

Una introducción (no demasiado breve) a ConT_EXt Mark IV

Una introducción (no demasiado breve) a ConT_EXt MarkIV

Versión 1.6 [2 de enero de 2021]

© 2020, Joaquín Ataz López

El autor del presente texto autoriza su libre distribución y uso, lo que incluye el derecho a copiar y redistribuir este documento en soporte digital con la condición de que se cite la autoría del mismo, y éste no se incluya en ningún paquete o conjunto de *software* o de documentación cuyas condiciones de uso o distribución no contemplen el libre derecho de copia y redistribución por parte de sus destinatarios. Se autoriza, asimismo, la traducción del documento, siempre que se indique la autoría del texto original y el texto traducido se distribuya bajo licencia FDL de la *Free Software Foundation*, licencia *Creative Commons* que autorice la copia y redistribución, o licencia similar.

No obstante lo anterior, la publicación y comercialización en papel de este documento, o de su traducción, requerirán autorización expresa y por escrito del autor.

Historial de versiones:

- 18 de agosto de 2020: Versión 1.0: Documento original.
- 23 de agosto de 2020: Versión 1.1: Corrección de pequeñas erratas y despistes del autor.
- 3 de septiembre de 2020: Versión 1.15: Más erratas y despistes.
- 5 de septiembre de 2020: Versión 1.16: Más erratas y despistes así como algunos pequeñísimos cambios que aumentan la claridad del texto (creo).
- 6 de septiembre de 2020: Versión 1.17: Es increíble la cantidad de pequeñas erratas que voy encontrando. Si quiero parar debo dejar de releer el documento.
- 21 de octubre de 2020: Versión 1.5: Introducción de sugerencias y corrección de errores reportados por usuarios de NTG-context.
- 2 de enero de 2021: Versión 1.6: Correcciones sugeridas tras una nueva y detenida lectura del documento, con ocasión de su traducción al inglés.

Sumario

Prefacio	6
I Qué es ConT_EXt y cómo se trabaja con él	14
1 Panorámica general de ConT_EXt	15
1.1 Pero entonces ¿Qué es ConT _E Xt?	15
1.2 Composición tipográfica de textos	16
1.3 Lenguajes de marcas	18
1.4 T _E X y sus derivados	19
1.5 ConT _E Xt	23
2 Nuestro primer fichero fuente	31
2.1 Preparación del experimento: Herramientas necesarias	31
2.2 El experimento propiamente dicho	33
2.3 La estructura de nuestro fichero de ejemplo	38
2.4 Algunos detalles adicionales sobre la forma de ejecutar «context»	39
2.5 Gestión de errores	40
3 Comandos y otros conceptos fundamentales de ConT_EXt	44
3.1 Los caracteres reservados de ConT _E Xt	45
3.2 Comandos propiamente dichos	48
3.3 Ámbito de aplicación de los comandos	52
3.4 Opciones de funcionamiento de los comandos	55
3.5 Recapitulación sobre la sintaxis de los comandos y de sus opciones, y sobre el uso de corchetes y llaves en las llamadas a los mismos	58
3.6 El listado oficial de comandos de ConT _E Xt	60
3.7 Definición de nuevos comandos	61
3.8 Otros conceptos fundamentales	65
3.9 Método para el autoaprendizaje de ConT _E Xt	69
4 Ficheros fuente y proyectos	72
4.1 Codificación de los ficheros fuente	72
4.2 Caracteres en el (o en los) fichero(s) fuente que ConT _E Xt trata de forma especial ..	75
4.3 Proyectos simples y proyectos multifichero	78
4.4 Estructura del fichero fuente en los proyectos simples	79
4.5 Gestión multifichero al estilo de T _E X	81
4.6 Proyectos de ConT _E Xt propiamente dichos	84

II Aspectos globales del documento	90
5 Páginas y paginación del documento	91
5.1 El tamaño de las páginas	91
5.2 Elementos en la página	96
5.3 El diseño de la página (<code>\setuplayout</code>)	99
5.4 Numeración de las páginas	103
5.5 Saltos de página forzados o sugeridos	106
5.6 Encabezados y pies de página	108
5.7 Inserción de elementos textuales en los bordes y márgenes de la página	112
6 Fuentes y colores en ConTeXt	115
6.1 Fuentes tipográficas incluidas en «ConTeXt Standalone»	115
6.2 Características de la fuente	116
6.3 Fijación de la fuente principal del documento	119
6.4 Cambiar la fuente o alguna de sus características	122
6.5 Otras cuestiones relacionadas con el uso de algunas alternativas	128
6.6 Uso y configuración de los colores	130
7 Estructura del documento	136
7.1 Divisiones estructurales en los documentos	136
7.2 Tipos de secciones y jerarquía de las mismas	138
7.3 Sintaxis común de los comandos de seccionado	139
7.4 Formateo y configuración de las secciones y sus títulos	141
7.5 Definición de nuevos comandos de seccionado	152
7.6 Macroestructura del documento	153
8 Índices	155
8.1 Índices de contenido	156
8.2 Listas, listas combinadas e índices creados a partir de una lista	167
8.3 Índices terminológicos	171
9 Remisiones e hiperenlaces	177
9.1 Tipos de remisiones	177
9.2 Remisiones internas	179
9.3 Documentos electrónicos interactivos	186
9.4 Hiperenlaces a documentos externos	188
9.5 Crear un índice de marcadores en el PDF final	192
III Cuestiones particulares	194
10 Caracteres, palabras, texto y espacio horizontal	195
10.1 Obtener caracteres no accesibles normalmente desde el teclado	195
10.2 Formatos especiales de los caracteres	204
10.3 Espacios de separación entre caracteres y palabras	209
10.4 Palabras compuestas	212
10.5 El idioma del texto	213

11	Párrafos, líneas y espaciado vertical	221
11.1	Formación y características de los párrafos	221
11.2	Espaciado vertical entre párrafos	224
11.3	Cómo construye ConT _E Xt las líneas que forman los párrafos	228
11.4	Interlineado	234
11.5	Otras cuestiones relacionadas con las líneas	235
11.6	Alineación horizontal y vertical	238
12	Construcciones y párrafos especiales	242
12.1	Notas al pie y notas finales	242
12.2	Párrafos con múltiples columnas	251
12.3	Listas estructuradas	256
12.4	Descripciones y enumeraciones	264
12.5	Líneas y marcos	267
12.6	Otros entornos y construcciones de interés	271
13	Imágenes, tablas y otros objetos flotantes	274
13.1	Qué son los objetos flotantes y para qué sirven	274
13.2	Imágenes externas	276
13.3	Tablas	284
13.4	Aspectos comunes a imágenes, tablas y otros objetos flotantes	291
13.5	Definición de objetos flotantes adicionales	297
Apéndices		300
A	Instalación, configuración y actualización de ConT_EXt	301
1	Instalación y configuración de «ConT _E Xt Standalone»	302
2	Instalación de LMTX	307
3	Usar en el mismo sistema varias versiones de ConT _E Xt (sólo para sistemas tipo Unix)	311
B	Comandos para generar símbolos, matemáticos y no matemáticos	312
C	Índice de comandos	316

Prefacio*

Gentil lector, he aquí un documento sobre ConT_EXt, un sistema de composición tipográfica derivado de T_EX, que, a su vez, es otro sistema de composición tipográfica diseñado entre 1977 y 1982 por DONALD E. KNUTH en la Universidad de Stanford.

ConT_EXt ha sido concebido para la creación de documentos de muy alta calidad tipográfica; bien se trate de documentos en papel, bien sean documentos pensados para ser mostrados en la pantalla de un dispositivo informático. No es un procesador de textos, o un programa de edición de textos, sino, como he dicho antes, un *sistema*, es decir: un conjunto de herramientas dirigidas a la composición tipográfica de documentos, entendiendo por tal la disposición gráfica y visual de los distintos elementos del documento en la página o en la pantalla. ConT_EXt, en suma, pretende proporcionar todas las herramientas necesarias para dotar a los documentos de la mejor apariencia posible. La idea es poder generar documentos que, además de bien escritos, sean «bellos». En tal sentido, podemos traer aquí a colación lo que escribió DONALD E. KNUTH, al presentar T_EX (el sistema en el que se basa ConT_EXt):

«Si usted simplemente desea producir un documento pasablemente bueno, algo aceptable y básicamente legible, pero no realmente hermoso, podrá conformarse con un sistema más simple. Pero con T_EX el objetivo es producir la más alta calidad, lo que requiere más atención al detalle; aunque no es tan difícil recorrer la distancia extra y, a cambio, podremos enorgullecernos especialmente del producto final.»

Cuando preparamos un manuscrito con ConT_EXt, indicamos exactamente cómo debe transformarse éste en páginas (o pantallas) cuya calidad tipográfica sea comparable a la que se obtendría con las mejores imprentas del mundo. Para ello, una vez que hayamos aprendido el sistema, no necesitaremos mucho más trabajo que el

* Este prefacio empezó intentando ser una traducción/adaptación a ConT_EXt del prefacio de «The T_EXBook», el documento en el que se explica *todo lo que es necesario saber sobre T_EX*. Finalmente tuve que desviarme del mismo; pero he dejado un par de fragmentos que espero que a quienes lo conozcan les ocasionen ciertas *resonancias* de él.

que se precisaría para teclear normalmente el documento en cualquier procesador o editor de textos. De hecho, una vez que tengamos cierta soltura en el manejo de ConT_EXt, nuestro trabajo total probablemente sea menor si tenemos en cuenta que en ConT_EXt los formatos principales del documento se describen globalmente y se trabaja con ficheros de texto que son —una vez uno se acostumbra— una forma mucho más natural de lidiar con la creación y edición de documentos; aparte de que este tipo de ficheros son mucho más ligeros y fáciles de manejar que los pesados ficheros binarios propios de los procesadores de texto.

Sobre ConT_EXt existe muchísima documentación, casi toda ella en inglés. La que podríamos considerar distribución *oficial* de ConT_EXt—llamada «ConT_EXt Standalone»¹—, por ejemplo, contiene unos 180 ficheros PDF de documentación (en inglés la mayoría, pero algunos en holandés o en alemán) que incluyen manuales, ejemplos y artículos técnicos; y en la web de Pragma ADE (la empresa donde nació ConT_EXt) se contienen (el día que hice el recuento, en mayo de 2020) 224 documentos descargables, donde están la mayor parte de los documentos distribuidos con «ConT_EXt Standalone», pero también algunos otros. Sin embargo esta ingente documentación, no es particularmente útil para aprender ConT_EXt, pues, en general, estos documentos no están pensados para un lector que no sepa nada del sistema y quiera aprenderlo. De los 56 ficheros PDF que «ConT_EXt Standalone» denomina «manuales», sólo hay uno que asume que el lector no sabe todavía nada de ConT_EXt. Se trata del documento titulado «ConT_EXt Mark IV, an Excursion», o, en español, «Una excursión por ConT_EXt Mark IV». Pero este documento, como su propio nombre indica, se limita a presentar el sistema, y explicar cómo se hacen algunas de las cosas que se pueden hacer con ConT_EXt. Sería una buena introducción si existiera después un manual de referencia algo más estructurado y sistemático. Pero, no existiendo ese manual, el salto entre el documento relativo a la excursión por ConT_EXt y el resto de la documentación es demasiado grande.

En 2001 sí se escribió un manual de referencia que se puede localizar en la [página web de Pragma ADE](#); pero, de un lado, a pesar de su título, no se había concebido para ser un *manual completo*, y de otro era (es) un texto pensado para la versión anterior de ConT_EXt (llamada Mark II) y que, por lo tanto, hoy está muy desactualizado. En 2013 el manual se actualizó parcialmente, pero muchas de sus secciones no llegaron a escribirse, y en él convive, además, información relativa a ConT_EXt Mark II y a ConT_EXt Mark IV (que es la versión actual), sin que quede siempre totalmente claro qué información se refiere a cada una de las versiones. Acaso esa sea la razón de que dicho manual no se encuentre entre los documentos

¹ En el momento en que se redactó la primera versión de este texto, lo que se dice en él era cierto; pero en la primavera de 2020 se actualizó la wiki de ConT_EXt y, desde entonces, hay que suponer que la distribución “oficial” de ConT_EXt ha pasado a ser LMTX. Aún así, para los recién llegados al mundo de ConT_EXt yo recomiendo que utilicen «ConT_EXt Standalone» pues es una distribución más estable. En el [apéndice A](#) se explica cómo instalar cualquiera de las dos distribuciones.

incluidos en «ConT_EXt Standalone». Pero a pesar de esos defectos, dicho manual, sigue siendo el mejor documento para iniciarse en el aprendizaje de ConT_EXt una vez que hemos leído la «excursión a ConT_EXt» introductoria. También es muy útil para iniciarse en ConT_EXt la información existente en su [wiki](#), que, en el momento de escribir estas líneas, se acaba de rediseñar y tiene una estructura mucho más clara, si bien también en ella se mezclan explicaciones que sólo funcionan en Mark II con otras para Mark IV o para ambas versiones. Falta de diferenciación ésta que también se da en el listado oficial de comandos de ConT_EXt¹ en el que no se precisa qué comandos funcionan sólo en alguna de las dos versiones.

Básicamente esta introducción se ha escrito a partir de esas cuatro fuentes de información: La excursión por ConT_EXt, el manual de 2013, el contenido de la wiki y el listado oficial de comandos que incluye, para cada uno de ellos, las opciones de configuración admisibles; además, claro está, de mis propias pruebas y conclusiones. Por ello, en realidad, esta introducción es el fruto de un trabajo de investigación, y durante algún tiempo estuve tentado de titularlo «Lo que sé de ConT_EXt Mark IV» o «Lo que he aprendido de ConT_EXt Mark IV». Finalmente descarté estos títulos por entender que, aunque sinceros, podían disuadir a alguien de adentrarse en ConT_EXt; y lo cierto es que aunque la documentación tenga (en mi opinión) deficiencias, se trata de una herramienta verdaderamente útil y versátil, en la que el esfuerzo que pueda suponer su aprendizaje merece, sin duda, la pena. Usando ConT_EXt podemos manipular y configurar los documentos de texto para conseguir cosas que quien no conoce el sistema simplemente no puede, ni siquiera, imaginar.

No puedo evitar —porque soy como soy— que mi lamento por la deficiente información se reproduzca de vez en cuando a lo largo de este documento. No quisiera que se malinterpretara: Estoy inmensamente agradecido a los creadores de ConT_EXt por haber diseñado una herramienta tan potente y haberla puesto a disposición del público. Es simplemente que no puedo evitar pensar que esta herramienta sería mucho más popular si su documentación mejorara: hay que invertir mucho tiempo en aprenderla, no tanto por su dificultad intrínseca (que la tiene, pero no es superior a la de otras herramientas similares sino más bien lo contrario), como por la falta de información clara, completa y sistematizada, que diferencie entre las dos versiones de ConT_EXt, explique qué funciona en cada una de ellas y, sobre todo, aclare para qué sirve cada comando, argumento y opción.

Es verdad que ese tipo de información exigiría una gran inversión de tiempo. Pero dado que muchos comandos comparten opciones con nombres similares, tal vez pudiera escribirse una especie de *glosario* de opciones, que ayudaría también a detectar algunas inconsistencias producidas cuando dos opciones del mismo nombre hacen cosas diferentes, o cuando para hacer lo mismo, se usan nombres de opciones diferentes en distintos comandos.

En cuanto al lector que se aproxime por primera vez a ConT_EXt, que mis quejas no le disuadan, pues aunque es cierto que la deficiente información incrementa el tiempo necesario para el aprendizaje, al menos para las materias tratadas en esta introducción, ese tiempo ya lo he

¹ Sobre este listado véase la [sección 3.6](#).

invertido yo, por lo que el lector no tendrá que hacerlo. Y con sólo lo que se puede aprender en esta introducción tendrá a su disposición una herramienta que le permitirá confeccionar documentos con utilidades que nunca sospechó.

Como lo que en este documento se explica está constituido en gran medida por mis propias conclusiones, es probable que, aunque he comprobado personalmente gran parte de lo que aquí se expone, algunas afirmaciones u opiniones sean incorrectas o poco ortodoxas. Agradeceré, por supuesto, cualquier corrección, matización o aclaración que se me quiera hacer, las cuales se me pueden enviar a joaquin@ataz.org. No obstante, para reducir las ocasiones en las que es probable que me equivoque he procurado no entrar en cuestiones sobre las que no he encontrado información y que no he podido (o querido) comprobar personalmente; a veces porque el resultado de mi comprobación no era concluyente, y otras veces porque no siempre lo he probado todo: el número de comandos y opciones de ConT_EXt es impresionante y si hubiera tenido que probarlo todo, nunca habría conseguido terminar esta introducción. Hay ocasiones, no obstante, en que no puedo evitar *conjeturar* algo, es decir: realizar una afirmación que veo probable pero de la que no estoy totalmente seguro. En estos casos, en el margen del párrafo donde se realiza tal suposición, se muestra la imagen que se puede ver a la izquierda de esta línea y que pretende representar gráficamente la conjetura¹. Otras veces no tengo más remedio que admitir que algo no lo sé y ni siquiera tengo al respecto una suposición razonable: en este segundo supuesto, la imagen que se inserta en el margen del párrafo es la que se ve a la izquierda de esta segunda línea²: Intenta representar más que la simple conjetura el desconocimiento. Pero como nunca he sido muy bueno con las representaciones gráficas, no estoy seguro de que las imágenes que he seleccionado realmente consigan transmitir tantos matices.



Esta introducción, por otra parte, ha sido escrita con la mente puesta en un lector que no sepa nada de T_EX ni de ConT_EXt, aunque espero que también pueda ser útil a quienes desde T_EX o desde L^AT_EX (el más popular de los derivados de T_EX) se aproximen por primera vez a ConT_EXt. Soy, no obstante, consciente, de que al intentar complacer a lectores tan diferentes, corro el riesgo de no conseguir satisfacer a nadie. Por ello, en caso de duda, he tenido siempre claro que el destinatario principal de este documento es el neófito en ConT_EXt; el recién llegado a este fascinante ecosistema.

Ser novato en ConT_EXt no implica serlo también en el manejo de herramientas informáticas; y aunque en esta introducción no asumo ningún nivel concreto de competencia informática en los lectores, si supongo un cierto “manejo razonable” que implique, por ejemplo, conocer en líneas generales la diferencia entre un editor

¹ La imagen no es diseño propio, la he descargado de Internet (<https://es.dreamstime.com/>), donde se informa de que es una imagen libre de derechos.

² Igualmente localizada en Internet (<https://www.freepik.es/>) donde se autoriza su uso gratuito.

y un procesador de textos, saber crear, abrir y manipular un fichero de texto, saber instalar un programa, saber abrir una terminal y ejecutar en ella un comando... y poco más.

Leyendo las partes de esta introducción ya escritas en el momento de redactar estas líneas, me doy cuenta de que en ocasiones me dejo llevar y entro en cuestiones informáticas que no son necesarias para el aprendizaje de ConT_EXt y que pueden espantar al neófito, mientras que otras veces, me entretengo en explicar cosas bastante evidentes que pueden aburrir al lector experimentado. Ruego la indulgencia de unos y otros. Racionalmente se que es muy difícil que un completo novato en el tratamiento informático de textos llegue siquiera a conocer la existencia de ConT_EXt, pero, desde otro punto de vista, en mi entorno profesional estoy rodeado de personas que batallan continuamente con textos usando procesadores de textos, y lo hacen razonablemente bien, pero sin embargo, al no haber trabajado nunca con ficheros de texto, ignoran cuestiones tan elementales como, por ejemplo, qué es la codificación o cuál es la diferencia entre un editor y un procesador de textos.

El que este manual haya sido concebido para personas que no sepan nada de ConT_EXt ni de T_EX, implica que en él he incluido información que, en puridad, no corresponde a ConT_EXt sino a T_EX; pero he entendido que no era preciso sobrecargar al lector con información que para él es poco relevante como puede ser la de si cierto comando que *de facto* funciona es realmente de ConT_EXt o pertenece a T_EX; por ello sólo en algunas ocasiones en las que me ha parecido útil, se aclara que ciertos comandos pertenecen en realidad a T_EX.

Respecto a la organización de este documento, la materia se agrupa en tres bloques:

- **La primera parte**, que comprende los cuatro capítulos iniciales, ofrece una panorámica global de ConT_EXt, explicando qué es y cómo se trabaja con él, mostrando un primer ejemplo de cómo transformar un documento, para después explicar algunos conceptos fundamentales de ConT_EXt así como ciertas cuestiones relativas a los ficheros fuente de ConT_EXt.

En su conjunto estos capítulos están pensados para lectores que hasta ahora solamente hayan conocido el trabajo con procesadores de texto. Un lector que ya conociera de antemano el trabajo con lenguajes de marcas, puede prescindir de los dos primeros capítulos; y si el lector ya conoce T_EX, o L^AT_EX, puede prescindir también de una buena parte del contenido de los capítulos 3º y 4º. Pero, aún así, yo le recomendaría que, al menos, leyera:

- La información relativa a los comandos de ConT_EXt (capítulo 3º), y en particular a la configuración de su funcionamiento, pues es allí donde reside la principal diferencia en la concepción y en la sintaxis de L^AT_EX y de ConT_EXt. Como esta introducción se refiere sólo a éste último, dichas diferencias no están expresamente señaladas como tales, pero quien lea dicho capítulo y conozca el funcionamiento de L^AT_EX, inmediatamente comprenderá las principales diferencias en la sintaxis de ambos lenguajes, así como la manera

en la que ConT_EXt permite que se configure y personalice el funcionamiento de casi todos sus comandos.

- La información relativa a los proyectos multifichero de ConT_EXt (capítulo 4º), que no se parece demasiado a la forma de trabajar de otros sistemas basados en T_EX.
- **La segunda parte**, que incluye los capítulos 5 a 9, se centra en los que podemos considerar principales aspectos globales de un documento de ConT_EXt:
 - Los dos aspectos que principalmente afectan a la apariencia de un documento son el tamaño y diseño de sus páginas y la fuente utilizada. A estas dos cuestiones se dedican los capítulos 5 y 6.
 - ★ El primero se centra en las páginas: tamaño, elementos que la componen, diseño (es decir, distribución en la página de sus distintos elementos), etc. Por razones sistemáticas se tratan aquí también aspectos más concretos como son los relativos a la paginación y a los mecanismos que nos permiten influir en ella.
 - ★ El capítulo 6 explica los comandos relacionados con las fuentes y su manipulación. Se incluye también aquí una explicación básica del uso y manipulación de los colores ya que, aunque éstos no son, en sentido estricto, una *característica* de la fuente, influyen igual que estas en la apariencia externa del documento.
 - Los capítulos 7 y 8 se concentran en la estructura del documento y en las herramientas que ConT_EXt pone a disposición del autor para escribir documentos bien estructurados. El capítulo 7 se concentra en la estructura propiamente dicha (divisiones estructurales del documento) y el capítulo 8 en el reflejo de la misma que tiene lugar en los índices de contenido; aunque, al hilo de la explicación de este tipo de índices, se aprovecha para explicar cómo generar con ConT_EXt índices analíticos y de otro tipo.
 - Por último, el capítulo 9 se concentra en un aspecto global del documento tan importante como es el de las remisiones en un documento a otros puntos del mismo documento (remisiones internas) y a otros documentos (remisiones externas) pero de estas últimas remisiones sólo nos ocupamos si la remisión se hace incluyendo un enlace que propicie un salto al documento externo. Estos *saltos* (que también pueden darse en las remisiones internas) convierten a nuestro documento en *interactivo*, y en este capítulo se explican algunas utilidades de ConT_EXt para la creación de dicho tipo de documentos.

Estos capítulos no necesitan ser leídos siguiendo ningún orden concreto, salvo acaso el capítulo 8, que posiblemente sea más fácil de entender si primero se ha

leído el 7. En todo caso he procurado que cuando en un capítulo o sección surja una cuestión que es objeto de tratamiento en otro punto de esta introducción, en el texto se incluya una mención de dicha circunstancia junto con un hipere enlace al punto en que dicha cuestión es objeto de tratamiento. No estoy, no obstante, en condiciones de garantizar que eso ocurra siempre.

- Finalmente **la tercera parte** (capítulos 10 y siguientes) se centra ya en aspectos más concretos. Ahora no sólo son independientes unos capítulos de otros, sino incluso unas secciones de otras (salvo, acaso, en el último capítulo). Dada la gran cantidad de utilidades que ConT_EXt incorpora esta tercera parte podría ser muy amplia; pero como, por otra parte, entiendo que al llegar aquí el lector ya estará preparado para bucear por su cuenta en la documentación de ConT_EXt, sólo he incluido en ella los siguientes capítulos:
 - Los capítulos 10 y 11 se ocupan de los que podríamos llamar *elementos nucleares* de todo documento de texto: El texto se compone de caracteres, los cuales forman palabras, que se agrupan en líneas, las cuales, a su vez, forman párrafos que se separan unos de otros mediante espacio vertical... En puridad todas estas cuestiones podrían haberse incluido en un solo capítulo; pero como este quedaría demasiado largo, he dividido esta materia en dos capítulos, uno que se ocupa de caracteres, palabras y espacio horizontal, y otro que se ocupa de líneas, párrafos y espacio vertical.
 - El capítulo 12 es una especie de *cajón de sastre* en el que se tratan elementos y construcciones que es habitual encontrar en numerosos documentos; principalmente si son académicos o científico-técnicos: Notas a pie de página, listas estructuradas, descripciones, enumeraciones, etc.
 - Por último el capítulo 13 se centra en los objetos flotantes y en sus dos casos más paradigmáticos: las imágenes insertadas en los documentos y las tablas.
- La introducción se cierra con tres **apéndices**. Uno relativo a la instalación de ConT_EXt, un segundo apéndice que recoge varias decenas de comandos que permiten generar símbolos variados; principalmente, pero no sólo, de uso matemático y un tercer apéndice que contiene un listado alfabético de comandos de ConT_EXt explicados o mencionados a lo largo de este texto.

Faltan muchas cuestiones por explicar: el tratamiento de las citas y referencias bibliográficas, la escritura de textos especiales (matemáticos, químicos...), la conexión con XML, la interfaz para código de Lua, los modos y la compilación basada en modos, el trabajo con MetaPost para el diseño de gráficos, etc. Por ello, porque no incluye una explicación completa de ConT_EXt, ni pretende hacerlo, he titulado a este documento como «Una introducción a ConT_EXt Mark IV»; y he añadido entre paréntesis la observación de que la introducción no es demasiado breve

porque, obviamente, así es: un texto que se deja tantísimas cosas en el tintero y que aún así sobrepasa las 300 páginas no es, desde luego, una introducción breve. Eso es porque intento en ella que el lector comprenda la lógica de ConT_EXt; o, al menos, la lógica tal y como yo la comprendo. No pretende ser un manual de referencia, sino, más bien, una guía para el autoaprendizaje que prepare al lector para confeccionar documentos de complejidad mediana (lo que incluye la mayor parte de los documentos posibles) y que, sobre todo, le enseñe a *imaginar* lo que se puede hacer con esta poderosa herramienta y a localizar en la documentación el cómo hacerlo. Este documento tampoco es un *tutorial*. Los tutoriales están pensados para ir incrementando progresivamente la dificultad, de tal modo que aquello que se pretende enseñar se vaya aprendiendo paso a paso; yo, en este sentido, he preferido, a partir de la segunda parte, en lugar de ordenar la materia atendiendo a su dificultad, ser más sistemático. Pero, aún no siendo un tutorial, sí he incluido abundantes ejemplos.

Es posible que a algunos lectores el título de este documento les recuerde el de un texto, escrito por OETIKER, PARTL, HYNÄ y SCHLEGL que está disponible en Internet y que es uno de los mejores documentos para introducirse en el mundo de L^AT_EX. Se trata de «*The Not So Short Introduction to L^AT_EX 2_ε*». Ello no es una coincidencia, sino un homenaje y un acto de agradecimiento: gracias a la generosa labor de quienes escriben textos como ese, es posible que muchas personas se inicien en el manejo de herramientas útiles y poderosas como son L^AT_EX y ConT_EXt. Dichos autores me ayudaron a iniciarme en L^AT_EX; yo pretendo hacer lo mismo para quien se quiera iniciar en ConT_EXt, aunque yo me ciño exclusivamente al público hispanoparlante que, de otro lado, tan falto está de documentación en su idioma. Espero que este documento cumpla con dicha finalidad.

Joaquín Ataz López
Verano de 2020

I

Qué es ConT_EXt y cómo se trabaja con él

Capítulo 1

Panorámica general de ConT_EXt

Sumario: 1.1 Pero entonces ¿Qué es ConT_EXt?; 1.2 Composición tipográfica de textos; 1.3 Lenguajes de marcas; 1.4 T_EX y sus derivados; 1.4.1 Motores de T_EX; 1.4.2 Formatos derivados de T_EX; 1.5 ConT_EXt; 1.5.1 Breve historia de ConT_EXt; 1.5.2 ConT_EXt versus L^AT_EX; 1.5.3 Comprender bien la dinámica de trabajo en ConT_EXt; 1.5.4 Obtener ayuda sobre ConT_EXt;

1.1 Pero entonces ¿Qué es ConT_EXt?

ConT_EXt es un *sistema de composición tipográfica*, es decir: Un amplio conjunto de herramientas dirigidas a otorgar al usuario un absoluto y completo control sobre el aspecto, apariencia y prestaciones de un determinado documento electrónico, pensado para ser impreso en papel o para ser mostrado en pantalla. En este capítulo se explicará qué significa eso. Pero antes, destaquemos alguna de las características de ConT_EXt.

- Existen dos *sabores* de ConT_EXt llamados, respectivamente, Mark II y Mark IV. ConT_EXt Mark II está congelado, es decir: se considera un lenguaje ya plenamente desarrollado al que no está previsto que se le introduzcan cambios o utilidades nuevas. Solamente en el caso de que se detecte algún fallo que deba ser corregido aparecerá una versión nueva. ConT_EXt Mark IV, por el contrario, sigue en la actualidad en desarrollo, por lo que con cierta periodicidad aparecen versiones nuevas que introducen alguna mejora o utilidad adicional. Pero, aunque aún en desarrollo, es un lenguaje ya muy maduro, en el que los cambios introducidos por las nuevas versiones son muy sutiles y afectan exclusivamente al funcionamiento a bajo nivel del sistema. Para el usuario medio estos cambios son totalmente transparentes; es como si no existieran. Y aunque ambos *sabores* tienen mucho en común, también hay entre ellos algunas incompatibilidades; por ello esta introducción se centra sólo en ConT_EXt Mark IV.
- ConT_EXt es software libre. El programa propiamente dicho (es decir, el conjunto de herramientas informáticas que componen ConT_EXt) se distribuye bajo la licencia *GNU General Public License*. La documentación se proporciona bajo licencia «*Creative Commons*» que permite copiarla y distribuirla libremente.

- ConT_EXt no es un programa de procesamiento de textos o un programa de edición de textos, sino un conjunto de herramientas diseñadas para *transformar* un texto que previamente hayamos escrito con nuestro editor de textos favorito. Por lo tanto, cuando trabajamos con ConT_EXt:
 - Empezamos por escribir uno o varios ficheros de texto con un editor de textos cualquiera.
 - En esos ficheros, junto con el texto que constituye el contenido propiamente dicho del documento, se incluyen una serie de instrucciones que indican a ConT_EXt qué apariencia debe tener el documento final que se genere a partir de los ficheros de texto originales. El conjunto completo de instrucciones de ConT_EXt constituye, en realidad, un *lenguaje*; y como ese lenguaje permite *programar* la transformación tipográfica de un texto, puede decirse que ConT_EXt es un *lenguaje de programación tipográfica*.
 - Una vez que hemos escrito los ficheros fuente, éstos serán procesados por un programa (que se llama también «**context**»¹), el cual, a partir de ellos, generará un fichero PDF preparado para ser enviado a una impresora o para ser mostrado en pantalla.
- En ConT_EXt, por lo tanto, debemos diferenciar entre el documento que nosotros escribimos, y el documento que ConT_EXt genera. Para evitar dudas, en esta introducción llamaré *fichero fuente* al documento de texto que contiene las instrucciones de formateo, y *documento final* al fichero PDF generado por ConT_EXt a partir del fichero fuente.

A continuación se desarrollarán algo más los anteriores puntos básicos.

1.2 Composición tipográfica de textos

Escribir un documento (libro, artículo, capítulo, folleto, impreso, cartel ...) y componerlo tipográficamente son dos actividades muy diferentes. Escribir el documento es tanto como redactarlo; cosa que hace el autor, que es quien decide su contenido y estructura. Al documento creado directamente por el autor, tal y como él lo

¹ ConT_EXt es simultáneamente un lenguaje y un programa (además de algunas otras cosas). Esta circunstancia, en un texto como el presente, plantea el problema de que a veces hay que distinguir entre los dos aspectos. Por ello he adoptado la convención tipográfica de escribir «ConT_EXt» con su logotipo (ConT_EXt) cuando me quiera referir exclusivamente al lenguaje, o indistintamente al lenguaje y al programa. Pero cuando me quiera referir exclusivamente al programa, escribiré «**context**» todo en minúsculas y con el tipo de letra monoespaciada propio de las terminales informáticas y de las máquinas de escribir que, por otra parte, utilizo también en los ejemplos y en las menciones de los comandos del lenguaje.

escribió, se le llama *manuscrito*. Al manuscrito, por su propia naturaleza, sólo tienen acceso el autor y aquellas personas a quienes éste permita leerlo. Su difusión más allá de este círculo íntimo, requiere que el manuscrito sea *publicado*. Hoy día publicar algo —en el sentido etimológico de hacerlo «accesible al público»— es tan sencillo como ponerlo en Internet, a disposición de cualquiera que lo pueda localizar y lo quiera leer. Pero hasta hace relativamente poco la publicación era un proceso que acarreaba costes, dependía de ciertos profesionales especializados en ella, y al que sólo accedían aquellos manuscritos que, por su contenido, o por su autor, se estimaban especialmente interesantes. Y todavía hoy tendemos a reservar la palabra *publicación* para ese tipo de *publicación profesional* en la que el manuscrito experimenta una serie de transformaciones en su apariencia dirigidas a mejorar la *legibilidad* del documento. Es a esta serie de transformaciones a las que se llama *composición tipográfica*.

El objetivo de la composición tipográfica es —en general, y dejando de lado los textos publicitarios que persiguen atraer la atención del lector— conseguir documentos con la máxima *legibilidad*, entendiendo por tal aquella cualidad de un texto impreso que invita a su lectura, o la facilita, y hace que el lector se sienta cómodo con ella. A que esto ocurra contribuyen numerosos aspectos; algunos, por supuesto, tienen que ver con *el contenido* del documento (calidad, claridad, sistematica...), pero otros dependen de cuestiones tales como el tipo y tamaño de letra utilizado, la distribución de espacios en blanco en la página, la separación visual entre párrafos, etc.; además de otro tipo de recursos, no tan gráficos o visuales, como la existencia o no en el documento de determinadas ayudas al lector tales como encabezados o pies de página, índices, glosarios, negritas, titulillos en los márgenes, etc. Al conocimiento y correcto manejo de todos los recursos a disposición de un compositor tipográfico lo podríamos llamar «arte de la composición tipográfica» o «arte de la impresión».

Históricamente, y hasta el advenimiento de la informática, las tareas y papeles del escritor y del compositor tipográfico se mantuvieron netamente diferenciadas. El autor escribía a mano o, desde mediados del siglo XIX, en una «máquina de escribir» cuyos recursos tipográficos eran incluso más limitados que los de la escritura a mano; y luego entregaba sus originales a la editorial o a la imprenta que se encargaba de transformarlos para, a partir de ellos, obtener el documento impreso.

Hoy día la informática ha facilitado que sea el propio autor quien decida la composición hasta en sus últimos detalles. Pero eso no borra el hecho de que las cualidades que necesita un buen autor no son las mismas que las que necesita un buen compositor tipográfico. El autor requiere, dependiendo del tipo de documento de que se trate, conocimiento de la materia sobre la que escribe, claridad expositiva, una mente bien estructurada que le permita crear un texto bien sistematizado, creatividad, sentido del ritmo, etc. Pero el compositor tipográfico tiene que reunir un

buen conocimiento de los recursos gráficos y conceptuales a su disposición, y el suficiente buen gusto para utilizarlos de forma armónica.

Con un buen programa procesador de textos¹ es posible conseguir una composición tipográfica razonablemente buena. Pero los procesadores de textos no están en general pensados para la composición tipográfica y sus resultados, aunque pueden ser correctos, no son comparables a los que se obtienen con otras herramientas diseñadas específicamente para controlar la composición del documento. De hecho los procesadores de texto son la evolución de las máquinas de escribir, y su uso, en la medida en que estas herramientas enmascaran la diferencia entre la autoría del texto y su composición tipográfica, tiende a producir textos desestructurados y tipográficamente inadecuados. Por el contrario, las herramientas como ConT_EXt son la evolución de la imprenta; ofrecen muchísimas más posibilidades de composición y, sobre todo, no es posible aprender a manejarlas sin adquirir también, en el camino, numerosas nociones relativas a la composición tipográfica, a diferencia de los procesadores de texto, que pueden ser usados durante muchos años sin que se llegue a aprender ni una palabra de tipografía.

1.3 Lenguajes de marcas

En los tiempos previos a la informática, como he dicho antes, el autor preparaba su manuscrito a mano o a máquina y lo entregaba al editor o impresor que era quien se ocupaba de la transformación del manuscrito en el texto impreso definitivo. Aunque en dicha transformación el autor intervenía relativamente poco, sí mantenía cierta intervención señalando, por ejemplo, que ciertas líneas del manuscrito eran los títulos de sus distintas partes (capítulos, secciones ...); o indicando que ciertos fragmentos debían destacarse tipográficamente de alguna manera. Estas indicaciones las hacía el autor en el propio manuscrito, a veces expresamente, y otras veces mediante ciertas convenciones que, con el paso del tiempo, se fueron desarrollando; y así, por ejemplo, los capítulos siempre se iniciaban en una página nueva, insertando varias líneas en blanco antes del título, subrayándolo, escribiéndolo con mayúsculas; o enmarcando el texto que se debía resaltar entre dos guiones bajos, aumentando el sangrado de un párrafo, etc.

El autor, en definitiva, *marcaba* el texto para dar algunas indicaciones relativas a la composición tipográfica del mismo. Y posteriormente el editor, escribía a mano en el original otras indicaciones para el impresor tales como, por ejemplo, el tipo de letra, o el tamaño.

¹ Por convención bastante antigua, se distingue entre los programas de *edición de textos* y los *procesadores de texto*. Los primeros manipulan ficheros de texto sin formato, y los segundos trabajan con ficheros binarios de texto formateado.

Hoy día, en un mundo informatizado, podemos seguir haciendo eso mismo para la generación de documentos electrónicos, mediante lo que se llama un *lenguaje de marcas*. En este tipo de lenguajes se utilizan una serie de *marcas* o indicaciones que el programa que procesa el fichero que las contiene sabe interpretar. Posiblemente a día de hoy el lenguaje de marcas más conocido del mundo sea HTML, pues en él se basan la mayoría de las páginas web. Un fichero HTML contiene el texto de una página web, junto con una serie de marcas que le indican al programa navegador con el que se carga la página, cómo debe mostrarse la misma. Al conjunto de marcas HTML comprensibles por los navegadores web, junto con las instrucciones relativas a cómo y dónde usarlas, se le llama «lenguaje HTML», el cual es un *lenguaje de marcas*. Pero además de HTML, hay muchos otros lenguajes de marcas; de hecho éstos están francamente en auge y así XML, que es el lenguaje de marcas por excelencia, resulta hoy día absolutamente omnipresente y se usa para casi todo: para diseño de bases de datos, para creación de lenguajes específicos, transmisión de datos estructurados, ficheros de configuración de aplicaciones, etc. Hay también lenguajes de marcado pensados para diseñar gráficos (SVG, TikZ o MetaPost), fórmulas matemáticas (MathML), música (Lilypond y MusicXML), finanzas, geomática, etc. También, por supuesto, los hay para la transformación tipográfica de textos, y entre ellos destacan T_EX y sus derivados.

A propósito de las marcas *tipográficas*, que indican el aspecto que debe tener un texto, las hay de dos tipos a los que podríamos referirnos como *marcado puramente tipográfico* y *marcado conceptual* o, si se prefiere, *marcado lógico*. Las marcas puramente tipográficas se limitan a indicar exactamente con qué recurso tipográfico se debe mostrar cierto texto; como cuando, por ejemplo, indicamos que cierto texto debe ir en negrita o en cursiva. El marcado conceptual, por el contrario, indica qué función cumple cierto texto en el total del documento, como cuando indicamos que algo es un título, o un subtítulo, o una cita. En general son más coherentes y más fáciles de componer los documentos que utilizan preferentemente este segundo tipo de marcas, pues en ellas se vuelve a manifestar la diferencia entre la autoría y la composición: el autor indica que esta línea es un título, o que este fragmento es una advertencia, o una cita; y el compositor decide cómo destacar tipográficamente todos los títulos, advertencias o citas; con lo que, de un lado, se garantiza la coherencia, ya que todos los fragmentos que cumplan la misma función tendrán el mismo aspecto, y, de otro, se ahorra tiempo, ya que el formato de cada tipo de fragmento sólo hay que indicarlo una vez.

1.4 T_EX y sus derivados

T_EX fue desarrollado a finales de los años 70 por DONALD E. KNUTH, profesor de teoría de la programación en la Universidad de Stanford, que implementó el programa para componer sus propias publicaciones y como ejemplo de un programa sistemáticamente desarrollado y anotado. Junto con T_EX, KNUTH desarrolló un

lenguaje de programación adicional llamado MetaFont, pensado para el diseño de fuentes tipográficas, con el que creó una fuente a la que bautizó como *Computer Modern*, la cual, junto con los caracteres habituales de cualquier fuente, incluía también un juego completo de «glifos»¹ pensados para la escritura de las matemáticas. A todo esto le añadió algunas utilidades adicionales y así nació el sistema de composición tipográfica llamado T_EX, que, por su potencia, calidad de resultados, flexibilidad de uso y amplias posibilidades, está considerado uno de los mejores sistemas informáticos de composición de textos. Fue pensado para textos en los que hubiera muchas matemáticas, pero pronto se vio que las posibilidades del sistema lo hacían idóneo para todo tipo de textos.

Internamente T_EX funciona como lo haría el cajista de una imprenta; porque para T_EX todo son *cajas*: Las letras se contienen en cajas, los espacios en blanco son también cajas, varias letras (las cajas que contienen varias letras) forman una caja nueva que encierra la palabra, y varias palabras, junto con el espacio en blanco entre ellas, forman una caja que contiene una línea, varias líneas se convierten en una caja que contiene el párrafo ... y así sucesivamente. Todo ello, además, con una precisión extraordinaria en el manejo de las medidas. Piénsese que la unidad más pequeña que maneja T_EX es 65 536 veces más pequeña que el punto tipográfico, con el que se miden los caracteres y las líneas, que suele ser la unidad más pequeña manejada por la mayor parte de los programas de procesado de textos. Ello significa que la unidad más pequeña que maneja T_EX es de, aproximadamente, 0.000005356 milímetros.

El nombre T_EX proviene de la raíz de la palabra griega τεχνη, escrita en mayúsculas (ΤΕΧΝΗ). Por ello, como la letra final del nombre T_EX no es una «X» latina, sino la «χ» griega, que se pronunciaba —según parece— como la «j» española, T_EX se debe pronunciar «Tej». Dicha palabra griega, por otra parte, significaba tanto «arte» como «técnica», siendo esta la razón de que KNUTH la eligiera para bautizar a su sistema. El propósito de este nombre —escribió— «es para recordarnos que T_EX se ocupa principalmente de manuscritos técnicos de alta calidad. Su énfasis está en el arte y en la tecnología, al igual que la palabra griega subyacente».

Por convención establecida por KNUTH el nombre de T_EX ha de escribirse:

- En textos tipográficamente formateados como el presente, mediante el logotipo que hasta ahora he venido usando: Las tres letras en mayúsculas, con la «E» central ligeramente desplazada hacia abajo para facilitar una mayor aproximación entre la «T» y la «X»; o sea: «T_EX».

Para facilitar la escritura de dicho logotipo, Knuth incluyó en T_EX una instrucción que lo escribe en el documento final: `\TeX`.

¹ En tipografía, un glifo es una representación gráfica de un carácter, de varios caracteres o de parte de un carácter y es el equivalente actual del tipo de imprenta (la pieza que tenía grabada la letra).

- En textos no formateados (como un correo electrónico, o un fichero de texto), con la «T» y la «X» en mayúscula, y la «e» central en minúscula; o sea: «T_EX».

Esta convención se viene siguiendo en todos los derivados de T_EX que lo incluyen en su propio nombre, como ocurre con ConT_EXt, que cuando se escribe en modo de texto ha de escribirse «ConT_EXt».

1.4.1 Motores de T_EX

El programa T_EX es software libre: su código fuente está al alcance del público y quien quiera puede utilizarlo o modificarlo a su gusto, con la única condición de que, si se introducen modificaciones, al resultado no se le puede llamar «T_EX». Esta es la razón de que, a lo largo del tiempo, hayan ido surgiendo ciertas adaptaciones del programa, que introducían diferentes mejoras en el mismo, y a las que en general se las denomina *motores de T_EX*. Aparte del programa original, los principales motores de T_EX son, por orden cronológico de aparición, pdfT_EX, ϵ -T_EX, X_YT_EX y LuaT_EX. Cada uno de ellos se supone que incorpora las mejoras del anterior. Estas mejoras, por otra parte, hasta la aparición de LuaT_EX, no afectaron al lenguaje propiamente dicho, sino exclusivamente a los ficheros de entrada, los ficheros de salida, el manejo de las fuentes y el funcionamiento a bajo nivel de las macros.

La cuestión de qué motor de T_EX utilizar es muy debatida en el Universo de T_EX. No la desarrollaré aquí porque ConT_EXt Mark IV sólo funciona con LuaT_EX. En realidad en el mundo de ConT_EXt la discusión sobre los *motores* de T_EX pasa a ser una discusión sobre si usar Mark II (que funciona con PdfT_EX y con XeT_EX) o Mark IV (que sólo funciona con LuaT_EX).

1.4.2 Formatos derivados de T_EX

El núcleo o corazón de T_EX sólo entiende un conjunto de aproximadamente 300 instrucciones muy básicas, llamadas *primitivas*, que son adecuadas para las operaciones de composición tipográfica y para funciones de programación. Estas instrucciones, en su gran mayoría, son de un muy *bajo nivel*, lo que en terminología informática significa que son más fácilmente comprensibles por el ordenador que por los seres humanos, pues se refieren a operaciones muy elementales del tipo «desplaza este carácter 0.000725 milímetros hacia arriba». Por ello KNUTH hizo que T_EX fuera extensible; es decir: que hubiera un mecanismo que permitiera definir instrucciones de más alto nivel, más fácilmente comprensibles por los seres humanos. A estas instrucciones, que en el momento de la ejecución se descomponen en otras instrucciones más simples, se las llama *macros*. Por ejemplo, la instrucción de T_EX que imprime su logotipo (`\TEX`), al ejecutarse se descompone en:

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

Pero para un ser humano es mucho más sencillo comprender y recordar que el simple comando «\T_EX» realiza las operaciones tipográficas necesarias para imprimir el logotipo.

La diferencia entre lo que son *macros* y lo que son *primitivas*, en realidad sólo tiene importancia desde el punto de vista del desarrollador de T_EX. Desde el punto de vista del usuario todo son *instrucciones* o, si se prefiere, *comandos*. Knuth las llamaba *secuencias de control*.

Esta posibilidad de extender el lenguaje mediante *macros* es una de las características que han convertido a T_EX en una herramienta tan potente. De hecho el propio KNUTH diseñó aproximadamente 600 macros que, junto con las 300 primitivas componen el formato denominado «Plain T_EX». Es bastante corriente confundir a T_EX propiamente dicho, con Plain T_EX y, de hecho, casi todo lo que se suele decir o escribir sobre T_EX, se refiere en realidad a Plain T_EX. Los libros que dicen tratar sobre T_EX (incluyendo el libro fundacional «*The T_EXBook*»), en realidad se refieren a Plain T_EX; y quienes creen manejar directamente T_EX en realidad están manejando Plain T_EX.

Plain T_EX es lo que en terminología de T_EX se llama un *formato*, consistente en un conjunto amplio de macros, junto con ciertas reglas de sintaxis relativas a cómo y de qué manera utilizarlos. Además de Plain T_EX se han desarrollado, con el paso del tiempo, otros *formatos* entre los que cabe destacar a L^AT_EX un amplio conjunto de macros para T_EX diseñado en 1985 por LESLIE LAMPORT y que probablemente es el derivado de T_EX más utilizado en el mundo académico, tecnológico y matemático. ConT_EXt es (o empezó siendo), al igual que L^AT_EX un formato derivado de T_EX.

Normalmente estos *formatos* van acompañados de un programa que carga en memoria las macros que los componen antes de llamar a «*tex*» (o al concreto motor que se utilice en la compilación) para procesar el fichero fuente. Pero aunque todos estos formatos, en realidad estén ejecutando T_EX, como cada uno de ellos tiene instrucciones distintas, y reglas de sintaxis diferentes, desde el punto de vista del usuario, podemos considerarlos *lenguajes distintos*. Todos ellos inspirados en T_EX, pero diferentes de T_EX y diferentes también entre ellos.

1.5 ConT_EXt

En realidad ConT_EXt que empezó siendo un *formato* de T_EX, hoy día es bastante más que eso. ConT_EXt incluye:

1. Un amplísimo conjunto de macros de T_EX. Si Plain T_EX consta de en torno a 900 instrucciones, ConT_EXt se aproxima a las 3500; y si sumamos los nombres de las distintas opciones que tales comandos admiten, estaremos hablando de un vocabulario en torno a las 4000 palabras. El vocabulario es así de amplio debido a que la estrategia de ConT_EXt para facilitar su aprendizaje, pasa por incluir numerosos sinónimos de comandos y opciones.

Lo que se pretende es que si se quiere conseguir cierto efecto, para cada una de las formas en las que un hablante de inglés llamaría a ese efecto, haya un comando o una opción que lo logre; lo que se supone que hace más sencillo el uso del lenguaje. Por ejemplo para conseguir simultáneamente una letra en negrita (en inglés *bold*) y en cursiva (en inglés *italic*), ConT_EXt contiene tres instrucciones idénticas en su resultado: `\bi`, `\italicbold` y `\bolditalic`.

2. Un también bastante amplio conjunto de macros para MetaPost, un lenguaje de programación gráfica derivado de MetaFont, que, a su vez, es el lenguaje de diseño de fuentes tipográficas que KNUTH desarrolló conjuntamente con T_EX.
3. Varios *scripts* desarrollados en PERL (los más antiguos), RUBY (algunos también antiguos y otros no tanto) y LUA (los más recientes).
4. Una interfaz que integra T_EX, MetaPost, LUA y XML, permitiendo escribir y procesar documentos en cualquiera de estos lenguajes, o que mezclen elementos de algunos de ellos.

¿No ha entendido gran cosa de la explicación anterior? No se preocupe. En ella he empleado mucha jerga informática y he mencionado muchos programas y lenguajes. Pero para usar ConT_EXt no es preciso saber de dónde vienen sus distintos componentes. Lo importante, a estas alturas del aprendizaje, es quedarse con la idea de que ConT_EXt integra numerosas herramientas de procedencias diferentes que forman un *sistema de composición tipográfica*.

Es por esta última característica de integración de herramientas de orígenes diversos, por lo que se dice de ConT_EXt que constituye una «tecnología híbrida» orientada a la composición tipográfica de documentos. Lo que entiendo que convierte a ConT_EXt en un sistema extraordinariamente avanzado y potente.

Pero aunque ConT_EXt sea mucho más que un conjunto de macros para T_EX, su base sigue estando en T_EX, y por ello este documento, que no pretende ser más que una *introducción*, se centra en tal aspecto.

ConT_EXt, por otra parte, es bastante más moderno que T_EX. Cuando T_EX se diseñó, apenas empezaba la eclosión de la informática, y se estaba todavía lejos

de vislumbrar lo que sería (lo que llegaría a ser) Internet, o el mundo multimedia. En este sentido ConT_EXt integra con naturalidad algunos elementos que en T_EX siempre han sido como una especie de cuerpo extraño tales como la inclusión de gráficos externos, el manejo de los colores, los hiperenlaces en documentos electrónicos, el asumir un tamaño de papel adecuado para un documento pensado para mostrarse en pantalla, etc.

1.5.1 Breve historia de ConT_EXt

ConT_EXt nació aproximadamente en 1991. Fue creado por HANS HAGEN y TON OTTEN en el seno de una empresa holandesa de diseño y composición de documentos llamada «*Pragma Advanced Document Engineering*», que se suele abreviar como Pragma ADE. Empezó siendo un conjunto de macros para T_EX con nombre en holandés, conocido oficiosamente como *Pragmatex*, y dirigido a los empleados no técnicos de la empresa, que tenían que gestionar los múltiples detalles de la composición de los documentos a editar, y que no estaban habituados a usar lenguajes de marcas ni interfaces que no fueran en holandés. Por ello la primera versión de ConT_EXt se escribió en holandés. La idea era crear un número suficiente de macros con una interfaz uniforme y coherente. Aproximadamente en 1994 el *paquete* era lo bastante estable como para que se escribiera un manual del usuario en holandés, y en 1996, por iniciativa de HANS HAGEN empezó a usarse el nombre «ConT_EXt» para referirse a él. Este nombre pretende significar «Texto con T_EX» (usando la preposición latina “con” que significa lo mismo que la española), pero, al mismo tiempo, juega con el término «Contexto», que en holandés (igual que en inglés) se escribe «context». Detrás del nombre hay, por lo tanto, un triple juego de palabras entre «T_EX», «texto» y «contexto».

Por ello, como en la base del nombre hay un juego de palabras, ConT_EXt aunque derive de T_EX (pronunciado «Tej»), no debe pronunciarse «Contejt» ya que ello haría que se perdiera el juego de palabras.

La interfaz empezó a traducirse al inglés aproximadamente en 2005, dando lugar a la versión conocida como ConT_EXt Mark II, en donde el «II» se explica porque en la mente de los desarrolladores, la versión previa en holandés había sido la versión «I», aunque lo cierto es que en realidad nunca llegó a denominarse así. Tras haber sido traducida la interfaz al inglés, empezó a extenderse el uso del sistema fuera de Holanda, traducándose la interfaz a otros idiomas europeos como el francés, el alemán, el italiano o el rumano. La documentación «oficial» de ConT_EXt, no obstante, se escribe normalmente sobre la versión en inglés, y por ello esa es la versión sobre la que se trabaja en este documento; a pesar de que el autor del mismo (o sea, yo), se siente más cómodo con el francés que con el inglés.

En su versión inicial ConT_EXt Mark II funcionaba con el *motor* de T_EX PdfT_EX. Más tarde, al surgir el *motor* X_YT_EX, ConT_EXt Mark II se modificó para permitir el uso de este nuevo motor que aportaba numerosas ventajas frente a PdfT_EX. Pero

cuando años más tarde se presentó LuaT_EX, se decidió reconfigurar internamente el funcionamiento de ConT_EXt para integrar en él todas las nuevas posibilidades que ofrecía dicho motor. Así nació ConT_EXt Mark IV, que fue presentado en 2007, inmediatamente después de que se presentara LuaT_EX. Muy probablemente en la decisión de reconfigurar ConT_EXt para adaptarlo a LuaT_EX influyó el hecho de que dos de los tres principales desarrolladores de ConT_EXt, HANS HAGEN y TACO HOEKWATER, están también en el equipo principal de desarrollo de LuaT_EX. Por ello ConT_EXt Mark IV y LuaT_EX nacieron simultáneamente y se fueron desarrollando al unísono. Hay una sinergia entre ConT_EXt y LuaT_EX que no existe con ningún otro derivado de T_EX; y no creo que ninguno de ellos aproveche las posibilidades de LuaT_EX como las aprovecha ConT_EXt.

Entre Mark II y Mark IV hay muchas diferencias, aunque la mayoría de ellas son *internas*, es decir: tienen que ver con cómo funciona realmente la macro a bajo nivel, de manera que desde la perspectiva del usuario la diferencia no es observable: el nombre y parámetros de la macro son los mismos. Hay, no obstante, algunas diferencias que sí afectan a la interfaz y obligan a hacer las cosas de modo diferente según con qué versión se esté trabajando. Estas diferencias son relativamente pocas, pero afectan a aspectos muy importantes como, por ejemplo, la codificación del fichero de entrada, o el manejo de las fuentes tipográficas instaladas en el sistema.



Sería, no obstante, muy de agradecer que en algún lugar hubiera un documento que explicara (o enumerara) las diferentes apreciables entre Mark II y Mark IV. En la wiki de ConT_EXt, por ejemplo, para cada comando de ConT_EXt se recogen *dos sintaxis* (muchas veces idénticas). Supongo que una es la de Mark II y la otra es la de Mark IV; y puestos a suponer, supongo también que la *primera versión* es la de Mark II. Pero lo cierto es que la wiki no informa de nada de eso.

El hecho de que las diferencias, a nivel de lenguaje, sean relativamente pocas, lleva a que en muchas ocasiones, más que de dos versiones se hable de dos «sabores» de ConT_EXt. Pero se les llame de una forma o de otra, lo cierto es que un documento preparado para Mark II normalmente no podrá ser compilado con Mark IV y viceversa; y si el documento mezcla ambas versiones, lo más probable es que no compile bien con ninguna de ellas; lo que implica que el autor del fichero fuente tiene que empezar decidiendo si lo escribirá para Mark II o para Mark IV.

Si hemos trabajado con las distintas versiones de ConT_EXt, un buen truco para diferenciar a simple vista los ficheros pensados para Mark II y los pensados para Mark IV consiste en usar una extensión diferente en el nombre de los ficheros. Así yo, por ejemplo, a mis ficheros escritos para Mark II les pongo, como extensión, «.mki» y a los escritos para Mark IV, «.mkiv». Es verdad que ConT_EXt espera que todos los ficheros fuente tengan la extensión «.tex», pero se puede cambiar la extensión siempre y cuando al invocar a un fichero se indique expresamente su extensión, si esta no es la que ConT_EXt espera por defecto.

La distribución de ConT_EXt que se instala desde su wiki, «ConT_EXt Standalone», incluye ambas versiones, y para evitar confusiones —supongo— utiliza un comando

distinto para compilar en cada una de ellas. Mark II se compila con el comando «`texexec`» y Mark IV con el comando «`context`».

En realidad tanto el comando «`context`» como «`texexec`» son *scripts* que arrancan, con diferentes opciones, «`mtxrun`» que, a su vez, es un *script* de Lua.

A día de hoy Mark II está congelada y Mark IV sigue en desarrollo, lo que significa que sólo se publican versiones nuevas de la primera cuando se detectan errores o fallos que hay que corregir, mientras que de Mark IV se siguen publicando versiones nuevas con asiduidad; a veces incluso dos o tres por mes; aunque en la mayor parte de los casos estas «nuevas versiones» no introducen cambios perceptibles en el lenguaje, sino que se limitan a mejorar de algún modo la implementación a bajo nivel de algún comando, o a actualizar alguno de los muchos manuales que se incluyen con la distribución. Aún así, si tenemos instalada la versión de desarrollo —que es la que recomiendo, y la que se instala por defecto con «ConT_EXt Standalone»—, conviene actualizar nuestra instalación de vez en cuando (Véase el [apéndice A](#) respecto al modo de actualizar la versión instalada de «ConT_EXt Standalone»).

LMTX y otras implementaciones alternativas de Mark IV

Los desarrolladores de ConT_EXt son de naturaleza inquieta, y por lo tanto no han detenido la evolución de ConT_EXt en Mark IV; se siguen probando y experimentando nuevas versiones, aunque éstas, en general, difieren de Mark IV en muy pocos aspectos, y no tienen la incompatibilidad de compilación que existe entre Mark IV y Mark II.

Así, se han desarrollado ciertas variantes menores de Mark IV llamadas, respectivamente, Mark VI, Mark IX y Mark XI. De ellas sólo he podido encontrar una pequeña referencia a Mark VI en la wiki de ConT_EXt en la que se dice que su única diferencia con Mark IV se encuentra en la posibilidad de definir comandos asignando a los parámetros no un número, como es tradicional en T_EX, sino un nombre, como suele hacerse en casi todos los lenguajes de programación.

Más importante que esas pequeñas variantes —creo— es la aparición en el universo de ConT_EXt (¿ConT_EXtverso?) de una nueva versión, llamada LMTX, nombre que es un acrónimo de LuaMetaT_EX: un nuevo *motor* de T_EX que es una versión simplificada de LuaT_EX, desarrollada con la vista puesta en el ahorro de recursos del ordenador; es decir LMTX requiere menos memoria y menos potencia de procesado que ConT_EXt Mark IV.

LMTX fue presentado en la primavera de 2019 y se supone que no implicará ninguna alteración externa del lenguaje Mark IV. Para el autor del documento no habrá diferencia a la hora de diseñarlo; pero en el momento de compilar podrá elegir entre hacerlo con LuaT_EX, o hacerlo con LuaMetaT_EX. En el [Apéndice A](#), relativo a la instalación de ConT_EXt se explica un procedimiento para asignar un nombre de comando distinto a cada una de las instalaciones ([sección 3](#)).

1.5.2 ConT_EXt versus L^AT_EX

Dado que el formato derivado de T_EX más popular es L^AT_EX, resulta inevitable la comparación entre este y ConT_EXt. Se trata, claro está, de lenguajes distintos

aunque, en cierto modo, emparentados entre sí por derivar ambos de T_EX; el parentesco es pues, similar, al que existe entre, por ejemplo, el español y el francés: idiomas que comparten un origen común (el latín) que afecta a que las sintaxis sean *parecidas* y muchas de las palabras de cada uno de estos idiomas tienen un reflejo en el otro. Pero aparte de ese *parecido de familia*, L^AT_EX y ConT_EXt difieren en la filosofía y en la implementación, pues los objetivos iniciales de uno y otro son, en cierto modo, contradictorios. L^AT_EX pretende facilitar el uso de T_EX, aislando al autor de los concretos detalles tipográficos para propiciar que el autor se centre en el contenido, y deje los detalles de la composición en manos del propio L^AT_EX. Es decir: la simplificación en el uso de T_EX se consigue a costa de limitar la inmensa flexibilidad de T_EX, predefiniendo los formatos fundamentales y limitando el número de cuestiones tipográficas que el autor debe decidir. Frente a esa filosofía, ConT_EXt nació en el seno de una empresa dedicada a la composición tipográfica de documentos. Por lo tanto, lejos de pretender aislar al autor de los detalles de composición tipográfica, lo que se intenta es otorgarle un absoluto y completo control sobre ellos. Para conseguirlo ConT_EXt proporciona una interfaz uniforme y coherente que se mantiene mucho más cerca del espíritu original de T_EX que L^AT_EX.

Esta diferencia en la filosofía y objetivos fundacionales, se traduce, a su vez, en una diferencia en la implementación. Porque L^AT_EX, que tiende a simplificar todo lo posible, no necesita usar todos los recursos de T_EX. Su núcleo es, en cierto modo, bastante simple. Por ello, cuando se quieren ampliar sus posibilidades, es necesario escribir expresamente un *paquete* que lo haga. Esa *paquetería* asociada a L^AT_EX es al mismo tiempo una virtud y un defecto: una virtud, porque la tremenda popularidad de L^AT_EX, junto con la generosidad de sus usuarios, hace que prácticamente cualquier necesidad que se nos plantee se le haya planteado antes a alguien, y exista un paquete que la implementa; pero también un defecto, porque estos paquetes son a menudo incompatibles entre sí, y su sintaxis no siempre es uniforme, lo que se traduce en que el manejo de L^AT_EX exija un continuo bucear en los miles de paquetes existentes para encontrar los que necesitamos y lograr que todos ellos puedan trabajar conjuntamente.

Frente a esa simplicidad del núcleo de L^AT_EX que se complementa con su extensibilidad mediante paquetes, ConT_EXt está pensado para albergar en su seno todas —o casi todas— las posibilidades tipográficas de T_EX, por lo que su concepción es mucho más monolítica, pero, al mismo tiempo, también es más modular: el núcleo de ConT_EXt permite hacerlo casi todo y está garantizado que no habrá incompatibilidades entre sus diferentes utilidades, no hay que investigar sobre las extensiones que se necesitan, y la sintaxis del lenguaje no cambia por el hecho de que necesitemos cierta utilidad.

Es cierto que en ConT_EXt existen los llamados *módulos* de extensión que alguien podría considerar que cumplen una función similar a la de los paquetes de L^AT_EX,

pero lo cierto es que la función de unos y otros es muy diferente: los módulos de ConT_EXt están pensados exclusivamente para albergar utilidades adicionales que, por estar en fase de experimentación, aún no se han incorporado al núcleo, o para permitir el acceso a extensiones cuya autoría es ajena al equipo de desarrollo de ConT_EXt.

No creo que pueda considerarse que alguna de estas dos *filosofías* es preferible a la otra. La cuestión más bien depende del perfil del usuario y de lo que pretenda. Si el usuario no desea lidiar con cuestiones tipográficas sino simplemente producir documentos estandarizados de muy alta calidad, probablemente sería preferible para él optar por un sistema como L^AT_EX; por el contrario, al usuario que guste de experimentar, o el que necesite controlar hasta el último detalle de sus documentos, o el que debe pergeñar un diseño especial para cierto documento, muy probablemente le convenga más usar un sistema como ConT_EXt, en donde el autor tiene en sus manos absolutamente todo el control; con el riesgo, claro está, de que no sepa hacer un uso correcto del mismo.

1.5.3 Comprender bien la dinámica de trabajo en ConT_EXt

Cuando trabajamos con ConT_EXt, empezamos siempre escribiendo, un fichero de texto (al que llamaremos *fichero fuente*), en el que junto con el contenido propiamente dicho de nuestro documento final, iremos incluyendo las instrucciones (en lenguaje ConT_EXt) que indican exactamente cómo queremos que el documento se formatee: qué apariencia general queremos que tengan sus páginas y párrafos, qué márgenes queremos aplicar a ciertos párrafos especiales, con qué tipo de letra se debe mostrar, qué fragmentos queremos que se muestren en un tipo de letra distinto, etc. Una vez que hemos escrito el fichero fuente, desde una terminal, le aplicaremos el programa «context», que lo procesará, y, a partir de él, generará un fichero distinto, en el que el contenido de nuestro documento se habrá formateado según las instrucciones que a tal fin se incluyeron en el fichero fuente. Este nuevo fichero podrá ser enviado a la impresora, mostrado en pantalla, alojado en Internet o distribuido entre nuestros contactos, amigos, clientes, profesores, alumnos ..., o, en definitiva, a cualquiera para quien hayamos escrito el documento.

Es decir: cuando se trabaja con ConT_EXt el autor actúa sobre un fichero cuya apariencia no tiene nada que ver con la del documento final: el fichero con el que el autor directamente trabaja es un fichero de texto que no está tipográficamente formateado. En esto ConT_EXt funciona de manera muy diferente a la forma en que se comportan los programas llamados *procesadores de texto* que van mostrando la apariencia final del documento editado al mismo tiempo que éste se va escribiendo. Para quien está acostumbrado a los procesadores de texto, al principio le parecerá extraña la forma de trabajar de ConT_EXt, e incluso es posible que le lleve algún

tiempo acostumbrarse. Sin embargo una vez que uno se acostumbra a ella comprende que en realidad esta otra forma de trabajar, diferenciando entre el fichero de trabajo y el resultado final, es, en realidad, una ventaja por muchas razones, entre las que aquí destacaré, sin seguir ningún orden concreto, las siguientes:

1. Porque los ficheros de texto son más “ligeros” de manejar que los ficheros binarios propios de los procesadores de texto y su edición requiere menos memoria del ordenador; son menos dados a corromperse, y no se vuelven ininteligibles si cambia la versión del programa con el que se crearon. Son también compatibles con cualquier sistema operativo, y se pueden editar con numerosos editores de texto, de tal modo que para que podamos trabajar con ellos no es preciso que el sistema informático tenga instalado el programa con el que el fichero fue creado: cualquier otro programa de edición valdrá; y en todo sistema informático hay siempre algún programa de edición de textos.
2. Porque diferenciar entre el documento de trabajo y el documento final, ayuda a distinguir lo que es contenido propiamente dicho del documento, de lo que será su apariencia, permitiendo que, en la fase de creación, el autor se concentre en el contenido, y en la fase de composición tipográfica, el autor se centre en la apariencia.
3. Porque permite cambiar con rapidez y precisión la apariencia del documento, ya que esta viene determinada por comandos de ConT_EXt que son fácilmente identificables.
4. Porque esta facilidad para cambiar la apariencia, por otra parte, permite que a partir de un sólo contenido, podamos generar con facilidad dos (o más) versiones diferentes: Tal vez una versión optimizada para su impresión en papel, y otra pensada para ser mostrada en pantalla, ajustada al tamaño de éstas y, quizás, incluyendo hiperenlaces que carecen de sentido en un documento impreso en papel.
5. Porque se evitan también con facilidad errores tipográficos que son comunes en los procesadores de texto tales como, por ejemplo, extender la letra cursiva más allá del último carácter que ha de llevarla.
6. Porque desde el momento en que el fichero de trabajo no será distribuido y es «sólo para nuestros ojos», es posible incorporar a él anotaciones y observaciones, comentarios y advertencias para nosotros mismos, de cara a ulteriores revisiones o versiones, con la tranquilidad de saber que las mismas no aparecerán en el fichero formateado que será objeto de distribución.
7. Porque la calidad que se puede obtener procesando simultáneamente todo el documento, es muy superior a la que es posible alcanzar con un programa que

tiene que ir tomando las decisiones tipográficas sobre la marcha, conforme el documento va siendo escrito.

8. Etcétera.

Todo lo anterior se traduce en que, de un lado, al trabajar con ConT_EXt, una vez le hemos cogido el tranquillo, seamos más eficaces y productivos, y que, de otro lado, la calidad tipográfica que obtendremos sea muy superior a la que se obtendría con los llamados *procesadores de texto*. Y aunque es verdad que, a cambio, éstos últimos son más fáciles de usar, en realidad no son *mucho* más fáciles de usar. Porque aunque es cierto que ConT_EXt consta, como antes dije, de cerca de 3500 instrucciones, un usuario normal de ConT_EXt no tendrá que conocerlas todas. Para hacer lo que se suele hacer con los procesadores de texto, le bastará con conocer las instrucciones que permiten indicar la estructura del documento, un par de instrucciones relativas a recursos tipográficos habituales, tales como la negrita o la cursiva, y, tal vez, el cómo generar una lista, o una nota a pié de página. En total, no más de 15 ó 20 instrucciones nos permitirán hacer casi todas las cosas que se hacen con el procesador de textos. El resto de instrucciones nos permiten hacer cosas distintas que con el procesador de textos normalmente no se pueden hacer, o son muy difíciles de conseguir; de forma que puede afirmarse que si bien es cierto que el aprendizaje de ConT_EXt es más difícil que el de un procesador de textos, ello es porque con ConT_EXt se pueden hacer muchísimas más cosas.

1.5.4 Obtener ayuda sobre ConT_EXt

Mientras seamos novatos, el mejor lugar para encontrar ayuda sobre ConT_EXt es, sin duda, su [wiki](#), la cual abunda en ejemplos y tiene un buen buscador, aunque exige, eso sí, entenderse bien con el idioma inglés. También podemos buscar ayuda en Internet, claro, pero aquí el juego de palabras en que consiste el nombre de ConT_EXt nos gastará una mala pasada porque una búsqueda de información sobre «context» devolvería millones de resultados y la mayoría no guardaría ninguna relación con lo que buscábamos. Para buscar información sobre ConT_EXt hay que añadir algo al nombre «context»; por ejemplo, «tex», o «Mark IV» o «Hans Hagen» (uno de los creadores de ConT_EXt) o «Pragma ADE», o algo similar. También puede ser útil buscar información por el nombre de la wiki: «contextgarden».

Cuando hayamos aprendido algo más de ConT_EXt, si nos manejamos bien con el inglés, podemos consultar alguno de los muchos documentos incluidos en «ConT_EXt Standalone», o pedir ayuda, bien en [TeX – LaTeX Stack Exchange](#), bien en la lista de distribución del propio ConT_EXt ([NTG-context](#)). En esta última intervienen las personas que más saben sobre ConT_EXt, sin embargo las normas de la buena educación «cibernética» exigen que antes de hacer una pregunta hayamos intentado por todos los medios hallar la respuesta por nosotros mismos.

Capítulo 2

Nuestro primer fichero fuente

Sumario: 2.1 Preparación del experimento: Herramientas necesarias; 2.2 El experimento propiamente dicho; 2.3 La estructura de nuestro fichero de ejemplo; 2.4 Algunos detalles adicionales sobre la forma de ejecutar «context»; 2.5 Gestión de errores;

El presente capítulo se dedica a realizar nuestro primer experimento; y al hilo del mismo se explicará la estructura básica de un documento de ConT_EXt así como las mejores estrategias para lidiar con los posibles errores.

2.1 Preparación del experimento: Herramientas necesarias

Para escribir y compilar un primer fichero fuente, necesitaremos tener instaladas en nuestro sistema, las siguientes herramientas.

1. **Un editor de textos** para escribir nuestro fichero de prueba. Hay muchísimos editores de textos y es difícilmente concebible un sistema informático que no tenga ya instalado alguno. Podemos usar cualquiera: los hay más simples, más complejos, más potentes, más sencillos, de pago, gratuitos, libres, especializados en sistemas T_EX, generales, etc. Si estamos acostumbrados a manejar un editor concreto, lo mejor es que sigamos trabajando con él; si no tenemos costumbre, hasta ahora, de trabajar con editores de texto, mi consejo es, inicialmente, escoger un editor sencillo, para no añadir a la dificultad de aprender ConT_EXt la de aprender el manejo del editor. Aunque también es cierto que muchas veces los programas que son más difíciles de aprender, son también los más potentes.

Este texto lo he escrito con GNU Emacs, que es uno de los editores de propósito general más potentes y versátiles que existen; es verdad que tiene sus peculiaridades y también sus detractores, pero en general hay más «*Emacslovers*» que «*Emacshaters*». Para trabajar con ficheros de T_EX o de alguno de sus derivados existe una extensión de GNU Emacs, llamada AucT_EX, que dota al editor de unas utilidades adicionales muy interesantes, aunque AucT_EX está

en general mejor preparado para batallar con ficheros L^AT_EX que para hacerlo con ficheros de ConT_EXt. GNU Emacs en combinación con AucTeX pueden ser una buena opción si no sabemos por qué editor decantarnos; ambos son programas de código libre, por lo que hay versiones de ellos para todos los sistemas operativos. De hecho, decir que GNU Emacs es *software libre*, es quedarse corto pues este programa encarna mejor que ningún otro el espíritu de lo que es y significa el *software libre*. A fin de cuentas su desarrollador principal fue RICHARD STALLMAN fundador e ideólogo del proyecto GNU y de la *Free Software Foundation*.

Además de GNU Emacs + AucTeX, otras buenas opciones si no sabe por qué editor decantarse son *Scite* y *TexWorks*. El primero, aunque es un editor de propósito general, no específicamente diseñado para trabajar con ficheros de ConT_EXt, es fácilmente personalizable y, como es el editor que en general utilizan los desarrolladores de ConT_EXt en «ConT_EXt Standalone» se contienen los ficheros de configuración de este editor diseñados por el propio HANS HAGEN. *TexWorks* es, por otra parte, un editor de textos rápido, y especializado en el manejo de ficheros de T_EX y de sus lenguajes derivados. Resulta bastante sencillo de configurar para trabajar con ConT_EXt y en «ConT_EXt Standalone» también se prevé su posible configuración.

Sea un editor o sea otro, lo que no debemos hacer es usar, como editor de textos, un *procesador de textos* como, por ejemplo, OpenOffice Writer o Microsoft Word. Estos programas, que son, en mi opinión, demasiado lentos y pesados, pueden ciertamente, si se les indica expresamente, grabar un fichero como de «sólo texto», pero no están pensados para ello y lo más probable es que terminaríamos almacenando nuestro fichero en algún formato binario incompatible con ConT_EXt.

2. **Una distribución de ConT_EXt** para procesar nuestro fichero de prueba. Si en nuestro sistema ya existe alguna instalación de T_EX (o de L^AT_EX) es posible que ya haya alguna versión de ConT_EXt instalada. Para comprobarlo basta con abrir una terminal y teclear en ella

```
$> context --version
```

NOTA para los muy novatos en el manejo de terminales, los dos primeros caracteres que he escrito («\$>») no hay que escribirlos en la terminal. Con ellos intento representar el llamado *prompt* de la terminal; esa lucecita parpadeante que indica que el terminal está esperando nuestras instrucciones.

Si ya hubiera instalada alguna versión de ConT_EXt, nos aparecerá algo así como


```

mtx-context | ConTeXt Process Management 1.03
mtx-context |
mtx-context | main context file: /home/jq/context/LMTX/tex/texmf-context/
            | tex/context/base/mkiv/context.mkiv
mtx-context | current version: 2020.04.30 11:15
mtx-context | main context file: /home/jq/context/LMTX/tex/texmf-context/
            | tex/context/base/mkiv/context.mxl
mtx-context | current version: 2020.04.30 11:15

```

En la última línea se nos informa de la fecha en que se liberó la versión instalada. Si ésta fuera muy antigua, nos conviene, bien actualizarla, bien instalar una versión nueva. Yo recomiendo la instalación de la distribución llamada «ConTeXt Standalone» cuyas instrucciones de instalación se pueden encontrar en la [wiki de ConTeXt](#). En el [apéndice A](#) se incluye un resumen de las mismas.

3. **Un programa visor de ficheros PDF**, para poder ver en pantalla el resultado de nuestro experimento. En sistemas Windows y Mac OS, el visor omnipresente es Adobe Acrobat Reader. No viene instalado por defecto (o no venía cuando yo dejé de usar Microsoft Windows, hace ya más de 15 años), pero lo hace la primera vez que se intenta abrir un fichero PDF por lo que lo más normal es que ya esté instalado. En sistemas Linux/Unix no hay versión actualizada de Acrobat Reader, pero tampoco hace falta, pues existen literalmente decenas de visores PDF gratuitos y muy buenos. En estos sistemas, además, casi siempre hay alguno instalado por defecto. Mi favorito, por su rapidez y facilidad de manejo, es MuPDF; aunque tiene algunas contraindicaciones como, por ejemplo, que no muestra el índice de marcadores, que no permite búsquedas de texto que incluyan caracteres inexistentes en el alfabeto inglés (como las vocales acentuadas o las eñes) o que no permite seleccionar texto, o enviar el documento a la impresora; es simplemente un visor; pero rapidísimo y comodísimo. Cuando necesito alguna de esas utilidades que en MuPDF no funcionan, suelo utilizar, bien Okular, bien qPdfView. Pero, de nuevo, la cuestión va en gustos: cada cual puede elegir el que prefiera.

Podemos elegir editor, podemos elegir visor de PDF, podemos elegir distribución de ConTeXt... ¡Bienvenidos al mundo del *software* libre!.

2.2 El experimento propiamente dicho

La escritura del fichero fuente

Si ya tenemos disponibles las herramientas mencionadas en el apartado anterior, debemos abrir nuestro editor de textos y crear en él un fichero al que llamaremos «avestruz.tex». Como contenido del fichero escribiremos el siguiente:

```
% Primera línea del documento

\mainlanguage[es] % Idioma = Español

\setuppapersize[S5] % Tamaño del papel

\setupbodyfont
  [modern,12pt] % Fuente = Latin Modern, 12 puntos

\setuphead      % Formato de los capítulos
  [chapter]
  [style=\bfc]

\starttext % Empieza el contenido del documento

\startchapter
  [title=El tren de los avestruces]

Recorriendo mil poblados, principados y reinados
ducados y archiducados, y también, algunos pocos
Estados tan desproletarizados que ya están en otro
estado, pasa el tren. Va tocando los poblados
más lujosos y adornados, enjoyados con tejados de
Belén, pero esos barrios dejados de la mano de los
hados son salteados con cuidado por el tren.

\stopchapter

\stoptext % Fin del documento
```

A la hora de escribirlo no importa si cambia algo; sobre todo si añade o quita espacios en blanco o saltos de línea. Lo que sí es importante es que las palabras tras los caracteres «\» se escriban exactamente igual, así como el contenido de los corchetes. En el resto puede haber variaciones.

Codificación del fichero

Una vez que hayamos escrito lo anterior, guardamos el fichero en disco asegurándonos de que la codificación del mismo es UTF-8. Esta codificación constituye a día de hoy el estándar y es la que se aplica por defecto en la mayoría de sistemas Linux/Unix. No se si ocurre o no lo mismo en Mac OS o en Windows, aunque sospecho que en este último es muy posible que se utilice la codificación ANSI. En todo caso, si no estamos seguros, desde el propio editor de textos podemos ver con qué codificación se guardará el fichero y, en su caso, cambiarla. Cómo hacerlo depende, claro está, del concreto editor con el que estemos trabajando. En GNU Emacs, por ejemplo, pulsando simultáneamente las teclas CTRL-X y después Return seguido de «f», en la última línea de la ventana (a la que GNU

Emacs denomina *mini-buffer*) aparecerá un mensaje solicitándonos una nueva codificación e informándonos de la codificación actual. En otros editores lo normal es que podamos acceder a la codificación en el menú «Guardar Como».

Una vez comprobado que la codificación es la correcta, y guardado el fichero en el disco cerraremos el editor para centrarnos en el análisis de lo que hemos escrito.

Un vistazo al contenido de nuestro primer fichero fuente para ConT_EXt

La primera línea empieza con el carácter «%». Este es un carácter reservado que indica a ConT_EXt que no debe procesar el texto entre dicho carácter y el final de la línea en la que éste se encuentre. Esa utilidad sirve para escribir comentarios en el fichero fuente que sólo podrá leer el autor, pues no se incorporarán al documento definitivo. En este ejemplo he usado ese carácter para llamar la atención sobre algunas líneas, explicando qué es lo que hacen.

Las siguientes líneas empiezan con el carácter «\» que es otro de los caracteres reservados de ConT_EXt y que indica que lo que viene a continuación es el nombre de un comando. En el ejemplo se incluyen varios comandos habituales en un fichero fuente para ConT_EXt: El idioma en el que está redactado el documento, el tamaño del papel, la fuente que se usará en el documento y el formato que han de tener los capítulos. Más adelante, en otros capítulos, iremos viendo los detalles de estos comandos, de momento sólo me interesa que el lector vea qué aspecto tienen: Empiezan siempre por el carácter «\», a continuación va el nombre del comando, y después, entre corchetes o entre llaves, según los casos, los datos que el comando necesita para producir sus efectos. Entre el nombre del comando, y los corchetes o llaves que le acompañan, puede haber espacios en blanco o saltos de línea.

En la línea 9^a de nuestro ejemplo (cuento sólo las líneas que tienen algún texto) está el importante comando `\starttext`: le indica a ConT_EXt que a partir de ese punto empieza el contenido del documento; y, en la última línea de nuestro ejemplo, vemos un comando `\stoptext` que indica que ahí termina el documento. Son dos comandos muy importantes sobre los que muy pronto diré algo más. Entre ellos se ubica el contenido propiamente dicho de nuestro documento que, en nuestro ejemplo, consiste en la primera estrofa de la canción «El tren de los avestruces», cuya letra se debe a JORGE DE LA VEGA. La he escrito en prosa para que se observe mejor el formateado de párrafos que realiza ConT_EXt.

Procesado del documento fuente

Para el siguiente paso, tras asegurarnos de que efectivamente en nuestro sistema se ha instalado ConT_EXt correctamente, debemos abrir una terminal en el directorio en el que se encuentre nuestro fichero «avestruz.tex».

Muchos editores de texto permiten compilar el documento con el que se está trabajando sin necesidad de abrir una terminal. Sin embargo el procedimiento *canónico* para procesar un documento con ConT_EXt implica hacerlo desde una terminal, ejecutando directamente el programa. Voy a hacerlo así (o a presuponer que se hace así) a lo largo de todo este documento por varias razones; la primera es que no tengo forma de saber con qué editor esta trabajando cada lector. Pero la más importante es la de que desde el terminal tendremos acceso a la salida a pantalla de «context» y podremos ver los mensajes emitidos por el programa.

Si la distribución de ConT_EXt que hemos instalado es «ConT_EXt Standalone», antes de nada debemos ejecutar el *script* que indica a la terminal las rutas y localización de los ficheros que ConT_EXt necesita para trabajar. En sistemas Linux/Unix ello se hace escribiendo el siguiente comando:

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

suponiendo que hayamos instalado ConT_EXt en un directorio llamado «context».

Respecto a la ejecución del *script* del que se acaba de hablar, véase lo que se dice en el [apéndice A](#) relativo a la instalación de «ConT_EXt Standalone».

Una vez que se han cargado en memoria las variables necesarias para la ejecución de «context», podemos ejecutarlo. Ello se hace escribiendo en el terminal

```
$> context avestruz
```

Obsérvese que aunque el fichero fuente se llama «avestruz.tex» en la llamada a «context» hemos omitido la extensión del fichero. Si hubiéramos denominado al fichero fuente, por ejemplo, «avestruz.mkiv» (cosa que yo suelo hacer para saber que ese fichero está escrito para Mark IV), habríamos tenido que indicar expresamente la extensión del fichero a compilar escribiendo «context avestruz.mkiv».

Tras ejecutar «context» en la terminal, empezarán a mostrarse por pantalla varias decenas de líneas, informando de lo que ConT_EXt está haciendo. La información se muestra a una velocidad imposible de seguir por un ser humano, pero no nos preocupemos, pues además de en la pantalla, dicha información se almacena también en un fichero auxiliar, de extensión «.log» que se genera con la compilación y que más tarde podremos consultar con toda tranquilidad si fuera preciso.

A los pocos segundos, si hemos escrito bien el texto de nuestro fichero fuente, sin cometer ningún error grave, terminará la emisión de mensajes a la terminal. El último de los mensajes nos informará del tiempo que se ha necesitado para la compilación. La primera vez que se compila un documento siempre se necesita algo más de tiempo, porque ConT_EXt tiene que cargar en memoria los ficheros con información sobre las fuentes que se utilizarán, los cuales quedan ya cargados para las ulteriores compilaciones. Con el mensaje relativo al tiempo invertido habrá terminado la compilación. Si todo ha ido bien, en el directorio en el que hemos ejecutado «context» habrán aparecido tres ficheros adicionales:

- `avestruz.pdf`
- `avestruz.log`
- `avestruz.tuc`

El primero de ellos es el resultado de nuestro procesamiento, es decir: el fichero PDF ya formateado. El segundo es el fichero «`.log`» en el que se ha almacenado toda la información que se mostró en pantalla durante la compilación; el tercero es un fichero auxiliar que ConT_EXt genera durante la compilación y que se usa para construir los índices y las referencias cruzadas. De momento, si todo ha funcionado como esperábamos, podemos borrar ambos ficheros (`avestruz.log` y `avestruz.tuc`). Si ha habido algún problema la información de esos ficheros nos puede ayudar a localizar su origen y determinar cómo solucionarlo.

Si no hemos obtenido esos resultados, probablemente ello se debe a que:

- O bien no hemos instalado correctamente nuestra distribución de ConT_EXt, caso este en el que al escribir en la terminal la orden «`context`», habrá salido un mensaje de «comando desconocido».
- O bien nuestro fichero no se codificó en UTF-8 y eso ha generado un error de compilación.
- O quizás la versión de ConT_EXt instalada en nuestro sistema es Mark II. En esta versión no se puede usar la codificación UTF-8 sin indicarlo expresamente en el propio fichero fuente. Podríamos arreglar el fichero fuente para que compilara bien, pero, dado que esta introducción se refiere a Mark IV, no tiene sentido seguir trabajando con Mark II: lo mejor es que nos instalemos «ConT_EXt Standalone».
- O bien hemos cometido algún error al escribir en el fichero fuente el nombre de alguno de los comandos o sus datos asociados.

Si tras ejecutar «`context`» la terminal empezó a emitir mensajes, pero luego se detuvo sin que reapareciera el *prompt*, antes de seguir debemos pulsar CTRL-X para abortar la ejecución de ConT_EXt que quedó interrumpida por el error.

Deberemos pues verificar cual es el caso, y solucionarlo, hasta que obtengamos una correcta compilación.

En la [figura 2.1](#) se muestra el contenido de «`avestruz.pdf`». Así veremos que ConT_EXt ha numerado la página, ha numerado también el capítulo y ha escrito el texto en la fuente que se le indicó. También ha partido la palabra «archiducados» entre la primera y la segunda línea, así como la palabra «barrios» entre la cuarta y la quinta líneas. ConT_EXt, por defecto, tiene activada la división silábica de palabras, para así conseguir que no haya excesivos espacios en blanco entre las palabras de un línea. Por eso es tan importante que informemos a ConT_EXt del idioma del documento, porque los patrones de división silábica varían según el idioma de que se trate. En nuestro ejemplo el primer comando del fichero fuente (`\mainlanguage[es]`) es el que se ha ocupado de ello.

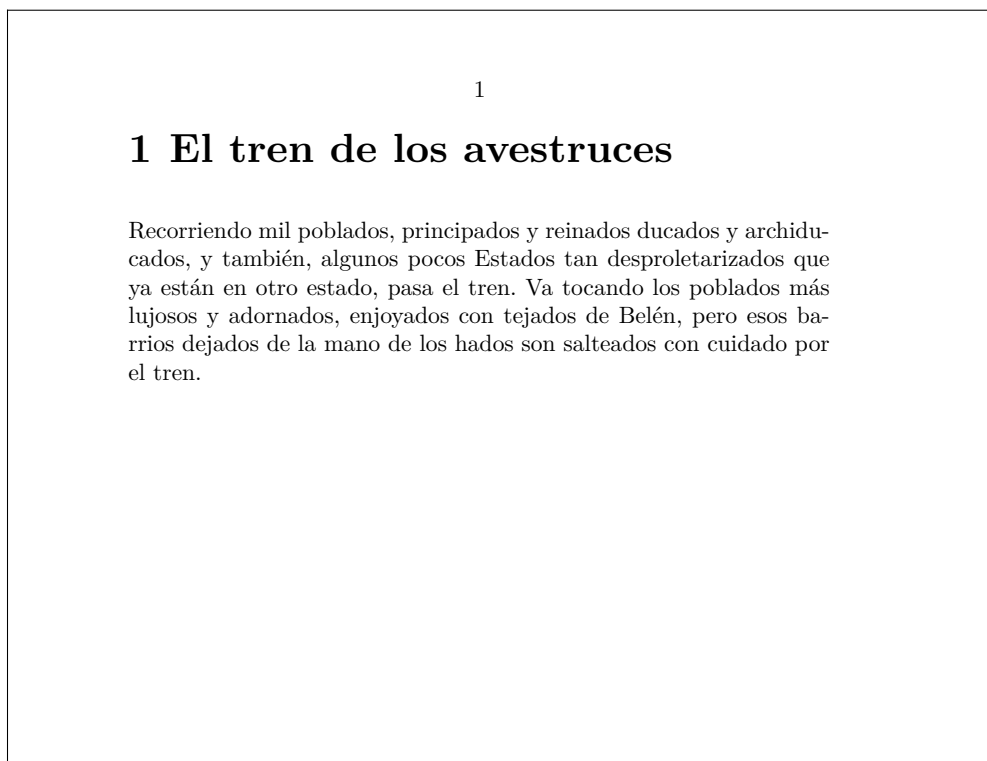


Figura 2.1 El tren de los avestruces

En definitiva: Con \TeX t ha transformado el fichero fuente y ha generado un fichero en el que tenemos un documento formateado según las instrucciones que se incluyeron en el fichero fuente. Han desaparecido de él los comentarios y, respecto de los comandos, lo que tenemos ahora no es su nombre, sino el resultado de su ejecución.

2.3 La estructura de nuestro fichero de ejemplo

En un proyecto que se desarrolle en un solo fichero fuente, la estructura de este es muy simple y viene marcada por los comandos `\starttext` ... `\stoptext`. Todo lo que haya entre la primera línea del fichero y el comando `\starttext` constituye el *preámbulo*. El contenido del documento propiamente dicho se inserta entre los comandos `\starttext` y `\stoptext`. En nuestro ejemplo el preámbulo incluye tres comandos de configuración global: uno para indicar el idioma de nuestro documento (`\mainlanguage`), otro para indicar el tamaño de las páginas (`\setuppapersize`) que en nuestro caso es «S5», que representa las proporciones de una pantalla de ordenador, y un tercer comando (`\setuphead`) que permite configurar el aspecto de los capítulos.

El cuerpo del documento se enmarca entre los comandos `\starttext` y `\stoptext`. Estos comandos indican, respectivamente, el punto inicial y el punto final del texto procesable: entre ellos se debe incluir todo el texto que queremos que ConTeXt procese, junto con aquellos comandos que no deban afectar a todo el documento sino solamente a fragmentos del mismo. De momento debemos asumir que los comandos `\starttext` y `\stoptext` son obligatorios en todo documento de ConTeXt, aunque más adelante, al hablar de los proyectos multifichero ([sección 4.6](#)) veremos que existe alguna excepción.

2.4 Algunos detalles adicionales sobre la forma de ejecutar «context»

El comando «context» con el que hemos procedido antes a procesar nuestro primer fichero fuente es, en realidad, un *script* de LUA, es decir: un pequeño programa de LUA que, tras hacer algunas comprobaciones, llama a LuaTeX para que sea éste quien procese el fichero fuente.

Podemos llamar a «context» con varias opciones. Las opciones se introducen inmediatamente después del nombre del comando, precedido su nombre de dos guiones. Si queremos introducir más de una opción, las separaremos con un espacio en blanco. La opción «help» nos da un listado de todas las opciones, con una breve explicación de cada una de ellas:

```
$>context --help
```

Algunas de las opciones más interesantes son las siguientes:

interface: Como ya dije en el capítulo introductorio, la interfaz de ConTeXt está traducida a varios idiomas. Por defecto se usa la interfaz en inglés, pero mediante esta opción podemos indicarle que utilice la versión holandesa (nl), francesa (fr), italiana (it), alemana (de) o rumana (ro).

purge, purgeall: Borra los ficheros auxiliares generados durante el procesamiento.

result=Nombre: Indica el nombre que debe tener el fichero PDF resultante. Por defecto será el mismo que el del fichero fuente a procesar, con la extensión .PDF.

usemodule=lista: Carga los módulos que se le indiquen antes de ejecutar ConTeXt (un módulo es una extensión de ConTeXt, que no forma parte de su núcleo, y que le dota de alguna utilidad adicional).

useenvironment=lista: Carga los ficheros de entorno que se le indiquen antes de ejecutar ConT_EXt (un fichero de entorno es un fichero con instrucciones de configuración).

version: Informa de la versión de ConT_EXt.

help: imprime información de ayuda sobre las opciones del programa.

noconsole: Suprime el envío de mensajes a pantalla durante la compilación. Estos mensajes, no obstante, se seguirán guardando en el fichero .log.

nonstopmode: Ejecuta la compilación sin detenerse ante los errores. Esto no significa que el error no se produzca, sino que cuando ConT_EXt encuentre un error, aunque sea recuperable, seguirá la compilación hasta terminar o hasta encontrar un error irrecuperable.

batchmode: Es una combinación de las dos opciones anteriores. Se ejecuta sin interrupciones y omite los mensajes en pantalla.

En los primeros pasos del aprendizaje de ConT_EXt no creo que sea una buena idea usar las tres últimas opciones, pues cuando se produzca un error, no tendremos pista alguna de dónde está o de qué lo ha producido. Y, creedme queridos lectores, antes o después se produce algún error de compilación.

2.5 Gestión de errores

Trabajando con ConT_EXt es inevitable que antes o después se produzca algún error en la compilación. Básicamente podemos agrupar los errores en alguna de las siguientes cuatro categorías:

1. **Errores de escritura.** Se producen cuando escribimos mal el nombre de algún comando. En tal caso le estaremos enviando al compilador una orden que no entiende. Como cuando, por ejemplo, en lugar de escribir el comando `\TeX` escribimos `\Tex` con la «X» final en minúsculas, puesto que ConT_EXt diferencia entre mayúsculas y minúsculas y por lo tanto considera que «TeX» y «Tex» son palabras diferentes; o si las opciones de funcionamiento de un comando, en lugar de encerrarlas entre corchetes las encerramos entre llaves, o si intentamos usar alguno de los caracteres reservados como si fuera un carácter normal, etc.
2. **Errores por omisión.** En ConT_EXt hay instrucciones que inician una tarea, la cual debe indicarse explícitamente cuándo hay que cerrarla; como el carácter reservado `$` que activa el modo matemático, el cual se mantiene hasta que sea desactivado, y si olvidamos desactivarlo se generará un error en cuanto se encuentre un texto o una instrucción que no tengan sentido en el modo matemático. Exactamente igual si iniciamos un bloque de texto, mediante el

carácter reservado «{» o mediante algún comando `\startLoQueSea` y, más adelante, no se encuentra el cierre explícito («}» o `\stopLoQueSea`).

3. **Errores de concepción.** Llamo así a aquellos errores que se producen cuando se llama a un comando que requiere ciertos argumentos, sin facilitárselos, o cuando la sintaxis de la llamada al comando no es la correcta.
4. **Errores de situación.** Hay algunos comandos que están pensados para funcionar solamente en ciertos contextos o entornos, por lo que fuera de ellos son desconocidos. Esto ocurre, particularmente, con el modo matemático: algunos comandos de ConT_EXt sólo funcionan en la escritura de fórmulas matemáticas y si son llamados en otros contextos generan un error.

¿Qué hacer cuando «context» nos avisa, durante la compilación, de que se ha producido un error? Lo primero, como es obvio, es determinar cuál es el error. Para ello tendremos que analizar el fichero «.log» generado durante la compilación; aunque a veces no hace falta pues el error ha sido de tal naturaleza que ha provocado que cese inmediatamente la compilación, caso este en el que el mensaje de error se podrá ver, todavía en la misma terminal en la que estamos ejecutando «context».

```
3      \setuppapersize % Tamaño del papel
4      [S5]
5
6      \setupbodyfont
7      [modern,12pt] % Fuente principal
8
9      \setuphead      % Capítulos en negrita
10     [chapter]
11     [style=\bfc]
12
13 >> \starttext % Empieza el documento propiamente dicho
14
15     \startchapter[title=El tren de los avestruces]
16
17     Recorriendo mil poblados, principados y reinados
18     ducados y archiducados, y también, algunos pocos
19     Estados tan desproletarizados que ya están en otro
20     estado, pasa el tren. Va tocando los poblados
21     más lujosos y adornados, enjoyados con tejados de
22     Belén, pero esos barrios dejados de la mano de los
23     hados son salteados con cuidado por el tren.

mtx-context      | fatal error: return code: 256
```

Figura 2.2 Salida por pantalla en caso de error de compilación

Por ejemplo, si en nuestro fichero de prueba, «avestruz.tex», por error, en lugar de `\starttext` hubiéramos escrito `\starttext` (con una sola «t»), lo que, por otra parte, es un error muy común, al ejecutar «context avestruz», cuando se detuviera la compilación, en la pantalla del terminal se podría ver la información que se muestra en la [figura 2.2](#). En ella podemos ver las líneas de nuestro fichero fuente numeradas, y en una de ellas, en este caso la número 13, entre el número y el texto de la línea el compilador ha añadido «>>» para indicar que es en esa

línea en la que ha encontrado el error. El fichero «avestruz.log» nos dará más pistas. En nuestro ejemplo no es un fichero demasiado extenso, porque la fuente que estábamos compilando es muy reducida; en otros caso puede contener una cantidad abrumadora de información. Pero debemos bucear en ella. Si abrimos «avestruz.log» con un editor de textos, veremos que en este fichero se va almacenando todo lo que va haciendo ConT_EXt. En él deberemos buscar una línea que empiece con una advertencia de error, para lo que podemos usar la función de búsqueda de texto del editor. Buscaremos la expresión “tex error”, y llegaremos a las siguientes líneas

```
tex error          > tex error on line 13 in file |  
                   /home/jq/context/docs/avestruz.tex: ! Undefined control sequence  
  
1.13 \starttext  
      % Empieza el documento propiamente dicho
```

Nota: La primera línea informativa del error, en el fichero «avestruz.log» es muy larga. Para que se viera bien, teniendo en cuenta el ancho de página La he partido en dos. El carácter «|» indica el punto en el que la he partido.

Si ponemos atención en las tres líneas del mensaje de error, vemos que en la primera se nos dice en qué número de línea se ha producido el error (la línea 13) y qué tipo de error es: «Undefined control sequence», o, lo que es lo mismo: Secuencia de control desconocida, o sea, comando desconocido. Las dos siguientes líneas del fichero log nos muestran la línea 13, partida en el punto en el que se produjo el error. Con lo que no hay dudas, el error está en `\starttext`. Lo leeremos con atención y, con suerte y experiencia, caeremos en la cuenta de que hemos escrito «starttext» y no «startttext» (con doble «t»).

Piénsese que los ordenadores son muy buenos y muy rápidos para ejecutar instrucciones, pero muy torpes para leer nuestra mente, y la palabra «starttext» no es la misma que «startttext». La segunda el programa sabe cómo ejecutarla; con la primera no sabe que hacer.

Otras veces la localización del error no será tan fácil. Particularmente cuando el error consiste en que se ha iniciado una tarea de la que no se ha especificado expresamente su terminación. A veces, en lugar de buscar en el fichero «.log» la expresión «tex error» deberemos buscar un asterisco. Este carácter al principio de una línea en dicho fichero representa, no un error fatal, sino una advertencia. Pero las advertencias pueden ser útiles para localizar el error.

Y si con la información del fichero «.log» no fuera suficiente, tendríamos que ir, poco a poco, localizando el lugar del error. Una buena estrategia para ello es ir cambiando de lugar el comando `\stoptext`. Recordemos que ConT_EXt deja de procesar el texto en cuanto encuentra dicho comando. Por lo tanto, si yo, en mi fichero fuente, escribo, más o menos a la altura de la mitad, un `\stoptext` y compilo, se

procesara sólo la primera mitad; si el error se repite sabré que este se encuentra en la primera mitad del fichero fuente, si no se repite significa que el error está en la segunda mitad... y así, poco a poco, cambiando de lugar el comando `\stoptext` podemos ir ubicando la localización del error. Una vez que lo hayamos localizado podremos intentar comprenderlo y corregirlo o, si no conseguimos comprender por qué se produce el error, al menos, localizado el punto en el que éste se encuentra, podremos intentar escribir las cosas de otra manera para evitar que el error se reproduzca. Esto último, claro está, sólo si somos los autores; si nos limitamos a componer un texto ajeno no podemos alterarlo y habrá que seguir investigando hasta que descubramos las razones del error y su posible solución.

En la práctica, cuando se confecciona con ConT_EXt un documento relativamente extenso lo que se suele hacer es ir compilando cada cierto tiempo, conforme se va redactando el documento, para que si se produce un error tengamos más o menos claro la parte nueva, desde la última compilación, que ha podido producirlo.

Capítulo 3

Comandos y otros conceptos fundamentales de ConT_EXt

Sumario: 3.1 Los caracteres reservados de ConT_EXt; 3.2 Comandos propiamente dichos; 3.3 Ámbito de aplicación de los comandos; 3.3.1 Comandos que requieren y comandos que no requieren que se les señale un ámbito de aplicación; 3.3.2 Comandos que requieren que se indique expresamente su inicio y su fin (entornos); 3.4 Opciones de funcionamiento de los comandos; 3.4.1 Comandos que pueden funcionar de varias maneras distintas; 3.4.2 Comandos que configuran el funcionamiento de otros comandos (`\setupAlgunaCosa`); 3.4.3 Creación de versiones personalizadas de comandos configurables (`\defineAlgunaCosa`); 3.5 Recapitulación sobre la sintaxis de los comandos y de sus opciones, y sobre el uso de corchetes y llaves en las llamadas a los mismos; 3.6 El listado oficial de comandos de ConT_EXt; 3.7 Definición de nuevos comandos; 3.7.1 Mecanismo general de definición de nuevos comandos; 3.7.2 Creación de nuevos entornos; 3.8 Otros conceptos fundamentales; 3.8.1 Grupos; 3.8.2 Dimensiones; 3.9 Método para el autoaprendizaje de ConT_EXt;

Ya hemos visto que en el fichero fuente, junto con el contenido propiamente dicho de nuestro futuro documento formateado, se insertan las instrucciones necesarias para explicarle a ConT_EXt cómo queremos que se transforme nuestro manuscrito. A estas instrucciones las podemos llamar, indistintamente, «comandos», «macros» o «secuencias de control».

Desde el punto de vista del funcionamiento interno de ConT_EXt (en realidad de T_EX), se diferencia entre *primitivas* y *macros*. Una primitiva es una instrucción simple que no se puede descomponer en otras instrucciones más simples. Una macro es una instrucción que se descompone en otras instrucciones más simples, las cuales, a su vez, tal vez puedan también descomponerse en otras, y así sucesivamente. La mayor parte de las instrucciones de ConT_EXt son, en realidad, macros. Desde la perspectiva del programador, la diferencia entre macros y primitivas es importante. Pero desde la perspectiva del usuario la cuestión no tiene demasiada trascendencia: en ambos casos lo que hay son instrucciones que se ejecutarán, sin importarnos demasiado cómo funcionan a bajo nivel. Por ello es corriente que en la documentación de ConT_EXt se hable de *comando* cuando se asume la perspectiva del usuario, y de *macro* cuando se asume la del programador. Como esta introducción asume sólo la perspectiva del usuario, usaré ambos términos indistintamente y como sinónimos.

Los *comandos* son órdenes a ConT_EXt para que haga algo; mediante ellos *controlamos* la actuación del programa. Por ello KNUTH, padre de T_EX, para referirse indistintamente a primitivas y macros usaba la denominación de *secuencias de control*, que me parece que es la más precisa de todas. Yo la usaré cuando sea importante diferenciar entre *símbolos de control* y *palabras de control*.

Las instrucciones de ConT_EXt son, básicamente, de dos tipos: caracteres reservados, y comandos propiamente dichos.

3.1 Los caracteres reservados de ConT_EXt

Cuando ConT_EXt va leyendo el fichero fuente, que, como es un fichero de texto, se compone sólo de caracteres de texto, necesita de alguna manera diferenciar entre lo que es texto propiamente dicho, que hay que formatear, y lo que son instrucciones que hay que ejecutar. Para diferenciar unas y otras es para lo que existen los caracteres reservados de ConT_EXt. En principio ConT_EXt asumirá que todo carácter en el fichero fuente es texto a procesar, salvo que sea alguno de los 11 caracteres reservados, los cuales serán tratados como *instrucción*.

¿Sólo 11 instrucciones? No. Sólo hay 11 caracteres reservados; pero como la función de uno de ellos («\») es convertir en instrucción el o los caracteres que se escriban inmediatamente detrás de él, en realidad el número potencial de comandos es ilimitado. ConT_EXt tiene en torno a 3000 comandos (sumando los comandos exclusivos de Mark II, los exclusivos de Mark IV y los comunes a ambas versiones).

Los caracteres reservados son los siguientes:

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

ConT_EXt los interpreta de la siguiente manera:

\ Este carácter, para nosotros, es el más importante de todos: indica que lo que viene inmediatamente detrás no debe ser interpretado como texto, sino como instrucción. Se le llama «Carácter de Escape» o «Secuencia de Escape» (aunque no tiene nada que ver con la tecla «Esc» presente en la mayor parte de los teclados)¹.

¹ En terminología informática se llama *carácter de escape* al que afecta a la interpretación del carácter siguiente. Por el contrario la tecla de *escape* de los teclados se llama así porque genera el carácter 27 del Código ASCII, que, en dicho Código, se usaba como carácter de escape. Hoy día los usos de la tecla de *Escape* están más asociados a la idea de cancelar alguna acción en curso.

- %** Le indica a ConT_EXt que lo que viene a continuación, y hasta el final de la línea, es un comentario que no debe ser procesado ni incluido en el fichero final formateado. La introducción de comentarios en el fichero fuente es extremadamente útil. Un comentario puede servir para explicar el por qué algo se ha hecho de cierta manera, lo que es muy útil en ficheros fuente complejos, de cara a su revisión posterior, cuando tal vez ya no recordemos por qué hicimos las cosas como las hicimos; o también sirve de recordatorio a nosotros mismos respecto de algún aspecto que quizás deberíamos revisar; o incluso puede servir para ayudar a localizar en qué lugar del fichero fuente se encuentra el origen de cierto error de compilación, pues poniendo una marca de comentario al inicio de cierta línea, la excluiríamos de la compilación, y podremos ver si era esa línea la que causaba el error; también sirve para almacenar dos versiones distintas de una misma macro, de tal manera que podamos compilar con unos resultados o con otros; o para eliminar de la compilación cierto fragmento del que no estamos muy seguros, pero sin borrarlo del fichero fuente para poder volver sobre él más tarde ..., etc. Una vez abierta la posibilidad de que nuestro fichero fuente contenga texto que nadie más que nosotros podrá ver, los usos que podemos hacer de ello sólo están limitados por nuestra imaginación. Yo admito que esta es una de las utilidades que más echo de menos cuando no tengo más remedio que escribir un texto con un procesador de textos.
- {** Este carácter abre un grupo. Los grupos son bloques de texto sometidos a ciertas características. Se hablará de ellos en la [sección 3.8.1](#).
- }** Este carácter cierra un grupo previamente abierto mediante **{**.
- #** Este carácter se usa en la definición de macros para hacer referencia a los argumentos de la macro. Véase en la [sección 3.7.1](#) de este mismo capítulo.
- ~** Introduce en el documento un espacio en blanco que no podrá sustituirse por un salto de línea, es decir: las dos palabras separadas por el carácter **~** estarán siempre en la misma línea. De esta instrucción, y los casos en los que se aconseja su uso, se hablará en la [sección 11.3.1](#).
- |** Este carácter se usa para indicar que dos palabras unidas por un elemento separador constituyen una palabra compuesta que puede dividirse silábicamente en el primer componente, pero no en el segundo. Véase la [sección 10.4](#).
- \$** Este carácter es un conmutador del modo matemático. Es decir: lo activa, si se encuentra desactivado, o lo desactiva si estaba activado. En el modo matemático ConT_EXt aplica unas fuentes y reglas de construcción diferentes de las normales, pensadas para optimizar la escritura de fórmulas matemáticas. Aunque un uso muy importante de ConT_EXt es el de la escritura de

matemáticas, no desarrollaré esos aspectos en esta introducción pues, siendo yo de letras, no me siento capacitado para ello.

- Este carácter se usa en modo matemático para indicar que lo que viene a continuación es un subíndice. Así, por ejemplo, para obtener x_1 , debemos escribir `x_1`.
- ^ Este carácter se usa en modo matemático para indicar que lo que viene a continuación es un superíndice. Así por ejemplo, para obtener $(x + i)^n$ deberíamos escribir `$(x+i)^{n^3}$`.
- & En la documentación de ConT_EXt se afirma que se trata de un carácter reservado, pero no se dice para qué se reserva. En Plain T_EX este carácter tiene, fundamentalmente, dos usos: sirve para alinear columnas en entornos tabulares básicos, y para indicar, en un contexto matemático, que lo que viene a continuación ha de ser tratado como texto normal. En el manual introductorio «ConT_EXt Mark IV, an Excursion», aunque no se dice para qué sirve, si hay ejemplos de su uso dentro de fórmulas matemáticas, aunque no para el uso que tenía en Plain T_EX, sino para alinear columnas dentro de funciones complejas. Como yo soy de letras, no me siento capaz de hacer más pruebas para ver exactamente cuál es la utilidad de este carácter reservado.



Cabe suponer que en la selección de cuáles habrían de ser los caracteres reservados, se buscó que se tratara de caracteres disponibles en la mayoría de los teclados, pero que normalmente no se incluirían en un texto escrito. No obstante, aunque no sea muy corriente, siempre cabe la posibilidad de que alguno de ellos deba figurar en nuestro documento, como si, por ejemplo, queremos escribir que algo costó 100 dólares (100\$), o que en España el porcentaje de conductores mayores de 65 años era del 16% en 2018. En tales casos no debemos escribir el carácter reservado directamente sino que deberemos insertar un *comando* que escriba en el fichero final dicho carácter. El comando para cada uno de los caracteres reservados se muestra en la [tabla 3.1](#).

Otra forma de obtener los caracteres reservados es mediante el comando `\type`. Este comando envía al documento final lo que reciba como argumento sin procesarlo de ninguna manera y, por lo tanto, sin interpretarlo. En el documento final el texto recibido mediante `\type` se mostrará en la fuente mono-espaciada típica de las terminales informáticas y de las máquinas de escribir.

Normalmente encerraremos entre llaves el texto que `\type` debe mostrar. Pero cuando dicho texto incluya llaves de apertura o de cierre, en lugar de entre llaves debemos encerrar el texto entre dos caracteres iguales que no formen parte del texto que constituye el argumento de `\type`. Por ejemplo: `\type*{*, o \type+}+`.

Carácter reservado	Comando que lo genera
\	<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>

Tabla 3.1 Escritura de caracteres reservados

Si por error usamos directamente alguno de los caracteres reservados, no con la finalidad para la que fueron concebidos, sino porque momentáneamente hemos olvidado que se trata de caracteres reservados que no podemos usar como caracteres normales, pueden ocurrir tres cosas:

1. Lo más normal, que se genere un error al compilar.
2. Que obtengamos un resultado inesperado. Esto pasa particularmente con «~» y con «%»; en el primer caso, en nuestro documento definitivo, en lugar del carácter «~» que esperábamos, se insertará un espacio en blanco; y en el segundo caso dejará de procesarse todo lo que haya en la misma línea, a partir del «%». También el uso inadecuado del carácter «\» puede producir un resultado inesperado, si el o los caracteres inmediatamente posteriores constituyen un comando conocido por ConT_EXt. Pero lo más normal cuando se usa incorrectamente «\» es que se produzca un error de compilación.
3. Que no haya ningún problema: Esto ocurre con los tres caracteres reservados que se usan principalmente en matemáticas ($_$ $^$ $\&$): si se usan fuera de dicho entorno, son tratados como caracteres normales.

El punto 3 es mi conclusión. Lo cierto es que en la documentación de ConT_EXt no he encontrado ningún lugar en el que se diga que estos tres caracteres reservados pueden usarse directamente; pero en mis pruebas, no he visto que se produzca ningún error cuando así se hace; a diferencia de lo que ocurre, por ejemplo, en L^AT_EX.



3.2 Comandos propiamente dichos

Los comandos propiamente dichos empiezan siempre por el carácter «\». Atendiendo a lo que haya inmediatamente después de la secuencia de escape se distingue entre:

- a. **Símbolos de control.** Un símbolo de control empieza por la secuencia de escape («\») y consta exclusivamente de un carácter que no sea una letra, como por ejemplo «\,», «\1», «\'» o «\%». Cualquier carácter o símbolo que no sea una letra en el sentido estricto de la palabra puede ser un símbolo de control, lo que incluye números, signos de puntuación, símbolos e incluso el espacio en blanco que, en este documento se representa, con el símbolo `_`, cuando sea preciso destacar la presencia de algún espacio en blanco. De hecho «_» (una barra invertida seguida de un espacio en blanco) es un símbolo de control que se usa bastante, como en seguida tendremos ocasión de ver.

El espacio en blanco es un carácter “invisible”, lo que, en un documento como este, en el que a veces hay que especificar con mucha claridad lo que se debe escribir en el fichero fuente, es un inconveniente. Ya Knuth se dio cuenta de dicho problema y en «The T_EXBook» inició la costumbre de representar con el símbolo «_» los espacios en blanco que sean significativos. Y así, por ejemplo, si quisiéramos resaltar que en el documento fuente hay que separar dos palabras con dos espacios en blanco, escribiríamos «palabra1__palabra2».

- b. **Palabras de control.** Si el carácter inmediatamente posterior a la barra invertida es una letra propiamente dicha el comando será una *Palabra de control*. Este grupo de comandos es el más numeroso y en él se da la característica de que el nombre del comando sólo puede estar constituido por letras; no se admiten números, signos de puntuación o símbolos de cualquier otro tipo. Exclusivamente letras, en mayúsculas o minúsculas. Téngase en cuenta, por otra parte, que ConT_EXt diferencia entre mayúsculas y minúsculas, por lo que para él los comandos `\micomando` y `\MiComando` tienen nombres diferentes. Pero `\Micomando1` y `\Micomando2` se consideran el mismo nombre, pues el «1» y el «2» no forman parte del nombre del comando ya que no son letras.



En el manual de referencia de ConT_EXt no se contiene ninguna regla sobre los nombres de los comandos, ni en el resto de los «manuales» que se incluyen con «ConT_EXt Standalone». Lo que he afirmado en el párrafo anterior es mi conclusión basada en que así ocurre en T_EX (donde, además, se consideran «no letras» los caracteres que no existen en el alfabeto inglés, como las vocales acentuadas), y en que esa regla permite explicar bien la absorción de espacios en blanco posteriores al nombre de un comando.

Cuando ConT_EXt está leyendo un fichero fuente y encuentra el carácter de escape («\») ya sabe que lo que viene a continuación es un comando. Entonces lee el primer carácter detrás de la secuencia de escape. Si no es una letra, significa que el comando es un símbolo de control y consta sólo de ese primer símbolo. Pero si, por el contrario, el primer carácter tras la secuencia de escape es una letra, entonces ConT_EXt seguirá leyendo carácter a carácter hasta que encuentre la primera «no letra», momento en el que sabrá que el nombre del comando ya se ha terminado. Esta es la razón de que en los nombres de los comandos que sean palabras de control no puede haber caracteres que no sean letras.

Cuando la «no letra» con la que acaba el nombre del comando es un espacio en blanco, se asume que dicho espacio en blanco no forma parte del texto a procesar, sino que se insertó exclusivamente para indicar que ahí terminaba el nombre del comando, y por ello ConT_EXt elimina dicho espacio en blanco. Esto produce un efecto que sorprende a los principiantes en ConT_EXt; pues cuando el efecto del comando en cuestión implicaba escribir algo en el documento final, lo escrito por el comando aparecerá pegado a la siguiente palabra. Así, por ejemplo, las siguientes dos frases en el fichero fuente

```
Saber \TeX ayuda a aprender \ConTeXt.
Saber \TeX, aunque no es imprescindible, ayuda a aprender \ConTeXt
```

producirían, respectivamente, los siguientes resultados

```
Saber TEXayuda a aprender ConTEXt.
Saber TEX, aunque no es imprescindible, ayuda a aprender ConTEXt.1
```

Obsérvese como, en el primer caso, la palabra «T_EX» aparece pegada a la siguiente palabra y en el segundo no. Eso es porque en el fichero fuente, en el primer caso la primera «no letra» tras el nombre del comando `\TeX` era un espacio en blanco, que se ha suprimido por asumir ConT_EXt que su presencia era sólo para indicar la terminación del nombre del comando, y en el segundo caso se trataba de una coma, que, al no ser un espacio en blanco, no se ha suprimido.

Por otra parte, este problema no se arregla simplemente añadiendo un espacio en blanco adicional, y escribir, por ejemplo

```
Saber \TeX_ ayuda a aprender \ConTeXt2.
```

no resolverá el problema, pues una regla de ConT_EXt (que veremos en la [sección 4.2.1](#)) es la de que un espacio en blanco absorbe a todos los espacios en blanco y tabuladores que le sigan. Por ello, cuando se nos plantea este problema (lo que por suerte no ocurre con demasiada frecuencia) debemos asegurarnos de que la primera «no letra» tras el nombre del comando no sea un espacio en blanco. Hay dos buenos candidatos para ello:

- Los caracteres reservados «{ }». El carácter reservado «{», como ya dije, abre un grupo, y «}» cierra un grupo, por lo tanto la secuencia «{ }» introduce un grupo vacío. Un grupo vacío no tiene ningún efecto en el documento final, pero sirve

¹ **Nota:** en los casos en los que en esta introducción, para ilustrar alguna cuestión, se escribe un fragmento del código fuente y el resultado de la compilación del mismo, se siguen dos convenciones: En ocasiones se colocan uno junto a otro en un párrafo a dos columnas el código y el resultado de su compilación; en otros casos el código se escribe en este tono de magenta oscuro que con carácter general se usa en este documento para representar los comandos de ConT_EXt, y el resultado de su compilación en color rojo.

² Sobre el símbolo «_» recuérdese la observación hecha en la [página 49](#).

para que ConT_EXt sepa que el nombre del comando anterior ya ha terminado. O también podríamos crear un grupo en torno al comando en cuestión, y escribir, por ejemplo «`\TeX`». En cualquiera de ambos casos conseguiríamos que la primera «no letra» tras `\TeX` no sea un espacio en blanco.

- El símbolo de control «`_`» (una barra invertida seguida de un espacio en blanco, véase la observación realizada en la [página 49](#)). El efecto de este símbolo de control es el de insertar un espacio en blanco en el documento final. Para entender bien la lógica de ConT_EXt, tal vez merezca la pena entretenerse un poco en ver qué pasa cuando ConT_EXt se encuentra con una palabra de control (como por ejemplo `\TeX`) seguida de un símbolo de control (como por ejemplo «`_`»):
 - ConT_EXt encuentra el carácter `\` seguido de una «T» y ya sabe que está ante una palabra de control, sigue leyendo caracteres hasta llegar a una «no letra», cosa que ocurre cuando llega al carácter `\` que introduce el próximo símbolo de control.
 - Una vez que sabe que el nombre del comando era `\TeX`, lo ejecuta e imprime en el documento final el texto T_EX. Tras ello vuelve al punto en el que detuvo la lectura para comprobar el carácter inmediatamente posterior a la segunda barra invertida.
 - Comprueba que es un espacio en blanco, es decir, una «no letra» lo que significa que la secuencia de control es exactamente esa, por lo tanto ya puede ejecutarla. Así lo hace e inserta un espacio en blanco.
 - Por último, una vez más, vuelve al punto en el que detuvo la lectura (el espacio en blanco que constituía el símbolo de control) y continúa, a partir de ahí, procesando el fichero fuente.

He explicado este mecanismo con cierto detenimiento, pues la eliminación de espacios en blanco suele sorprender a los recién llegados. No obstante hay que señalar que el problema se plantea relativamente poco, pues en general las palabras de control no imprimen directamente nada en el documento final, sino que afectan al formato y a la apariencia. Por el contrario, si es bastante corriente que los símbolos de control impriman algo en el documento final.

Hay un tercer procedimiento para evitar el problema de los espacios en blanco, consistente en definir (al estilo de T_EX) un comando similar e incluir una «no letra» al final del nombre del comando. Por ejemplo, la siguiente secuencia:

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

creará un comando llamado `\txt`, que hará exactamente lo mismo que el comando `\TeX` y sólo funcionará correctamente si es llamado con un guión detrás de su nombre `\txt-`. Ese guión no forma técnicamente parte del nombre del comando, pero éste no funcionará si no se escribe el nombre seguido de un guión. El por qué eso es así tiene que ver con el mecanismo de definición de macros de T_EX, y es demasiado complejo de explicar aquí. Pero funciona: una vez definido ese comando, cada vez que usemos `\txt-` ConT_EXt lo sustituirá por `\TeX` eliminando el guión,

pero usándolo internamente para saber que el nombre del comando ya se ha terminado, por lo que un espacio en blanco inmediatamente detrás de él no sería eliminado.

Este «truco» no funciona correctamente con el comando `\define`, que es el comando específico de ConT_EXt para definir macros.

3.3 Ámbito de aplicación de los comandos

3.3.1 Comandos que requieren y comandos que no requieren que se les señale un ámbito de aplicación

Gran parte de los comandos de ConT_EXt, y en particular los comandos que afectan de modo directo a las características de formato de las fuentes (negrita, cursiva, versalitas, etc.), activan cierta cualidad que se queda activada hasta que se encuentre otro comando que la desactive, o que active alguna otra característica incompatible con ella. Por ejemplo, el comando `\bf` activa la negrita, y ésta seguirá activa hasta que se encuentre un comando *incompatible* como, por ejemplo `\tf`, o `\it`.

Este tipo de comandos no necesita recibir ningún argumento, pues no están diseñados para aplicarse sólo a cierto texto. Es como si se limitaran a *encender* la función que sea (las negritas, la cursiva, la letra sans serif, cierto tamaño de letra, etc.).

Cuando estos comandos se ejecutan dentro de un *grupo* (véase la [sección 3.8.1](#)), también pierden su eficacia cuando se cierre el grupo en el que se ejecutaron. Por ello, muchas veces, para hacer que estos comandos afecten sólo a una porción de texto, lo que se hace es generar un grupo que contenga a dicho comando y al texto al que queremos que afecte. Un grupo se crea encerrándolo entre llaves. Por lo tanto, el siguiente texto

```
En {\it The \TeX Book}, {\sc Knuth}
explicó todo lo que hay que saber
sobre \TeX.
```

En *The T_EX Book*, KNUTH explicó todo lo que hay que saber sobre T_EX.

crea dos grupos, uno para delimitar el ámbito del comando `\it` (que establece la cursiva) y otro para delimitar el ámbito del comando `\sc` (que establece las versalitas).

Frente a este tipo de comandos, hay otros que, por el tipo de efecto que producen, o por otras razones, requieren que se les indique expresamente a qué texto han de

aplicarse. En estos casos el texto que se haya de ver afectado por el comando se encierra entre llaves *inmediatamente después del comando*. Como ejemplo de este tipo de comandos podemos mencionar a `\framed`: este comando dibuja un marco en torno al texto que recibe como argumento, de modo que

```
\framed{Tararí que te vi, Mariví}
```

producirá

Tararí que te vi, Mariví

Obsérvese que, aunque en el primer grupo de comandos (los que no requieren argumento) también se usan a veces llaves para delimitar su campo de actuación, esto no es necesario para la eficacia del comando. El comando está diseñado para aplicarse a partir del punto en el que aparece. Por ello, cuando se delimita su campo de aplicación mediante llaves, el comando se ubica *dentro de las llaves*, a diferencia de lo que ocurre en el segundo grupo de comandos, en los que las llaves que encierran el texto al que se aplicará el comando se encuentran *detrás* del comando.

En el caso del comando `\framed` está claro que el efecto que produce en cierto modo exige que reciba como argumento un texto al que aplicarse. En otros casos, el que el comando sea de un tipo u otro depende de lo que decida su diseñador. Y así, por ejemplo lo que hacen los comandos `\it` y `\color` es bastante parecido: aplican una característica (de formato o de color) al texto. Pero se decidió diseñar al primero como comando sin argumento, y al segundo como comando con argumento.

3.3.2 Comandos que requieren que se indique expresamente su inicio y su fin (entornos)

Hay ciertos comandos que delimitan su ámbito de aplicación indicando exactamente el punto en el que empezarán a aplicarse y el punto en el que dejarán de hacerlo. Estos comandos, por lo tanto se desdoblán en dos: una sentencia que indica cuándo se inicia la acción del comando, y otra sentencia que indica cuándo cesa la acción del comando. Para indicar el inicio se usa el término «start» seguido del nombre del comando, y para indicar el final se usa el término «stop», seguido también del nombre del comando. Así por ejemplo el comando «itemize» se convierte en `\startitemize` para indicar el inicio de la *itemización* y `\stopitemize` para indicar el fin de la misma.

No hay en la documentación oficial de ConT_EXt un nombre especial para estos comandos por parejas. El manual de referencia y la introducción los llaman simplemente comandos «start ... stop». A veces se les llama *entornos*, que es el nombre con el que se conoce en L^AT_EX a un tipo de construcción similar, aunque ello tiene el inconveniente de que en ConT_EXt el término «entorno» se usa para otra cosa

(un tipo especial de fichero que veremos al hablar de los proyectos multifichero, en la [sección 4.6](#)). Aún así, como el término entorno es claro, y, por el contexto, creo que es fácil diferenciar si se habla de *comandos de entorno* o de *ficheros de entorno*, usaré esa denominación.

Los entornos, pues, constan de un comando que los abre o inicia y de otro que los cierra o termina. Si en el fichero fuente se encuentra un comando de apertura del entorno que no está expresamente cerrado, normalmente se generará un error¹. Este tipo de errores son, por otra parte, los más difíciles de localizar, pues el error puede producirse mucho después del punto en que se encontraba el comando de apertura. A veces el fichero «.log» nos informará de la línea en donde se inició el entorno no expresamente cerrado; pero otras veces la falta de cierre del entorno se traduce en que ConT_EXt interprete inadecuadamente cierto pasaje y crea que el error está en tal pasaje, y no en el entorno no expresamente cerrado, con lo que la información del fichero «.log» no nos ayudará tanto a descubrir dónde está el problema.

Los entornos se pueden anidar, y dentro de un entorno abrir otro entorno; si bien en tal caso, cuando hay entornos anidados, un entorno se tiene que cerrar dentro del entorno en el que se abrió, es decir, el orden de cierre de los distintos entornos ha de ser coherente con el orden de su apertura. En el siguiente ejemplo creo que se verá más claro:

```
\startUnaCosa
...
  \startOtraCosa
  ...
    \startUnaTerceraCosa
    ...
    \stopUnaTerceraCosa
  \stopOtraCosa
\stopUnacosa
```

Véase como, en el ejemplo, si el entorno «UnaTerceraCosa» se abre dentro del entorno «OtraCosa» se tiene que cerrar también dentro de él. Lo contrario generaría un error en el momento de la compilación.

En general se diseñan como *entornos* los comandos que implementan algún cambio que está pensado para aplicarse a unidades de texto no inferiores al párrafo. Por ejemplo, el entorno «**narrower**», que altera los márgenes, sólo tiene sentido aplicado a nivel de párrafo; o el entorno «**framedtext**» que enmarca uno o varios párrafos. Este último entorno quizás sirva bien para comprender por qué algunos

¹ Aunque no siempre; depende del entorno de que se trate y de la disposición del resto del documento. En esto ConT_EXt se diferencia de L^AT_EX que es mucho más estricto.

comandos se diseñan como entornos y otros se diseñan como comandos individuales: Si queremos enmarcar una o varias palabras, pero todas ellas en la misma línea, usaremos el comando `\framed`, pero si lo que queremos enmarcar es todo un párrafo (o varios párrafos) entonces usaremos el entorno «`framedtext`».

Por otra parte, el texto ubicado dentro de un concreto entorno, normalmente constituye un *grupo* (véase la [sección 3.8.1](#)), lo que significa que si dentro de un entorno se encuentra un comando de activación, de los que se aplican a todo el texto que le sigue, este comando se aplicará sólo hasta el final del entorno en el que se encuentre; y de hecho ConT_EXt cuenta con un *entorno sin nombre* que se inicia con el comando `\start` (no seguido de ningún otro texto; *start* a secas; por eso lo he llamado *entorno sin nombre*) y se termina con el comando `\stop` cuya única función (sospecho) es la de crear un grupo.



Que uno de los efectos de los entornos sea el de agrupar su contenido no lo he leído en la documentación de ConT_EXt, pero así resulta de mis pruebas con varios de los entornos predefinidos; aunque debo admitir que las pruebas no han sido demasiado exhaustivas sino que me he limitado a comprobarlo con algunos entornos elegidos al azar. Mis pruebas demuestran, eso sí, que tal afirmación, de ser cierta, sólo lo sería para algunos de los entornos predefinidos: los creados mediante el comando `\definestartstop` (que se explica en la [sección 3.7.2](#)) no crean ningún grupo, salvo que en la definición del nuevo entorno incluyamos los comandos necesarios para crear el grupo (véase la [sección 3.8.1](#)).

También es conjetura mía la de que el entorno al que he llamado *sin nombre* (`\start`) sólo sirve para crear un grupo: efectivamente crea un grupo, pero si tiene o no alguna otra utilidad no lo se. Este es uno de los comandos no documentados en el manual de referencia.

3.4 Opciones de funcionamiento de los comandos

3.4.1 Comandos que pueden funcionar de varias maneras distintas

Muchos comandos pueden funcionar de más de una manera. En tales casos hay siempre una forma predeterminada de funcionamiento que se puede alterar indicando entre corchetes, tras el nombre del comando los parámetros correspondientes al funcionamiento deseado.

Un buen ejemplo de lo que se acaba de decir lo tenemos con el comando `\framed`, mencionado en el apartado anterior. Este comando dibuja un marco alrededor del texto que recibe como argumento. Por defecto el marco tiene la altura y la anchura del texto al que se debe aplicar; pero podemos indicar una altura y anchura diferentes. Así puede observarse la diferencia entre el funcionamiento por defecto de `\framed`

```
\framed{Tararí}
```



y un funcionamiento personalizado:

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



En el segundo ejemplo, entre corchetes, hemos indicado una anchura y altura específica para el marco que rodea al texto recibido como argumento. Dentro de los corchetes, las distintas opciones de configuración se separan mediante una coma; los espacios en blanco e incluso los saltos de línea (siempre que no sean un doble salto de línea) entre dos o más opciones no se toman en consideración, de modo que, por ejemplo, las siguientes cuatro versiones del mismo comando producen exactamente el mismo resultado:

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

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

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

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

Parece claro que la última versión es más fácil de leer: se puede ver a simple vista cuántas opciones se han usado y cuáles son. En un ejemplo como este, con sólo dos opciones, tal vez ello no sea especialmente importante; pero en casos en los que hay una larga lista de opciones, que cada una de ellas tenga su propia línea en el fichero fuente ayuda a *comprender* mejor lo que el fichero fuente está pidiéndole a ConT_EXt y también, en su caso, a localizar un posible error. Por ello esta última (o alguna similar) es la forma de escribir los comandos «favorita» de los usuarios.

En cuanto a la sintaxis de las opciones de configuración, véase más adelante ([sección 3.5](#)).

3.4.2 Comandos que configuran el funcionamiento de otros comandos (`\setupAlgunaCosa`)

Ya hemos visto que los comandos que admiten varias posibilidades en su funcionamiento, tienen siempre previsto un funcionamiento por defecto. Si en nuestro

fichero fuente se llama a uno de esos comandos varias veces, y en todas ellas queremos alterar el funcionamiento por defecto, mucho más cómodo y eficiente que indicar en cada llamada las opciones de funcionamiento, resulta cambiar el funcionamiento por defecto. Para ello casi siempre se dispone de un comando cuyo nombre empieza con `\setup` seguido por el nombre del comando cuyo funcionamiento por defecto queremos alterar.

El comando `\framed` que nos viene sirviendo de ejemplo en esta sección, sigue siendo un buen ejemplo. Así, si en nuestro documento se usarán muchos marcos, pero todos ellos deben tener unas medidas precisas, lo mejor es reconfigurar el funcionamiento de `\framed` mediante `\setupframed`. Y así

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

hará que, a partir de ese momento, cada vez que llamemos a `\framed` este vaya a generar, por defecto, un marco de 3 centímetros de ancho por un centímetro de alto, sin necesidad de especificárselo expresamente en cada llamada.

En ConT_EXt hay cerca de 300 comandos que permiten configurar el funcionamiento de otros comandos. Así podemos configurar el funcionamiento por defecto de los marcos (`\framed`), de las listas («`itemize`») de los títulos de capítulos (`\chapter`), o de los de sección (`\section`), etc.

3.4.3 Creación de versiones personalizadas de comandos configurables (`\defineAlgunaCosa`)

Claro está que, siguiendo con el ejemplo de `\framed`, si nuestro documento usa varios tipos de marcos, cada uno de ellos con diferentes medidas, lo ideal sería que pudiéramos *predefinir* diferentes configuraciones de `\framed`, y asociar cada una de ellas a un nombre concreto para poder usar una u otra según convenga. Eso podríamos lograrlo en ConT_EXt mediante el comando `\defineframed` cuya sintaxis es:

```
\defineframed[Nombre][Configuración]
```

donde *Nombre* es el nombre que se asignará al tipo específico de marco que se configurará; y *Configuración* es la concreta configuración asociada a dicho nombre.

El efecto de todo ello será que la configuración indicada queda asociada al nombre que hayamos establecido, el cual, a todos los efectos, funcionará como si fuera un nuevo comando, que podremos usar en cualquier contexto en el que habríamos podido usar el comando original (`\framed`).

Esta posibilidad no sólo existe en el caso concreto de `\framed`, sino en la mayor parte de los comandos para los que existe un comando `\setup`. La combinación `\defineAlgunaCosa + \setupAlgunaCosa`, constituye un mecanismo que dota a ConT_EXt de una extremada potencia y flexibilidad. Si analizamos detenidamente lo que el comando `\defineAlgo` hace, resulta que:

- En primer lugar, clona un determinado comando susceptible de admitir varias configuraciones.
- Asocia ese clon al nombre de un nuevo comando.
- Por último, establece una configuración predeterminada para el clon, distinta de la configuración del comando original.

En el ejemplo que hemos puesto configurábamos nuestro marco especial en el mismo momento en el que lo creábamos. Pero también podemos crearlo primero y configurarlo más tarde, pues, como he dicho, una vez creado el clon este se puede usar donde se hubiera podido usar el original. Y así, por ejemplo, si hemos creado un tipo de marco al que hemos llamado «MiMarcoEspecial» podremos configurarlo con `\setupframed` indicando qué marco concreto queremos configurar. En este caso el comando `\setup` recibirá un argumento nuevo con el nombre del marco a configurar:

```
\defineframed[MiMarcoEspecial]

\setupframed
  [MiMarcoEspecial]
  [ ... ]
```

3.5 Recapitulación sobre la sintaxis de los comandos y de sus opciones, y sobre el uso de corchetes y llaves en las llamadas a los mismos

Recapitulando lo hasta ahora visto, resulta que en ConT_EXt

- Los comandos propiamente dichos siempre empiezan con el carácter «\».
- Algunos comandos pueden recibir uno o varios argumentos.
- Los argumentos que dicen al comando *cómo* debe funcionar o, de alguna manera afectan a lo que hará, se introducen entre corchetes.
- Los argumentos que dicen al comando sobre qué porción de texto debe actuar, se introducen entre llaves.

Cuando el comando actuará solamente sobre una letra, como es, por ejemplo, el caso de `\buildtextcedilla` (por poner un ejemplo), pueden omitirse las llaves alrededor del argumento: el comando se aplicará al primer carácter que no sea un espacio en blanco.

- Algunos de los argumentos pueden ser opcionales, en cuyo caso, podemos omitirlos. Pero lo que no puede hacerse nunca es alterar el orden de los argumentos que el comando espera.

Los argumentos introducidos entre corchetes, por su parte, pueden ser de varios tipos. Principalmente:

- Pueden recibir un solo valor, que casi siempre consistirá en un nombre o frase.
- Pueden recibir varias opciones, en cuyo caso estas pueden
 - Estar representadas por una sola palabra, que puede ser un nombre simbólico (del que ConT_EXt conoce el significado), una medida o dimensión, un número, el nombre de algún otro comando, etc.
 - Consistir en nombres de variables a las que hay que asignarles un valor. En este caso en la definición oficial del comando (véase [sección 3.6](#)) siempre se nos dice qué tipo de valor espera cada una de las opciones.
 - ★ Cuando el valor que espera una opción es un texto, éste puede contener espacios en blanco y también comandos. A veces, en estos casos, conviene encerrar entre llaves el valor de la opción.
 - ★ Cuando el valor que espera una opción es un comando, normalmente podremos indicar como valor de la opción más de un comando, aunque a veces deberemos encerrar todos los comandos asignados a la opción entre llaves. También deberemos encerrar entre llaves el contenido de la opción si alguno de los comandos incluidos en ella recibe alguna opción entre corchetes.

En uno y otro caso las distintas opciones que haya de recibir el mismo argumento se separarán por comas. Los espacios en blanco y saltos de línea (que no sean dobles) entre las distintas opciones son ignorados. También se ignoran los espacios en blanco y saltos de línea entre los distintos argumentos de un comando.

- Por último, en ConT_EXt nunca se da el caso de que un mismo argumento reciba simultáneamente opciones consistentes en una palabra y opciones consistentes en una variable a la que haya que asignar explícitamente un valor. Es decir: Puede haber opciones del tipo

`\comando[Opción1, Opción2, ...]`

y otras del tipo

```
\comando[Variable1=valor, Variable2=valor, ...]
```

Pero nunca encontraremos una mezcla de ambas:

```
\comando[Opción1, Variable1=valor, ...]
```

3.6 El listado oficial de comandos de ConT_EXt

En la documentación de ConT_EXt hay un documento especialmente importante en el que hay un listado de todos los comandos indicando, para cada uno de ellos, cuántos argumentos espera, y de qué tipo es así como las distintas opciones que contempla y sus valores admisibles. Este documento se llama «`setup-en.pdf`», es generado automáticamente para cada versión nueva de ConT_EXt y se localiza en el directorio «`tex/texmf-context/doc/context/documents/general/qrcs`».

En realidad, en el directorio «qrc» hay siete versiones de dicho documento, una en cada uno de los idiomas para los que existe una interfaz de ConT_EXt: alemán, checo, francés, holandés, inglés, italiano y rumano. Para cada uno de esos idiomas hay dos documentos en el directorio: uno llamado «`setup-CodIdioma`» (donde `CodIdioma` son las dos letras de identificación internacional del idioma de que se trate) y un segundo documento llamado «`setup-mapping-CodIdioma`». Este segundo documento contiene un listado de comandos ordenados alfabéticamente e indicando el *prototipo* del comando, pero sin más información sobre los posibles valores de cada argumento.

Este documento es fundamental para aprender a usar ConT_EXt, pues en él podemos buscar si existe o no cierto comando; lo que es particularmente útil teniendo en cuenta la combinación de COMANDO (o ENTORNO) + `setupCOMANDO` + `defineCOMANDO`. Por ejemplo, si yo sé que una línea en blanco se introduce con el comando `\blank` puedo buscar si existe un comando que me permita configurarlas llamado `\setupblank` y otro que me permita crear configuraciones personalizadas de líneas en blanco (`\defineblank`).

«`setup-en.pdf`» es, por lo tanto, fundamental, para el aprendizaje de ConT_EXt. Me gustaría, no obstante, que, en primer lugar, informara de si un comando sólo funciona en Mark II o en Mark IV, y, sobre todo, que además de informar sobre el número y tipo de argumentos que cada comando admite, dijera para qué sirven dichos argumentos. Ello reduciría mucho las deficiencias de la documentación de ConT_EXt. Hay algunos comandos que admiten argumentos opcionales que yo ni menciono en esta introducción porque no sé para qué sirven y, al ser opcionales, tampoco es imprescindible mencionar dicho argumento. Ello resulta tremendamente frustrante.

3.7 Definición de nuevos comandos

3.7.1 Mecanismo general de definición de nuevos comandos

Acabamos de ver como mediante `\defineAlgo` podemos clonar un comando pre-existente y, a partir de él, elaborar una nueva versión del mismo, que funcionará, a todos los efectos, como un comando nuevo.

Junto con esa posibilidad, que sólo está disponible para algunos comandos concretos (bastantes, ciertamente, pero no todos), ConT_EXt incorpora un mecanismo general para la definición de nuevos comandos, el cual es extremadamente potente pero, en algunos de sus usos, también muy complejo. En un texto como el presente, pensado para principiantes, creo que lo mejor es introducirlo a partir de algunos de sus usos más simples. El más simple de todos es el de la asociación de fragmentos concretos de texto a una palabra, de tal modo que cada vez que en el fichero fuente aparezca dicha palabra, esta sea reemplazada por el texto vinculado a ella. Ello nos permitirá, de un lado, ahorrarnos mucho tiempo de tecleado y además, como ventaja extra, se reducen las posibilidades de cometer errores al teclear, al tiempo que nos aseguramos de que el texto de que se trate siempre se escriba igual.

Imaginemos, por ejemplo, que estamos escribiendo un tratado sobre la aliteración en los textos latinos y en él se cita con cierta frecuencia la frase latina «O Tite tute Tati tibi tanta tyranne tulisti» (¡Oh Tito Tacio, tirano, tú mismo te produjiste tan terribles desgracias!). Es una frase relativamente larga, en la que dos de las palabras son nombres propios y deben ir en mayúsculas, y en la que, admitámoslo, por muy amantes que seamos de la poesía latina, es fácil que nos “trastabillemos” al escribirla. En tal caso podríamos, simplemente, escribir en el preámbulo de nuestro fichero fuente:

```
\define\Tite{«O Tite tute Tati tibi tanta tyranne tulisti»}
```

A partir de tal definición, cada vez que en nuestro fichero fuente aparezca el comando `\Tite`, será sustituido por la frase indicada, la cual, además, se entrecomillará exactamente igual que estaba entrecomillada en la definición original, lo que nos permite asegurarnos de que la forma en que aparezca esa frase será siempre la misma. Podríamos también haberla escrito con cursivas, con un tamaño de letra más grande... como queramos. Lo importante es que sólo tenemos que escribirla una vez, y, a lo largo del texto, se reproducirá exactamente igual a como se escribió, tantas veces como se desee. También podríamos crear dos versiones del comando, llamadas, respectivamente `\Tite` y `\tite`, según la frase se deba escribir o no con la primera letra en mayúsculas. El texto de reemplazo puede, por otra parte, ser puro texto, o incluir comandos, o formar expresiones matemáticas en las que hay

más posibilidades de equivocarse al teclearlas (al menos para mí) y así, por ejemplo si en nuestro texto continuamente debe aparecer la expresión (x_1, \dots, x_n) , podemos crear un comando que la represente. Por ejemplo

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

de tal modo que siempre que en el texto aparezca `\xvec` sea sustituido por la expresión asociada a él.

La sintaxis general del comando `\define` es la siguiente:

```
\define[NumArgumentos]\NombreComando{TeXtoReemplazo}
```

donde

- **NumArgumentos** se refiere al número de argumentos que recibirá el nuevo comando. Si no ha de recibir ninguno, como en los ejemplos hasta ahora puestos, esta parte se omitirá.
- **NombreComando** se refiere al nombre que tendrá el nuevo comando. Aquí se aplican las reglas generales relativas a los nombres de los comandos de tal modo que el nombre puede ser un sólo carácter que no sea una letra, o una o más letras, sin incluir ningún carácter que sea una «no letra».
- **TextoReemplazo** contiene el texto que sustituirá al nombre del nuevo comando cada vez que este sea encontrado en el fichero fuente.

La posibilidad de dotar a los nuevos comandos de argumentos en su definición, dota a este mecanismo de gran flexibilidad, pues permite definir un texto de reemplazo variable según cuáles hayan sido los argumentos recibidos.

Por ejemplo: imaginemos que queremos escribir un comando que escriba el principio de una carta comercial. Una versión muy simple del mismo sería:

```
\define\EncabezadoCarta{
  \rightaligned{Pedro Navajas}\par
  \rightaligned{Consultor}\par
  En Murcia, a \date\par
  Muy Señor mío:\par
}
```

pero sería preferible una versión del comando que, en el encabezado, escribiera el nombre del destinatario. Para ello habría que usar un parámetro que comunicara al nuevo comando el nombre del destinatario. Ello exigiría redefinir el comando de la siguiente manera:

```
\define[1]\EncabezadoCarta{
  \rightaligned{Pedro Navajas}\par
  \rightaligned{Consultor}\par
  En Murcia, a \date\par
  Estimado Sr. #1:\par
}
```

Obsérvese que hemos introducido dos cambios en la definición. En primer lugar, entre la palabra clave `\define` y el nombre de nuestro nuevo comando, hemos incluido un 1 entre corchetes ([1]), con esto le indicamos a ConT_EXt que el comando que estamos definiendo recibirá un argumento. Más adelante, en la última línea de la definición del comando, hemos escrito «Estimado Sr: #1», usando el carácter reservado «#». Con ello indicamos que en el punto del texto de reemplazo en donde aparece el «#1», debe insertarse, en su lugar, el contenido del primer argumento. Si hubiera dos parámetros, «#1» se referiría al primer parámetro y «#2» se referiría al segundo. Cuando un comando admite argumentos, para invocar al comando (en el fichero fuente) tras el nombre del comando hay que incluir, entre llaves, los argumentos. Cada argumento con sus propias llaves. De modo que al comando que acabamos de definir, en el texto de nuestro fichero fuente habría que invocarlo de la siguiente manera:

```
\EncabezadoCarta{Nombre del Destinatario}
```

Por ejemplo: `\EncabezadoCarta{Antonio Moreno Gómez}`.

Todavía podríamos mejorar algo la anterior función, pues en ella se asume que la carta se envía a un hombre (pone «Estimado Sr.») tal vez podríamos incluir otro parámetro para diferenciar entre destinatario masculino y destinatario femenino. Por ejemplo:

```
\define[2]\EncabezadoCarta{
  \rightaligned{Pedro Navajas}\par
  \rightaligned{Consultor}\par
  En Murcia, a \date\par
  #1\ #2:\par
}
```

de tal modo que a la función se la llamaría, por ejemplo con

```
\EncabezadoCarta{Estimada Sra.}{Eloísa Martínez López}
```

aunque esto último no es muy elegante (desde el punto de vista de la programación). Sería preferible que para el primer argumento se definieran valores simbólicos (hombre/mujer; 0/1; h/m) de tal modo que la propia macro escogiera el texto adecuado según dicho valor. Pero explicar cómo conseguir eso exige meternos en más vericuetos de los que a estas alturas creo que el lector novato puede entender.

3.7.2 Creación de nuevos entornos

Para crear un nuevo entorno ConT_EXt proporciona el comando `\definestartstop` cuya sintaxis es la siguiente:

```
\definestartstop[Nombre] [Opciones]
```



En la definición *oficial* de `\definestartstop` (véase [sección 3.6](#)) hay un argumento adicional que no he puesto más arriba porque es opcional y no he conseguido averiguar para qué sirve.

Ni la excursión introductoria de ConT_EXt ni el incompleto manual de referencia lo explican. Supuse que ese argumento (que se debe introducir entre el nombre y la configuración) pudiera ser el nombre de algún entorno ya existente que sirviera de modelo inicial al nuevo entorno, pero mis pruebas han demostrado que mi suposición es errónea. He buscado en la lista de correo de ConT_EXt y no he llegado a ver en ella ningún uso de ese posible argumento.

donde

- **Nombre** es el nombre que tendrá el nuevo entorno.
- **Configuración** permite configurar el comportamiento del nuevo entorno. Para configurarlo contamos con las siguientes variables:
 - **before** – Comandos que deben ejecutarse antes de entrar en el entorno.
 - **after** – Comandos que se ejecutarán al salir del entorno.
 - **style** – Estilo que ha de tener el texto del nuevo entorno.
 - **setups** – Conjunto de comandos creado mediante `\startsetups ... \stopsetups`. Este comando y su uso no es objeto de explicación en la presente introducción.
 - **color, inbetween, left, right** – Opciones no documentadas que no he conseguido hacer funcionar. Algunas, por su nombre, cabría suponer para qué sirven, por ejemplo **color**, pero por más pruebas que he hecho, indicando algún valor para esa opción no observo ningún cambio en el interior del entorno.



Un ejemplo de definición de entorno podría ser el siguiente:

```
\definestartstop
[TextoConBarra]
[before=\startmarginrule\noindeatation,
  after=\stopmarginrule,
  style=\ss\sl
]

\starttext

Las dos primeras leyes fundamentales de la estupidez humana afirman
sin ambigüedad que:

\startTextoConBarra

\startitemize[n,broad]

\item Siempre e inevitablemente cada uno de nosotros subestima el
  número de individuos estúpidos que circulan por el mundo.

\item La probabilidad de que una persona determinada sea estúpida es
  independiente de cualquier otra característica de la misma
  persona.

\stopitemize
```



```
\stopTextoConBarra
```

```
\stoptext
```

El resultado sería:

Las dos primeras leyes fundamentales de la estupidez humana afirman sin ambigüedad que:

1. *Siempre e inevitablemente cada uno de nosotros subestima el número de individuos estúpidos que circulan por el mundo.*
2. *La probabilidad de que una persona determinada sea estúpida es independiente de cualquier otra característica de la misma persona.*

Si queremos que nuestro nuevo entorno sea un grupo (sección 3.8.1), de tal modo que cualquier alteración del funcionamiento normal de ConT_EXt que se haga dentro de su interior, desaparezca al salir del entorno, debemos incluir en la opción «before» el comando `\bgroup` y en la opción «after» el comando `\egroup`.

3.8 Otros conceptos fundamentales

Hay otras nociones, además de los comandos, que resultan fundamentales para comprender la lógica del funcionamiento de ConT_EXt algunas de ellas, por su complejidad, no son propias de una introducción y, por ello, no se llegarán a tratar en este documento; pero hay dos nociones que conviene examinar ya: los grupos y las dimensiones.

3.8.1 Grupos

Un grupo es un fragmento del fichero fuente bien delimitado que ConT_EXt usa como *unidad de trabajo* (enseguida se explica lo que eso significa). Todo grupo tiene un principio y un final que han de indicarse expresamente. Un grupo se inicia:

- Con el carácter reservado «{» o con el comando `\bgroup`.
- Con el comando `\begingroup`
- Con el comando `\start`

- Con la apertura de ciertos entornos (comandos `\startAlgo`).
- Al iniciar un entorno matemático (con el carácter reservado «\$»).

y se cerrará

- Con el carácter reservado «}» o con el comando `\egroup`.
- Con el comando `\endgroup`
- Con el comando `\stop`
- Con el cierre del entorno (comandos `\stopAlgo`).
- Al salir de un entorno matemático (con el carácter reservado «\$»).

También ciertos comandos generan automáticamente un grupo como, por ejemplo, `\hbox`, `\vbox` y, en general, los comandos vinculados con la creación de cajas¹. Fuera de esos últimos casos (grupos generados automáticamente por ciertos comandos), la forma de cerrar un grupo ha de ser coherente con la forma en que se abrió. Es decir: un grupo iniciado con «{» se debe cerrar con «}», y un grupo iniciado con `\begingroup` se debe cerrar con `\endgroup`. Esta regla sólo tiene una excepción y es que el grupo iniciado con «{» se puede cerrar con `\egroup`, y el grupo iniciado con `\bgroup` se puede cerrar con «}»; lo que, en realidad, significa que «{» y `\bgroup` son totalmente sinónimos e intercambiables entre sí, al igual que «}» y `\egroup`.

Los comandos `\bgroup` y `\egroup` se diseñaron para que pudieran definirse comandos que abrieran un grupo y otros que lo cerraran. Para ello, por razones internas relativas a la sintaxis de T_EX, esos grupos no podían abrirse y cerrarse con las llaves, pues ello generaría, en el fichero fuente, la presencia de llaves no balanceadas, cosa que genera siempre un error de compilación.

Los comandos `\begingroup` y `\endgroup`, por el contrario, no son intercambiables con las llaves o con los comandos `\bgroup` ... `\egroup`, por lo que un grupo iniciado con `\begingroup` tiene que cerrarse necesariamente con `\endgroup`. Estos últimos comandos se diseñaron para poder hacer una comprobación de errores mucho más profunda. En general los usuarios normales no tienen por qué utilizarlos.

Puede haber grupos anidados (es decir, un grupo dentro de otro grupo), pero, en tal caso, el orden de cierre de los grupos debe ser coherente con el orden en el que se abrieron: todo subgrupo se debe cerrar dentro del grupo en el que se inició. También puede haber grupos vacíos, que se generan con «{}». Un grupo vacío, en principio, no produce ningún efecto sobre el documento final, pero puede servir, por ejemplo, para indicar el fin del nombre de un comando.

El principal efecto de los grupos es el del encapsulamiento de su contenido: como regla general, las definiciones, formatos y asignaciones de valor que se realicen dentro de un grupo, se “olvidan” una vez que se sale del grupo. De esta manera, si queremos que ConT_EXt temporalmente altere su funcionamiento normal, lo más

¹ La noción de *caja* es también una noción central de ConT_EXt, pero su explicación no se incluye en la presente introducción.

eficiente es crear un grupo y, dentro de él, alterar ese funcionamiento. Así, al salir del grupo, se restaurarán todos los valores y formatos previos a él. Ya hemos visto algunos ejemplos de este funcionamiento al mencionar comandos tales como `\it`, `\bf`, `\sc`, etc. Pero esto no pasa sólo con los comandos de formato: el grupo en cierto modo *aisla* su contenido, de tal modo que cualquier cambio en cualquiera de las muchas variables internas que ConT_EXt continuamente maneja, mantendrá su eficacia sólo mientras estemos dentro del grupo en el que dicho cambio tuvo lugar. Asimismo, un comando definido dentro de un grupo, no será conocido fuera de él.

Así, si procesamos el siguiente ejemplo

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

veremos que la primera vez que ejecutamos el comando `\A`, el resultado se corresponde con el de su definición inicial («B»). Luego, hemos creado un grupo y redefinido el comando `\A` dentro de él. Si ahora ejecutamos, dentro del grupo, el comando, nos dará la nueva definición (en nuestro ejemplo, «C»), pero, cuando salimos del grupo en el que el comando `\A`, se redefinió, si volvemos a ejecutarlo, volverá a escribir «B». La definición hecha dentro del grupo se «olvida» una vez que salimos de él.

Otro uso posible de los grupos tiene que ver con aquellos comandos o instrucciones diseñados para aplicarse exclusivamente al carácter que se escriba a continuación de los mismos. En tal caso, si queremos que el comando se aplique a más de un carácter, debemos encerrar en un grupo aquellos caracteres a los que queremos que se aplique el comando o instrucción. Así, por ejemplo, el carácter reservado «`^`» que ya sabemos que, usado en un entorno matemático, convierte en superíndice el carácter que vaya a continuación; y así, por ejemplo, si escribimos «`4^2x`» obtendremos « 4^2x ». Pero si escribimos «`4^{2x}`» obtendremos « 4^{2x} ».

En fin: una tercera utilidad del agrupamiento es la de indicar a ConT_EXt que lo encerrado dentro del grupo debe ser tratado como una sola cosa. Esta es la razón de que antes ([sección 3.5](#)) se haya dicho que en ciertas ocasiones conviene encerrar entre llaves el contenido de alguna opción de un comando.

3.8.2 Dimensiones

Aunque podemos usar perfectamente ConT_EXt sin preocuparnos por las dimensiones, no podremos hacer uso de todas sus posibilidades de configuración sin tenerlas muy en consideración. Porque en gran medida la perfección tipográfica lograda por

T_EX y sus derivados está en la gran atención que internamente el sistema presta a las dimensiones. Los caracteres tienen dimensiones; el espacio entre palabras, o entre líneas, o entre párrafos, tienen dimensiones, las líneas tienen dimensiones; los márgenes, los encabezados y pies de página... Para casi todo elemento de la página en el que podamos pensar habrá alguna dimensión.

En ConT_EXt las dimensiones se indican mediante un número decimal seguido de la unidad de medida. Se pueden usar las unidades de medida recogidas en la [tabla 3.2](#).

Nombre	Nombre en ConT _E Xt	Equivalencia
Pulgada	in	1 in = 2.54 cm
Centímetro	cm	2.54 cm = 1 pulgada
Milímetro	mm	100 mm = 1 cm
Punto	pt	72.27 pt = 1 pulgada
Punto grande	bp	72 bp = 1 pulgada
Punto escalado	sp	65536 sp = 1 punto
Pica	pc	1 pc = 12 puntos
Punto Didot	dd	1157 dd = 1238 puntos
Cícero	cc	1 cc = 12 didots
	em	
	ex	

Tabla 3.2 Unidades de medida en ConT_EXt

Las tres primeras unidades de la [tabla 3.2](#) son medidas de longitud estándar; la primera usada en el mundo anglosajón y las otras dos usadas fuera de él. Las restantes unidades proceden del mundo de la tipografía. Las dos últimas, para las que no he puesto equivalencia son unidades de medida relativa, basadas en la fuente activa en cada momento. 1 «em» es igual a la anchura de la «M» y un «ex» equivale a la altura de una «x». El uso de medidas relativas al tamaño de la fuente permite diseñar macros que se vean igual de bien sea cual sea la fuente utilizada en cada instante. Por ello, en general, es algo que se recomienda.

Salvo algunas excepciones muy contadas, podemos usar la unidad de medida que prefiramos, pues ConT_EXt la convertirá internamente. Pero siempre que se indique una dimensión es obligatorio indicar la unidad de medida, incluso aunque queramos indicar una medida de “0”, hay que decir “0pt” ó “0cm”. Entre el número y el nombre de la unidad, podemos o no dejar un espacio en blanco. Si la unidad tiene una parte decimal, podemos usar como separador de decimales, indistintamente, el punto (.) o la coma (,).

Las medidas se usan, normalmente, como opción de algún comando. Pero también podemos asignar directamente un valor a alguna medida interna de ConT_EXt siempre que conozcamos el nombre de la misma. Por ejemplo: la indentación de la primera línea de un párrafo ordinario, es controlada internamente por ConT_EXt

mediante una variable llamada `\parindent`. Asignando expresamente un valor a dicha variable habremos alterado, a partir del punto en el que lo hagamos, la medida que utilizará ConT_EXt. Y así, por ejemplo si queremos suprimir la indentación de la primera línea nos bastará con escribir en nuestro fichero fuente:

```
\parindent=0pt
```

También podríamos haber escrito `\parindent 0pt` (sin el signo igual) o `\parindent0pt` sin espacio de separación entre el nombre de la medida y su valor.

No obstante, asignar directamente un valor a una medida interna, se considera «poco elegante», en general se recomienda para ello usar los comandos que controlan esa variable, y hacerlo en el preámbulo del fichero fuente. Lo contrario se traduce en ficheros fuente muy difíciles de depurar, por no tener todos los comandos de configuración en el mismo lugar, y en los que es realmente difícil obtener cierta coherencia en las características tipográficas.

Algunas dimensiones utilizadas por ConT_EXt son «elásticas», es decir, según el contexto pueden adoptar una medida u otra. Esas medidas se asignan con la siguiente sintaxis:

```
\NombreMedida plus MaxIncremento minus MaxDecremento
```

Por ejemplo

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

Con esta instrucción le estamos indicando a ConT_EXt que asigne a `\parskip` (que representa el espacio de separación entre párrafos) una medida *normal* de 3 puntos, pero que si las necesidades de composición de la página se lo exigen, la medida pueda llegar a ser de hasta 5 puntos (3 plus 2) o de sólo 2 puntos (3 minus 1). En estos casos se dejará que sea ConT_EXt quien elija, para cada página la separación entre un mínimo de 2 puntos y un máximo de 5 puntos.

3.9 Método para el autoaprendizaje de ConT_EXt

La inmensa cantidad de comandos y opciones de ConT_EXt resulta verdaderamente abrumadora y puede producirnos la sensación de que nunca llegaremos a aprender bien a trabajar con él. Esa impresión sería errónea pues una de las ventajas de ConT_EXt es el tratamiento uniforme que da a todas sus estructuras: aprendiendo bien unas cuantas estructuras, y sabiendo, más o menos, para qué sirven las restantes, cuando necesitemos alguna utilidad extra será relativamente fácil aprender

a usarla. Por ello yo concibo esta introducción como una especie de *entrenamiento* que nos preparará para indagar por nuestra cuenta.

Para crear un documento con ConT_EXt probablemente sólo sea imprescindible el conocimiento de los siguientes cinco aspectos (lo que podríamos llamar el *Top Five* de ConT_EXt):

1. Saber crear el fichero fuente o en su caso el proyecto; lo que es objeto de explicación en el [capítulo 4](#) de esta introducción.
2. Establecer la fuente principal del documento, y conocer los comandos básicos para cambiar la fuente y el color ([capítulo ??](#)).
3. Conocer los comandos básicos para estructurar el contenido de nuestro documento, tales como capítulos, secciones, subsecciones, etc. Todo ello es objeto de explicación en nuestro [capítulo 7](#).
4. Tal vez, saber manejar el entorno *itemize* que es objeto de explicación relativamente detallada en la [sección 12.3](#).
5. ... y poco más.

Del resto lo único que necesitamos es saber que es posible hacerlo. Ciertamente nadie usará una utilidad si no sabe que ésta existe. En esta introducción se explican muchas de ellas; pero, sobre todo, se ve de forma reiterada cómo actúa ConT_EXt siempre ante cierto tipo de construcción:

- En primer lugar habrá un comando que permita realizarla.
- En segundo lugar hay casi siempre un comando que permite configurar y pre-determinar cómo se llevará a cabo la tarea; comando éste cuyo nombre empieza por `\setup` y suele coincidir con el del comando básico.
- Por último suele existir también la posibilidad de crear algún comando nuevo que sirva para realizar tareas similares, aunque con una configuración diferente.

Para ver si estos comandos existen o no, se puede usar el listado oficial de comandos (véase la [sección 3.6](#)), el cual también nos informará de las opciones de configuración que estos comandos admiten. Y aunque en una primera aproximación los nombres de estas opciones nos pueden parecer *crípticos*, pronto comprobaremos que hay opciones que se repiten en muchos comandos y que funcionan igual en todos ellos, o de forma muy parecida. Si tenemos dudas sobre qué hace una opción, o cómo trabaja, bastará con generar un documento y probarla. También podemos indagar en la abundante documentación de ConT_EXt. Tal y como es corriente en el mundo del software libre, «ConT_EXt Standalone» incluye en la distribución las fuentes de casi toda su documentación. Una utilidad como «**grep**» (para sistemas

GNU Linux) puede ayudarnos a buscar si en alguno de esos ficheros fuente se utiliza el comando o la opción que nos plantea dudas, para que podamos tener un ejemplo a la vista.

Así concebido el aprendizaje de ConT_EXt esta introducción explica con detalle los cinco (en realidad, cuatro) aspectos que he destacado, y muchos más: conforme vayamos leyendo se irá formando en nuestra mente una imagen clara de la secuencia: *comando que realiza la tarea – comando que configura al anterior – comando que permite crear un comando similar*. Aprenderemos también algunas de las principales estructuras de ConT_EXt, y sabremos para qué sirven.

Capítulo 4

Ficheros fuente y proyectos

Sumario: 4.1 Codificación de los ficheros fuente; 4.2 Caracteres en el (o en los) fichero(s) fuente que ConT_EXt trata de forma especial; 4.2.1 Espacios en blanco y tabuladores; 4.2.2 Saltos de línea; 4.2.3 Guiones; 4.3 Proyectos simples y proyectos multifichero; 4.4 Estructura del fichero fuente en los proyectos simples; 4.5 Gestión multifichero al estilo de T_EX; 4.5.1 El comando `\input`; 4.5.2 `\ReadFile` y `\readfile`; 4.6 Proyectos de ConT_EXt propiamente dichos; 4.6.1 Ficheros de entorno (*environment*); 4.6.2 Componentes y productos; 4.6.3 Proyectos propiamente dichos; 4.6.4 Aspectos comunes a entornos, componentes, productos y proyectos;

Como ya sabemos, cuando se trabaja con ConT_EXt se parte siempre de un fichero de texto en el que junto con el contenido del futuro documento, se incluyen ciertas instrucciones que explican a ConT_EXt las transformaciones que debe aplicar para, a partir del fichero fuente, generar nuestro documento final en PDF, correctamente formateado.

Pensando en lectores que hasta ahora no conozcan una forma de trabajar distinta de la propia de los procesadores de texto, creo que merece la pena detenerse un momento en el fichero fuente propiamente dicho. O mejor: en los ficheros fuente, pues hay casos en los que habrá un solo fichero fuente, y otros en los que, para un mismo documento final, se usarán varios ficheros fuente diferentes. En este último supuesto se puede hablar de «proyectos multifichero».

4.1 Codificación de los ficheros fuente

El —o los— ficheros fuente, han de ser ficheros de texto. En terminología informática se llama así a un fichero que contiene solamente texto formado por caracteres legibles por los humanos, y que no incluye en su interior códigos binarios. A estos ficheros se les llama también ficheros de *texto simple* o *texto plano*, en lo que es una traducción demasiado literal de la expresión inglesa «*plain text*».

Como quiera que internamente los sistemas informáticos sólo procesan números binarios, un fichero de texto, en realidad, está compuesto de *números* que representan *caracteres*. Para vincular un número con un concreto carácter se utiliza

una *tabla*. Existen, para los ficheros de texto, varias tablas posibles. Con la expresión *codificación de un fichero de texto* se hace referencia a la concreta tabla de correspondencia de caracteres que un concreto fichero de texto utilice.

La existencia de distintas tablas de codificación para los ficheros de texto es una consecuencia de la propia historia de la informática. En los primeros momentos de desarrollo, cuando la memoria y la capacidad de almacenamiento de los dispositivos informáticos eran escasas, se decidió usar para codificar los ficheros de texto una tabla, llamada ASCII (siglas de «*American Standard Code for Information Interchange*») que sólo admitía 128 caracteres y que había sido establecida en 1963 por el Comité Estadounidense de Estándares. Es obvio que 128 caracteres no es suficiente para representar todos los caracteres y símbolos usados en todos los idiomas del mundo; pero era más que suficiente para representar el inglés que es, de todas las lenguas occidentales, la que menos caracteres tiene, por no conocer ningún tipo de símbolos diacríticos. La ventaja que ofrecía el uso de ASCII estaba en que los ficheros de texto ocuparían muy poco espacio, pues, como 127 (el número más alto de la tabla) se puede representar con un número binario de 7 cifras, y los primeros ordenadores usaban como unidad de medida de la memoria el byte, un número binario de 8 cifras, cualquier carácter de la tabla cabría en un solo byte. Incluso, como el byte tiene 8 cifras y ASCII sólo usaba 7 cifras, quedaba espacio para añadir algunos otros caracteres que permitieran representar a otras lenguas.

Pero cuando el uso de la informática se expandió quedó manifiesta la insuficiencia de ASCII y se hizo preciso elaborar tablas *alternativas* que incluyeran caracteres no conocidos por el alfabeto inglés tales como, por ejemplo, la «ñ» española, las vocales acentuadas, los signos de apertura de interrogaciones y exclamaciones, la «ç» catalana o francesa, etc. Aunque, por otra parte, tampoco hubo un acuerdo inicial respecto de cuáles deberían ser esas *tablas alternativas* de ASCII, sino que las distintas empresas dedicadas a la informática fueron abordando, cada una por su cuenta, el problema; y así se fueron creando, no sólo tablas específicas para diferentes idiomas o grupos de idiomas, sino también tablas distintas según la empresa que las hubiera creado (Microsoft, Apple, IBM, etc.).

Sólo con el incremento de la memoria de los ordenadores y el abaratamiento de los dispositivos de almacenaje y correlativo aumento de su capacidad, surgió la idea de crear una sola tabla que sirviera para todos los idiomas. Pero, una vez más, en realidad no se creó una sola tabla que contuviera todos los caracteres, sino un estándar de codificación (llamado Unicode) y distintas maneras de representarlo (UTF-8, UTF-16, UTF-32, etc.) De todos estos sistemas el que ha terminado por imponerse como estándar de facto es UTF-8, que permite representar prácticamente cualquier lengua viva, y muchas lenguas ya extintas, así como numerosos símbolos adicionales, todo ello usando números de longitud variable (entre 1 y 4 bytes), lo que, a su vez, contribuye a optimizar el tamaño de los ficheros de texto, que no se incrementa *demasiado* respecto de los ficheros que usan ASCII puro.

Hasta la aparición de \LaTeX , los sistemas basados en \TeX —que también nació en los Estados Unidos y cuya lengua natal es, por lo tanto, el inglés— presuponían la codificación en ASCII puro; de tal modo que para utilizar una codificación diferente había que indicarlo de alguna manera en el fichero fuente.

Con \TeX Mark IV asume que la codificación será UTF-8. Aun así, en sistemas informáticos poco actualizados es posible que se siga usando, por defecto, una codificación distinta. No estoy tampoco muy seguro de la codificación por defecto que usan los sistemas Windows, dado que como la estrategia de Microsoft para llegar al gran público consiste en ocultar la complejidad (que sigue subyaciendo,

por supuesto), no hay mucha información disponible (o yo no he sabido localizarla) sobre qué sistema de codificación se usa por defecto.

En todo caso, sea cual sea la codificación por defecto, cualquier editor de textos permite guardar el fichero en la codificación deseada. Los ficheros fuente destinados a ser procesados por ConT_EXt Mark IV deben ser guardados en UTF-8, salvo, claro está, que haya una muy buena razón para utilizar una codificación diferente (aunque no se me ocurre cuál pueda ser esta razón).

Si deseamos procesar un fichero escrito con otra codificación (tal vez un fichero antiguo) podemos

- a. Convertir el fichero a UTF-8, opción recomendada, para la que existen diferentes herramientas; en Linux, por ejemplo, los comandos `iconv` o `recode`.
- b. Indicar a ConT_EXt en el fichero fuente que la codificación no es UTF-8. Para ello hay que usar el comando `\enableregime`, cuya sintaxis es:

`\enableregime[Codificación]`

donde *Codificación* se refiere al nombre por el que ConT_EXt conoce la concreta codificación del fichero en cuestión. En la [tabla 4.1](#) se recogen las distintas codificaciones y los nombres por los que ConT_EXt las conoce.

Codificación	Nombre(s) en ConT _E Xt	Observaciones
Windows CP 1250	cp1250, windows-1250	Europa occidental
Windows CP 1251	cp1251, windows-1251	Cirílico
Windows CP 1252	cp1252, win, windows-1252	Europa occidental
Windows CP 1253	cp1253, windows-1253	Griego
Windows CP 1254	cp1254, windows-1254	Turco
Windows CP 1257	cp1257, windows-1257	Báltico
ISO-8859-1, ISO Latin 1	iso-8859-1, latin1, il1	Europa occidental
ISO-8859-2, ISO Latin 2	iso-8859-2, latin2, il2	Europa oriental
ISO-8859-15, ISO Latin 9	iso-8859-15, latin9, il9	Europa occidental
ISO-8859-7	iso-8859-7, grk	Griego
Mac Roman	mac	Europa occidental
IBM PC DOS	ibm	Europa occidental
UTF-8	utf	Unicode
VISCII	vis, viscii	Vietnamita
DOS CP 866	cp866, cp866nav	Cirílico
KOI8	koi8-r, koi8-u, koi8-ru	Cirílico
Mac Cyrillic	maccyr, macukr	Cirílico
Otras	cp855, cp866av, cp866mav, cp866tat, ctt, dbk, iso88595, isoir111, mik, mls, mnk, mos, ncc	Varias

Tabla 4.1 Principales codificaciones en ConT_EXt

Para documentos en español, las codificaciones más corrientes, además de UTF-8 son las llamadas Latin-1 y Latin-9. De todas formas ConT_EXt Mark IV recomienda

encarecidamente que se utilice UTF-8; recomendación con la que estoy de acuerdo. A partir de aquí en la presente introducción se asume que la codificación será siempre UTF-8.



Junto con `\enableregime` ConT_EXt incluye el comando `\useregime` que admite como argumento el código de una o varias codificaciones. No he encontrado información sobre este comando ni sobre su diferencia con `\enableregime`, sólo algún ejemplo de su uso. Sospecho que `\useregime` está pensado para proyectos complejos en los que se utilicen varios ficheros fuente, previendo que no todos ellos tengan la misma codificación. Pero es sólo una conjetura.

4.2 Caracteres en el (o en los) fichero(s) fuente que ConT_EXt trata de forma especial

Llamaré *caracteres especiales* a un grupo de caracteres diferente de los *caracteres reservados*. Estos últimos son, tal y como se vio en la [sección 3.1](#), aquellos que tienen una significación especial para ConT_EXt de tal modo que no pueden ser usados directamente como caracteres en el fichero fuente. Junto a estos, existe otro grupo de caracteres que, aunque sí son tratados como tales por ConT_EXt cuando los encuentra en el fichero fuente, los somete a reglas especiales. En este grupo hay que incluir espacios en blanco, tabuladores, saltos de línea y guiones.

4.2.1 Espacios en blanco y tabuladores

En el fichero fuente los tabuladores y los espacios en blanco se equiparan a todos los efectos. Un carácter de tabulación (que se obtiene en el teclado con la tecla Tab) será transformado por ConT_EXt en un espacio en blanco. Los espacios en blanco, por su parte, absorben a cualquier otro espacio en blanco (o tabulador) que se encuentre inmediatamente detrás. Así, en el fichero fuente da absolutamente igual escribir

Tararí que vi, Mariví.

o

Tararí que te vi, Mariví.

Las dos frases serían consideradas idénticas por ConT_EXt. Por lo tanto, si queremos introducir entre dos palabras un espacio en blanco adicional, tendremos que usar algún comando de ConT_EXt que se encargue de ello. Normalmente valdrá con «`_`», es decir, el carácter `\` seguido de un espacio en blanco. Pero hay otros procedimientos que serán examinados en el [capítulo 10.3](#) a propósito del espacio horizontal.

La absorción de espacios en blanco consecutivos, permite escribir el fichero fuente destacando visualmente las partes que nos interesa destacar, simplemente aumentando o disminuyendo la indentación usada, con la tranquilidad de saber que ello no afectará para nada al documento final. Así, en el siguiente ejemplo

```
El grupo musical madrileño de finales de los años setenta {\em La
  Romántica Banda Local} compuso canciones de un estilo ecléctico y
muy difícil de clasificar. En su canción "El Egipcio", por ejemplo,
decían: «{\em Esto es una farsa más que una comedia, página muy
  seria de la histeria musical; sueños de princesa, vicios de gitano
  pueden en su mano acariciar la verdad}», mezclando frases
simplemente porque entre ellas hay rimas internas
(comedia-histeria-seria, gitano-mano).
```

puede verse cómo algunas líneas aparecen ligeramente sangradas a la derecha. Son líneas que forman parte de fragmentos en cursiva. El mayor sangrado de las mismas ayuda (al autor) a localizar el final de la cursiva.

Tal vez alguien piense ¡Vaya tabarra! ¡Tener que molestarme en ir sangrando las líneas! Lo cierto es que ese sangrado especial lo realiza automáticamente mi editor de textos (GNU Emacs) cuando está editando un fichero fuente de ConT_EXt. Son ese tipo de pequeñas ayudas, las que hacen que se elija trabajar con cierto editor de textos y no con otro.

La regla de que los espacios en blanco son absorbidos se aplica exclusivamente a los espacios en blanco del fichero fuente que en él sean consecutivos. Por tanto si entre dos espacios en blanco, se introduce en el fichero fuente, por ejemplo, un grupo vacío («{}»), aunque el grupo vacío no producirá nada en el fichero final, su presencia servirá para considerar que los dos espacios en blanco no son consecutivos. Por ejemplo, si escribimos «Tararí {} que te vi, Mariví», obtendremos «Tararí que te vi, Mariví», en donde si se examina con el suficiente detenimiento se podrán apreciar dos espacios consecutivos entre las dos primeras palabras.

Lo mismo ocurre con el carácter reservado «~», aunque su efecto consiste en generar un espacio en blanco, como realmente no lo es, un espacio en blanco seguido de un ~ no absorberá a este último, y un espacio en blanco detrás de ~ tampoco será absorbido.

4.2.2 Saltos de línea

En la mayor parte de los editores de texto, cuando una línea supera la anchura máxima prevista, se inserta automáticamente un salto de línea. También podemos insertar expresamente un salto de línea pulsando la tecla «↵» o «Intro», que en algunos teclados se denomina «Enter» o «Return».

ConT_EXt aplica a los saltos de línea las siguientes reglas:

- a. Un salto de línea individual, se equipara, a todos los efectos, con un espacio en blanco. Por lo tanto, si inmediatamente antes o después del salto de línea

hay algún espacio en blanco o tabulador, éstos serán absorbidos por el salto de línea o por el primer espacio en blanco, y en el documento definitivo se insertará simplemente un espacio en blanco.

- b. Dos o más saltos de línea consecutivos, provocan un salto de párrafo. A estos efectos, se considera que dos saltos de línea son consecutivos si entre el primer salto y el segundo no hay nada salvo espacios en blanco o tabuladores (porque éstos son absorbidos por el primer salto de línea); lo que, en definitiva, significa que una o más líneas consecutivas absolutamente en blanco en el fichero fuente (sin ningún carácter en ellas, o sólo con espacios en blanco o tabuladores) se convierten en un salto de párrafo.

Obsérvese que he dicho «dos o más saltos de línea consecutivos» y luego «una o más líneas en blanco consecutivas», lo que significa que si queremos aumentar la separación entre dos párrafos, no lo conseguiremos simplemente insertando más saltos de línea. Debemos usar para ello algún comando que aumente el espacio vertical. Si sólo queremos una línea extra de separación, podemos usar el comando `\blank`. Pero hay otros procedimientos para aumentar el espaciado vertical. Me remito a la [sección 11.2](#).

En algunas ocasiones, que los saltos de línea se conviertan en espacios en blanco puede dar lugar a la aparición de algún espacio en blanco inesperado e indeseado. Sobre todo cuando escribimos macros, en las que es fácil que «se cuele» un espacio en blanco con el que no se contaba. Para evitar ese efecto puede usarse el carácter reservado «%» que como se sabe, provoca que la línea en donde se encuentre deje de procesarse, lo que implica que el salto final de la línea tampoco se procesará. Y así, por ejemplo, el comando

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

que escribe en cursiva su primer argumento, en negrita el segundo y en versalitas el tercero, insertará un espacio en blanco entre cada uno de los argumentos, mientras que

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

no insertaría ningún espacio en blanco entre ellos, pues el carácter reservado % impide que se procesen los saltos de línea y se conviertan en espacios en blanco.

4.2.3 Guiones

Los guiones son un buen ejemplo de las diferencias entre un teclado de ordenador y un texto impreso. En un teclado normal sólo hay un carácter para el guión («-»); pero en los textos impresos se usan guiones de hasta cuatro longitudes distintas:

- Guiones cortos, como los que se utilizan para la separación silábica de las palabras al final de una línea (-).
- Guiones intermedios, ligeramente más largos que los anteriores (–) y que tienen varios usos, como indicar la entrada de un interlocutor en un diálogo, o para separar la cifra inferior y la superior dentro de un rango, como, por ejemplo en «Págs. 12–33».
- Guiones largos (—), usados a modo de paréntesis dentro de una frase para intercalar una frase dentro de otra.
- Guiones matemáticos (−) con los que se representa una resta, o un número negativo.

Hoy día, en la codificación UTF-8 están disponibles todos esos guiones, y algunos más. Pero como los mismos no se pueden generar desde el teclado, resultan incómodos de introducir en nuestro fichero fuente. Por suerte T_EX previó la necesidad de incluir en nuestro documento más guiones de los que se pueden generar desde el teclado, y diseñó un procedimiento sencillo para conseguirlo. ConT_EXt ha complementado dicho procedimiento añadiendo comandos que también generan estos diferentes guiones. De modo que podemos generar cuatro tipos de guiones por dos procedimientos: mediante el ordinario en ConT_EXt, consistente en la introducción de un comando, o directamente desde el teclado. En la [tabla 4.2](#) se recogen estos procedimientos:

Tipo de guión	Apariencia	Escritura directa	Comando
Guión corto	-	-	<code>\hyphen</code>
Guión intermedio	–	--	<code>\endash</code>
Guión largo	—	---	<code>\emdash</code>
Guión matemático	−	\$-\$	<code>\minus</code>

Tabla 4.2 Guiones en ConT_EXt

Los nombres de los comandos `\hyphen` y `\minus` coinciden con los nombres que se usan en inglés para referirse, respectivamente al guión simple de partición de sílabas y al signo negativo. Los nombres `\endash` y `\emdash` proceden de la terminología tipográfica. «Dash» en inglés es «línea» mientras que «en» y «em» son nombres de unidades de medida usados en la tipografía. Un «en» representa la anchura de una «n» con la fuente actual; un «em» representa la anchura de una «m».

4.3 Proyectos simples y proyectos multifichero

En ConT_EXt podemos usar un solo fichero fuente que incluya absolutamente todo el contenido del documento final, así como todos los detalles relativos a él, en cuyo

caso podríamos hablar de «proyectos simples», o, por el contrario, podemos utilizar varios ficheros fuente entre los que se reparta el contenido de nuestro documento final, caso este en el que hablaríamos de «proyectos multifichero».

Los escenarios típicos en los que puede ser conveniente trabajar con más de un fichero fuente son los siguientes:

- Si se está escribiendo un documento en el que colaboran varios autores, teniendo cada uno de ellos a su cargo una parte concreta, distinta de la de los demás; como por ejemplo, si estamos confeccionando un libro homenaje, con aportaciones de diferentes autores, o un número de una revista, etc.
- Si se está escribiendo un documento extenso en el que cada parte (capítulo) tiene relativa autonomía, de tal modo que la ordenación final de los elementos admita varias posibilidades y deba decidirse al final. Esto ocurre con relativa frecuencia en ciertos textos académicos (manuales, introducciones y similares) en los que el que un capítulo deba ir antes o después de otros admite cierto grado de variabilidad.
- Si se están escribiendo varios documentos relacionados entre sí que compartan características de estilo.
- Si, simplemente, el documento en el que estamos trabajando es muy extenso, de tal modo que el ordenador trabaja lento, bien al editar el documento, bien al compilarlo; caso este en el que un fraccionamiento en varios ficheros fuente acelerará considerablemente las compilaciones parciales.
- También, si hemos escrito varias macros que queremos aplicar en varios (o en todos) nuestros documentos, o si hemos generado una plantilla que controle el estilo de los documentos y queremos aplicarla a varios de ellos, etc.

4.4 Estructura del fichero fuente en los proyectos simples

En los proyectos simples, que se desarrollan en un solo fichero, la estructura de éste es muy sencilla y gira alrededor del entorno «`text`», que obligatoriamente debe aparecer en el fichero. A partir de este entorno se diferencia entre:

- **El preámbulo del documento:** todo lo que haya desde la primera línea del fichero hasta la línea de inicio del entorno «`text`» (`\starttext`).
- **El cuerpo del documento:** está constituido por el contenido del entorno «`text`»; es decir: todo lo que haya entre `\starttext` y `\stoptext`.

```
% Primera línea del documento

% Zona de Preámbulo:
% Contienen los comandos de configuración
% global del documento

\starttext % Aquí empieza el cuerpo del documento

...
... % Contenido del documento
...

\stoptext % Fin del documento
```

Figura 4.1 Fichero que contiene un proyecto simple

En la [figura 4.1](#) se muestra un fichero fuente muy simple. Absolutamente todo lo que haya antes del comando `\starttext` (que en la imagen se encuentra en la línea 5ª, contando sólo las que tienen algún texto), constituye el preámbulo; todo lo que haya entre `\starttext` y `\stoptext` constituye el cuerpo del documento. Lo que pueda haber después de `\stoptext` será ignorado.

El preámbulo se utiliza para incluir en él los comandos que deban afectar a la totalidad del documento, que serán los que determinen su configuración general. No es imprescindible que en el preámbulo se llegue a escribir algún comando. Si no se hace, ConTeXt asumirá su configuración por defecto, la cual no es muy elaborada, pero puede servir para muchos documentos. En un documento bien planificado, en el preámbulo se encontrarán todos los comandos que hayan de afectar al documento en su totalidad, así como las macros y comandos personalizados que se usarán en el fichero fuente. Esto incluiría, en un preámbulo típico, los siguientes aspectos:

- La indicación del idioma principal del documento (Véase [sección 10.5](#)).
- La indicación del tamaño del papel ([sección 5.1](#)) y del diseño de las páginas ([sección 5.3](#)).
- Las características de la fuente principal del documento ([sección 6.3](#)).
- La personalización de los comandos de seccionado que se usarán ([sección 7.4](#)) y, en su caso, la definición de comandos de seccionado nuevos ([sección 7.5](#)).
- El diseño de los encabezados y pies de página ([sección 5.6](#)).
- El establecimiento de nuestras propias macros ([sección 3.7](#)).
- Etc.

El preámbulo está pensado para la configuración global del documento; por lo tanto en él no debe incluirse nada que sea *contenido* del documento o texto procesable. En teoría el texto procesable que se incluya en el preámbulo será ignorado, aunque en alguna ocasión su existencia puede provocar un error de compilación.

El cuerpo del documento, enmarcado entre los comandos `\starttext` y `\stoptext` incluye el contenido propiamente dicho, es decir: el texto procesable, así como los comandos de ConT_EXt que no deban afectar a todo el documento.

4.5 Gestión multifichero al estilo de T_EX

Para trabajar con más de un fichero fuente, T_EX incluyó la primitiva `\input`, que también funciona en ConT_EXt, aunque en éste se incorporan dos comandos específicos que, en cierto modo, perfeccionan el funcionamiento de aquella.

4.5.1 El comando `\input`

El comando `\input` inserta el contenido del fichero que se le indique. Su formato es:

`\input NombreFichero`

donde *NombreFichero* es el nombre del fichero a insertar. Obsérvese que no es preciso que el nombre del fichero se encierre entre llaves, aunque si así se hace no se producirá ningún error. No debe, sin embargo, introducirse entre corchetes. La extensión del fichero puede omitirse si es «`.tex`».

Cuando ConT_EXt está compilando un documento y encuentra un comando `\input`, busca el fichero indicado y continúa la compilación como si ese fichero formara parte del que lo ha llamado. Cuando termina de compilarlo, vuelve al fichero original y continúa con su compilación en el punto en el que la dejó; el resultado práctico es, por lo tanto, el de que el contenido del fichero llamado mediante `\input` se inserta en el punto en el que es llamado. El fichero llamado con `\input` ha de tener un nombre válido en nuestro sistema operativo y sin espacios en blanco dentro del nombre. ConT_EXt lo buscará en el directorio de trabajo y, si no lo encuentra allí, lo buscará en los directorios incluidos en la variable de entorno TEXROOT. Si finalmente el fichero no es localizado, se producirá un error de compilación.

El uso más corriente del comando `\input` es el siguiente: se escribe un fichero, llamémosle «`principal.tex`», que se usará como contenedor para, desde él, ir llamando mediante `\input` a los distintos ficheros que componen nuestro proyecto. Esto se muestra en el siguiente ejemplo:

```
% Comandos de configuración general:
```

```
\input MiConfiguracion
```

```
\starttext
```

```
\input PagTitulo
```

```
\input Prefacio
```

```
\input Cap1
```

```
\input Cap2
```

```
\input Cap3
```

```
...
```

```
\stoptext
```

Obsérvese como para la configuración general del documento hemos llamado al fichero «MiConfiguracion.tex» que se supone que contiene los comandos globales que deseamos aplicar. Luego, entre los comandos `\starttext` y `\stoptext` vamos llamando a los distintos ficheros que contienen las diferentes partes de nuestro documento. Si en un momento dado, para acelerar la compilación, queremos omitir la compilación de alguna de las partes, basta con poner una marca de comentario al principio de la línea que contiene la llamada a dicho fichero. Por ejemplo, si estamos escribiendo el capítulo tercero y queremos compilar simplemente para asegurarnos de que en él no hay errores, no necesitaríamos compilar el resto, por lo que podríamos escribir:

```
% Comandos de configuración general:
```

```
\input MiConfiguracion
```

```
\starttext
```

```
% \input PagTitulo
```

```
% \input Prefacio
```

```
% \input Cap1
```

```
% \input Cap2
```

```
\input Cap3
```

```
...
```

```
\stoptext
```

con lo que sólo se compilaría el capítulo 3. Obsérvese como, por otra parte, cambiar el orden de los capítulos es tan sencillo como cambiar el orden de las líneas que los llaman.

Cuando se excluye de la compilación algún fichero de un proyecto multifichero, se gana en velocidad de compilación, pero a cambio, todas las referencias que la parte que se compila hace a partes del documento que no se compilan, dejarán de funcionar. Véase la [sección 9.2](#).

Es importante tener claro que cuando trabajamos con `\input` solamente el fichero principal, desde el que se llama a los demás, debe incluir los comandos `\start-text` y `\stoptext`, pues si los demás ficheros también los incluyen, se produciría un error. Ello, por otra parte, significa que no podemos compilar directamente los distintos ficheros que componen el proyecto, sino que hay que compilarlos necesariamente desde el fichero principal que es el que alberga la estructura básica del documento.

4.5.2 `\ReadFile` y `\readfile`

Como acabamos de ver, si ConTeXt no consigue localizar al fichero llamado mediante `\input`, se generará un error. Para el caso de que queramos insertar un determinado fichero sólo en el caso de que éste realmente exista, pero admitiendo la posibilidad de que no exista, ConTeXt ofrece una variación del comando `\input`. Se trata de

`\ReadFile{NombreFichero}`

Este comando es en todo similar a `\input` en cuanto a sus efectos, con la única salvedad de que si no se encuentra el fichero a insertar, continuará la compilación sin generarse ningún tipo de error. Hay también una diferencia con `\input` respecto a la sintaxis, pues sabemos que con `\input` no es preciso introducir entre llaves el nombre del fichero a insertar. Pero con `\ReadFile` eso es imprescindible. Si no se utilizan llaves ConTeXt considerará que el nombre del fichero a buscar es igual al del primer carácter tras el comando `\ReadFile`, seguido de la extensión `.tex`. Así, por ejemplo, si escribimos

`\ReadFile MiFichero`

ConTeXt entenderá que el fichero que hay que leer se llama «`M.tex`», pues el carácter inmediatamente posterior al comando `\ReadFile` (excluidos los espacios en blanco que, como se sabe, son ignorados al final del nombre de un comando) es una «`M`». Como ConTeXt, normalmente, no encontrará un fichero llamado «`M.tex`», pero `\ReadFile` no genera ningún error si el fichero no se encuentra, ConTeXt continuará la compilación justo después de la «`M`» de «`MiFichero`», e insertará el texto «`ifichero`».

Una versión más depurada de `\ReadFile` es `\readfile` el formato de este último comando es

`\readfile{NombreFichero}{TextoSiExiste}{TextoSiNoExiste}`

El primer argumento es similar al de `\ReadFile` el nombre de un fichero, encerrado entre llaves. El segundo argumento incluye el texto que hay que escribir, si el fichero existe, antes de insertar el contenido del fichero. El tercer argumento incluye el texto que habría que insertar si no se consigue localizar el fichero en cuestión.

Es decir: dependiendo de que se localice o no el fichero introducido como primer argumento, se ejecutará el segundo argumento (si el fichero existe) o el tercero (si el fichero no existe).

4.6 Proyectos de ConT_EXt propiamente dichos

El tercer mecanismo que ofrece ConT_EXt para proyectos multifichero, es el más completo y complejo: Parte de distinguir entre ficheros de proyecto, ficheros de producto, ficheros de componente y ficheros de entorno. Para entender las relaciones y funcionamiento de cada uno de estos tipos de ficheros creo que lo mejor es explicarlos por partes:

4.6.1 Ficheros de entorno (*environment*)

Un fichero de entorno es aquel que almacena las macros y configuraciones de un concreto estilo que se pretende aplicar a varios documentos, bien sean documentos totalmente independientes, bien sean partes de un documento complejo. El fichero de entorno, por lo tanto, puede incluir todo lo que normalmente escribiríamos antes del comando `\starttext`; es decir: la configuración general del documento.

He mantenido la denominación de «ficheros de entorno» para este tipo de ficheros, por no alejarme de la terminología oficial de ConT_EXt; aunque pienso que probablemente sería más afortunada la denominación «ficheros de formato» o «ficheros de configuración global».

Como todos los ficheros fuente de ConT_EXt, los ficheros de entorno son ficheros de texto, y se asume que su extensión será «`.tex`»; aunque si queremos podemos cambiarla, tal vez por «`.env`». No obstante en ConT_EXt eso no suele hacerse. Lo más normal es que el fichero de entorno se identifique por empezar o terminar el nombre con el texto “env”. Por ejemplo: «`MisMacros_env.tex`» o «`env_MisMacros.tex`». Visto por dentro un fichero de entorno tendría el siguiente aspecto:

```
\startenvironment MiEntorno

\mainlanguage[es]

\setupbodyfont
  [modern]

\setupwhitespace
  [big]

...

\stopenvironment
```

Es decir: las definiciones y comandos de configuración se encierran entre `\startenvironment` y `\stopenvironment`. Inmediatamente detrás de `\startenvironment` se escribe el nombre por el que se identificará al entorno en cuestión, y a continuación se incluyen todos los comandos que deseamos que compongan el entorno.

Respecto del nombre del entorno, de acuerdo con mis pruebas, el nombre que se introduce inmediatamente después de `\startenvironment` es meramente indicativo y de hecho si no se introduce ningún nombre, no pasa nada.

Los ficheros de entorno están pensados para trabajar junto con componentes y productos (que en seguida se explicarán). Por ello desde un componente o un producto se puede llamar a uno o varios entornos mediante el comando `\environment`. Pero este comando también funciona si es utilizado en la zona de configuración de cualquier fichero fuente de ConT_EXt; aunque no sea un fichero fuente pensado para compilar por partes.

El comando `\environment` admite los siguientes dos formatos de llamada:

`\environment Fichero`

`\environment [Fichero]`

En cualquiera de ambos casos, el efecto de este comando será el de cargar el contenido del fichero recibido como argumento. Si ese fichero no se encuentra, seguirá la compilación de modo normal sin generar ningún error. Si la extensión del fichero es «`.tex`», puede omitirse.

4.6.2 Componentes y productos

Si pensamos en un libro del que cada capítulo se contiene en un fichero diferente, diríamos que los capítulos son los *componentes* y el libro el *producto*. Es decir: un *componente* es una parte autónoma de un *producto*, susceptible de tener su propio estilo y de ser compilado de modo independiente. Para cada componente habrá un fichero distinto, y, además habrá un fichero de producto que permita unificar a todos los componentes.

Un fichero de componente típico sería como el siguiente

```
\environment MiEntorno
\environment MisMacros

\startcomponent Capitulo1

  \startchapter
    [title={Capítulo 1º}]

  ...

  \stopchapter

\stopcomponent
```

Y un fichero de producto tendría el siguiente aspecto:

```
\environment MiEntorno
\environment MisMacros

\startproduct MiLibro

  \component Capitulo1
  \component Capitulo2
  \component Capitulo3

  ...

\stopproduct
```

Obsérvese que el verdadero contenido de nuestro documento estará repartido entre los distintos ficheros de “componente” y el fichero de producto se limita a establecer el orden de los componentes. Por otra parte, tanto los componentes (individualmente) como los productos, son directamente compilables. La compilación de un producto generará un fichero PDF que contenga todos los componentes del mismo. Si, por el contrario, se compila individualmente alguno de los componentes, se generará un fichero pdf que contenga exclusivamente el componente compilado.

Dentro de un fichero de componente, y antes de la orden `\startcomponent`, podemos llamar a uno o varios entornos mediante la orden `\environment NombreEntorno`. Lo mismo podemos hacer en un fichero de producto, antes de `\startproduct`. Se pueden cargar simultáneamente varios ficheros de entorno. Podemos, tal vez, tener en ficheros diferentes nuestra colección favorita de macros personalizadas y los diferentes estilos que aplicamos a nuestros documentos. Téngase en cuenta, no obstante, que cuando se usan dos o más entornos, éstos se cargan en el orden en el que son llamados, de tal manera que si se ha incluido el mismo comando de configuración en más de un entorno, y lo ha sido con valores distintos, se terminarán aplicando los valores del último entorno cargado. Los ficheros de entorno, por otra parte, se cargan una sola vez, por lo que, en los ejemplos anteriores, en los que se llama al entorno desde el fichero del producto y desde los concretos ficheros de los componentes, si compilamos el producto, será en ese momento en el que se carguen los entornos, y lo harán en el orden allí indicado; cuando desde alguno de los componentes se llame a algún entorno ConT_EXt comprobará si ese entorno está ya cargado, en cuyo caso no hará nada.

El nombre del componente al que se llama desde un producto, ha de ser el nombre del fichero que contiene el componente de que se trate, aunque, si la extensión de dicho fichero es «.tex», puede omitirse.

4.6.3 Proyectos propiamente dichos

Con la distinción entre productos y componentes, será suficiente en la mayor parte de los casos. Pero todavía ConT_EXt incluye un elemento superior, pensado para agrupar a varios productos: el «proyecto» (*project*, en inglés).

Un fichero de proyecto típico sería más o menos así

```
\startproject MiColeccion

\environment MiEntorno
\environment MisMacros

\product Libro01
\product Libro02
\product Libro03

...

\stopproject
```

Un escenario en el que necesitaríamos un proyecto sería, por ejemplo, aquel en el que queremos editar una colección de libros, todos ellos con las mismas especificaciones de formato; o si somos editores de una revista: La colección de libros, o la revista como tal, sería el proyecto; cada libro, o cada número de la revista, sería un producto; y cada capítulo de libro, o cada artículo en algún número de la revista, sería un componente.

Los proyectos, por otra parte, no están pensados para ser compilados directamente. Piénsese que, por definición, cada producto del proyecto (cada libro de la colección, o cada número de la revista) debería compilarse aparte y generar su propio PDF. Por lo tanto el comando `\product` incluido dentro de él para indicar qué productos pertenecen al proyecto, en realidad no hace nada: sirve simplemente como recordatorio para el autor.

Alguien, claro, podría preguntar para qué sirven los proyectos, si no se pueden compilar: la respuesta está en que el fichero de proyecto vincula a determinados entornos con el proyecto. Por ello, si dentro de un producto o componente incluimos la orden `\project NombreProyecto` ConT_EXt leerá el fichero del proyecto y cargará automáticamente los entornos vinculados con él. Esta es la razón de que en los proyectos la orden `\environment` se debe incluir después de `\startproject`; pero, sin embargo, en productos y componentes, `\environment` debe ubicarse *antes* de `\startproduct` o `\startcomponent`.

Al igual que ocurre con los comandos `\environment` y `\component`, el comando `\project` admite que el nombre del proyecto se indique entre corchetes, o sin los

corchetes. Es decir: `\project NombreFichero` y `\Project[NombreFichero]` son comandos equivalentes.

Resumen sobre las distintas formas de cargar un entorno

De lo hasta ahora dicho se desprende que un entorno se puede cargar por cualquiera de los siguientes procedimientos:

- a. Insertando la orden `\environment NombreEntorno` antes de `\starttext` o `\startcomponent`. Esto cargará el entorno para la compilación exclusivamente del fichero en cuestión.
- b. Insertando la orden `\environment NombreEntorno` en un fichero de producto, antes de `\startproduct`. Esto cargará el entorno cuando sea compilado el producto, pero no si se compilan individualmente sus componentes.
- c. Insertando la orden `\project` dentro de un producto o entorno: esto cargará todos los entornos vinculados a dicho proyecto (en el fichero del proyecto).

4.6.4 Aspectos comunes a entornos, componentes, productos y proyectos

Nombre de entornos, componentes, productos y proyectos: Ya hemos visto que en todos estos elementos, tras el comando `\start` que inicia un concreto entorno, componente, producto o proyecto, se introduce directamente el nombre del mismo. Este nombre, como regla, ha de coincidir con el nombre del fichero que contiene el entorno, componente o producto, pues cuando context, por ejemplo, está compilando un producto y, de acuerdo con el fichero del producto debe cargar un entorno o componente, no tiene forma de saber en qué fichero está ese entorno o componente, salvo que el fichero se llame igual que el elemento a cargar.

Por lo demás, según mis pruebas, en los ficheros de producto y ficheros de entorno el nombre del producto o entorno que se escribe a continuación de `\startproduct` o `\startenvironment` es indicativo. Si se omite, o no coincide con el nombre del fichero, no pasa nada. Por el contrario, en el caso de los componentes, sí es importante que el nombre del mismo coincida con el del fichero que lo contiene.

Estructura de directorios del proyecto: Sabemos que, por defecto, ConTeXt busca ficheros en el directorio de trabajo y dentro de la ruta indicada por la variable TEXROOT. Sin embargo, cuando se usan las órdenes `\project`, `\product`, `\component` o `\environment` se asume que el proyecto tiene una estructura de directorios en la que los elementos comunes se encuentran en el directorio padre, y los específicos en algún directorio hijo, por lo que si no

se encuentra el fichero indicado en el directorio de trabajo, se buscará en su directorio padre, y si allí tampoco se encuentra, en el padre de éste, y así sucesivamente.

II

Aspectos globales del documento

Capítulo 5

Páginas y paginación del documento

Sumario: 5.1 El tamaño de las páginas; 5.1.1 Establecimiento del tamaño de página; 5.1.2 Uso de tamaños de página no estandarizados; 5.1.3 Cambiar el tamaño de página en algún punto del documento; 5.1.4 Ajustar el tamaño de página a su contenido; 5.2 Elementos en la página; 5.3 El diseño de la página (`\setuplayout`); 5.3.1 Asignación de tamaño a los distintos componentes de la página; 5.3.2 Adaptación del diseño de página; 5.3.3 Utilización de múltiples diseños de página; 5.3.4 Otras cuestiones relacionadas con el diseño de la página; A Distinción entre páginas pares e impares; B Páginas con más de una columna; 5.4 Numeración de las páginas; 5.5 Saltos de página forzados o sugeridos; 5.5.1 El comando `\page`; 5.5.2 Unir ciertas líneas o párrafos para impedir que entre ellas se inserte un salto de página; 5.6 Encabezados y pies de página; 5.6.1 Comandos para fijar el contenido de los encabezados y pies; 5.6.2 Formato del encabezado y pie; 5.6.3 Definir encabezados o pies específicos y vincularlos a comandos de seccionado; 5.7 Inserción de elementos textuales en los bordes y márgenes de la página;

ConT_EXt transforma el documento fuente en *páginas* correctamente formateadas. El cómo sean esas páginas, cómo se distribuya en ellos el texto y los espacios en blanco y de qué elementos dispongan son aspectos fundamentales de cara a obtener una buena composición tipográfica. A todas esas cuestiones, y a algunas otras relacionadas con la paginación, se dedica el presente capítulo.

5.1 El tamaño de las páginas

5.1.1 Establecimiento del tamaño de página

Por defecto ConT_EXt asume que las páginas del documento serán del tamaño A4, que es el estándar en Europa. Podemos establecer un tamaño distinto mediante `\setuppapersize` que es el típico comando que se ubica en el preámbulo del documento. La sintaxis *normal* de este comando es:

```
\setuppapersize [PagLógica] [PagFísica]
```

donde ambos argumentos reciben nombres simbólicos¹. el primer argumento, al que he llamado *PagLógica* representa el tamaño de página que hay que tomar en consideración para la composición tipográfica; y el segundo argumento *PagFísica* representa el tamaño de página en que se imprimirá. Normalmente ambos tamaños son el mismo, y el segundo argumento se puede omitir; pero en ocasiones puede haber una diferencia entre ambos tamaños, como, por ejemplo, cuando se imprime un libro en pliegos de 8 o 16 páginas (técnica de impresión relativamente habitual, sobre todo en libros académicos, hasta aproximadamente los años sesenta del pasado siglo). En estos casos ConT_EXt permite diferenciar entre ambos tamaños; y si la idea es que en un solo pliego de papel se impriman varias páginas, también se puede indicar el esquema de plegado que se seguirá mediante el comando `\setuparranging` (que no se explicaré en esta introducción).

Como tamaño para la composición podemos indicar: Cualquier tamaño predefinido por la industria papelera (o por nosotros mismos). Esto incluye

- Cualquiera de los tamaños de las series A, B y C predefinidos por el estándar ISO-216 (de A0 a A10, de B0 a B10 y de C0 a C10), usados en general en Europa.
- Cualquiera de los tamaños estándar en Estados Unidos: letter, ledger, tabloid, legal, folio, executive.
- Cualquiera de los tamaños RA y RSA definidos por el estándares ISO-217.
- Los tamaños G5 y E5 definidos por el estándar sueco SIS-014711 (para tesis doctorales).
- Para sobres: Cualquiera de los tamaños definidos por el estándar norteamericano (envelope 9 a envelope 14) o por el estándar ISO-269 (C6/C5, DL, E4).
- CD, para portadas de CDs.
- S3 – S6, S8, SM, SW para tamaños de pantalla en documentos pensados para no ser impresos sino mostrados en pantalla.

Junto con el tamaño del papel, con `\setuppapersize`, podemos indicar la orientación: «portrait» (vertical) o «landscape» (apaisado).

Otras opciones que, según la wiki de ConT_EXt admite `\setuppapersize` son «rotated», «90», «180», «270», «mirrored» y «negative». En mis pruebas sólo he conseguido observar algún cambio perceptible con el valor «rotated» que invierte la página; aunque tampoco es exactamente una inversión. Los valores numéricos se supone que deberían producir rotaciones

¹ Recuérdese que en la [sección 3.5](#) señalé que las opciones recibidas por los comandos de ConT_EXt son básicamente de dos tipos: nombres simbólicos, cuyo significado es ya conocido por ConT_EXt, o variables a las que hay que asignar explícitamente algún valor.



equivalentes, por sí solos, o en combinación con «rotated», pero no he conseguido hacerlos funcionar. Tampoco he descubierto exactamente qué hacen las opciones «mirrored» y «negative».

El segundo argumento de `\setuppapersize`, que ya he dicho que se puede omitir cuando el tamaño de impresión no sea distinto del de composición, puede recibir los mismos valores que el primero, indicativos del tamaño y orientación del papel. Puede también recibir el valor «oversized» que —dice la wiki de ConTeXt— añade un centímetro y medio en cada esquina del papel.

De acuerdo con la wiki hay otros valores posibles para el segundo argumento: «undersized», «doublesized» y «doubleoversized». Pero en mis pruebas no he conseguido ver ningún cambio por el hecho de introducir cualquiera de ellos; tampoco la definición oficial del comando (véase [sección 3.6](#)) menciona estas opciones.

5.1.2 Uso de tamaños de página no estandarizados

Si queremos usar un tamaño de página no estandarizado, podemos hacer dos cosas:

1. Usar una sintaxis alternativa de `\setuppapersize` que permite introducir la altura y anchura del papel como dimensiones.
2. Definir nuestro propio tamaño de página, asignarle un nombre y usarlo como si fuera un tamaño estándar.

Sintaxis alternativa de `\setuppapersize`

Además de la sintaxis ya vista, `\setuppapersize` admite esta otra sintaxis:

`\setuppapersize`[Nombre][Opciones]

donde *Nombre* es un argumento opcional que representa el nombre asignado a algún tamaño de papel mediante `\definepapersize` (que examinaremos a continuación), y *Opciones* son opciones del tipo de asignación explícito de valor. Entre las opciones admisibles se pueden destacar las siguientes:

- **width**, **height** que representan, respectivamente, la anchura y la altura de la página.
- **page**, **paper**. La primera se refiere al tamaño de página para la composición tipográfica, y la segunda al tamaño de página para la impresión física. Es decir «page» equivale al primer argumento de `\setuppapersize` en su sintaxis normal (explicada antes) y «paper» al segundo argumento. Estas opciones pueden recibir los mismos valores que antes se indicaron (A4, S3, etc.).

- **scale**, indica un factor de escalado de la página.
- **topspace**, **backspace**, **offset** distancias adicionales.

Definición de un tamaño de página personalizado

Para definir un tamaño de página personalizado se usa el comando `\definepapersize` cuya sintaxis es

`\definepapersize[Nombre] [Opciones]`

donde *Nombre* se refiere al nombre que se asignará al nuevo tamaño y *Opciones* pueden ser

- Cualquiera de los valores admisibles para `\setuppapersize` en su sintaxis normal (A4, A3, B5, CD, etc).
- Las medidas `width` (anchura del papel), `height` (altura del papel) y `offset` (desplazamiento), o un valor de escalado («**scale**»).

Lo que no es posible es mezclar valores admisibles para `\setuppapersize` con medidas o factor de escalado. Esto es así porque en el primer caso las opciones consisten en palabras simbólicas y en el segundo en variables a las que hay que asignar un valor explícito; y en ConT_EXt como ya se dijo, no es posible mezclar ambos tipos de opciones.

Cuando usamos `\definepapersize` para indicar un tamaño de papel que coincide con alguna de las medidas estándar, en realidad, más que definir un nuevo tamaño de papel, lo que estamos es definiendo un nuevo nombre para un tamaño de papel ya existente. Esto puede ser útil si queremos combinar un tamaño de papel con una orientación y así, por ejemplo, podríamos escribir

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

5.1.3 Cambiar el tamaño de página en algún punto del documento

En la mayor parte de los casos, los documentos sólo tienen un tamaño de página y por ello `\setuppapersize` es el típico comando que se incluye en el preámbulo y que se usa una sola vez en cada documento. No obstante en algunas ocasiones puede ser necesario cambiar el tamaño en algún punto del documento; como, por ejemplo, si a partir de cierto punto se incluye un anexo en el que las hojas son apaisadas.

En tales casos se puede usar `\setuppapersize` en el punto exacto en el que deseamos que cambie el tamaño. Pero como el tamaño cambiará inmediatamente, para

evitar resultados inesperados lo normal es insertar antes de `\setuppapersize` un salto de página forzado.

Si sólo necesitamos cambiar el tamaño de página para una página concreta, en lugar de usar dos veces `\setuppapersize`, una para cambiar al nuevo tamaño, y otra para restaurar el valor original, podemos usar `\adaptpapersize` que cambia el tamaño de página y, tras exactamente una página, automáticamente establece el valor previo a su llamada. Al igual que con `\setuppapersize`, antes de usar `\adaptpapersize` conviene insertar un salto de página forzado.

5.1.4 Ajustar el tamaño de página a su contenido

Hay tres entornos en ConTeXt que generan páginas del tamaño exacto para almacenar su contenido. se trata de `\startMPpage`, `\startpagefigure` y `\startTEXpage`. El primero crea una página que contiene un gráfico generado con MetaPost, lenguaje para el diseño de gráficos que se integra en ConTeXt, pero del que no me ocuparé en esta introducción. El segundo hace lo mismo con una imagen y, tal vez, algo de texto bajo ella. Recibe dos argumentos el primero identifica el fichero que contiene la imagen, me ocuparé de él en el capítulo dedicado a las imágenes externas. El tercero (`\startTEXpage`) encierra en una página el texto que constituya su contenido. Su sintaxis es:

```
\startTEXpage[Opciones] ... \stopTEXpage
```

donde las opciones pueden ser cualquiera de las siguientes:

- **strut**. No estoy seguro de la utilidad de esta opción. En la terminología de ConTeXt un *strut* (que en inglés significa pilar o basamento), es un carácter sin anchura, pero con la máxima altura y profundidad, pero no termino de ver qué tiene eso que ver con la utilidad general de este comando. Según la wiki esta opción admite los valores «yes», «no», «global» y «local», teniendo por defecto el valor «no».
- **align**. Indica la alineación del texto. Puede ser «normal», «flushleft», «flushright», «middle», «high», «low» o «lohi».
- **offset** para indicar la cantidad de espacio en blanco alrededor del texto. Puede ser «none», «overlay» si se desea un efecto de superposición, o una concreta dimensión.
- **width**, **height** donde podemos indicar una anchura y una altura para la página, o el valor «fit» para que anchura y altura sean las que necesita el texto introducido en el entorno.
- **frame** que por defecto es «off» pero puede asumir el valor de «on» si deseamos que se establezca un marco en torno al texto que es el contenido de la página.



5.2 Elementos en la página

ConT_EXt reconoce en las páginas distintos elementos cuyas dimensiones se pueden configurar mediante `\setuplayout`, que inmediatamente veremos; pero antes conviene describir cada uno de los elementos de la página, indicando el nombre por el que `\setuplayout` conoce a cada uno de ellos:

- **Los bordes:** espacio en blanco que enmarca la zona de texto. Aunque muchos programas de procesamiento de texto los llama «márgenes», es preferible usar la terminología de ConT_EXt pues, ello nos permitirá diferenciar entre el borde propiamente dicho, en el que nunca hay texto (aunque, en documentos electrónicos, puede haber botones de navegación y elementos similares), y los márgenes, en los que ocasionalmente pueden insertarse ciertos elementos textuales como, por ejemplo, las notas marginales.
 - La altura del borde superior está controlada por dos medidas: la del borde superior propiamente dicho («`top`») y la distancia entre el borde superior y el encabezado («`topdistance`»). A la suma de estas dos medidas se la llama «`topspace`».
 - El tamaño de los bordes izquierdo y derecho depende, respectivamente, de las medidas «`leftedge`» y «`rightedge`». Si queremos que ambos sean de la misma longitud, podemos configurarlos simultáneamente con la opción «`edge`».

En documentos pensados para ser impresos a doble cara, no se habla de borde izquierdo o derecho, sino de borde interior y exterior; el primero es el borde más próximo al punto en el que los folios se graparán o coserán, esto es: en las páginas impares el borde izquierdo, y en las pares el derecho. El borde exterior es el contrario al interior.

- La altura del borde inferior se denomina «`bottom`».
- **Los márgenes** propiamente dichos. ConT_EXt sólo llama márgenes a los laterales (izquierdo y derecho). Los márgenes se ubican entre el borde y la zona principal de texto y están pensados para albergar ciertos elementos textuales como, por ejemplo, las notas marginales, los títulos de las secciones o la numeración de los mismos.

Las dimensiones que controlan el tamaño de los márgenes son:

- **margin:** se usa cuando se quiere fijar simultáneamente los dos márgenes con el mismo tamaño.
- **leftmargin, rightmargin:** Fijan, respectivamente, el tamaño del margen izquierdo y el del margen derecho.

- **edgedistance**: Distancia entre el borde y el margen.
 - **leftedgedistance, rightedgedistance**: Respectivamente la distancia entre el borde y el margen izquierdo, y entre el borde y el margen derecho.
 - **margindistance**: Distancia entre el margen y la zona principal de texto.
 - **leftmargindistance, rightmargindistance**: Distancia entre, respectivamente los márgenes izquierdo y derecho y la zona principal de texto.
 - **backspace**: Esta medida representa el espacio entre la esquina izquierda del papel y el inicio de la zona principal de texto. Constituye, por lo tanto, la suma de «leftedge» + «leftedgedistance» + «leftmargin» + «leftmargindistance».
- **El encabezado y el pie de página**: El encabezado y el pie de la página son dos zonas que se ubican, respectivamente en la parte superior o inferior de la zona escrita de la página. En ellas se suele ubicar información que ayude a contextualizar el texto, tal como, por ejemplo, el número de página, el nombre del autor, el título de la obra, el título del capítulo o sección, etc. En ConTeXt estas dos zonas de la página se encuentran afectadas por las siguientes dimensiones:
 - **header**: Altura del encabezado.
 - **footer**: Altura del pie de página.
 - **headerdistance**: Distancia entre el encabezado y la zona de texto principal de la página.
 - **footerdistance**: Distancia entre el pie y la zona de texto principal de la página.
 - **topdistance, bottomdistance**: Ya se mencionaron antes. Respectivamente son la distancia entre el borde superior y el encabezado o entre el borde inferior y el pie de página.
 - **La zona de texto principal**: Es la zona más amplia de la página y, en ella, se ubica el texto del documento propiamente dicho. Su tamaño depende de las variables «width» (anchura) y «textheight» (altura). La variable «height», por su parte, mide la suma de «header», «headerdistance», «textheight», «footerdistance» y «footer».

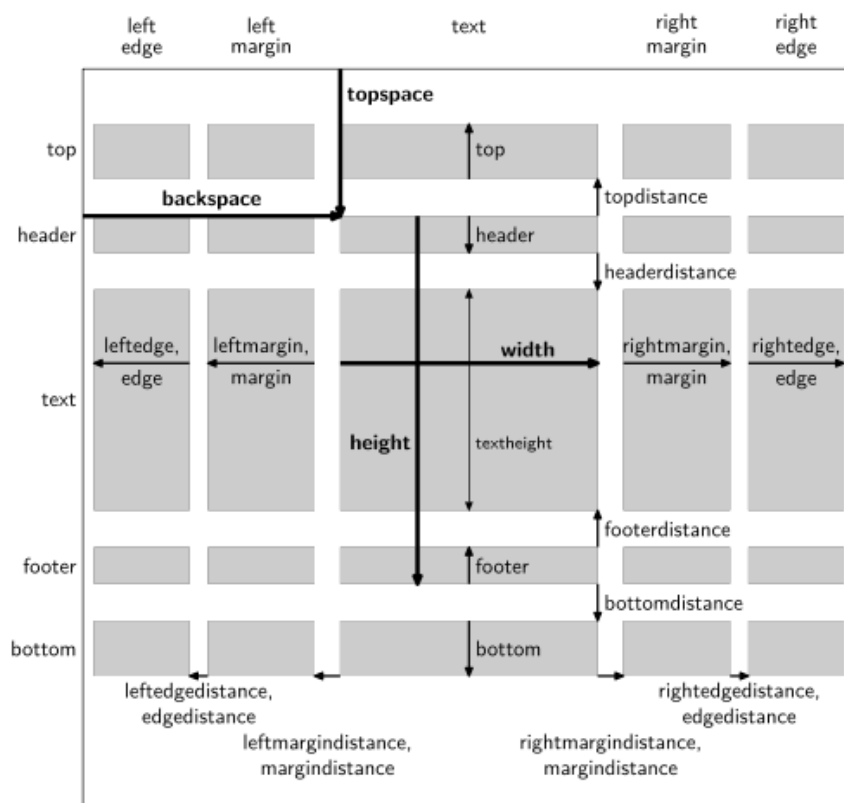


Figura 5.1 Zonas y medidas en una página

En la [imagen 5.1](#) se pueden ver todas estas zonas, con los nombres en inglés de las correspondientes medidas, y flechas indicativas de su extensión. El grosor de las flechas junto con el tamaño de los nombres de las mismas pretenden reflejar la importancia que para el diseño de la página tiene cada una de estas distancias. Si nos fijamos bien, veremos como dicha imagen muestra que una página se puede representar como una tabla con 9 filas y 9 columnas, o, si no tomamos en consideración los valores de separación entre las distintas zonas, habría cinco filas y cinco columnas de las que sólo puede haber texto en tres filas y en tres columnas. El cruce de la fila central con la columna central constituye la zona de texto principal y, normalmente, ocupará la mayor parte de la página.

En la fase de diseño de nuestro documento, podemos ver todas las medidas de una página mediante el comando `\showsetups`. Para ver visualmente indicadas las líneas principales de distribución de texto en la página podemos usar `\showframe`; y con `\showlayout` obtendremos una combinación de los dos anteriores comandos.

5.3 El diseño de la página (`\setuplayout`)

5.3.1 Asignación de tamaño a los distintos componentes de la página

El diseño de la página implica asignar tamaños concretos a las respectivas zonas de la página. Ello se hace con `\setuplayout`. Este comando nos permite alterar cualquiera de las dimensiones que se han mencionado en el apartado anterior. Su sintaxis es la siguiente:

`\setuplayout` [*Nombre*] [*Opciones*]

en donde *Nombre* es un argumento opcional que se usa sólo para el caso de que hayamos definido múltiples diseños (véase [sección 5.3.3](#)), y las *opciones* son, además de otras que en seguida se verán, cualquiera de las medidas que se acaban de mencionar. Téngase, en cuenta, no obstante que estas medidas están relacionadas entre sí pues la suma total de las componentes que afectan a la anchura o de los que afectan a la altura debería coincidir con la anchura y altura de la página. Lo que en principio significaría que al cambiar una longitud horizontal, deberíamos ajustar las restantes longitudes horizontales; y lo mismo si lo que se cambia es una longitud vertical.

Por defecto ConT_EXt sólo realiza automáticamente el ajuste de dimensiones en algunos casos, los cuales, por otra parte, no están indicados de forma sistemática y completa en su documentación. Realizando varias pruebas he podido comprobar que, por ejemplo, un aumento o disminución manual en la altura del encabezado o pie de página conlleva un ajuste de «`textheight`»; pero un cambio manual de alguno de los márgenes, no ajusta automáticamente (según mis pruebas) la anchura del texto («`width`»). Por ello, lo más eficiente para no generar una inconsistencia entre el tamaño de la página (fijado con `\setuppapersize`) y el tamaño de sus respectivos componentes, es:

- En cuanto a las medidas horizontales:
 - Ajustar «`backspace`» (que incluye «`leftedge`» y «`leftmargin`»).
 - Ajustar «`width`» (anchura del texto) no con una dimensión, sino con los valores «`fit`» o «`middle`»:
 - ★ `fit` calcula la anchura del texto a partir de la anchura del resto de los componentes horizontales de la página.
 - ★ `middle` hace lo mismo, pero antes iguala los márgenes derecho e izquierdo.
- Respecto a las medidas verticales:

- Ajustar «`topspace`».
 - Asignar a «`height`» los valores «`fit`» o «`middle`» que funcionan igual que en el caso de la anchura. El primero calcula la altura a partir del resto de los componentes, y el segundo primero iguala el margen superior y el inferior, y luego calcula la altura del texto.
 - Una vez ajustado «`height`», ajustar la altura del encabezado o del pie de página si fuera preciso, sabiendo que en tal caso «`textheight`» se reajustará automáticamente.
- Otra posibilidad para determinar de modo indirecto la altura de la zona principal de texto, es hacerlo indicando el número de líneas que cabrán en ella (teniendo en cuenta el tamaño de letra e interlineado actual). Para ello `\setuplayout` incluye la opción «`lines`».

Ubicación de la página lógica dentro de la página física

Para el caso de que el tamaño de la página lógica no coincida con el de la página física (véase la [sección 5.1.1](#)) `\setuplayout` permite configurar algunas opciones adicionales que afectan a la ubicación de la página lógica dentro de la página física:

- **location:** Esta opción determina el lugar en el que la página se colocará en el papel físico. Sus valores posibles son `left`, `middle`, `right`, `top`, `bottom`, `singlesided`, `doublesided` o `duplex`.
- **scale:** Indica un factor de escalado de la página, antes de ubicarla en el papel físico.
- **marking:** Imprimirá, en el papel físico, marcas visuales para indicar el lugar por donde se debe guillotinar la página.
- **horoffset, veroffset, clipoffset, cropoffset, trimoffset, bleedoffset, artoffset:** Medidas varias que indican diferentes desplazamientos dentro del papel físico. La mayoría están explicadas en el manual de referencia de 2013.

Estas opciones de `\setuplayout` deben ser compaginadas con las indicaciones de `\setuparranging` que indica cómo se deben ordenar las páginas lógicas en el papel físico. No explico estos últimos comandos en la introducción pues no he hecho pruebas sobre ellos.

Obtener la anchura y altura de la zona de escritura

Los comandos `\textwidth` y `\textheight` devuelven, respectivamente, la anchura y la altura de la zona de texto. Los valores que ofrecen estos comandos no se

pueden mostrar directamente en el documento final, pero sí pueden ser utilizados por otros comandos para fijar las medidas de anchura o altura de los mismos. Y así, por ejemplo, para indicar que deseamos una imagen cuya anchura sea el 60% de la anchura de la línea deberíamos indicar, como valor de la opción «width» de la imagen: «width=0.6\textwidth».

5.3.2 Adaptación del diseño de página

Puede ocurrir que nuestro diseño de página en alguna página concreta produzca un resultado indeseado; como, por ejemplo, una página final de un capítulo con sólo una o dos líneas, lo que estética y tipográficamente no es muy recomendable. Para solventar estos casos ConTeXt proporciona el comando `\adaptlayout` que permite alterar, en una o varias páginas el tamaño de la zona de texto. Este comando está pensado para ser usado solamente cuando ya hemos terminado de escribir nuestro documento y estamos realizando los pequeños ajustes finales. Por ello su ubicación natural es el preámbulo del documento. La sintaxis del comando es:

`\adaptlayout[Páginas] [Opciones]`

donde *Páginas* se refiere al número de la página o páginas cuyo diseño queremos alterar. Es un argumento opcional que se debe usar sólo cuando `\adaptlayout` se ubique en el preámbulo. Podemos indicar una sola página, o varias páginas, separando su número por comas. Si se omite este primer argumento, `\adaptlayout` afectará exclusivamente a la página en la que se encuentre el comando.

En cuanto a las opciones, pueden ser:

- **height:** Permite indicar, como una dimensión, la altura que tendrá que tener la página en cuestión. Se puede indicar una altura absoluta (ej. “19cm”) o una altura relativa (ejs., “+1cm”, “-0.7cm”).
- **lines:** Podemos incluir el número de líneas a añadir o a restar. Para añadir líneas se precede el valor del signo +, y para restarlas se precede del signo –.

Piénsese que cuando cambiamos la cantidad de líneas que admitirá una página, con ello podemos afectar al paginado del resto del documento. Por ello es por lo que la recomendación es la de usar `\adaptlayout` sólo al final, cuando el documento no vaya ya a sufrir cambios, y hacerlo en el preámbulo. Allí iremos a la primera página que queremos adaptar, la adaptaremos y comprobaremos como afecta eso a las páginas posteriores; si afecta de tal modo que alguna requiere también ser adaptada, se añade su número y se vuelve a compilar; y así sucesivamente.

5.3.3 Utilización de múltiples diseños de página

Si necesitamos usar distintos diseños en diferentes partes del documento, lo mejor es, empezar definiendo el diseño *general* y luego ir definiendo los distintos diseños

alternativos, en los que sólo cambiaremos la dimensión que haya de ser diferente. Estos diseños alternativos heredarán todas las características del diseño global que no se alteren en su definición. Para especificar un diseño alternativo y asignarle un nombre con el que posteriormente podamos invocarlo, se usa el comando `\definelayou` cuya sintaxis general es:

```
\definelayou [Nombre/Número] [Configuración]
```

donde *Nombre/Número* es el nombre que se asociará al nuevo diseño, o el número de página en el que se activará automáticamente el nuevo diseño, y *Configuración* contendrá los aspectos del diseño que se quieren cambiar respecto del diseño global.

Cuando el nuevo diseño se asocia a un nombre, para invocarlo en un punto concreto del documento, se usará:

```
\setuplayout [NombreDiseño]
```

y para regresar al diseño general:

```
\setuplayout [reset]
```

Si, por el contrario, el nuevo diseño se asoció a un número concreto de página, se activará automáticamente al llegar a dicha página. Pero, una vez activado, para regresar al diseño general habrá que indicarlo explícitamente. Aunque eso último también podemos *semiautomatizarlo*. Por ejemplo, si quisiéramos aplicar un diseño exclusivamente para las páginas 1ª y 2ª podríamos escribir, en el preámbulo del documento:

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

El efecto de estos comandos será el de que, en la página 1 se activa el diseño definido en la primera línea; y en la página 3 se activa otro diseño cuya función es sólo la de regresar al diseño general.

Con `\definelayou [even]` crearemos un diseño que se activará automáticamente en todas las páginas pares; y con `\definelayou [odd]` el diseño se activará en las impares.

5.3.4 Otras cuestiones relacionadas con el diseño de la página

A. Distinción entre páginas pares e impares

En documentos impresos a doble cara, en ocasiones se estipula que el encabezado, la numeración de páginas y los márgenes laterales difieran entre las páginas pares y

las impares. A las páginas pares se las denomina también páginas izquierda y a las impares, páginas derecha. En estos casos también es corriente que la terminología relativa a los márgenes cambie y se hable de margen interior y exterior. El primero se encuentra en el punto más próximo al lugar donde las páginas se coserán o graparán y el segundo en el extremo contrario. El margen interior se corresponde, en las páginas impares, con el margen izquierdo y en las pares con el margen derecho.

`\setuplayout` no tiene ninguna opción que expresamente nos permita decirle que deseamos diferenciar entre el diseño de las páginas pares y el de las impares. Esto es así porque para ConT_EXt la diferenciación entre ambos tipos de páginas se establece mediante una opción diferente: `\setuppagenumbering` que veremos en la [sección 5.4](#). Una vez establecida esa diferenciación, ConT_EXt asume que la página descrita con `\setuplayout` era la página impar, y construye la página par invirtiendo en ella los valores de la página impar: es decir: las especificaciones que en la página impar se aplican al lado izquierdo, en la par se aplicarán al izquierdo; y al revés: las aplicables en la página impar al lado derecho, se aplicarán, en la par, al izquierdo.

B. Páginas con más de una columna

Mediante `\setuplayout` podemos también establecer que el texto de nuestro documento se distribuya en dos o más columnas, al modo en que lo hacen, por ejemplo, los periódicos y algunas revistas. Ello es controlado por la opción «`columns`» cuyo valor ha de ser un número entero. Cuando hay más de una columna, la distancia entre columnas se indica mediante la opción «`columndistance`».

Esta opción está pensada para documentos en los que todo el texto (o su mayor parte) esté distribuido en múltiples columnas. Si lo que deseamos es, en un documento que en general está compuesto con una sola columna, que un fragmento concreto se componga con doble o triple columna, no hay que alterar el diseño de página sino, simplemente, utilizar el entorno «`columns`» (véase la [sección 12.2](#)).

5.4 Numeración de las páginas

Por defecto ConT_EXt numera las páginas con números arábigos y el número se muestra en la parte central del encabezado. Para alterar estas características ConT_EXt dispone de distintos procedimientos lo que, en mi opinión, le aporta una innecesaria complejidad en este punto.

En primer lugar, las características fundamentales de la numeración están controladas por dos comandos diferentes: `\setuppagenumbering` y `\setupuserpagenumber`.

`\setuppagenumbering` admite las siguientes opciones:

- **alternative:** Esta opción controla si el documento está concebido para que el encabezado y pie de página sean idénticos en todas las páginas («`singlesided`»), o diferenciando entre páginas pares e impares («`doublesided`»). Cuando esta opción asume este último valor, automáticamente los valores de diseño de página introducidos mediante «`setuplayout`» se ven afectados, de tal forma que se asume que lo indicado en «`setuplayout`» se refiere sólo a las páginas impares, y que, por lo tanto, lo dispuesto para el margen izquierdo en realidad se refiere al margen interior (que en las páginas pares está en el lado derecho) y que lo dispuesto para el lado derecho, en realidad se refiere al margen exterior, que en las páginas pares está en el lado izquierdo.
- **state:** Indica si se mostrará o no el número de página. Admite dos valores: `start` (se mostrará el número de página) y `stop` (se suprimen los números de página). La denominación de estos valores (`start` y `stop`) pudiera hacernos pensar que cuando «`state=stop`» dejan de numerarse las páginas y cuando «`state=start`» se reinicia la numeración. Pero ello no es así: estos valores afectan sólo a si el número de página se mostrará o no.
- **location:** Indica en qué lugar se mostrará. Normalmente en esta opción hay que indicar dos valores, separados entre sí por una coma. En primer lugar debemos especificar si deseamos el número de página en el encabezado («`header`») o en el pie de página («`footer`»), y, a continuación, en qué lugar del encabezado o pie debe imprimirse puede ser «`left`», «`middle`», «`right`», «`inleft`», «`inright`», «`margin`», «`inmargin`», «`atmargin`» o «`marginedge`». Por ejemplo: para mostrar la numeración alineada a la derecha en el pie de página deberíamos indicar «`location={footer,right}`». Véase, por otra parte, cómo hemos rodeado los dos valores de esta opción con llaves para que ConT_EXt interprete correctamente a la coma que los separa.
- **style:** Indica el estilo y tamaño de la fuente que se usará para el número de página.
- **color:** Indica el color que se aplicará al número de página.
- **left:** Recoge un comando o texto que se ejecutará o escribirá a la izquierda del número de página.
- **right:** Recoge un comando o texto que se ejecutará o escribirá a la derecha del número de página.
- **command:** Recoge un comando al que se le pasará como parámetro el número de página.

- **width:** Indica la anchura que ocupará el número de página.
- **strut:** No estoy muy seguro. En mis pruebas cuando «**strut=no**» el número se imprime exactamente en el borde superior del encabezado o en el inferior del pie de página, mientras que cuando «**strut=yes**» (valor por defecto) se aplica un espacio de separación entre el número y el borde.

`\setupuserpagenumber`, por su parte, admite estas otras opciones:

- **numberconversion:** Controla el tipo de numeración puede ser arábigo («**n**», «**numbers**»), letras en minúsculas («**a**», «**characters**»), en mayúsculas («**A**», «**Characters**»), en versalitas («**KA**»), números romanos en mayúsculas («**i**», «**r**», «**romannumerals**»), en mayúsculas («**I**», «**R**», «**Romannumerals**») o en versalitas («**KR**»).
- **number:** Indica el número a asignar a la primera página, a partir del cual se calcularán los restantes.
- **numberorder:** Asignando a esta opción el valor «**reverse**» se consigue que la numeración de páginas se haga en orden decreciente; es decir: que la última página sea el 1, la penúltima el 2, etc.
- **way:** Permite indicar el procedimiento de numeración. Puede ser: `byblock`, `bychapter`, `bysection`, `bysubsection`, etc.
- **prefix:** Permite indicar un prefijo para los números de página.
- **numberconversionset:** Se explica a continuación.

Junto a estos dos comandos, también hay que tener en cuenta el control de numeración que se puede realizar atendiendo a los grandes bloques que componen la macroestructura del documento (véase [sección 7.6](#)). Desde este punto de vista el comando `\defineconversionset` nos permitirá indicar un tipo de numeración distinta para cada uno de estos macrobloques estructurales. Por ejemplo:

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

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

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

hará que en nuestro documento el bloque inicial (`frontmatter`) se numere con números romanos en minúsculas, el bloque central (`bodymatter`) lo haga con números arábigos y los apéndices con letras mayúsculas.

Para obtener el número de página podemos usar los siguientes comandos:

- `\userpagenumber`: Devuelve el número de página, tal y como se haya configurado con `\setuppagenumbering` y con `\setupuserpagenumber`.
- `\pagenumber`: Devuelve el mismo número que el anterior comando, pero siempre en números arábigos.
- `\realpagenumber`: Devuelve, en números arábigos, el número real de página, sin tomar en consideración ninguna de estas especificaciones.

Para obtener el número de la última página del documento, disponemos también de tres comandos, que son paralelos a los anteriores. Se trata de `\lastuserpagenumber`, `\lastpagenumber` y `\lastrealpagenumber`.

5.5 Saltos de página forzados o sugeridos

5.5.1 El comando `\page`

El algoritmo de distribución del texto en páginas de ConT_EXt es bastante complejo y se basa en multitud de cálculos y variables internas que informan al programa del mejor punto posible, desde la perspectiva de la corrección tipográfica, para introducir un concreto salto de página. El comando `\page` nos permite influir en dicho algoritmo:

- a. Sugiriendo ciertos puntos como lugares óptimos, o inadecuados para incluir en ellos un salto de página.
 - **no**: Indica que el lugar en donde se encuentra el comando no es un buen candidato para insertar en él un salto de página, por lo que, en la medida de lo posible, el salto debería realizarse en otro punto del documento.
 - **preference**: Indica a ConT_EXt que el punto donde se encuentre el comando, es un *buen lugar* para intentar un salto de página, aunque no lo fuerza.
 - **bigpreference**: Indica que el punto donde se encuentre el comando es un *muy buen lugar* para intentar un salto de página, pero tampoco llega a forzarlo.

Obsérvese que estas tres opciones, no fuerzan ni impiden los saltos de página, sólo informan a ConT_EXt para que a la hora de buscar el mejor lugar para un salto de página, tenga en cuenta lo que con este comando se le indica. Pero en última instancia el punto exacto donde se producirá el salto de página seguirá siendo decidido por ConT_EXt.

- b. Forzando un salto de página en cierto punto; caso este en el que podemos, además, indicar cuántos saltos de página deben hacerse así como ciertas características de las páginas que se insertarán.
- **yes**: Fuerza, en ese punto, un salto de página.
 - **makeup**: Similar a «yes», pero el salto que se fuerza es inmediato, sin colocar primero los objetos flotantes pendientes de ubicación (véase la [sección 13.1](#)).
 - **empty**: Inserta en el documento una página totalmente en blanco.
 - **even**: Inserta el número de páginas que sea preciso para que la próxima página sea par.
 - **odd**: Inserta el número de páginas que sea preciso para que la próxima página sea impar.
 - **left, right**: Similar a las dos opciones anteriores, pero aplicable sólo a documentos impresos a doble cara, con encabezados, pies o márgenes diferentes según la página sea par o impar.
 - **quadruple**: Inserta las páginas necesarias para que la próxima página sea múltiplo de 4.

Junto a estas opciones, que propiamente controlan la paginación, `\page` incluye otras opciones que afectan a su propio funcionamiento. En particular la opción «**disable**» que provoca que ConTeXt ignore los comandos `\page` que encuentre a partir de ese punto y la opción «**reset**» que produce el efecto contrario, restaurando la efectividad de los futuros comandos `\page`.

5.5.2 Unir ciertas líneas o párrafos para impedir que entre ellas se inserte un salto de página

En ocasiones, si queremos impedir un salto de página entre varios párrafos, el uso del comando `\page` puede ser trabajoso, pues habría que escribirlo en todos los puntos en los que fuera posible que se insertara un salto de página. Un procedimiento más sencillo para ello es empaquetar el material que queremos que se mantenga en la misma página en lo que T_EX llama una *caja vertical*.

Al principio de este documento (en la [página 20](#)) señalé que internamente para T_EX todo son *cajas*. La noción de cajas es fundamental en T_EX para cualquier aplicación *avanzada*; pero su manejo es demasiado complejo como para incluirlo en esta introducción. Por ello sólo ocasionalmente se hace referencia a ellas.

Las cajas de T_EX, una vez creadas, son indivisibles, por lo que no puede insertarse un salto de página que parta en dos una caja. Por ello, si empaquetamos el material que queremos tener unido en una caja invisible, evitaremos que se inserte un salto de página que separe dicho material. El comando para hacer eso es `\vbox` cuya sintaxis es

```
\vbox{Material}
```

donde *Material* está constituido por el texto que queremos mantener unido.

Algunos entornos de ConT_EXt empaquetan su contenido en una caja. Por ejemplo «`framedtext`», por lo que si encerramos el material que queremos mantener unido en dicho entorno y, además, hacemos que el marco generado por el mismo sea invisible (lo que se consigue con la opción `frame=off`), habremos conseguido lo mismo.

5.6 Encabezados y pies de página

5.6.1 Comandos para fijar el contenido de los encabezados y pies

Si en el diseño de la página hemos asignado algún tamaño al encabezado o al pie de página, podemos incluir texto en ellos mediante los comandos `\setupheadertexts` y `\setupfootertexts`. Son dos comandos similares con la única diferencia de que el primero actúa sobre el encabezado de las páginas y el segundo sobre los pies de página. Ambos admiten entre uno y cinco argumentos.

1. Usados con un solo argumento éste contendrá el texto del encabezado o pie de página que se ubicará en el centro del mismo. Por ejemplo: `\setupfootertexts[pagenumbers]` escribirá el número de página en el centro del pie de página.
2. Usados con dos argumentos, el contenido del primer argumento se ubicará en el lado izquierdo del encabezado o pie de página, y el del segundo argumento en el lado derecho. Por ejemplo `\setupheadertexts[Prefacio][pagenumbers]` compondrá un encabezado de página en el que, en el lado izquierdo se escriba la palabra «prefacio» y en el lado derecho se imprima el número de página.
3. Si utilizamos tres argumentos, el primero indicará *la zona* en la que se han de imprimir los otros dos. Y aquí por *zona* hago referencia a las *zonas* de la página mencionadas en la [sección 5.2](#), es decir: borde (edge), margen (margin) o encabezado propiamente dicho (text). Los restantes dos argumentos contienen el texto que se ubicará, respectivamente, en el borde, margen o lado izquierdo y en el borde, margen o lado derecho.

El uso con cuatro o cinco argumentos equivale al uso con dos o tres argumentos, en los casos en los que se diferencie entre páginas pares e impares, cosa que ocurre, como ya sabemos, cuando se establece «`alternative=doublesided`» con `\setuppagenumbering`. En tal caso se añaden dos argumentos posibles para reflejar el contenido de los lados izquierdo y derecho de las páginas pares.

Una característica importante de estos dos comandos es la de que cuando se usan con dos argumentos, el encabezado o pie de página central previo (si existiera) no se reescribe, lo que nos permite, escribir un texto distinto en cada zona siempre y cuando primero escribamos el texto central (llamando al comando con un solo argumento) y después escribamos los textos laterales (llamándolo de nuevo, ahora con dos argumentos). Así, por ejemplo, si escribimos los siguientes comandos:

```
\setupheadertexts[que te vi]
\setupheadertexts[Tararí][Mariví]
```

El primer comando escribirá en el centro del encabezado la frase “que te vi” y el segundo escribirá en el lado izquierdo la frase “Tararí” y en el lado derecho la frase “Mariví”, dejando inalterada la zona central, puesto que no se le ha ordenado reescribirla, por lo que el encabezado resultante mostraría la frase

Tararí

que te vi

Mariví



La explicación que acabo de dar del funcionamiento de estos comandos es mi conclusión después de varias pruebas. La explicación que de estos comandos se da en la *excursión* por ConT_EXt se basa en la versión con cinco argumentos; y la que se hace en el manual de referencia de 2013 parte de la versión con tres argumentos. Creo que la mía es más clara. De otro lado, el que la segunda llamada al comando no sobrescribe a una llamada anterior, no lo he visto explicado, pero funciona así si primero se escribe el encabezado o pie central y luego los laterales. Por el contrario, si primero escribimos los encabezados o pies laterales, la ulterior llamada al comando para escribir el central borrará los encabezados o pies previos. ¿Por qué? No lo se. Creo que estos pequeños detalles introducen una innecesaria complicación, y deberían estar claramente explicados en la documentación oficial.

Por lo demás, como contenido propiamente dicho del encabezado o del pie de página, podemos indicar cualquier combinación de texto y comandos. Pero también los siguientes valores:

- **date**, **currentdate**: Escribirán (cualquiera de ellos) la fecha actual.
- **pagenumber**: Escribirá el número de página.
- **part**, **chapter**, **section...**: Escribirán el título correspondiente a la parte, capítulo, sección... o división estructural de que se trate.
- **partnumber**, **chapternumber**, **sectionnumber...**: Escribirán número de parte, capítulo sección ... o división estructural de que se trate.

Ojo: Estos nombres simbólicos (**date**, **currentdate**, **pagenumber**, **chapter**, **chapternumber**, etc.) sólo son interpretados como tales si el único contenido del argumento es el nombre simbólico propiamente dicho; pero si añadimos algún otro texto, o comando de formateo, estas palabras se interpretarán literalmente,

y así, por ejemplo, si escribimos `\setupheadertexts[chapternumber]` obtendremos el número del capítulo actual; pero si escribimos `\setupheadertexts[Capítulo chapternumber]` obtendremos la siguiente frase: “Capítulo chapternumber”. En estos casos, cuando el contenido del comando no es sólo la palabra simbólica, debemos:

- Para `date`, `currentdate` y `pagenumber` usar, no la palabra simbólica sino el comando del mismo nombre (`\date`, `\currentdate` o `\pagenumber`).
- Para `part`, `partnumber`, `chapter`, `chapternumber`, etc. usar el comando `\getmarking[Marca]` que devuelve el contenido de la *Marca* que se le solicite. Y así, por ejemplo, `\getmarking[chapter]` devolverá el título del capítulo actual, mientras que `\getmarking[chapternumber]` devolverá el número del capítulo actual.

Para desactivar los encabezados y pies en una página concreta, se usa el comando `\noheaderandfooterlines` que actúa exclusivamente sobre la página en la que se ubique. Si sólo queremos suprimir el número de página en una página concreta, hay que usar el comando `\page[blank]`.

5.6.2 Formato del encabezado y pie

El concreto formato en el que se muestre el texto del encabezado o pie de página lo podemos indicar en los argumentos de `\setupheadertexts` o `\setupfootertexts` mediante los correspondientes comandos de formato. Pero también podemos configurarlo globalmente mediante los comandos `\setupheader` y `\setupfooter` que admiten las siguientes opciones:

- **state**: Admite los siguientes valores: `start`, `stop`, `empty`, `high`, `none`, `normal` o `nomarking`.
- **style**, **leftstyle**, **rightstyle**: Configuración del estilo del texto del encabezado o pie. `style` afecta a todas las páginas, `leftstyle` a las páginas pares y `rightstyle` a las impares.
- **color**, **leftcolor**, **rightcolor**: Color del encabezado o pie. Puede afectar a todas las páginas (opción `color`) o sólo a las pares (`leftcolor`) o impares (`rightcolor`).
- **width**, **leftwidth**, **rightwidth**: Anchura de todos los encabezados o pies (`width`) o de los encabezados o pies pares (`leftwidth`) o impares (`rightwidth`).
- **before**: Comando a ejecutar antes de escribir el encabezado o pie.

- **after**: Comando a ejecutar después de haber escrito el encabezado o pie.
- **strut**: Si vale «yes» se establece un espacio vertical de separación entre el encabezado y el borde. Cuando vale no, el encabezado o el pie de página aparecen pegados a límite de la zona del borde superior o inferior.

5.6.3 Definir encabezados o pies específicos y vincularlos a comandos de seccionado

El sistema de encabezados y pies de página de ConT_EXt permite que automáticamente cambie el texto del encabezado o pie cuando se cambia de capítulo o sección; o cuando se cambia de página, si se han establecido encabezados o pies diferentes para las páginas pares y las impares. Pero lo que no permite es diferenciar entre la primera página (del documento, o de un capítulo o sección) y el resto de las páginas. Para lograr esto último debemos:

1. Definir un encabezado o pie de página específico.
2. Vincularlo a la sección de que se trate.

La definición de encabezados o pies de página específicos se lleva a cabo con el comando `\definetext` cuyo formato es:

```
\definetext
  [Nombre] [Tipo]
  [Contenido1] [Contenido2] [Contenido3]
  [Contenido4] [Contenido5]
```

donde *Nombre* es el nombre asignado al encabezado o pie de que se trate; *Tipo* puede ser `header` o `footer`, según estemos definiendo un encabezado o un pie de página y los restantes cinco argumentos recogen los posibles contenidos del nuevo encabezado o pie de modo similar a como ya hemos visto que funcionan `\setupheadertexts` y `\setupfootertexts`. Una vez que hemos hecho esto, debemos vincular este nuevo encabezado o pie a algún tipo concreto de sección mediante `\setuphead` usando las opciones (que no se explican en el [capítulo 7](#)) `header` y `footer`.

Así el siguiente ejemplo hará que en la primera página de cada capítulo se oculte el encabezado y como pie de página se muestre el número de página centrado:

```
\definetext[ChapterPrimPag] [footer] [pagenumber]
\setuphead
  [chapter]
  [header=high, footer=ChapterPrimPag]
```

5.7 Inserción de elementos textuales en los bordes y márgenes de la página

Los bordes superior e inferior y los márgenes derecho e izquierdo habitualmente no contienen texto de ningún tipo. Sin embargo ConT_EXt permite ubicar allí ciertos elementos textuales. En concreto se dispone para ello de los siguientes comandos:

- `\setuptoptexts`: Permite ubicar texto en el borde superior de la página (por encima de la zona dedicada al encabezado).
- `\setupbottomtexts`: Permite ubicar texto en el borde inferior de la página (por debajo de la zona dedicada al pie de página).
- `\margintext`, `\atleftmargin`, `\atrighmargin`, `\ininner`, `\ininneredge`, `\ininnermargin`, `\inleft`, `\inleftedge`, `\inleftmargin`, `\inmargin`, `\inother`, `\inouter`, `\inouteredge`, `\inoutermargin`, `\inright`, `\inrightedge`, `\inrightmargin`: Permiten ubicar texto en los márgenes y bordes laterales del documento.

Los dos primeros comandos funcionan exactamente igual que `\setupheadertexts` y `\setupfootertexts`, e incluso se puede configurar de antemano el formato de estos textos mediante `\setuptop` y `\setupbottom` de modo similar a como `\setupheader` permite configurar el texto establecido con `\setupheadertexts`. Por todo ello me remito aquí a lo ya dicho en la [sección 5.6](#). La única matización que hay que añadir es la de que el texto establecido mediante `\setuptoptexts` o `\setupbottomtexts` no será visible si en el diseño de la página no se ha reservado algún espacio para los bordes superior (`top`) o inferior (`bottom`). Véase, al respecto, la [sección 5.3.1](#).

En cuanto a los comandos dirigidos a ubicar texto en los márgenes del documento, todos tienen una sintaxis similar:

`\NombreComando [Referencia] [Configuración] {Texto}`

donde *Referencia* y *Configuración* son argumentos opcionales; el primero se usa para posibles referencias cruzadas y el segundo permite configurar el texto marginal. El último argumento, encerrado entre llaves, contiene el texto a ubicar en el margen.

De estos comandos el más general es `\margintext` pues permite ubicar texto en cualquiera de los márgenes o bordes laterales de la página. Los restantes, tal y como su nombre indica, ubican el texto bien en el margen propiamente dicho (derecho o izquierdo, interior o exterior), bien en el borde (derecho o izquierdo, interior o exterior). Estos comandos están muy relacionados con el diseño de la página

pues si, por ejemplo, usamos `\inrightedge` y en el diseño de la página no se ha reservado ningún espacio para el borde derecho, no se verá nada.

Las opciones de configuración de `\margintext` son las siguientes:

- **location:** Indica en qué margen se ubicará el texto. Puede ser `left`, `right` o, en documentos a doble cara, `outer` o `inner`. Por defecto es `left` en documentos a una sola cara y `outer` en documentos a doble cara.
- **width:** Anchura de que dispondrá el texto para imprimirse. Por defecto se usa toda la anchura del margen.
- **margin:** Indica si el texto se colocará en el margen propiamente dicho (`margin`) o en el borde (`edge`).
- **align:** Alineación del texto. Se usan aquí los mismos valores que en `\setupalign` 11.6.1.
- **line:** Permite indicar un número de líneas de desplazamiento del texto al margen. Así, `line=1` desplazará el texto una línea hacia abajo y `line=-1` una línea hacia arriba.
- **style:** Comando o comandos para indicar el estilo del texto a ubicar en el margen.
- **color:** Color del texto marginal.
- **command:** Nombre de un comando al que se pasará como argumento el texto a colocar en el margen. Este comando se ejecutará antes de escribir el texto. Por ejemplo, si queremos dibujar un marco en torno al texto, podríamos usar «`[command=\framed]{Texto}`».

Los restantes comandos admiten las mismas opciones, salvo `location` y `margin`. En particular los comandos `\atrightmargin` y `\atleftmargin` ubican el texto totalmente pegado al cuerpo de la página. Podemos establecer un espacio de separación con la opción `distance`, que no he mencionado a propósito de `\margintext` porque, en mis pruebas, con dicho comando parecía no surtir ningún efecto.



Además de las opciones mencionadas, estos comandos admiten también otras opciones (`strut`, `anchor`, `method`, `category`, `scope`, `option`, `hoffset`, `voffset`, `dy`, `bottomspace`, `threshold` y `stack`) que no he mencionado porque no están documentadas y, francamente, no estoy muy seguro de para qué sirven. Las que tienen nombres de *distancias* podemos intuirlo pero el resto ¿? En la wiki sólo se menciona la opción `stack` diciendo que se usa para emular el comando `\marginpars` de L^AT_EX, lo que no me parece particularmente claro.

El comando `\setupmargindata` permite configurar globalmente los textos de cada margen. Así, por ejemplo,

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

hará que todos los textos ubicados en el margen derecho se escriban con letra inclinada.

También podemos crear nuestro propio comando personalizado con

```
\definemargindata[Nombre][Configuración]
```

Capítulo 6

Fuentes y colores en ConT_EXt

Sumario: 6.1 Fuentes tipográficas incluidas en «ConT_EXt Standalone»; 6.2 Características de la fuente; 6.2.1 Fuentes propiamente dichas, *estilos* y variantes del estilo; 6.2.2 Tamaño de la fuente; 6.3 Fijación de la fuente principal del documento; 6.4 Cambiar la fuente o alguna de sus características; 6.4.1 Los comandos `\setupbodyfont` y `\switchtobodyfont`; 6.4.2 Cambio rápido de estilo, alternativa y tamaño; 6.4.3 Definir comandos y palabras clave para los tamaños, estilos y alternativas de las fuentes; 6.5 Otras cuestiones relacionadas con el uso de algunas alternativas; 6.5.1 Cursiva, letra inclinada y enfatización; 6.5.2 Versalitas y pseudoversalitas; 6.6 Uso y configuración de los colores; 6.6.1 Procedimientos para escribir fragmentos de texto en color; 6.6.2 Cambiar el color del fondo y del primer plano del documento; 6.6.3 Comandos para colorear fragmentos concretos de texto; 6.6.4 Colores predefinidos; 6.6.5 Ver los colores disponibles; 6.6.6 Definir nuestros propios colores;

6.1 Fuentes tipográficas incluidas en «ConT_EXt Standalone»

El sistema de fuentes de ConT_EXt ofrece muchas posibilidades, pero también es bastante complejo. No entraré en este manual en analizar todas las posibilidades avanzadas en materia de fuentes, sino que me limitaré a asumir que se trabaja con alguna de las 21 fuentes proporcionadas con la instalación de ConT_EXt Suite, que son las que se muestran en la [tabla 6.1](#).

La columna central de la [tabla 6.1](#) indica el nombre o nombres por los que ConT_EXt conoce la fuente de que se trate. Cuando hay dos nombres, ambos funcionan como sinónimos. En la última columna se pone un ejemplo de la fuente. En cuanto al orden en el que se muestran las fuentes, la primera es la fuente que ConT_EXt usa por defecto, las restantes fuentes se ordenan alfabéticamente, colocándose al final tres fuentes específicamente diseñadas para las matemáticas. Obsérvese que la fuente Euler no puede representar directamente letras acentuadas.

De cara a los lectores que procedan del mundo Windows y de sus fuentes por defecto, señalaré que *heros* es equivalente a la fuente llamada en Windows Arial,

Nombre oficial	Nombre(s) en ConT _E Xt	Ejemplo
Latin Modern	modern, modern-base	Pablito clavó un clavo
Antykwa Poltawskiego	antykwapoltawskiego	Pablito clavó un clavo
Antykwa Toruńska	antykwa	Pablito clavó un clavo
Cambria	cambria	Pablito clavó un clavo
DejaVu	dejavu	Pablito clavó un clavo
DejaVu Condensed	dejavu-condensed	Pablito clavó un clavo
Gentium	gentium	Pablito clavó un clavo
Iwona	iwona	Pablito clavó un clavo
Latin Modern Variable	modernvariable, modern-variable	Pablito clavó un clavo
PostScript	postscript	Pablito clavó un clavo
TeX Gyre Adventor	adventor, avantgarde	Pablito clavó un clavo
TeX Gyre Bonum	bonum, bookman	Pablito clavó un clavo
TeX Gyre Cursor	cursor, courier	Pablito clavó un clavo
TeX Gyre Heros	heros, helvetica	Pablito clavó un clavo
TeX Gyre Schola	schola, schoolbook	Pablito clavó un clavo
TeX Gyre Chorus	chorus, chancery	<i>Pablito clavó un clavo</i>
TeX Gyre Pagella	pagella, palatino	Pablito clavó un clavo
TeX Gyre Termes	termes, times	Pablito clavó un clavo
Euler	eulernova	Pablito clav un clavo
Stix2	stix	Pablito clavó un clavo
Xits	xits	Pablito clavó un clavo

Tabla 6.1 Fuentes incluidas en la distribución de ConT_EXt

mientras que *termes* equivale a Times New Roman. No son exactamente las mismas fuentes, aunque sí bastante parecidas, hasta el punto de que hay que ser muy observador para apreciar la diferencia.

Las fuentes usadas por Windows no son *software libre* (en realidad casi nada en Windows es *software libre*), por lo que no pueden incluirse en la distribución de ConT_EXt. No obstante si ConT_EXt se instala en un sistema Windows donde dichas fuentes ya estén instaladas, podrá hacer uso de ellas como de cualquier otra fuente instalada en el sistema en el que se ejecuta ConT_EXt. Pero en esta introducción no trataré de cómo usar fuentes ya instaladas en el sistema. Puede encontrarse ayuda sobre esa cuestión en la [wiki de ConT_EXt](#).

6.2 Características de la fuente

6.2.1 Fuentes propiamente dichas, *estilos* y *variantes del estilo*

La terminología en materia de fuentes es algo confusa, porque a veces se llama fuente a lo que en puridad es una *familia de fuentes* que incluye distintos estilos y variantes que comparten un diseño básico. No entraré aquí a discutir qué terminología es más correcta; me interesa solamente aclarar la terminología que se usa en ConT_EXt. En ella se distingue entre fuentes, estilos y variantes (o alternativas) de cada estilo. Las *fuentes* incluidas en la distribución de ConT_EXt (que en realidad

son *familias de fuentes*) las hemos visto en el apartado anterior. Veremos ahora los *estilos* y las *alternativas*.

Estilos de las fuentes

DONALD E. KNUTH diseñó para T_EX la fuente *Computer Modern*, a la que dotó de tres *estilos* distintos, llamados *roman*, *sans serif* y *teletype*. El estilo *roman* (o romano) es un diseño en el que los caracteres aparecen rematados por trazos decorativos a los que en terminología tipográfica se denomina *serifas*, por esto a este estilo de fuentes se le llama también *serif* (= serifa, en inglés). Este estilo estaba pensado para constituir la fuente *normal* o por defecto. El estilo *sans serif*, como su propio nombre indica, carece de los trazos decorativos, por lo que es una fuente más sobria y estilizada, a la que en la jerga tipográfica española se la llama, a veces, *paloseco*; esta fuente puede ser la fuente principal del documento, pero también es adecuada para utilizarla en ciertos fragmentos de un texto cuya fuente principal utilice el estilo *roman* como, por ejemplo, los títulos o los encabezados de página. Por último, el estilo *teletype* fue incluido en *Computer Roman* porque la misma había sido diseñada para escribir libros relativos a la programación informática, en los que había grandes fragmentos de *código* informático que, convencionalmente, se suelen representar en los textos impresos con un estilo mono-espaciado que imita al de las terminales informáticas y las viejas máquinas de escribir.

Como T_EX se escribió con una fuente que disponía de esos tres estilos, se le dotó de comandos para activar o desactivar con facilidad cualquiera de ellos. Esta es la razón de que todos los derivados de T_EX, y entre ellos ConT_EXt, asuman que las fuentes tendrán, al menos, esos tres *estilos*, aunque lo cierto es que no todas las fuentes incluyen en su diseño los tres estilos.

A estos tres *estilos* de la fuente, habría que añadir un cuarto estilo pensado para fragmentos matemáticos. Pero como T_EX activa automáticamente dicho estilo cuando entra en modo matemático, no incluye comandos que expresamente lo activen o desactiven, y tampoco tiene las *variantes* o alternativas de los otros estilos, no se le suele considerar un *estilo* propiamente dicho.

ConT_EXt, por su parte, incluye comandos para dos posibles estilos adicionales: escritura a mano y escritura caligráfica. No tengo claro cuál es la diferencia exacta entre ellos pues, de un lado, ninguna de las fuentes incluidas en la distribución de ConT_EXt incluye en su diseño estos estilos y, de otro lado, a mi modo de ver, la escritura caligráfica es también una escritura a mano. Estos comandos que incluye ConT_EXt para activar tales estilos, si se usan con una fuente que no los implementa, no provocan ningún error de compilación: simplemente no hacen nada.

Formas alternativas de la fuente

Cada *estilo* admite varias formas alternativas, a las que ConT_EXt llama, simplemente, *alternativas* (*alternative*):

- Regular o normal («**tf**», de *typeface*).
- Negrita («**bf**», de *boldface*).
- Cursiva («**it**» de *italic*)
- Negrita y cursiva («**bi**» de *bold italic*)
- Inclínada («**sl**» de *slanted*)
- Negrita inclinada («**bs**» de *bold slanted*)
- Versalitas («**sc**» de *small caps*)
- Medieval («**os**» de *old style*)

Estas *alternativas*, como su propio nombre indica, son excluyentes entre sí: Cuando se activa una de ellas se desactivan las restantes. Por ello ConT_EXt proporciona comandos para activarlas, pero no para desactivarlas; porque al activar una alternativa, se desactiva la que hasta entonces se estuviera utilizando; y así, por ejemplo, si estamos escribiendo en cursiva y activamos la negrita, se desactivará la cursiva. Si queremos usar simultáneamente negrita y cursiva no hay que activar una y luego otra, sino que hay que activar la alternativa que las incluye a ambas («**bi**»).

De otro lado, hay que tener en cuenta que aunque ConT_EXt asume que toda fuente tendrá estas alternativas, y, por lo tanto, proporciona comandos para activarlas, dichos comandos necesitan para funcionar y producir algún efecto perceptible en el documento final, que la fuente contenga en su diseño formas específicas para cada estilo y alternativa.

En particular, son muchas las fuentes que no establecen en su diseño ninguna diferencia entre la letra inclinada y la letra cursiva; o que no incluyen formas especiales para las versalitas.

Diferencia entre cursiva y letra inclinada

El parecido en la función tipográfica desempeñada por la cursiva y la letra inclinada lleva a que mucha gente confunda estas dos alternativas. La letra inclinada, llamada en inglés *slanted* se obtiene rotando ligeramente la forma regular. Por el contrario la cursiva implica —al menos en ciertas fuentes— un diseño distinto en el que las letras *parecen* inclinadas porque han sido dibujadas para parecerlo; pero en realidad no hay auténtica inclinación. Eso lo podemos comprobar en el siguiente ejemplo, en el que hemos escrito tres veces las mismas palabras a un tamaño lo suficientemente grande como para que sea fácil apreciar las diferencias. En la primera versión se usa la forma regular, en la segunda la forma inclinada, y en la tercera la cursiva:

Letra cursiva – *Letra cursiva* – *Letra cursiva*

Obsérvese como el diseño de los caracteres es el mismo en los dos primeros ejemplos, pero en el tercero hay sutiles diferencias en el trazo de algunas letras, lo que resulta

muy obvio, sobre todo, en el dibujo de la «a», aunque las diferencias en realidad se dan en casi todos los caracteres.

Los usos habituales de la cursiva y de la letra inclinada son similares y cada cual decide si prefiere usar una u otra. Aquí hay libertad, aunque es preciso indicar que un documento estará mejor compuesto y tendrá mejor aspecto si el uso de cursivas y letra inclinada es *consistente*. En muchas fuentes, además, la cursiva no tiene un diseño diferente de la letra inclinada, por lo que es indiferente usar una u otra.

Tanto la cursiva como la letra inclinada son, por otra parte, alternativas de la fuente, lo que significa principalmente dos cosas:

1. Sólo podemos usarlas cuando están definidas en la fuente.
2. Al activar una de ellas, desactivamos la alternativa que se estuviera usando hasta el momento.

Junto a los comandos para la letra cursiva y la inclinada, ConT_EXt ofrece un comando adicional dirigido a *enfaticar* un determinado texto, cuyo uso implica sutiles diferencias con respecto al uso de la cursiva o la letra inclinada. Véase al respecto en la [sección 6.5.1](#).

6.2.2 Tamaño de la fuente

Todas las fuentes que maneja ConT_EXt se basan en gráficos vectoriales por lo que, en teoría, se pueden mostrar a prácticamente cualquier tamaño, aunque, como veremos, eso depende de la concreta instrucción que usemos para indicar el tamaño de la fuente. Si no se indica otra cosa se asume que el tamaño será de 12 puntos.

Que todas las fuentes usadas por ConT_EXt se basen en gráficos vectoriales, y sean por tanto fuentes Opentype o Type 1, implica que las fuentes cuyo origen es anterior a esta tecnología han sido reimplementadas. En particular la fuente por defecto de T_EX, *Computer Modern*, diseñada por Knuth, sólo existía en ciertos tamaños por lo que fue reimplementada en una fuente llamada *Latin Modern* que es la que utiliza ConT_EXt, aunque en muchos documentos se la sigue llamando *Computer Modern* por el fuerte simbolismo que esa fuente tiene para los sistemas T_EX, ya que estos empezaron funcionando sólo con ella, la cual fue diseñada, además, al mismo tiempo que se desarrollaba el propio T_EX con un programa también de la autoría de Knuth llamado MetaFont pensado para diseñar fuentes que pudieran trabajar con T_EX.

6.3 Fijación de la fuente principal del documento

Por defecto, si no se indica ninguna fuente concreta, ConT_EXt usará *Latin Modern Roman* a un tamaño de 12 puntos como fuente principal. Esta fuente fue originalmente diseñada por el propio KNUTH para implementarla en T_EX. Es una

fuente elegante del tipo llamado «romano» (de ahí el nombre, en inglés, de *roman*), con gran armonía de proporciones y con «remates» decorativos —llamados *serifas* (en inglés *serif*)— en ciertos trazos, que resulta muy apropiada tanto para textos impresos como para ser mostrada en pantalla; aunque —y esto es una opinión personal— no resulta tan indicada para pantallas pequeñas tipo *smartphone*, pues en ellas las *serifas* o remates de las letras tienden a amontonarse dificultando la lectura.

Para establecer una fuente distinta se emplea `\setupbodyfont` que, nos permite no sólo cambiar la fuente propiamente dicha, sino también el tamaño y el estilo de la misma. Cuando queremos que esta orden afecte a todo el documento, debemos incluirla en el preámbulo del fichero fuente. Por el contrario si pretendemos simplemente cambiar la fuente a partir de un punto concreto, será allí donde deberá incluirse.

El formato de `\setupbodyfont` es el siguiente:

`\setupbodyfont[Opciones]`

donde las distintas opciones del comando nos permiten indicar:

- **El nombre de la fuente**, que puede ser cualquiera de los nombres simbólicos de las fuentes que se recogen en la [tabla 6.1](#).
- **El tamaño**, que se puede indicar bien mediante sus dimensiones (usando el punto como unidad de medida) o mediante ciertos nombres simbólicos. Téngase en cuenta, por otra parte, que aunque antes dije que las fuentes que usa ConT_EXt se pueden escalar a prácticamente cualquier tamaño, en `\setupbodyfont` sólo se admiten tamaños consistentes en números enteros entre el 4 y el 12, así como los valores 14.4 y 17.3. Por defecto se asume un tamaño de 12 puntos.

`\setupbodyfont`, por otra parte, establece lo que podríamos llamar el *tamaño base* del documento; es decir: el tamaño de la letra *normal* a partir del cual se calculan otros tamaños como, por ejemplo, los títulos o las notas a pie de página. Al cambiar el tamaño principal con `\setupbodyfont` también se estarán cambiando todos estos otros tamaños que se calculan en relación a la fuente principal.

Además de indicar directamente la dimensión de la letra (10pt, 11pt, 12pt, etc.) también podemos usar ciertos nombres simbólicos que calculan el tamaño de letra a aplicar a partir del tamaño actual. Los nombres simbólicos en cuestión son, de mayor a menor: *big*, *small*, *script*, *x*, *scriptscript* y *xx*. Así, por ejemplo, si queremos establecer con `\setupbodyfont` un cuerpo de texto superior a 12 puntos debemos indicar como tamaño «*big*».

- **El estilo de la fuente**, que, tal y como ya hemos indicado, puede ser romano (con serifas), palo seco (sin serifas), o estilo de máquina de escribir, y en

algunas fuentes se admiten también los estilos escritura a mano y caligráfico. `\setupbodyfont` admite distintos nombres simbólicos para indicar los diferentes estilos. Estos se recogen en la tabla 6.2:

Estilo	Nombres simbólicos que se admiten
Roman	<code>rm</code> , <code>roman</code> , <code>serif</code> , <code>regular</code>
Sans Serif	<code>ss</code> , <code>sans</code> , <code>support</code> , <code>sansserif</code>
Monoespacio	<code>tt</code> , <code>modo</code> , <code>type</code> , <code>teletype</code>
Escritura a mano	<code>hw</code> , <code>handwritten</code>
Caligráfico	<code>cg</code> , <code>calligraphic</code>

Tabla 6.2 Estilos en `setupbodyfont`

Los distintos nombres que se admiten para cada uno de los estilos son, hasta donde yo se, totalmente sinónimos.

Ver la apariencia de una fuente

Antes de decidirnos por usar una fuente concreta en nuestro documento, lo normal es que queramos ver su apariencia. Ello se puede hacer casi siempre desde el sistema operativo pues es habitual que exista alguna utilidad que permita examinar la apariencia de las fuentes instaladas en el sistema; pero para mayor comodidad, ConT_EXt ofrece una utilidad que nos permite ver la apariencia de cualquiera de las fuentes habilitadas en el propio ConT_EXt. Se trata de `\showbodyfont`, que genera una tabla con ejemplos de la fuente indicada.

El formato de `\showbodyfont` es el siguiente:

`\showbodyfont[Opciones]`

Donde como opciones podemos indicar exactamente los mismos nombres simbólicos que en `\setupbodyfont`. Así, por ejemplo `\showbodyfont[schola, 8pt]` nos mostrará la tabla que más abajo se ve, en la que aparecen distintos ejemplos de la fuente schola a un tamaño base de 8 puntos:

[schola] [schola,8pt]													
	<code>\tf</code>	<code>\tf</code>	<code>\bf</code>	<code>\sl</code>	<code>\it</code>	<code>\bs</code>	<code>\bi</code>	<code>\tfx</code>	<code>\tfx</code>	<code>\tfa</code>	<code>\tfa</code>	<code>\tfc</code>	<code>\tfd</code>
<code>\rm</code>	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag
<code>\ss</code>	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag
<code>\tt</code>	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag
<code>\mr</code>	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag

Obsérvese que en la primera fila y en la primera columna de la tabla mostrada se ven ciertos comandos. Más adelante, cuando se haya explicado el significado de esos comandos, volveremos a examinar las tablas generadas por `\showbodyfont`.

Si queremos ver el juego completo de caracteres de una determinada fuente, podemos usar el comando `\showfont [NombreFuente]`. Este comando mostrará el diseño principal de cada uno de los caracteres, sin aplicar comandos de estilo ni alternativas.

6.4 Cambiar la fuente o alguna de sus características

6.4.1 Los comandos `\setupbodyfont` y `\switchtobodyfont`

Para cambiar la fuente, el estilo o el tamaño podemos usar el mismo comando con el que se establece la fuente al principio del documento cuando no queremos usar la fuente por defecto de ConT_EXt: `\setupbodyfont`. Basta con insertar dicho comando en el punto del documento a partir del cual queremos operar el cambio de fuente. Ello producirá un cambio *permanente* de la fuente, es decir: quedará afectada de modo directo la fuente principal y de modo indirecto todas las fuentes que se establecen en relación con la principal.

Muy parecido a `\setupbodyfont` es `\switchtobodyfont`. Ambos comandos permiten cambiar los mismos aspectos de la fuente (la fuente, el estilo y el tamaño) pero internamente funcionan de manera distinta y están pensados para usos diferentes. El primero (`\setupbodyfont`) está pensado para *establecer* la fuente principal (y normalmente única) del documento; por ello lo habitual es que en un documento sólo se utilice una vez ya que no es corriente —ni tipográficamente correcto— que un mismo documento tenga más de una fuente principal (por eso se llama *principal*). Por el contrario `\switchtobodyfont` está pensado para escribir fragmentos de texto con una fuente diferente, o para asignar cierta fuente a algún tipo especial de párrafo que pensemos definir en nuestro documento.

Aparte de lo anterior —que en realidad afecta al funcionamiento interno de cada uno de estos dos comandos—, desde el punto de vista del usuario hay algunas diferencias entre usar uno u otro comando. En particular:

1. Como ya sabemos `\setupbodyfont` está limitado a un rango concreto de tamaños, mientras que `\switchtobodyfont` permite indicar prácticamente cualquier tamaño, de tal modo que si la fuente no está disponible en dicho tamaño, procederá a escalarla.
2. `\switchtobodyfont` no afecta de ninguna manera a los elementos de texto distintos de aquel donde se use, a diferencia de `\setupbodyfont` que, como antes se dijo, establece la fuente principal y, al alterar esta, altera también la

fuente de todos aquellos elementos textuales cuya fuente se calcule a partir de la fuente principal.

Ambos comandos, de otro lado, cambian no solo la fuente, el estilo y el tamaño, sino también otros aspectos asociados a la fuente usada tales como, por ejemplo, el interlineado.

`\setupbodyfont` genera un error de compilación si se solicita un tamaño de fuente no permitido; pero no lo genera si se solicita una fuente no existente, caso este último en el que se activará la fuente por defecto (*Latin Modern Roman*). `\switchtobodyfont` actúa igual respecto de la fuente, y en cuanto al tamaño, como ya he dicho, intenta conseguirlo escalando la fuente. No obstante hay fuentes que no pueden ser escaladas a ciertos tamaños, caso este en el que una vez más se activará la fuente por defecto.

6.4.2 Cambio rápido de estilo, alternativa y tamaño

Cambio de estilo y de alternativa

Además de `\switchtobodyfont`, ConT_EXt proporciona un grupo de comandos que permiten cambiar con rapidez el estilo, la alternativa, o el tamaño. Respecto a estos comandos advierte la wiki de ConT_EXt que en ocasiones, cuando figuran al principio de un párrafo, pueden producir algunos efectos colaterales indeseados, por lo que se recomienda que en tal caso el comando en cuestión sea precedido del comando `\dontleavehmode`.

Estilo	Comandos que lo habilitan
Roman	<code>\rm</code> , <code>\roman</code> , <code>\serif</code> , <code>\regular</code>
Sans Serif	<code>\ss</code> , <code>\sans</code> , <code>\support</code> , <code>\sansserif</code>
Monoespaciado	<code>\tt</code> , <code>\mono</code> , <code>\teletype</code> ,
Escritura a mano	<code>\hw</code> , <code>\handwritten</code> ,
Caligráfico	<code>\cf</code> , <code>\calligraphic</code>

Tabla 6.3 Comandos para cambiar entre los distintos estilos

La [tabla 6.3](#) contiene los comandos que permiten cambiar el estilo, sin alterar ningún otro aspecto; y la [tabla 6.4](#) contiene los comandos que permiten cambiar exclusivamente la alternativa.

Alternativa	Comandos que la activan
Normal	<code>\tf</code> , <code>\normal</code>
Cursiva	<code>\it</code> , <code>\italic</code>
Negrita	<code>\bf</code> , <code>\bold</code>
Negrita-Cursiva	<code>\bi</code> , <code>\bolditalic</code> , <code>\italicbold</code>
Inclinada	<code>\sl</code> , <code>\slanted</code>
Negrita-Inclinada	<code>\bs</code> , <code>\boldslanted</code> , <code>\slantedbold</code>
Versalitas	<code>\sc</code> , <code>\smallcaps</code>
Medieval	<code>\os</code> , <code>\mediaeval</code>

Tabla 6.4 Comandos para activar una alternativa concreta

Todos estos comandos mantienen su eficacia hasta que se activa explícitamente otro estilo u otra alternativa o se termina el *grupo* dentro del que se declaró el comando. Por ello, cuando queremos que el comando sólo afecte a un fragmento de texto lo que hay que hacer es encerrar dicho fragmento en un grupo, como en el siguiente ejemplo, en el que cada vez que aparece la palabra *cuento* en el sentido de relato, se ha puesto en cursiva, creando un grupo para ello.

```
Cuando cuentas {\it cuentos}, cuenta
cuántos {\it cuentos} cuentas, porque
si no cuentas cuántos {\it cuentos}
cuentas, nunca sabrás cuántos
{\it cuentos} cuentas tú.
```

```
Cuando cuentas cuentos, cuenta cuántos cuen-
tos cuentas, porque si no cuentas cuántos cuen-
tos cuentas, nunca sabrás cuántos cuentos cuen-
```

Sufijos para cambiar el tamaño junto con el estilo y la alternativa

Los comandos que cambian el estilo o la alternativa, en su versión de dos letras (`\tf`, `\it`, `\bf`, etc.) admiten una serie de *sufijos* que afectan al tamaño de la fuente. Los sufijos a, b, c y d, aumentan el tamaño de la fuente multiplicándolo, respectivamente, por 1.2, 1.2^2 (= 1.44), 1.2^3 (= 1.728) o 1.2^4 (= 2.42). Véase un ejemplo:

```
\tf prueba, \tfa prueba, \tfb prueba, \tfc prueba, \tfd prueba
```

prueba, prueba, prueba, prueba, prueba

los sufijos x y xx reducen el tamaño de la fuente multiplicándolo, respectivamente, por 0.8 y 0.6:

```
\tf prueba, \tfx prueba, \tfxx prueba
```

prueba, prueba, prueba

Los sufijos «x» y «xx» aplicados a `\tf` permiten abreviar el comando de tal modo que `\tfx` se puede escribir como `\tx` y `\tfxx` como `\txx`.

La disponibilidad de los diferentes sufijos, por otra parte, depende de la concreta implementación de la fuente. Según el manual de referencia de ConT_EXt (en su última versión, de 2013 y pensado en gran medida para Mark II) el único sufijo que se garantiza que siempre funcionará es «x» los restantes pueden o no estar implementados; y pueden estarlo solo para algunas alternativas.

En todo caso, para salir de dudas, podemos usar `\showbodyfont`, de la que ya antes se habló (en la [sección](#)). Este comando muestra un cuadro que no sólo nos permite apreciar la apariencia de la fuente, sino también ver cómo es la fuente

en cada uno de sus estilos y alternativas, así como que sufijos de alteración del tamaño están disponibles.

Volvamos a mirar la tabla mostrada por `\showbodyfont`:

[modern]													
	<code>\tf</code>	<code>\tf</code>	<code>\bf</code>	<code>\sl</code>	<code>\it</code>	<code>\bs</code>	<code>\bi</code>	<code>\tfx</code>	<code>\tfxx</code>	<code>\tfa</code>	<code>\tfb</code>	<code>\tfc</code>	<code>\tfd</code>
<code>\rm</code>	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag
<code>\ss</code>	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag
<code>\tt</code>	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag
<code>\mr</code>	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag

Si nos fijamos bien en la tabla veremos que la primera columna recoge los estilos de la fuente (`\rm`, `\ss` y `\tt`). La primera fila recoge, en el lado izquierdo, las alternativas (`\tf`, `\sc`, `\sl`, `\it`, `\bf`, `\bs` y `\bi`), mientras el lado derecho de la primera fila recoge los sufijos disponibles, aunque, solo con la alternativa regular.

Es importante tener en cuenta que un cambio en el tamaño de la letra realizado mediante alguno de estos sufijos, sólo cambiará el tamaño de la letra en el sentido estricto, dejando intactos otros valores que normalmente van asociados con el tamaño de la letra como puede ser, por ejemplo, el interlineado.

Personalización del factor de escalado de los sufijos

Para personalizar el factor de escalado, podemos usar `\definebodyfontenvironment` cuyo formato puede ser:

```
\definebodyfontenvironment[tamaño concreto][escalado]
\definebodyfontenvironment[default][escalado]
```

En la primera versión redefiniríamos el escalado para un tamaño concreto de la fuente principal establecido mediante `\setupbodyfont` o mediante `\switchto-bodyfont`. Por ejemplo:

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

haría que cuando la fuente principal sea de 10 puntos, el sufijo “a” cambie a 12 puntos, el sufijo “b” cambie a 14, el sufijo “c” multiplique la fuente original por 2 y el sufijo “d” lo multiplique por 3. Obsérvese que para a y b se ha indicado una

dimensión fija, pero para c y d se ha indicado un factor multiplicador del tamaño original.

Pero cuando el primer argumento de `\definebodyfontenvironment` es igual a «`default`», entonces estaremos redefiniendo el valor de escalado para todos los tamaños posibles de fuente, y como valor de escalado sólo podremos introducir un número multiplicador. De modo que si, por ejemplo, escribimos:

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

estaremos indicando que sea cual sea el tamaño de la fuente principal, el sufijo a la multiplique por 1.3, el b por 1.6, el c por 2 y el d por 4.

Además de a los sufijos xx, x, a, b, c y d, mediante `\definebodyfontenvironment` podemos asignar un valor de escalado a las palabras clave «`big`», «`small`», «`script`» y «`scriptscript`». Estos valores se aplican a los tamaños que en `\setupbodyfont` y `\switchtobodyfont` se asocian a dichas palabras clave. También se aplican en los siguientes comandos, cuya utilidad se deduce (creo) de su propio nombre:

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

Si queremos ver los tamaños por defecto de una determinada fuente podemos usar `\showbodyfontenvironment[Fuente]`. Este comando, aplicado a la fuente `modern`, por ejemplo, ofrece el siguiente resultado:

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

6.4.3 Definir comandos y palabras clave para los tamaños, estilos y alternativas de las fuentes

Los comandos predefinidos para el cambio de estilo, tamaño o variante de la fuente son suficientes, en mi opinión, para casi todo lo que queramos hacer. Pero además ConT_EXt nos permite:

1. Añadir nuestros propios comandos de cambio de estilo, tamaño o variante de la fuente.
2. Añadir sinónimos a los nombres de estilo o variante, que sean reconocidos por `\switchtobodyfont`.

Para ello se dispone de los siguientes comandos:

- `\definebodyfontswitch`: Permite definir comandos de cambio del tamaño de la fuente. Por ejemplo, si queremos definir el comando `\ocho` (o el comando `\viii`¹) para establecer una fuente de 8 puntos, deberíamos escribir:

```
\definebodyfontswitch[ocho][8pt] o \definebodyfontswitch[viii][8pt]
```

- `\definefontstyle`: Permite definir una o más palabras que podrán usarse en `\setupbodyfont` o en `\switchtobodyfont` para establecer un concreto estilo

¹ Recuérdese que, salvo en el caso de los símbolos de control, los nombres de los comandos de ConT_EXt sólo pueden constar de letras.

de fuente; de modo que si queremos, por ejemplo, llamar «paloseco» a la fuente *sans serif* podríamos escribir

```
\definefontstyle[paloseco][ss]
```

Una peculiaridad de `\definefontstyle` es que permite asociar simultáneamente varias palabras a un mismo estilo, como, por ejemplo en

```
\definefontstyle[paloseco, sosa, sinrebordes][ss]
```

- `\definealternativestyle`: Permite asociar un nombre a una variante de la fuente. Ese nombre podrá funcionar como comando, o ser reconocido por la opción `style` de aquellos comandos que permite configurar el estilo que aplicarán. Así, por ejemplo, el siguiente fragmento

```
\definealternativestyle[negrita][\bf]
```

habilitará el comando `\negrita` y la palabra clave «**negrita**» que será reconocida por la opción `style` de aquellos comandos que la admiten.



El tercer argumento de `\definealternativestyle` no se para qué sirve. No es opcional y, por lo tanto, no puede omitirse; pero la única información que he encontrado sobre él está en el manual de referencia de ConT_EXt donde se dice que este tercer argumento sólo tiene importancia en los títulos de capítulos y secciones «*donde, aparte de `\cap`, debemos obedecer la fuente usada aquí*» (¿?)

6.5 Otras cuestiones relacionadas con el uso de algunas alternativas

Dentro de las distintas alternativas de una fuente, hay dos cuyo uso requiere ciertas precisiones

6.5.1 Cursiva, letra inclinada y enfatización

Tanto la cursiva como la letra inclinada se usan principalmente para destacar tipográficamente un fragmento del texto; para llamar la atención sobre él. Es decir: para *enfatizarlo*.

Podemos, claro está, enfatizar un texto activando explícitamente la cursiva o la letra inclinada. Pero ConT_EXt ofrece un comando alternativo, mucho más útil e interesante, dirigido específicamente a la enfatización de un fragmento de texto. Se trata del comando `\em`, cuyo nombre procede de la palabra inglesa *emphasis*. Frente a los comandos `\it` y `\sl`, que son comandos puramente tipográficos, `\em` es un comando *conceptual*; funciona de una manera distinta y es, por ello, más versátil, hasta el punto de que la documentación de ConT_EXt recomienda usar

`\em` con preferencia a `\it` o `\sl`. Cuando usamos estos dos últimos comandos le estamos diciendo a ConT_EXt qué alternativa de la fuente queremos que utilice; pero cuando usamos `\em` le estamos diciendo qué efecto pretendemos conseguir, dejando en manos de ConT_EXt el cómo conseguirlo. Normalmente, para conseguir el efecto de enfatizar o destacar el texto lo que hará es activar la letra inclinada (o la cursiva), pero ello depende del contexto y así, si usamos `\em` dentro de un texto que ya está inclinado —o en cursiva— el comando provocará que, para destacarlo de su entorno, se desactive la cursiva.

Así el siguiente ejemplo:

```
Ya lo dijo Salomón: {\em Oh bella
reina de Saba, vámonos a
{\em Mazarrón}}.
```

```
Ya lo dijo Salomón: Oh, bella reina de Saba,
vámonos a Mazarrón.
```

Obsérvese que el primer `\em` activa la letra inclinada para toda la frase del Rey Salomón, y el segundo la desactiva para la palabra “Mazarrón”.

Otra ventaja de `\em` es que no se trata de una alternativa, por lo que no desactiva la alternativa que hubiera antes y así, por ejemplo, dentro un texto en negrita `\em` provocará la negrita inclinada sin necesidad de invocar explícitamente a `\bs`. Asimismo, si dentro un texto que se está enfatizando, aparece el comando `\bf`, eso no hará que la enfatización cese.

Por defecto `\em` activa la letra inclinada y no la cursiva, pero eso podemos cambiarlo mediante `\setupbodyfontenvironment[default][em=italic]`.

6.5.2 Versalitas y pseudoversalitas

Las versalitas constituyen un recurso tipográfico que en numerosas ocasiones queda mucho mejor que el uso de las mayúsculas propiamente dichas, pues la versalita, aunque proporciona a los caracteres la forma de la letra mayúscula, no sobresale por su altura dentro de la línea de texto. A pesar de su apariencia formal, se trata de una variante estilística de la minúscula, por lo que —dice la Real Academia de la Lengua en su Libro de Estilo (pág. 162)—, cuando se emplea, la mayúscula inicial debe mantenerse en aquellas palabras que lo requieran. Este tipo de letra se usa fundamentalmente —sigo citando la obra anterior— “para reemplazar a la mayúscula en determinados contextos, favoreciendo el equilibrio tipográfico del texto y evitando el exceso de mancha, así como para resaltar una palabra o fragmento en aquellos casos en los que el uso de la cursiva o la negrita no es lo más indicado, o para destacarlos en un texto que ya presenta estos estilos”. Es especialmente útil su uso en la escritura de números romanos, o en los títulos de textos o capítulos. En textos académicos también es costumbre usar versalitas para escribir los nombres de los autores citados.

El problema es que, de un lado, no todas las fuentes implementan las versalitas, y las que lo hacen no siempre las contemplan para todos los *estilos de fuente*. Además, al ser las versalitas una alternativa a la cursiva, la negrita o la letra inclinada, de acuerdo con las reglas generales que hemos expuesto en este mismo capítulo, no podrían usarse simultáneamente todas estas características tipográficas.

Estos problemas se pueden resolver utilizando las *pseudoversalitas* que ConT_EXt permite crear mediante los comandos `\cap` y `\Cap`; véase al respecto la [sección 10.2.1](#).

6.6 Uso y configuración de los colores

ConT_EXt proporciona comandos para cambiar el color de todo el documento, de alguno de sus elementos, o de ciertos fragmentos de texto. También proporciona comandos para cargar en memoria cientos de colores predefinidos, y para ver cuáles son sus componentes.

6.6.1 Procedimientos para escribir fragmentos de texto en color

La mayor parte de los comandos configurables de ConT_EXt admite una opción llamada «color» que nos permite indicar el color en el que se debe escribir el texto afectado por dicho comando. Así, por ejemplo, para indicar que los títulos de los capítulos se escriban en color azul basta con escribir:

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

Por este procedimiento podemos colorear los títulos, encabezados, notas a pie de página, notas al margen, barras y líneas, tablas, títulos de tablas o imágenes, etc. La ventaja de usar este procedimiento está en que el resultado final será coherente (todos los textos que cumplen la misma función se escribirán con el mismo color) y más fácil de cambiar globalmente.

También podemos colorear directamente una porción o fragmento de texto, aunque, para evitar un uso demasiado abigarrado de los colores, que no es agradable desde una perspectiva tipográfica, o un uso inconsistente, en general se recomienda huir del coloreado directo y utilizar lo que podríamos llamar un *coloreado semántico*, es decir: en lugar de, por ejemplo, escribir

```
\color[red]{Texto muy importante}
```

definir un comando para el texto muy importante que lo coloree. Por ejemplo

```
\definehighlight[importante][color=red]
\importante{Texto muy importante}
```

6.6.2 Cambiar el color del fondo y del primer plano del documento

Si lo que deseamos es cambiar el color de todo el documento, dependiendo de que deseamos alterar el color del fondo o el color del primer plano (texto) usaremos `\setupbackgrounds` o `\setupcolors`. Así, por ejemplo

```
\setupbackgrounds
[page]
[background=color,backgroundcolor=blue]
```

Este comando establecerá el color del fondo de las páginas en azul. Como valor para «`backgroundcolor`» podemos usar el nombre de cualquier color predefinido.

Para cambiar, con carácter global, el color del primer plano en todo el documento (desde el punto en el que se inserte el comando) se usa `\setupcolors`, donde la opción «`textcolor`» controla el color del texto. Por ejemplo:

```
\setupcolors[textcolor=red]
```

hará que el color del texto sea el rojo.

6.6.3 Comandos para colorear fragmentos concretos de texto

El comando general para colorear porciones pequeñas de texto es

```
\color[NombreColor]{Texto a colorear}
```

Para porciones más grandes de texto es preferible usar

```
\startcolor[NombreColor] ... \stopcolor
```

Tanto uno como otro reciben como argumento el nombre de algún color predefinido. Si queremos definir el color sobre la marcha, podemos usar el comando `\colored`. Por ejemplo:

```
Tres gatos \colored[r=0.1, g=0.8, b=0.8] Tres gatos coloreados.
{coloreados}
```

6.6.4 Colores predefinidos

ConT_EXt carga por defecto la predefinición de los colores más corrientes cuyos nombres en inglés se recogen en la [tabla 6.5](#)¹.

Nombre (español)	Nombre (inglés)	Tono claro	Tono medio	Tono oscuro
Negro	black			
Blanco	white			
Gris	gray	lightgray	middlegray	darkgray
Rojo	red	lightred	middlered	darkred
Verde	green	lightgreen	middlegreen	darkgreen
Azul	blue	lightblue	middleblue	darkblue
Cian	cyan		middlecyan	darkcyan
Magenta	magenta		middlemagenta	darmagenta
Amarillo	yellow		middleyellow	darkyellow

Tabla 6.5 Colores predefinidos en ConT_EXt

Pero también dispone de otras colecciones de colores que no se cargan por defecto, pero que se pueden cargar mediante el comando

`\usecolors[NombreColección]`

donde nombrecolección puede ser

- «**crayola**», 235 colores que imitan el tono de los rotuladores.
- «**dem**», 91 colores.
- «**ema**», 540 definiciones de color basados en los colores usados por Emacs.
- «**rainbow**», 91 colores pensados para ser usados en fórmulas matemáticas.
- «**ral**», 213 definiciones de color del *Deutsches Institut für Gütesicherung und Kennzeichnung* (Instituto Alemán de Aseguramiento de la Calidad y Etiquetado).
- «**rgb**», 223 colores.
- «**solarized**», 16 colores basados en un esquema de solarización.
- «**svg**», 147 colores.
- «**x11**», 450 colores estándar para X11.
- «**xwi**», 124 colores.

Los ficheros de definición de colores se incluyen en el directorio «context/base/mkiv» de la distribución y su nombre responde al esquema «colo-imp-NOMBRE.mkiv». La información que acabo de exponer sobre las distintas colecciones de colores predefinidos, se basa en mi concreta distribución. Las concretas colecciones, o el número de colores definidos en ellas, podrían cambiar en versiones futuras.

¹ Esta lista se encuentra en el manual de referencia y en la wiki de ConT_EXt pero estoy bastante seguro de que es una lista incompleta pues en este documento, por ejemplo, sin haber cargado ningún color adicional, se usa el color «orange» —que no está en la [tabla 6.5](#)— para los títulos de los epígrafes.

Para ver qué colores contiene cada una de estas colecciones podemos usar el comando `\showcolor[NombreColección]` que se describe a continuación. Para usar alguno de estos colores, primero hay que cargar en memoria la colección (`\usecolors[NombreColección]`) y luego hay que indicar al comando `\color` o a `\startcolor` el nombre del color. Por ejemplo la siguiente secuencia:







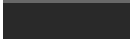
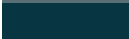
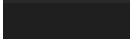
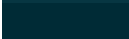


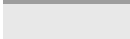
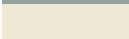
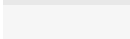
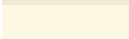

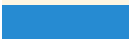








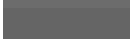

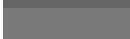

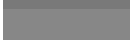

```
\usecolors[xwi]
\color[darkgoldenrod]{Tararí que te vi, Mariví}
```

escribirá

Tararí que te vi, Mariví

6.6.5 Ver los colores disponibles

El comando `\showcolor` muestra un listado de colores en el que se puede ver la apariencia del color, su apariencia cuando el color se usa en escala de grises, los componentes rojo, verde y azul del color, y el nombre por el que ConTeXt lo conoce. Usado sin ningún argumento `\showcolor` mostrará los colores usados en el documento actual. Pero como argumento podemos indicar cualquiera de las colecciones predefinidas de colores de las que se habló en la [sección 6.6.4](#), y así, por ejemplo, `\showcolor[solarized]` nos mostrará los 16 colores solarizados de dicha colección:

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

Si lo que queremos es ver las componentes rgb de algún concreto color, podemos usar `\showcolorcomponents[NombreColor]`. Esto es útil si buscamos definir un concreto color, para ver la composición de algún color que se le aproxime. Así por ejemplo, `\showcolorcomponents[darkgoldenrod]` nos mostrará:

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

6.6.6 Definir nuestros propios colores

`\definecolor` nos permite, bien clonar un color ya existente, bien definir un nuevo color. Clonar un color ya existente es tanto como crear un nombre alternativo para el mismo. Para ello habría que escribir:

```
\definecolor[Color nuevo][Color viejo]
```

Esa orden hará que “*Color nuevo*” sea exactamente el mismo color que “*Color viejo*”.

Pero la principal utilidad de `\definecolor` está en la creación de colores nuevos. Para ello el comando se debe usar de la siguiente forma:

```
\definecolor[NombreColor][Definición]
```

donde *Definición* puede hacerse aplicando hasta seis esquemas distintos de generación de colores:

- **Colores RGB:** La definición de colores RGB es una de las más extendidas; se basa en la idea de que es posible representar un color mediante la mezcla, por adición, de los tres colores primarios: rojo («r» de *red*), verde («g» de *green*) y azul («b» de *blue*). Cada uno de estos componentes se indica como un número decimal entre 0 y 1. Por ejemplo

```
\definecolor[lima 1][r=0.75, g=1, b=0]: Texto en color “lima 1”.
```

- **Colores Hex:** Esta forma de representar los colores se basa también en el esquema RGB, pero los componentes de rojo, verde y azul se indican como un número hexadecimal de tres bytes en el que el primer byte representa el valor de rojo, el segundo el valor de verde y el tercero el valor de azul. Por ejemplo:

```
\definecolor[lima 2][x=BFFF00]: Texto en color “lima 2”.
```

- **Colores CMYK:** Este modelo de generación de colores, es lo que se llama un «modelo sustractivo» y se basa en la mezcla de pigmentos de los siguientes colores: cian («c»), magenta («m»), amarillo («y», de *yellow*) y negro («k», de *key*). Cada uno de estos componentes se indica como un número decimal entre 0 y 1. Por ejemplo:

```
\definecolor[lima 3][c=0.25, m=0, y=1, k=0]: Texto en color “lima 3”.
```

- **Colores HSL/HSV:** Este modelo de colores parte de medir el matiz («h», de *hue*), la saturación («s») y la luminosidad («l» o a veces «v», de *value*). El matiz se corresponde con un número entre 0 y 360; la saturación y la luminosidad han de ser un número decimal entre 0 y 1. Por ejemplo:

```
\definecolor[lima 4][h=75, s=1, v=1]: Texto en color “lima 4”
```

- **Colores HWB:** El modelo HWB es un estándar sugerido para CSS4 que mide el matiz («h», de *hue*), y el nivel de blanco («w», de *whiteness*) y de negro («b», de *blackness*). El matiz se corresponde con un número entre 0 y 360, mientras que la blancura y la negrura se representan mediante un número decimal entre 0 y 1.

```
\definecolor[Azulón][h=75, w=0.2, b=0.7] Texto en color “Azulón”.
```

- **Escala de grises:** Se basa en un solo componente («s», de *scale*) que mide el valor de gris. Ha de ser un número decimal entre 0 y 1. Por ejemplo:

```
\definecolor[gris pachucho][s=0.65]: Texto en “gris pachucho”.
```

También es posible definir un nuevo color a partir de otro color. Por ejemplo, el color en el que en esta introducción se escriben los títulos viene definido como

```
\definecolor[ColorPrincipal][0.6(orange)]
```

Capítulo 7

Estructura del documento

Sumario: 7.1 Divisiones estructurales en los documentos; 7.2 Tipos de secciones y jerarquía de las mismas; 7.3 Sintaxis común de los comandos de seccionado; 7.4 Formateo y configuración de las secciones y sus títulos; 7.4.1 Los comandos `\setuphead` y `\setupheads`; 7.4.2 Las partes del título de una sección; 7.4.3 Control de la numeración (en las secciones numeradas); 7.4.4 Estilo y color de los títulos; 7.4.5 Ubicación del número y del texto del título; 7.4.6 Comandos o acciones a realizar antes o después de imprimir el título; 7.4.7 Otras características configurables; 7.4.8 Otras opciones de `\setuphead`; 7.5 Definición de nuevos comandos de seccionado; 7.6 Macroestructura del documento;

7.1 Divisiones estructurales en los documentos

Salvo en el caso de textos muy pequeños (como, por ejemplo, una carta), lo normal es que un documento se estructure dividiéndose en bloques o partes del texto que, en general, siguen un orden jerárquico. No hay un estándar para la denominación de estos bloques: en novelas, por ejemplo, las divisiones estructurales suelen denominarse «capítulo», aunque algunas —las más largas— incluyen también bloques más amplios, llamados habitualmente «parte», que agrupan a varios capítulos. Las obras de teatro diferencian entre «actos» y «escenas». Los manuales académicos se dividen en «partes» (a veces) y «lecciones», «temas» o «capítulos», los cuales a su vez suelen tener también divisiones internas; este mismo tipo de divisiones jerárquicas complejas suele existir también en otros documentos académicos o técnicos (como, por ejemplo, los textos como el presente dedicados a explicar un programa o sistema informático). Incluso las leyes se estructuran en «libros» (las más largas y complejas, como los Códigos), «títulos», «capítulos», «secciones» y «subsecciones». También los documentos científico-técnicos pueden llegar a alcanzar hasta seis, siete e incluso en ocasiones ocho niveles de profundidad en el anidamiento de este tipo de divisiones.

El presente capítulo se dedica a analizar el mecanismo que ConT_EXt ofrece para soportar estas divisiones estructurales a las que me referiré globalmente como «secciones».

No existe en castellano —y sospecho que tampoco en inglés— una denominación clara que permita referirse genéricamente a este tipo de divisiones estructurales. El “Libro de Estilo de la Lengua Española”, de la Real Academia de la Lengua, se refiere a ellas como “Elementos de titulación”; pero creo que el término *Título* es más adecuado para referirse a uno de los elementos de que estas unidades estructurales constan: el texto o rótulo inicial que le da nombre. El término *Epígrafe* creo que tiene el mismo inconveniente, pone el acento en la denominación global del bloque, y no en el bloque en sí mismo considerado. El término «sección», que es por el que finalmente me he decantado, tiene la ventaja de que fija la atención más en la división estructural de que se trata que en su título, pero tiene el inconveniente de que una de las divisiones estructurales predeterminadas de ConT_EXt se denomina «section». Espero que ello no ocasione confusiones, pues creo que por el contexto será fácil determinar cuando se está hablando globalmente de este tipo de divisiones estructurales y cuando se está hablando de un tipo concreto de división estructural llamada en inglés «section» y, en castellano, «sección».

Cada «sección» implica:

- *Una división estructural del documento* más o menos larga, que puede, a su vez incluir otras divisiones de menor nivel jerárquico. Desde este punto de vista las «secciones» implican bloques de texto, con una relación jerárquica entre ellos. Desde el punto de vista de sus secciones, todo el documento se puede examinar como un árbol. El documento, en sí mismo considerado es el tronco, cada uno de los capítulos constituye una rama, la cual, a su vez, puede tener subramas, que también pueden subdividirse, y así sucesivamente.

Que esta estructura esté clara es muy importante para la legibilidad y aprehensibilidad del documento. Pero esta tarea corresponde al autor, no a quien se ocupa de la composición tipográfica. Aunque no está en la mano de ConT_EXt el convertirnos en mejores autores de lo que somos, el completo juego de comandos de seccionado que incluye, en el que la jerarquía entre ellos está bastante clara, si puede ayudarnos a escribir documentos bien estructurados.

- *Una denominación* a la que podemos llamar «título» o «rótulo». Esta denominación se imprime:
 - Siempre (o casi) en el punto del documento en el que empieza la división estructural.
 - A veces también en el índice, en el encabezado o en el pie de las páginas ocupadas por la sección de que se trate.

ConT_EXt permite automatizar todas estas tareas de tal modo que sólo haya que indicar una vez las características de formato en que se debe imprimir el título de una unidad estructural, y si el mismo debe incluirse también —o no— en los índices, o en los encabezados o pies. Para hacer eso ConT_EXt sólo necesita saber dónde empieza y dónde acaba cada unidad estructural, cómo se titula, y cuál es su nivel jerárquico.

7.2 Tipos de secciones y jerarquía de las mismas

ConT_EXt diferencia entre secciones *numeradas* y secciones *no numeradas*. Las primeras, como su propio nombre indica, son numeradas automáticamente y enviadas a la tabla de contenido del documento así como, en ocasiones, a los encabezados o pies de página.

Ordenados jerárquicamente, ConT_EXt tiene predefinidos los comandos de seccionado que se recogen en la [tabla 7.1](#).

Nivel	Secciones numeradas	Secciones no numeradas
1	\part	—
2	\chapter	\title
3	\section	\subject
4	\subsection	\subsubject
5	\subsubsection	\subsubsubject
6	\subsubsubsection	\subsubsubsubject
...

Tabla 7.1 Comandos de seccionado en ConT_FXt

En relación con las secciones predefinidas, conviene hacer las siguientes precisiones:

- En la [tabla 7.1](#) los comandos de seccionado se muestran en su forma tradicional. Inmediatamente veremos que también se pueden usar como *entornos* (`\startchapter ... \stopchapter`, por ejemplo) y que, de hecho, se recomienda ese estilo.
- En la tabla se recogen sólo los 6 primeros niveles de seccionado. Pero, según mis pruebas, se admiten hasta 12 niveles: Después de `\subsubsubsection` vendrá `\subsubsubsubsection`, y así sucesivamente hasta `\subsubsubsubsubsubsubsubsubsubsubsection`, o `\subsubsubsubsubsubsubsubsubsubsubject`.

Debe tenerse en cuenta que tan malo para la aprehensibilidad de un documento es un bajo nivel de profundidad en su seccionado, como un nivel excesivo. En el primer caso tendremos secciones largas que inevitablemente tratarán de varios aspectos y harán más difícil que el lector *aprehenda* su contenido. Pero una profundidad excesiva puede hacer que se pierda la visión de conjunto produciéndose una sensación de excesiva fragmentación de la materia. En general entiendo que con cuatro niveles de seccionado suele ser suficiente; ocasionalmente puede llegarse a los seis o siete niveles; pero raramente será buena idea una mayor profundidad.

Desde la perspectiva de la escritura del fichero fuente, por otra parte, el hecho de que a partir de la subsección cada nivel se limite a añadir un «sub» adicional al nombre del nivel previo, puede hacer que el fichero fuente resulte poco legible: si para identificar el nivel

de profundidad de un comando llamado «subsubsubsubsubsection» me tengo que poner a contar los «subs»..., mal asunto. Por ello mi consejo es que si realmente necesitamos tantos niveles de profundidad, a partir del quinto nivel (subsubsection) definamos nuestros propios comandos de seccionado (véase la [sección 7.5](#)) dotándoles de nombres más claros que los preestablecidos.

- El nivel de seccionado más alto (`\part`) sólo existe para los títulos numerados y tiene la peculiaridad de que no imprime el título de la parte. Pero, aunque no se imprima el título, sí se introduce una página en blanco (en la que se supone que se imprimirá el título una vez el usuario haya reconfigurado el comando) y la numeración de la *parte* se tiene en cuenta para calcular la numeración de los capítulos y demás secciones.

La razón por la que de forma predeterminada `\part` no imprime nada es, según la wiki de ConT_EXt, la consideración de que casi siempre el título de este nivel requiere un diseño específico; lo que, siendo cierto, no me parece que sea una razón suficiente, pues, en la práctica, también suelen redefinirse los capítulos y secciones, y el hecho de que las partes no impriman nada, obliga al usuario novato a *bucear* en la documentación para ver qué es lo que está fallando.

- Aunque el primer nivel de seccionado es la «parte», eso es así sólo de forma teórica y abstracta. En un documento concreto, el primer nivel de seccionado será el correspondiente al primer comando de seccionado que haya en el documento. Es decir: en un documento que no incluya partes pero sí capítulos, ese será el primer nivel. Pero si el documento tampoco incluye capítulos, sino secciones, la jerarquía, para ese documento, empezará en las secciones.

7.3 Sintaxis común de los comandos de seccionado

Todos los comandos de seccionado, incluyendo los posibles niveles creados por el usuario (véase la [sección 7.5](#)), admiten las siguientes sintaxis alternativas (se usa, para el ejemplo, el nivel «`section`»):

```
\section [Etiqueta] {Título}
\section [Opciones]
\startsection [Opciones] [Variables] ... \stopsection
```

En las tres modalidades, los argumentos entre corchetes son opcionales y pueden omitirse. Veámoslas por separado, pero antes conviene dejar claro que en Mark IV se recomienda siempre usar la tercera modalidad.

- En la primera sintaxis, a la que podríamos bautizar como «*clásica*», el comando recibe dos argumentos, uno, opcional, entre corchetes, y otro obligatorio entre

llaves. El argumento opcional sirve para asociar la sección a una etiqueta que será usada para remisiones internas (véase la [sección 9.2](#)). El obligatorio recoge, entre llaves, el título de la sección.

- Las otras dos sintaxis son más del estilo de ConT_EXt: todo lo que el comando necesita saber se le comunica mediante valores y opciones, introducidas entre corchetes.

Recuérdese que en las [secciones 3.3.1](#) y [3.4](#) dije que, en ConT_EXt entre llaves se indica el ámbito de aplicación del comando, y entre corchetes sus opciones. Pero si lo pensamos bien el título de un concreto comando de seccionado no constituye su ámbito de aplicación, por lo que, para ser coherentes con la sintaxis general, no debería introducirse entre llaves, sino como una opción. ConT_EXt admite la excepción, porque es el modo clásico de hacer las cosas de T_EX, pero proporciona sintaxis alternativas, más coherentes con su diseño general.

Las opciones son del tipo de asignación de valor (NombreOpción=Valor), y son las siguientes:

- **reference**: Etiqueta para referencias cruzadas.
- **title**: El título de la sección que se imprimirá en el cuerpo del documento.
- **list**: El título de la sección que se imprimirá en el índice.
- **marking**: El título de la sección que se imprimirá en los encabezados o pies de página.
- **bookmark**: El título de la sección que se convertirá en *marcador* en el fichero PDF.
- **ownnumber**: Esta opción se usa en el caso de que se trate de una sección que no se numera automáticamente; supuesto este en el que esta opción recogerá el número asignado a la sección de que se trate.

Como es lógico, las opciones «**list**», «**marking**» y «**bookmark**» sólo se deben usar si queremos que en esos lugares se use un texto diferente al título principal establecido con la opción «**title**». Esto es muy útil, por ejemplo, cuando el título es demasiado largo para el encabezado; aunque para conseguir ese efecto (más bien un efecto parecido) también pueden usarse los comandos `\nomarking` y `\nolist`.

De otro lado, debe tenerse en cuenta que si el texto del título (opción «**title**») incluye alguna coma, habrá que encerrar entre llaves, bien el texto completo del título, bien la coma, para asegurarnos de que ConT_EXt sepa que la coma es parte del título. Igual ocurre en las opciones «**list**», «**marking**» y «**bookmark**». Por ello, para no tener que estar pendientes de si en el título hay o no alguna

coma, creo que es una buena idea coger el hábito de encerrar siempre entre llaves el valor de cualquiera de estas opciones.

Así, por ejemplo, las siguientes líneas crearán un capítulo con el título «Capítulo de prueba» al que se asociará la etiqueta «prueba» para referencias cruzadas, y que enviará al encabezado no el nombre del capítulo, sino el texto «Prueba de capítulo»

```
\chapter
[
  title={Capítulo de prueba},
  reference=prueba,
  marking={Prueba de capítulo}
]
```

La sintaxis `\startTipoSección` convierte a la sección en un *entorno*. Eso es más coherente con el hecho de que, como dije al principio, en el fondo cada sección es un bloque de texto diferenciado, aunque, ConT_EXt, por defecto, a los *entornos* generados a partir de los comandos de seccionado no los considera *grupos*. Aún así este procedimiento es el que Mark IV recomienda; muy posiblemente porque esta forma de establecer secciones exige que se indique expresamente dónde empieza y donde acaba cada sección, lo que facilita el que la estructura sea coherente, y, muy probablemente, tenga un mejor soporte para la salida XML y EPUB.

Cuando se usa `\startNombreSección` se admite, como argumento adicional entre corchetes, una o más variables cuyo valor podrá ser recuperado en otro punto del documento mediante el comando `\structureuservariable`.

El establecimiento de variables de usuario, permite usos muy avanzados de ConT_EXt en virtud de los cuales se podrían tomar decisiones relativas a si compilar o no un fragmento, o a de qué manera o con qué plantilla compilarlo, según el valor de cierta variable. Estas utilidades de ConT_EXt, sin embargo, exceden la materia que tengo pensado tratar en la presente introducción.

7.4 Formateo y configuración de las secciones y sus títulos

7.4.1 Los comandos `\setuphead` y `\setupheads`

Por defecto ConT_EXt asigna a cada nivel de seccionado ciertas características que afectan, fundamentalmente (aunque no sólo) al formato en el que se muestra el título en el cuerpo principal del documento; no así a la forma en que se muestra ese mismo título en el índice o en los encabezados o pies de página. Estas características las podemos cambiar con el comando `\setuphead` cuya sintaxis es:

`\setuphead[Secciones][Opciones]`

en donde

- **Secciones** se refiere al nombre de una o varias secciones (separadas por coma) que se verán afectadas por el comando. Puede ser
 - Alguna de las secciones predefinidas (`part`, `chapter`, `title`, etc.), en cuyo caso podemos referirnos a ellas, bien por su nombre, bien por su nivel. Para referirnos a ellas por su nivel se usa la palabra «`section-NumNivel`», donde *NumNivel* es el número de nivel de la sección de que se trate. Así, «`section-1`» equivale a «`part`», «`section-2`» equivale a «`chapter`», etc.
 - Algún tipo de sección definido por nosotros mismos. Véase, al respecto, en la [sección 7.5](#).
- **Opciones** son las opciones de configuración. Estas opciones son del tipo de asignación explícita de valor (`NombreOpción = valor`). El número de opciones admisibles es muy elevado (más de sesenta) y por ello las explicaré agrupándolas en categorías según su función. Debo señalar, no obstante, que en algunas de las opciones no he conseguido determinar para qué sirven o cómo se usan. De esas opciones no hablaré.

Antes he dicho que `\setuphead` afecta a las secciones que expresamente se le indiquen. Pero ello no significa que la modificación de una concreta sección no haya de afectar de ninguna manera a las demás salvo que se las haya mencionado expresamente en el comando. De hecho ocurre lo contrario: la modificación de una sección afecta a otras secciones *vinculadas* con ella, aunque no se haya explicitado así en el comando. La vinculación entre las distintas secciones es de dos tipos:

- Los comandos no numerados están vinculados con el correspondiente comando numerado del mismo nivel; de tal forma que un cambio en el aspecto del comando numerado, afectará al comando no numerado del mismo nivel; pero no al revés: el cambio en el comando no numerado no afecta al comando numerado. Es decir: si cambiamos, por ejemplo, el aspecto de «`chapter`» (nivel 2) cambiaremos también el aspecto de «`title`»; pero cambiando «`title`» no afectamos a «`chapter`».
- Los comandos están vinculados jerárquicamente, de tal modo que si cambiamos *ciertas características* en un nivel concreto, el cambio afectará a todos los niveles posteriores. Esto ocurre sólo con ciertas características. Por ejemplo, el color: si establecemos que las subsecciones se muestren en color rojo, también se mostrarán en color rojo las subsubsecciones, subsubsubsecciones, etc. Pero no pasa lo mismo con otras características, como, por ejemplo, el estilo de fuente.

Junto a `\setuphead` ConT_EXt dispone del comando `\setupheads` que afecta globalmente a todos los comandos de seccionado. De este comando dice la wiki de ConT_EXt que algunas personas han informado de que no funciona. De acuerdo con mis pruebas este comando sólo funciona en algunas opciones, pero no en otras. En particular no funciona con la opción «`style`», lo que es llamativo, pues el estilo de los títulos es lo que más probablemente se querrá cambiar de modo global para todos ellos. Pero sí funciona, según mis pruebas, con otras opciones como, por ejemplo, «`number`» o «`color`» y así, por ejemplo, `\setupheads[color=blue]` hará que todos los títulos de nuestro documento se impriman en color azul.

Como quiera que me da cierta pereza ir comprobando, opción a opción, si funciona o no con `\setupheads` (recordemos que hay más de sesenta opciones) en lo sucesivo me referiré sólo a `\setuphead`.

En fin: antes de entrar al examen de las concretas opciones, es preciso advertir algo que se dice en la wiki de ConT_EXt, aunque probablemente no se diga en el lugar adecuado: algunas opciones sólo funcionan si se ha usado para el comando la sintaxis `\startNombreSección`.

Esta información se contiene a propósito de `\setupheads`, pero no a propósito de `\setuphead` que es donde se explica el grueso de las opciones y donde, si sólo se debía decir en un lugar, me parece más razonable que se diga. La información, por otra parte, sólo menciona a la opción «`insidesection`», sin dejar claro si eso sucede también con alguna otra opción.

7.4.2 Las partes del título de una sección

Antes de entrar en las concretas opciones que permiten configurar la apariencia de los títulos, conviene empezar señalando que un título de sección puede llegar a tener hasta tres partes diferentes, las cuales ConT_EXt permite que se formateen conjuntamente o por separado. Estos elementos del título son los siguientes:

- **El título propiamente dicho**, es decir: el texto que lo compone. En principio este título siempre se muestra, salvo en las secciones de tipo «`part`» en las que, por defecto, no se muestra. La opción que controla que el título se muestre o no se muestre es «`placehead`» que puede asumir los valores «`yes`», «`no`», «`hidden`», «`empty`» o «`section`». El significado de los dos primeros está claro. Respecto del efecto de los restantes valores posibles para esta opción no estoy muy seguro.

Por lo tanto, si queremos que se muestre el título en las secciones del primer nivel, deberemos establecer:

```
\setuphead
[part]
[placehead=yes]
```



El título de ciertas secciones, como ya sabemos, se puede enviar automáticamente a los encabezados y al índice. Mediante las opciones `list` y `marking` de los comandos de seccionado podemos indicar un título alternativo que se envíe a esos lugares. También es posible, al escribir el título, usar los comandos `\nolist` o `\nomarking` para conseguir que ciertos fragmentos del título se sustituyan, en el índice o en el encabezado, por puntos suspensivos. Por ejemplo:

```
\chapter{Influencias \nomarking{en el siglo XXI} del impresionismo
        \nomarking{decimonónico}}
```

Escribirá en el encabezado “Influencias ... del impresionismo ...”.

- **La numeración.** Este elemento sólo existe en las secciones numeradas (`part`, `chapter`, `section`, `subsection`...), pero no en las no numeradas (`title`, `subject`, `subsubject`). En realidad que un determinado tipo de sección sea o no numerado depende de las opciones «`number`» e «`incrementnumber`» cuyos valores posibles son «`yes`» y «`no`». En las secciones numeradas ambas opciones se establecen en `yes` y en las no numeradas en `no`.

¿Por qué hay dos opciones para controlar una sola cosa? En realidad las dos opciones controlan dos cosas; una es si la sección es o no numerada (`incrementnumber`) y la otra si el número se muestra o no (`number`). Si para un tipo de sección se establece `incrementnumber=yes` y `number=no` obtendríamos una sección externamente no numerada pero internamente sí. Eso sería útil para incluir ese tipo de sección en los índices, ya que éstos sólo pueden incluir, de forma automática, secciones numeradas. Véase al respecto el apartado A de la [sección 8.1.7](#).

- **El rótulo o etiqueta del título.** En principio este elemento de los títulos está vacío. Pero podemos asociarle algún valor, en cuyo caso, antes del número y del título propiamente dicho, se imprimiría la etiqueta que hayamos asignado a dicho nivel. Por ejemplo, podemos hacer que antes de los capítulos se escriba la palabra «Capítulo», o que antes de las partes se escriba la palabra «Parte». Para ello no se usa `\setuphead` sino el comando `\setuplabeltext`. Este comando permite asignar un valor textual a las etiquetas de los distintos niveles de seccionado. Y así, por ejemplo, si queremos que en nuestro documento antes de los capítulos se escriba “Capítulo”, deberíamos establecer:

```
\setuplabeltext
[chapter=Capítulo~]
```

En el ejemplo, tras el nombre asignado, he incluido el carácter reservado «`~`» que inserta un espacio en blanco de no separación. Si no nos importa que, eventualmente, se inserte un salto de línea entre la etiqueta y el número, podemos escribir simplemente un espacio en blanco. Pero ese espacio en blanco es

importante; sin él el número se mostrará pegado a la etiqueta, y se verá, por ejemplo, «Capítulo1» en lugar de «Capítulo 1».

7.4.3 Control de la numeración (en las secciones numeradas)

Ya sabemos que hay secciones numeradas predefinidas (part, chapter, section...) y que el que un tipo concreto de sección sea o no numerado depende de las opciones «number» e «incrementnumber» que se establecen con `\setuphead`.

Por defecto la numeración de los distintos niveles es automática, salvo que se asigne a la opción «ownnumber» el valor de «yes». Cuando «ownnumber=yes» habrá que indicar el número asignado a cada comando. Eso se hace:

- Si el comando es invocado con la sintaxis clásica, añadiendo un argumento con el número antes del texto del título. Por ejemplo: `\chapter{13}{Título del capítulo}` generará un capítulo al que se le asigna manualmente el número 13.
- Si el comando es invocado con la sintaxis específica de ConT_EXt (`\TipoSección [Opciones]` o `\startTipoSección [Opciones]`), mediante la opción «ownnumber». Por ejemplo: `\chapter[title=Título del capítulo, ownnumber=13]`, generará un capítulo al que manualmente se le asigna el número 13.

Cuando ConT_EXt realiza automáticamente la numeración, se vale de contadores internos que almacenan el número de los distintos niveles; así hay un contador de partes, otro de capítulos, otro de secciones, etc. Cada vez que ConT_EXt encuentra un comando de seccionado realiza las siguientes acciones:

- Incrementa en «1» el contador asociado al nivel correspondiente a dicho comando.
- Pone a 0 los contadores asociados a todos los niveles inferiores al del comando en cuestión.

Es decir: cada vez que, por ejemplo, se encuentra un nuevo capítulo, se incrementa en 1 el contador de capítulos y se ponen a 0 los contadores de secciones, subsecciones, subsubsecciones, etc.; pero el contador de partes no se ve afectado.

Para alterar el número a partir del cual se debe empezar a contar, se usa el comando `\setupheadnumber` de la siguiente forma:

`\setupheadnumber[TipoSección][Número desde el que contar]`

Donde *Número desde el que contar* es el número a partir del cual se empezarán a contar las secciones del tipo que sea. Así, si *Número desde el que contar* es igual a cero, la primera sección sería 1; pero si es igual a 10, la primera sección sería igual a 11.

Este comando también permite alterar la pauta para el incremento automático; con lo que podemos, por ejemplo, conseguir que los capítulos o secciones se cuenten de dos en dos, o de tres en tres. Así, `\setupheadnumber[section][+5]` hará que las secciones se numeren de 5 en cinco; y `\setupheadnumber[chapter][14, +5]` hará que el primer capítulo empiece en el número 15 (14+1), el segundo lleve por número 20 (15+5), el tercero 25, etc.

Por defecto la numeración de las secciones se muestra con números arábigos, y se incluye la numeración de todos los niveles previos. Es decir: en un documento en el que haya partes, capítulos, secciones y subsecciones, en una concreta subsección se indicará a qué parte, capítulo y sección corresponde. Así la cuarta subsección de la segunda sección, del tercer capítulo de la primera parte será «1.3.2.4».

Las dos opciones fundamentales que controlan cómo se mostrará la numeración son:

- **conversion:** Esta opción controla el tipo de numeración que se utilizará. Admite numerosos valores dependiendo del tipo de numeración que deseemos:
 - **Numeración con números arábigos:** Es la numeración clásica: 1, 2, 3, ... Se obtiene con los valores `n`, `N` o `numbers`.
 - **Numeración con números romanos.** Admite tres modalidades:
 - ★ Números romanos en mayúsculas: `I`, `R`, `Romannumerals`.
 - ★ Números romanos en minúsculas: `i`, `r`, `romannumerals`.
 - ★ Números romanos en versalitas: `KR`, `RK`.
 - **Numeración con letras.** Admite también tres modalidades:
 - ★ Letras en mayúsculas: `A`, `Character`
 - ★ Letras en minúsculas: `a`, `character`
 - ★ Letras en versalitas: `AK`, `KA`
 - **Numeración con palabras.** Es decir: se escribe la palabra que designa al número y así, por ejemplo, «3» se convierte en «tres». Admite dos modalidades:
 - ★ Palabras con la primera letra en mayúscula: `Words`.
 - ★ Palabras totalmente en minúsculas: `words`.
 - **Numeración con símbolos:** La numeración basada en símbolos utiliza distintos juegos de símbolos en los que se asigna a cada símbolo un valor numérico. Como los juegos de símbolos que ConT_EXt utiliza tienen un número muy limitado de ellos, sólo es adecuado usar este tipo de numeración cuando el máximo número que se pretende alcanzar no es demasiado alto.

ConT_EXt prevé cuatro distintos juegos de símbolos denominados, respectivamente, **set 0**, **set 1**, **set 2** y **set 3**. A continuación se muestran los símbolos que cada uno de estos conjuntos utiliza para las numeraciones. Obsérvese que el número máximo que se puede alcanzar es de 9 en **set 0** y **set 1** y de 12 en **set 2** y **set 3**:

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

- **sectionsegments**: Esta opción permite controlar si se mostrará o no la numeración de los niveles precedentes. Mediante ella podemos indicar qué niveles previos se mostrarán. Ello se hace identificando el nivel inicial y el nivel final que se mostrará. La identificación del nivel la podemos hacer por su número (part=1, chapter=2, section=3, etc.), o por su nombre (part, chapter, section, etc.). Así, por ejemplo, «**sectionsegments=2:3**» indica que se debe mostrar la numeración del capítulo y la de la sección. Es exactamente lo mismo que decir «**sectionsegments=chapter:section**». Si queremos indicar que se muestren todos los números a partir de cierto nivel podemos usar, como valor de «**optionsegments**» *NivelInicial:all*, o *NivelInicial:**. Por ejemplo, «**sectionsegments=3:***» indica que se muestre la numeración a partir del nivel 3 (section).

Así, por ejemplo, imaginemos que queremos que las partes se numeren con romanos en mayúsculas; los capítulos con arábigos, pero sin incluir el número de la parte a que pertenecen; las secciones y subsecciones con números arábigos incluyendo los números de capítulo y sección, y las subsubsecciones con letras mayúsculas. Deberíamos escribir lo siguiente:

```
\setuphead[part][conversion=I]
\setuphead[chapter][conversion=n, sectionsegments=2]
\setuphead[section][conversion=n, sectionsegments=2:3]
\setuphead[subsection][conversion=n, sectionsgments=2:4]
\setuphead[subsubsection][conversion=A, sectionsegments=5]
```

7.4.4 Estilo y color de los títulos

Para controlar estilo y color disponemos de las siguientes opciones:

- **El estilo** se controla con las opciones «**style**», «**numberstyle**» y «**textstyle**» según queramos afectar a todo el título, sólo a la numeración, o sólo al texto. Mediante cualquiera de estas opciones podemos incluir comandos que

afecten a la fuente; a saber: fuente concreta, estilo (roman, sans serif o typewriter), alternativa (cursiva, negrita, inclinada...) y tamaño. Si sólo queremos indicar una característica de estilo podemos hacerlo, bien usando el nombre de la misma (por ejemplo, «bold» para la negrita), bien indicando su abreviatura («bf»), bien el comando que la genera (`\bf`, en el caso de la negrita). Si queremos indicar simultáneamente varias características, debemos hacerlo mediante los comandos que las generan, escribiéndolos uno tras otro. Téngase en cuenta, por otro lado, que si indicamos sólo una característica, el resto de las características del estilo se establecerán automáticamente con los valores por defecto del documento, razón esta por la que raramente es aconsejable establecer sólo una característica de estilo.

- **El color** se establece mediante las opciones «color», «numbercolor» y «textcolor» según queramos configurar el color de todo el título, o sólo el color de la numeración o el del texto. El color que aquí se indique puede ser alguno de los colores predefinidos en ConT_EXt o algún color definido por nosotros mismos al que le hayamos previamente asignado un nombre. No puede, sin embargo, usarse aquí directamente un comando de definición de color.

Junto a estas seis opciones, para establecer algunas características más sofisticadas aún se dispone de otras cinco opciones adicionales con las que podemos hacer prácticamente lo que deseemos. Se trata de «command», «numbercommand», «textcommand», «deepnumbercommand» y «deeptextcommand». Empecemos por explicar los tres primeros:

- **command** indica un comando que recibirá dos argumentos, el número y el título de la sección. Puede ser un comando normal de ConT_EXt o algún comando que hayamos definido nosotros mismos.
- **numbercommand** es similar a «command», pero este comando sólo recibirá un argumento con el número de la sección.
- **textcommand** es también similar a «command», pero sólo recibirá un argumento con el texto del título.

Estas tres opciones nos permiten hacer prácticamente lo que queramos. Por ejemplo, si quiero que las secciones se muestren alineadas a la derecha, encerradas en un marco y con un salto de línea entre el número y el texto, me bastará con crear un comando que haga eso, y luego indicar dicho comando como valor de la opción «command». Eso lo conseguiría con las siguientes líneas:

```

\define[2]\AlinearSección
  {\framed[frame=on, width=broad, align=flushright]{#1\#2}}

\setuphead
  [section]
  [command=\AlinearSección]

```

Cuando establecemos simultáneamente la opción «**command**» y la opción «**style**», el comando se aplica al título con su estilo. Es decir, si, por ejemplo, hemos establecido «**textstyle**=\em», y «**textcommand**=\WORD», el comando \WORD (que pone en mayúsculas el texto que reciba como argumento) se aplicará al título con su estilo, o sea: \WORD{\em Texto del título}. Si queremos que se haga al revés, es decir, que el estilo se aplique al contenido del título una vez se le haya aplicado el comando, debemos usar, en lugar de las opciones «**textcommand**» y «**numbercommand**» las opciones «**deeptextcommand**» y «**deepnumbercommand**». Esto, en el ejemplo que hemos puesto antes, generaría «{\em\WORD{Texto del título}}».

En la mayor parte de los casos no habrá ninguna diferencia entre hacerlo de un modo o del otro. Pero en algún caso puede que sí la haya.

7.4.5 Ubicación del número y del texto del título

La opción «**alternative**» controla simultáneamente dos cosas: la ubicación de la numeración respecto del texto del título, y la ubicación del título propiamente dicho (incluyendo número y texto) respecto de la página en la que se muestra y el contenido de la sección. Son dos cosas distintas, pero, al estar ambas regidas por la misma opción, se controlan simultáneamente.

La ubicación del título en relación con la página y con el primer párrafo del contenido de la sección, se controla mediante los siguientes valores posibles de «**alternative**»:

- **text**: El título de la sección se integra con el primer párrafo del contenido de la misma. El efecto es similar al que se produce en L^AT_EX con \paragraph y \subparagraph.
- **paragraph**: El título de la sección será un párrafo independiente.
- **normal**: El título de la sección se ubicará en el lugar que por defecto tenga previsto ConT_EXt para el tipo de sección concreta de que se trate. Normalmente es «**paragraph**».
- **middle**: El título se escribe como párrafo autónomo, centrado. Si se trata de un comando numerado, el número y el texto se separan en líneas diferentes, ambas centradas.

Un efecto parecido al que se obtiene con «`alternative=middle`» se obtiene con la opción «`align`» que controla la alineación del título. Puede asumir los valores «`left`», «`middle`» o «`flushright`». Pero si centramos el título con esta opción, el número y el texto aparecerán en la misma línea.

- **margin***text*: Esta opción provoca que todo el título (numeración y texto) se imprima en el espacio reservado para el margen.

La ubicación del número en relación con el texto del título se indica mediante los siguientes valores posibles de «`alternative`»:

- **margin/inmargin**: El título compone un párrafo independiente. La numeración se escribe en el espacio reservado para el margen. No he llegado a averiguar la diferencia entre usar «`margin`» y usar «`inmargin`».
- **reverse**: El título compone un párrafo independiente, pero se invierte el orden normal, y se imprime en primer lugar el texto y después el número.
- **top/bottom**: En títulos cuyo texto ocupe más de una línea, estas dos opciones controlan, respectivamente, si la numeración se alineará con la primera línea del título o con la última.

7.4.6 Comandos o acciones a realizar antes o después de imprimir el título

Es posible indicar uno o varios comandos que se ejecuten antes de escribir el título (opción «`before`») o después (opción «`after`»). Estas opciones son muy utilizadas para marcar visualmente al título. Por ejemplo: si queremos añadir un mayor espacio vertical entre el título y el texto que le precede, mediante «`before=\blank`» añadiríamos una línea en blanco. Para añadir un mayor espacio podríamos escribir, «`before={\blank[3*big]}`», caso este en el que hemos rodeado con llaves el valor de la opción para evitar un error. También podríamos indicar visualmente la separación del texto previo y del siguiente con «`before=\hairline`, `after=\hairline`», lo que dibujaría una línea horizontal antes y después del título.



Muy parecidas a las opciones «`before`» y «`after`» son «`commandbefore`» y «`commandafter`». De acuerdo con mis pruebas deduzco que la diferencia está en que las dos primeras ejecutan acciones antes y después de empezar a escribir el título, mientras que las segundas se refieren a comandos que se ejecutarán antes y después de escribir *el texto del título*.

Si lo que queremos es insertar un salto de página antes del título, hay que usar la opción «`page`» que admite, entre otros valores los de “yes” para insertar un salto de página, “left” para insertar tantos saltos de página como sea preciso para asegurarse de que el título empieza en una página par, “right” que se asegura de

que el título empiece una página impar o “no” si lo que queremos es desactivar el salto de página forzado. Esta opción, por otra parte, para niveles inferiores a «chapter» sólo funcionará siempre si la opción «continue=no», en otro caso no funcionará si la sección, subsección o comando que sea está en la primera página de un capítulo.

Por defecto en ConT_EXt los capítulos inician una nueva página. Si se establece que las secciones también inicien página, se plantea el problema de qué hacer con la primera sección de un capítulo que, tal vez, se encuentre al principio del capítulo: si esa sección inicia también un salto de página, tendríamos que la página que abre el capítulo sólo contendría el título de éste, lo que no es muy estético. Para ello se establece la opción «continue» cuyo nombre, por cierto, no me parece particularmente claro: Si «continue=yes», el salto de página no se aplicará en las secciones que se encuentren en la primera página de un capítulo. Si «continue=no» el salto de página se aplicará siempre.

Si en lugar de comandos de seccionado usamos entornos de seccionado (`\start ... \stop`), disponemos también de la opción «insidesection» mediante la que podemos indicar uno o varios comandos que se ejecutarán una vez que se haya escrito el título y estemos ya dentro de la sección. Esta opción nos permitiría, por ejemplo, asegurarnos de que inmediatamente después de iniciar un capítulo, se escribirá, automáticamente, un índice del mismo («insidesection=\placecontent»)

7.4.7 Otras características configurables

Además de las ya vistas, mediante `\setuphead` podemos configurar las siguientes características adicionales:

- **Interlineado.** Se controla mediante la opción «interlinespace» que recibe como valor el nombre de un comando de interlineado creado previamente con `\defineinterlinespace` y configurado con `\setupinterlinespace`.
- **Alineación.** La opción «align» afecta a la alineación del párrafo que contiene al título. Puede tener, entre otros, los siguientes valores: «flushleft» (izquierda), «flushright» (derecha), «middle» (centrado), «inner» (margen interior) y «outer» (margen exterior).
- **Margen.** Mediante la opción «margin» podemos fijar manualmente el margen del título.
- **Indentación del primer párrafo.** El valor de la opción «indentnext» (que puede ser “yes”, “no” o “auto”) controla si se indentará o no la primera línea del primer párrafo de la sección. Que se deba indentar o no (en un documento en el que la primera línea de los párrafos, en general, se indenta) es cuestión de gustos.

- **Anchura.** Por defecto los títulos ocupan la anchura que necesiten salvo que esta sea superior a la anchura de la línea, en cuyo caso el título ocupará más de una línea. Pero mediante la opción «`width`» podemos asignar una anchura concreta para el título. Las opciones «`numberwidth`» y «`textwidth`» asignan, respectivamente, la anchura de la numeración, o del texto del título.
- **Separación entre el número y el texto.** Las opciones «`distance`» y «`text-distance`» permiten controlar la separación entre la numeración y el texto del mismo.
- **Estilo de los encabezados y pies de página de la sección.** Para ello se usan las opciones «`header`» y «`footer`»

7.4.8 Otras opciones de `\setuphead`



Con las opciones ya vistas, se verá que las posibilidades de configuración de los títulos de las secciones son casi ilimitadas. Pero `\setuphead` tiene todavía cerca de treinta opciones que no he mencionado. La mayoría porque no he descubierto para qué sirven o cómo se usan, unas pocas, porque su explicación me obligaría a introducirme en aspectos que no tengo previsto tratar en esta introducción.

7.5 Definición de nuevos comandos de seccionado

Podemos definir nuestros propios comandos de seccionado mediante `\definehead` cuya sintaxis es:

```
\definehead [NombreComando] [Modelo] [Configuración]
```

donde

- **NombreComando** representa el nombre que tendrá el nuevo comando de seccionado.
- **Modelo** es el nombre de un comando de seccionado ya existente que se usará como modelo del que el nuevo comando heredará inicialmente todas sus características.

En realidad el nuevo comando hereda del modelo mucho más que las características iniciales: se convierte en una especie de instancia personalizada del modelo, pero comparte con él, por ejemplo, el contador interno que recoge la numeración.

- **Configuración** es la configuración personalizada de nuestro nuevo comando. Aquí se pueden usar exactamente las mismas opciones que en `\setuphead`.

No es necesario configurar el nuevo comando en el momento de su creación. Puede hacerse más tarde mediante `\setuphead` y, de hecho, en los ejemplos que se ponen en los manuales de ConT_EXt así como en su wiki, eso parece ser que es lo normal.

7.6 Macroestructura del documento

Capítulos, secciones, subsecciones, títulos..., estructuran al documento; lo sistematizan. Pero junto con la estructura compuesta por este tipo de comandos, en ciertos libros impresos, sobre todo en los procedentes del mundo académico existe una especie de *macroordenación* del material del libro atendiendo no a su contenido sino a la función que en el libro desempeña cada una de estas grandes partes. Así se diferencia entre:

- La parte inicial del documento en donde se contiene la página del título, la de agradecimientos, la dedicatoria, el índice sumario, tal vez el prefacio, la presentación, etc.
- El cuerpo principal del documento, en donde se contiene el texto fundamental del mismo, dividido, en partes, capítulos, secciones, subsecciones, etc. Esta parte suele ser la más extensa e importante.
- El material adicional constituido por los apéndices o anexos que desarrollen o ejemplifiquen alguna cuestión tratada en el cuerpo principal, o aporten documentación adicional no escrita por el autor del cuerpo principal, etc.
- La parte final del documento en donde podemos encontrar la bibliografía, los índices que no se hayan ubicado en la parte inicial, algún glosario, etc.

Podemos delimitar, en el fichero fuente, cada una de estas partes, mediante los entornos que se recogen en la [tabla 7.2](#).

Parte del documento	Comando
Parte inicial	<code>\startfrontmatter [Opciones] ... \stopfrontmatter</code>
Cuerpo principal	<code>\startbodymatter [Opciones] ... \stopbodymatter</code>
Apéndices	<code>\startappendices [Opciones] ... \stopappendices</code>
Parte final	<code>\startbackmatter [Opciones] ... \stopbackmatter</code>

Tabla 7.2 Entornos que reflejan la macroestructura del documento

Los cuatro entornos admiten las mismas cuatro opciones: «**page**», «**before**», «**after**» y «**number**», y sus valores y utilidad son similares a los que estas mismas opciones tienen en `\setuphead` (véase la [sección 7.4](#)), si bien hay que advertir que aquí la opción «**number=no**» eliminará la numeración de todos los comandos de seccionado que se encuentren en el interior del entorno.

Incluir en nuestro documento alguno de estos grandes apartados, sólo tiene sentido si es para establecer algún tipo de diferenciación entre ellos. Tal vez el encabezado de las páginas en *frontmatter*, o quizás la numeración de páginas. Esta configuración de cada uno de estos bloques se consigue mediante `\setupsectionblock` cuya sintaxis es:

```
\setupsectionblock[Nombre del bloque] [Opciones]
```

donde *Nombre del bloque* puede ser `frontpart`, `bodypart`, `appendix` o `backpart` y las opciones pueden ser las mismas que se acaban de mencionar: «`page`», «`number`», «`before`» y «`after`». Así, por ejemplo, para conseguir que en *frontmatter* las páginas se numeren con números romanos, deberíamos escribir en el preámbulo de nuestro documento:

```
\setupsectionblock
[frontpart]
[
  before={\setuppagenumbering[conversion=Romannumerals]}
]
```

La configuración por defecto de ConT_EXt para estos cuatro bloques implica:

- Los cuatro bloques inician página nueva.
- En cada uno de estos bloques, cambia la numeración de las secciones:
 - En `frontmatter` y en `backmatter` por defecto se anula la numeración de todas las secciones numeradas.
 - En `bodymatter` los capítulos se numeran con números arábigos.
 - En `appendices` los capítulos se numeran con letras mayúsculas.

También es posible crear nuevos bloques de sección mediante `\definesectionblock`.

Capítulo 8

Índices

Sumario: **8.1 Índices de contenido;** 8.1.1 Visión general de los índices de contenido; 8.1.2 Índices totalmente automáticos con título; 8.1.3 Índices automáticos sin título; 8.1.4 Elementos a incorporar al índice: La opción `criterion`; 8.1.5 Diseño del índice: la opción `alternative`; 8.1.6 Formato de las entradas del índice; 8.1.7 Ajustes manuales de los índices; A Incluir en el índice secciones no numeradas; B Agregar manualmente entradas al índice; C Excluir del índice una concreta sección perteneciente a un tipo de secciones que están incluidas en el índice; D Texto del título de la sección diferente en el índice y en el cuerpo del documento; **8.2 Listas, listas combinadas e índices creados a partir de una lista;** 8.2.1 Listas en ConTeXt; 8.2.2 Índices de imágenes, tablas y otros elementos; 8.2.3 Listas combinadas; **8.3 Índices terminológicos;** 8.3.1 Generación del índice; A La previa definición de las entradas del índice y el marcado de los puntos del fichero fuente que se refieren a ellas; B Generación del índice propiamente dicho; 8.3.2 Formateo del índice terminológico; 8.3.3 Crear otros índices terminológicos;

Un elemento global del documento es el índice. Existen distintos tipos de índices: de contenido, también llamados sistemáticos o generales; terminológicos, llamados a veces «alfabéticos» o «índices de materias», onomásticos, de imágenes, de tablas, etc.

Nos ocuparemos de todos ellos en el presente capítulo; aunque la atención se centra principalmente en los índices de contenido porque son, en la práctica, los más importantes: casi todos los documentos incluyen uno, mientras que los otros tipos de índice sólo existen en ciertos documentos.

La terminología tradicional española reservaba el término «índice» para el índice de contenido o sistemático (véase, por ejemplo, el significado 2º de la voz «Índice» en el Diccionario de la Real Academia de la Lengua). Pero en inglés parece ser que el término «índice» se asoció principalmente a lo que en España llamamos índice analítico o terminológico. La preponderancia actual del inglés, y, sobre todo, de herramientas de composición de textos nacidas en el ámbito cultural anglosajón, lleva a que hoy día exista en España (y no sé si también en el resto de los países de habla hispana) una cierta vacilación sobre el uso del término «índice», siendo cada vez más habituales los libros que al índice de contenido lo llaman «Tabla de contenido» o simplemente «Contenido». Personalmente entiendo que, a la vista del significado del verbo *indexar* (o *indizar*). Todas estas construcciones merecen la denominación de índice, y para evitar confusiones procuraré aclarar siempre a qué tipo de índice me refiero en cada caso.

En cuanto a la terminología de ConTeXt, se reserva el término «índice» (*index*) para los comandos vinculados con el índice terminológico.

Todos los índices, cualquiera que sea su tipo, implican una aplicación particular del mecanismo de las remisiones internas cuya explicación se incluye en la [sección 9.2](#).

8.1 Índices de contenido

8.1.1 Visión general de los índices de contenido

En el capítulo anterior se han examinado los comandos que permiten ir estableciendo la estructura de un documento conforme éste va siendo escrito. La presente sección se centra en los índices de contenido, a los que a veces se llama «índice general», «tabla de contenido», «sumario», o «índice» a secas; los cuales constituyen un elemento del documento en el que, en cierto modo, se *refleja* su estructura. Resultan muy útiles para hacerse una idea de conjunto del documento (lo que ayuda a contextualizar su información) y para buscar el punto exacto en el que se encuentra cierto pasaje concreto. En libros de estructura muy compleja, con numerosas secciones y subsecciones de varios niveles de profundidad, estas dos utilidades de los índices parecen exigir un tipo de índice diferente ya que un índice poco detallado (tal vez con solo los dos o tres primeros niveles de seccionado) ayuda mucho a hacerse una idea global del contenido del documento, pero no es muy útil para localizar un concreto pasaje; a diferencia de un índice muy detallado en el cual, sin embargo, es fácil que los árboles no dejen ver el bosque y que se pierda la visión de conjunto del documento. Por ello en ocasiones los libros con estructura especialmente compleja incluyen más de un índice: uno no demasiado detallado al principio que muestre los principales apartados, y un índice detallado del contenido de cada capítulo al principio del mismo; o un índice completo al final.

Todos estos índices pueden ser generados por ConT_EXt de forma automática con relativa facilidad. Podemos:

- Generar índices completos e índices parciales en cualquier punto del documento.
- Decidir el contenido de cada índice.
- Configurar hasta el más mínimo detalle su apariencia.
- Incluir hiperenlaces en el índice que permitan saltar directamente desde él hasta la sección de que se trate.

De hecho esta última utilidad se incluye por defecto en todos los índices siempre que en el documento se haya activado la función de interactividad. Véase, al respecto, la [sección 9.3](#).

La explicación que de esta materia se hace en el manual de referencia de ConT_EXt es, en mi opinión, algo confusa, lo que creo que se debe a que se introduce de

golpe mucha información. El mecanismo de construcción de los índices de ConT_EXt incluye numerosas piezas; y es difícil que un texto que las intente explicar simultáneamente resulte claro. Sobre todo para el lector novato. Por el contrario la explicación de la wiki, prácticamente se limita a poner ejemplos: muy útil para aprender *trucos* pero insuficiente —pienso— para comprender el mecanismo de funcionamiento. Por ello la estrategia de explicación que he decidido abordar en esta introducción parte de empezar asumiendo una afirmación que no es estrictamente cierta (o que no es totalmente cierta): que en ConT_EXt existe algo llamado *índice de contenido*. A partir de ahí se explican los comandos *normales* para generar los índices, y cuando se conocen bien estos comandos y su posible configuración, creo que es el momento de introducir —ya a nivel más teórico que práctico— la información sobre las piezas del mecanismo que hasta entonces se han omitido. El conocimiento de estas *piezas* adicionales permite crear índices mucho más personalizados que los que podríamos llamar *índices normales*, creados con los comandos explicados hasta entonces; pero, en la mayor parte de los casos, no necesitaremos hacerlo.

8.1.2 Índices totalmente automáticos con título

Los comandos básicos para generar un índice generado automáticamente a partir de las secciones numeradas del documento (`part`, `chapter`, `section`, etc.) son `\completecontent` y `\placecontent`. La diferencia principal entre ambos comandos se encuentra en que el primero, añade un *título* al índice; para lo cual inserta inmediatamente antes del índice un *capítulo no numerado* titulado por defecto «Contenido» (en documentos en español).

Por tanto `\completecontent`:

- Inserta, en el punto en el que se encuentra, un nuevo capítulo no numerado titulado «Contenido».

Recordemos que en ConT_EXt el comando que se utiliza para generar una sección no numerada del mismo nivel que los capítulos es `\title` (véase la [sección 7.2](#)). Por lo tanto en realidad `\completecontent` no inserta un *Capítulo* (`\chapter`), sino un *Título* (`\title`). No lo he dicho así porque pienso que puede ser confuso para el lector usar aquí los nombres de los comandos de sección no numerados, dado que el término *Título* en nuestro idioma tiene un sentido muy amplio y es fácil que el lector no lo identifique con el concreto nivel de seccionado a que nos estamos refiriendo.

- Dicho *capítulo* (en realidad `\title`) se formatea exactamente igual que el resto de los capítulos no numerados del documento; lo que por defecto incluye un salto de página.
- Inmediatamente detrás del título del índice (que para documentos en español es por defecto, como ya se ha dicho, «Contenido»), se imprime el índice.

En principio el índice que se genera es *completo*, tal y como cabría deducir del nombre del comando que lo genera (`\completecontent`). Pero, de un lado, podemos limitar el nivel de profundidad del índice tal y como se explica en la [sección 8.1.3](#) y, de otro, como este comando es *sensible* al lugar del fichero fuente en el que se encuentra (véase lo que más adelante se dice sobre `\placecontent`), si `\completecontent` no se encuentra al principio del documento es posible que el índice generado no sea completo; y en algunos puntos del fichero fuente es posible incluso que el comando sea aparentemente ignorado. Si ocurriera eso la solución está en invocar al comando con la opción «`criterium=all`». Sobre esta opción véase, también, la [sección 8.1.3](#).

Para cambiar el título asignado por defecto al índice se usa el comando `\setupheadtext` cuya sintaxis es

```
\setupheadtext[Idioma][Elemento=Nombre]
```

Donde *Idioma* es opcional y se refiere al identificador del idioma usado por ConTeXt (véase la [sección 10.5](#)), *Elemento* se refiere al elemento cuya denominación queremos cambiar («`content`» en el caso del índice de contenido) y *Nombre* es el nombre o título que queremos asignar a nuestro índice. Por ejemplo

```
\setupheadtext[es][content=Sumario]
```

hará que el índice generado por `\completecontent` se titule «Sumario» en lugar de «Contenido».

Por lo demás `\completecontent` admite las mismas opciones de configuración que `\placecontent`, a cuya explicación (el próximo epígrafe) me remito.

8.1.3 Índices automáticos sin título

El comando general para insertar índices sin título, generados automáticamente a partir de los comandos de seccionado del documento es `\placecontent`, cuya sintaxis es:

```
\placecontent[Opciones]
```

En principio el índice recogerá absolutamente todas las secciones numeradas; aunque podemos limitar su nivel de profundidad con el comando `\setupcombinedlist` (del que se hablará más adelante). Y así, por ejemplo:

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

limitará el contenido del índice a capítulos y secciones.

Una peculiaridad de este comando es la de que resulta sensible a su situación en el fichero fuente. Esto es muy fácil de explicar con unos pocos ejemplos, pero mucho

más difícil si lo que se quiere es precisar exactamente cómo funciona el comando y qué epígrafes se incluyen en el índice en cada caso. Empecemos pues, por los ejemplos:

- `\placecontent` ubicado al principio del documento, antes del primer comando de seccionado (parte, capítulo o sección, según los casos) generará un índice completo.

No estoy en realidad seguro de que el índice generado por defecto sea *completo*, creo que sí incluye suficientes niveles de seccionado para que sea completo en la mayor parte de los casos; pero sospecho que no irá más allá del octavo nivel de particionado. En todo caso, como ya se ha dicho antes, podemos ajustar el nivel de seccionado al que llega el índice con

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

- Por el contrario, este mismo comando ubicado en el interior de una parte, capítulo o sección, generará exclusivamente un índice del contenido de dicho elemento, es decir: capítulos, secciones y demás niveles inferiores de seccionado de una parte concreta, o secciones (y demás niveles) de un capítulo concreto, o subsecciones de una sección concreta.

En cuanto a la explicación técnica y detallada, para comprender bien el funcionamiento por defecto de `\placecontent` es imprescindible recordar que las distintas secciones son, en realidad, para ConT_EXt Mark IV, *entornos* que empiezan con `\startTipoDeSección` y terminan con `\stopTipoDeSección` y que pueden contener dentro de sí otros comandos de seccionado de nivel inferior. Pues bien: teniendo eso en cuenta, podemos decir que `\placecontent` genera por defecto un índice que sólo incluirá:

- Elementos que pertenezcan al *entorno* (nivel de seccionado) en el que se ubique el comando. Es decir: el comando ubicado en un capítulo, no incluirá secciones o subsecciones de otros capítulos.
- Elementos que tengan un nivel de seccionado inferior al nivel correspondiente al punto en el que se encuentra el comando. O sea: si el comando está en un capítulo, sólo se incluyen secciones, subsecciones y demás niveles inferiores; pero si el comando está en una sección, se partirá para hacer el índice del nivel subsección.

Además para que el índice se genere, se exige que `\placecontent` se encuentre *antes* de la primera sección del capítulo en el que se ubique, o antes de la primera subsección de la sección en que se ubique, etc.

No estoy muy seguro de haber sido claro en la explicación anterior. Tal vez con un ejemplo algo más detallado que los anteriores se entienda mejor lo que quiero decir: Imaginemos la siguiente estructura de un documento:

- Capítulo 1
 - Sección 1.1
 - Sección 1.2
 - ★ Subsección 1.2.1
 - ★ Subsección 1.2.2
 - ★ Subsección 1.2.3
 - Sección 1.3
 - Sección 1.4
- Capítulo 2

Pues bien: `\placecontent` ubicado antes del capítulo 1 generará un índice completo, similar al generado por `\completecontent` aunque sin título. Pero si el comando se ubica dentro del capítulo 1 y antes de la sección 1.1, el índice será sólo del capítulo; y si se ubica al principio de la sección 1.2., el índice recogerá sólo el contenido de la misma. Pero si el comando se ubica, por ejemplo, entre las secciones 1.1 y 1.2 será ignorado. También será ignorado si se ubica al final de una sección, o al final del documento.

Todo ello, claro está, se refiere sólo al caso de que el comando no incluya opciones. En particular, la opción `criterium` alterará ese comportamiento predeterminado.

De las opciones admitidas por `\placecontent` sólo explicaré dos que son, las más importantes para configurar el índice, y, además, las únicas que están documentadas (parcialmente) en el manual de referencia de ConT_EXt. La opción `criterium`, que afecta al contenido del índice en relación con el lugar del fichero fuente en que se encuentre el comando; y la opción `alternative`, que afecta al diseño general del índice que se generará.

8.1.4 Elementos a incorporar al índice: La opción `criterium`

Más arriba se ha explicado el funcionamiento por defecto de `\placecontent` en relación con la posición del comando en el fichero fuente. La opción `criterium` altera este funcionamiento. Puede asumir, entre otros, los siguientes valores:

- `all`: El índice será completo, con independencia del lugar del fichero fuente en que se encuentre el comando.
- `previous`: El índice sólo incluirá los comandos de seccionado (del nivel en que nos encontremos) anteriores a `\placecontent`. Esta opción está pensada para índices que se escriban al final del documento o sección de que se trate.
- `part`, `chapter`, `section`, `subsection`...: Implica que el índice debe ceñirse exclusivamente al nivel de seccionado que se le indica.
- `component`: En proyectos multifichero (véase la [sección 4.6](#)), generará solamente el índice correspondiente al elemento *component* en que se ubique el comando `\placecontent` o `\completecontent`.

8.1.5 Diseño del índice: la opción `alternative`

La opción `alternative` controla el diseño general del índice. Sus principales valores se recogen en la [tabla 8.1](#).

<code>alternative</code>	Contenido de las entradas del índice	Observaciones
a	Número – Título – Página	Una línea por entrada
b	Número – Título – Espacios – Página	Una línea por entrada
c	Número – Título – Puntos – Página	Una línea por entrada
d	Número – Título – Página	Índice continuo
e	Título	Enmarcado
f	Título	Alineado a la izquierda, a la derecha o centrado
g	Título	Centrado

Tabla 8.1 Formas de formatear el índice de contenido

Los primeros cuatro valores de `alternative` ofrecen toda la información de cada sección (su número, su título y el número de página en donde empieza), y por ello son adecuados tanto para documentos en papel como para documentos electrónicos. Las tres últimas alternativas sólo informan del título, por lo que sólo son adecuadas para documentos electrónicos en los que no es preciso saber el número de página en el que empieza una sección, siempre que el índice incluya un hipervínculo a ella, cosa que en ConT_EXt sucede por defecto.

Por lo demás considero que para apreciar verdaderamente las diferencias entre las distintas alternativas, lo mejor es que el lector genere un documento de prueba en el que pueda analizarlas con detenimiento.

8.1.6 Formato de las entradas del índice

Hemos visto que la opción `alternative` de `\placecontent` o `\completecontent` permite controlar el *diseño* general del índice, es decir: qué información se mostrará de cada epígrafe, y si habrá o no saltos de líneas que separen a los distintos epígrafes. El ajuste fino de cada entrada del índice se logra mediante el comando `\setuplist` cuya sintaxis es la siguiente:

`\setuplist[Elemento][Configuración]`

donde *Elemento* se refiere a un tipo concreto de sección. Puede ser: `part`, `chapter`, `section`, etc. Podemos también configurar simultáneamente más de un elemento, separándolos mediante comas. *Configuración* incluye hasta 54 posibilidades, muchas de ellas, como de costumbre, sin documentar expresamente; lo que por otra parte no impide que las sí documentadas, o aquellas cuya denominación es lo suficientemente explícita, permitan realizar un ajuste completo del índice.

A continuación explicaré las opciones más importantes, agrupándolas según su utilidad, pero antes de entrar en ello recordemos que una entrada del índice, dependiendo del valor de `alternative` puede llegar a tener hasta tres componentes distintos: El número de sección, el título de la sección, y el número de página. Las opciones de configuración permiten configurar los distintos componentes de modo global o separadamente:

- *Inclusión (o no) de los distintos componentes:* Si hemos elegido una alternativa que incluya, además del título, el número de sección y el número de página (alternativas «a», «b», «c» o «d»), las opciones `headnumber=no` o `pagenumber=no` hacen que para el nivel concreto que estemos configurando no se muestre, el número de sección (`headnumber`) o el número de página (`pagenumber`).
- *Color y estilo* Ya sabemos que la entrada que genera una concreta sección en el índice puede llegar a tener (dependiendo de la alternativa) hasta tres componentes distintos: Número de sección, título y número de página. Podemos indicar conjuntamente el estilo y el color para los tres componentes mediante las opciones `style` y `color`, o hacerlo individualmente para cada uno de los componentes mediante `numberstyle`, `textstyle` o `pagestyle` (para el estilo) y `numbercolor`, `textcolor` o `pagecolor` para el color.

Para controlar el aspecto de cada entrada, además de los comandos de estilo propiamente dicho, podemos aplicar algún comando a toda la entrada o a alguno de sus distintos elementos. Para ello estás las opciones `command`, `numbercommand`, `pagecommand` y `textcommand`. El comando indicado aquí puede ser un comando estándar de ConTeXt o un comando de nuestra propia creación. Al comando indicado por la opción `command` se le pasarán como argumentos el número de sección, el texto del título y el número de página, mientras que a `numbercommand` se le pasará como argumento el número de sección, a `textcommand` se le pasará como argumento el título de la sección y a `pagecommand` se le pasará como argumento el número de página. Así, por ejemplo, la siguiente sentencia hará que el título de las secciones se escriba en (falsas) versalitas:

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

- *Separación de los demás elementos del índice:* Las opciones `before` y `after` permiten indicar los comandos que se ejecutarán antes (`before`) y después (`after`) de escribir la entrada del índice. Normalmente estos comandos se utilizan para establecer, bien el espacio de separación, bien algún elemento separador de las entradas anteriores y posteriores.
- *Sangrado del elemento:* se establece con la opción `margin` que permite establecer la cantidad de sangrado izquierdo que tendrán las entradas del nivel que estamos configurando.

- *Hiperenlaces embebidos en el índice*: Por defecto las entradas del índice incluyen un hiperenlace a la página del documento en el que empieza la sección de que se trate. Mediante la opción `interaction` podemos desactivar ese funcionamiento (`interaction=no`) o limitar la parte de la entrada del índice en la que estará el hiperenlace que puede ser el número de la sección (`interaction=number` o `interaction=sectionnumber`), el título de la sección (`interaction=text` o `interaction=title`) o el número de página (`interaction=page` o `interaction=pagenumber`).
- *Otros aspectos*:
 - `width`: Especifica la distancia de separación entre el número y el título de la sección. Puede ser una dimensión, o la palabra clave `fit` que establece la anchura exacta del número de la sección.
 - `symbol`: Permite sustituir el número de sección por un *símbolo*. Se admiten tres valores posibles: `one`, `two` y `three`. El valor `none` para esta opción elimina del índice el número de la sección.
 - `numberalign`: Indica la alineación de los elementos de numeración; puede ser `left`, `right`, `middle`, `flushright`, `flushleft`.

No hay, entre las múltiples opciones de configuración del índice, ninguna que nos permita controlar directamente el tamaño de su interlineado. Este será, por defecto, el general del documento. En muchas ocasiones, sin embargo, es preferible que las líneas del índice estén ligeramente más *apretadas* que las del resto del documento. Para conseguir esto deberíamos encerrar el comando que genera el índice (`\placecontent` o `\completecontent`) dentro de un grupo en el que se establezca un interlineado diferente. Por ejemplo:

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

8.1.7 Ajustes manuales de los índices

Hemos explicado ya los dos comandos fundamentales para generar índices de contenido (`\placecontent` y `\completecontent`), así como sus opciones. Con estos dos comandos se generan, automáticamente, índices contruidos a partir de las secciones numeradas existentes en el documento, o en el bloque o segmento del documento a que se refiera el índice. A continuación explicaré ciertos *ajustes* que podemos hacer para que el contenido del índice no sea tan *automático*. Ello implica:

- La posibilidad de incluir también en el índice ciertos tipos de sección no numerada.
- La posibilidad de enviar, manualmente, alguna entrada concreta al índice que no se corresponda con la presencia de una sección numerada.
- La posibilidad de excluir del índice alguna concreta sección numerada.
- La posibilidad de que el título que para una sección concreta refleja el índice no coincida exactamente con el título que se incluye en el cuerpo del documento.

A. Incluir en el índice secciones no numeradas

El mecanismo de construcción del índice de ConT_EXt incluye en él todas las secciones que se numeran automáticamente, lo que, como ya se dijo (véase la [sección 7.4.2](#)) depende de dos opciones (`number` e `incrementnumber`) que, para cada tipo de sección podemos cambiar con `\setuphead`. También en aquel lugar se explicó que un tipo de sección en el que se hubiera establecido `incrementnumber=yes` y `number=no` sería una sección numerada internamente pero no externamente.

Por lo tanto, si queremos que cierto tipo de sección no numerada —por ejemplo, `title`— se incluya en el índice, debemos cambiar para dicho tipo de sección el valor de la opción `incrementnumber` estableciéndolo como `yes` y, después, incluir ese tipo de sección entre los que deben mostrarse en el índice lo que se haría, como se explicó más arriba, con `\setupcombinedlist`:

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

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

A continuación podremos formatear, si así lo deseamos, dicha entrada del índice mediante `\setuplist` exactamente igual que cualquiera de las restantes; por ejemplo:

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

Nota: El procedimiento que se acaba de explicar incluirá en el índice todas las instancias en nuestro documento del tipo de sección no numerada de que se trate (en nuestro ejemplo las secciones de tipo `title`). Si sólo deseamos incluir en el índice una aparición concreta de ese tipo de sección, es preferible hacerlo por el procedimiento que se explica inmediatamente.

B. Agregar manualmente entradas al índice

Desde cualquier punto del fichero fuente podemos enviar al índice de contenido, bien una entrada (simulando la existencia de una sección que en realidad no existe), bien un comando.

Para enviar una entrada que simule la existencia de una sección que en realidad no existe se usa el comando `\writetolist` cuya sintaxis es:

```
\writetolist[TipoSección] [Opciones] {Número}{Texto}
```

en donde

- El primer argumento indica el nivel que debe tener esta entrada en el índice: `chapter`, `section`, `subsection`, etc.
- El segundo argumento, que es opcional, permite configurar de forma particular esta entrada. Si se omite la entrada manualmente enviada se formateará como todas las del nivel indicado con el primer argumento; aunque, debo señalar, que en mis pruebas no he conseguido que funcione



Tanto en el listado oficial de comandos de ConT_EXt (véase [sección 3.6](#)) como en la wiki se informa de que este argumento admite los mismos valores que `\setuplist` que es el comando que permite formatear las distintas entradas del índice. Pero, insisto, en mis pruebas no he conseguido cambiar la apariencia de la entrada del índice enviada manualmente.



- El tercer argumento se supone que recoge la numeración que habrá de tener el elemento que se envía al índice, pero tampoco he conseguido hacerlo funcionar en mis pruebas.
- El último argumento recoge el texto que se envía al índice.

Esto es útil, por ejemplo, si queremos enviar al índice una concreta sección no numerada, pero sólo esa. En la [sección A](#) se ha explicado como conseguir que toda una categoría de secciones no numeradas se envíen al índice; pero si solo queremos enviar al índice una aparición concreta de un tipo de sección, es más cómodo usar el comando `\writetolist`. Y así, por ejemplo, si deseamos que la sección de nuestro documento en la que se recoge la bibliografía, no sea una sección numerada, pero aún así se incluya en el índice, deberíamos escribir:

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

Véase como usamos para la sección la versión no numerada de `section`, que es `subject`, pero la enviamos al índice, manualmente, como si fuera una sección numerada (`section`.)

Otro comando pensado para influir manualmente en el índice es `\writebetweenlist` que sirve para enviar al índice, desde cierto punto del documento, no una entrada propiamente dicha, sino un *comando*. Por ejemplo, si queremos introducir una línea entre dos elementos del índice, podríamos escribir lo siguiente, en cualquier punto del documento ubicado entre las dos secciones de que se trate:

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

C. Excluir del índice una concreta sección perteneciente a un tipo de secciones que están incluidas en el índice

El índice se construye a partir de *tipos de sección*, establecidos, como ya sabemos, mediante la opción `list` de `\setupcombinedlist`, por lo que si cierto *tipo* de sección debe aparecer en el índice, no hay manera de excluir de él alguna concreta sección que, por las razones que sean, no queremos que figure en él.

Normalmente, si no queremos que una sección figure en el índice, lo que haremos será usar el equivalente *no numerado* de dicha sección; es decir: `title` en lugar de `chapter`, `subject` en lugar de `section`, etc. Estas secciones no se envían al índice, pero tampoco se numeran.

No obstante, si por alguna razón deseamos que cierta sección esté numerada pero no aparezca en el índice, aunque las restantes de su tipo si lo hagan, podemos utilizar un *truco* consistente en crear un nuevo tipo de sección que sea un clon del tipo de sección de que se trate. Por ejemplo:

```
\definehead[MiSubsección][subsection]
\section{Primera sección}
\subsection{Primera subsección}
\MiSubsección{Segunda subsección}
\subsection{Tercera subsección}
```

Esto haría que al insertar una sección del tipo `MiSubSección` se incrementara el contador de subsecciones, pues esta sección es un *clon* de las subsecciones, pero el índice no quedara alterado, pues en el índice, por defecto, no se incluyen las secciones del tipo `MiSubSección`.

D. Texto del título de la sección diferente en el índice y en el cuerpo del documento

Si no queremos que el título de una concreta sección que se incluye en el índice sea idéntico al que se muestra en el cuerpo del documento, disponemos de dos procedimientos:

- Crear la sección no con la sintaxis tradicional (`\TipoSección{Título}`) sino bien con la sintaxis `\TipoSección[Opciones]`, bien con `\startTipoSección[Opciones]`, y asignar a la opción `list` el texto que queremos que se escriba en el índice (véase la [sección 7.3](#)).
- Al escribir, en el cuerpo del documento, el título de la sección de que se trate, usar el comando `\nolist`: este comando hace que el texto que recibe como argumento se sustituya en el índice por unos puntos suspensivos. Por ejemplo:

```
\chapter
  [title={Introducción \nolist{meramente aproximativa y
    ligeramente reiterativa} a la realidad de lo obvio}]
```

escribiría, como título del capítulo, en el cuerpo del documento, «Introducción meramente aproximativa y ligeramente reiterativa a la realidad de lo obvio», pero enviaría al índice el texto «Introducción ... a la realidad de lo obvio».

Atención: Lo que acabo de señalar del comando `\nolist` está afirmado tanto en el manual de referencia de ConT_EXt como en la [wiki](#). A mi, sin embargo, me produce un error de compilación en el que se me informa de que el comando `\nolist` está sin definir.

8.2 Listas, listas combinadas e índices creados a partir de una lista

Desde el punto de vista interno, para ConT_EXt un índice de contenido no es sino una *lista combinada*, la cual, a su vez, como su propio nombre indica, consiste en una combinación de listas simples. Por lo tanto la noción básica a partir de la cual ConT_EXt construye los índices es la de lista. Varias listas se combinan para dar lugar a un índice de contenido. Por defecto ConT_EXt contiene una lista combinada predefinida llamada «`content`» y es con ella con la que trabajan los comandos hasta ahora examinados: `\placecontent` y `\completecontent`.

8.2.1 Listas en ConT_EXt

En ConT_EXt una *lista* es una matriz de elementos numerados de los que hay que recordar tres aspectos:

1. El número.
2. El nombre o título.
3. La página en la que cada elemento se encuentra.

Esto ocurre con las secciones numeradas; pero también con otros elementos de los documentos como, por ejemplo, las imágenes, las tablas, etc. En general aquellos

elementos para los que hay un comando cuyo nombre empiece por `\place` que los ubica como `\placetable`, `\placefigure`, etc.

En todos estos casos ConTeXt genera automáticamente una lista que recoge las diferentes apariciones del tipo de elemento de que se trate, su número, título y página. Así, por ejemplo, hay una lista de capítulos, llamada `chapter`, otra de secciones, llamada `section`; pero también otra de tablas (llamada `table`) o de imágenes (llamada `figure`). Las listas generadas automáticamente por ConTeXt siempre se llaman igual que el elemento que almacenan.

También se generará automáticamente una lista si creamos, por ejemplo, un nuevo tipo de sección numerada: al crearla implícitamente estaremos creando también la lista que las almacena. Y si para una sección no numerada por defecto, establecemos la opción `incrementnumber=yes` convirtiéndola en sección numerada, estaremos también implícitamente creando una lista con dicho nombre.

Junto con las listas implícitas (definidas automáticamente por ConTeXt) podemos crear nuestras propias listas con `\definelist` cuya sintaxis es

```
\definelist[NombreLista][Configuración]
```

Los elementos de la lista se añaden:

- En listas predefinidas por ConTeXt, o creadas por este como consecuencia de la creación de un nuevo objeto flotante (véase la [sección 13.5](#)), automáticamente cada vez que se inserta en el documento un elemento de la lista, bien mediante un comando de seccionado, bien mediante el comando `\placeLoQueSea` para otro tipo de listas. Por ejemplo: `\placefigure` insertará en el documento la imagen que sea, pero también insertará en la lista la entrada correspondiente a la misma.
- En cualquier tipo de lista, manualmente mediante `\writetolist[NombreLista]`, ya explicado en el [apartado B](#) de la [sección 8.1.7](#). También se dispone del comando `\writebetweenlist` explicado en el mismo lugar.

Una vez creada una lista, e incluidos en ella todos sus elementos, los tres comandos básicos relacionados con ella son `\setuplist`, `\placelist` y `\completelist`. El primero permite configurar el aspecto de los elementos de la lista; los dos últimos insertan la lista de que se trate en el punto del documento en el que se encuentren. La diferencia entre `\placelist` y `\completelist` es similar a la que hay entre `\placecontent` y `\completecontent` (véanse las secciones [8.1.2](#) y [8.1.3](#)).

Así, por ejemplo,

```
\placelist[section]
```


insertará una lista de las secciones, incluyendo un hipervínculo a las mismas si la interactividad del documento está activada y en `\setuplist` no hemos establecido `interaction=no`. Una lista de secciones no es exactamente lo mismo que un índice de secciones: la idea de índice suele incluir también los niveles inferiores (subsecciones, subsubsecciones, etc.). Pero una lista de secciones recogerá solo las secciones propiamente dichas.

La sintaxis de estos comandos es:

```
\placelist[NombreLista][Opciones]
```

```
\setuplist[NombreLista][Configuración]
```

Las opciones de `\setuplist` ya las hemos examinado en la [sección 8.1.6](#), y las de `\placelist` son las mismas que las de `\placecontent` (véase la [sección 8.1.3](#)).

8.2.2 Índices de imágenes, tablas y otros elementos

De lo hasta ahora dicho se desprende que, dado que ConTeXt crea automáticamente una lista de imágenes insertadas en el documento mediante el comando `\placefigure`, generar un índice de imágenes en un punto concreto de nuestro documento es tan simple como incluir en él el comando `\placelist[figure]`. Y si queremos generar un índice con título (similar al obtenido con `\completecontent`) bastaría con `\completelist[figure]`. Lo mismo podemos hacer con los otros cuatro tipos de objetos flotantes predefinidos por ConTeXt: tablas («table»), Gráficos («graphic»), intermezzos («intermezzo») y fórmulas químicas («chemical»), aunque para el caso concreto de estos índices, ConTeXt ya incluye un comando que los genera sin título (`\placelistoffigures`, `\placelistoftables`, `\placelistofgraphics`, `\placelistofintermezzi` y `\placelistofchemicals`), y otro que los genera con título (`\completelistoffigures`, `\completelistoftables`, `\completelistofgraphics`, `\completelistofintermezzi` y `\completelistofchemicals`), de modo similar a `\completecontent`.

De la misma manera, para los objetos flotantes creados por nosotros mismos (véase la [sección 13.5](#)) se crearán automáticamente las órdenes `\placelistof<NombreFloat>` y `\completelistof<NombreFloat>`.

Para las listas creadas por nosotros mismos mediante `\definelist` podremos crear un índice con `\placelist[NombreLista]` o con `\completelist[NombreLista]`.

8.2.3 Listas combinadas

Una lista combinada es, como su propio nombre indica, una lista que combina elementos de diferentes listas previamente definidas. ConTeXt por defecto define

una lista combinada para los índices de contenido cuyo nombre es «`content`», pero podemos crear otras listas combinadas mediante `\definecombinedlist` cuya sintaxis es:

```
\definecombinedlist[Nombre] [Listas] [Opciones]
```

donde

- *Nombre*: es el nombre que tendrá la nueva lista combinada.
- *Listas*: se refiere a los nombres de las listas que hay que combinar, separados por comas.
- *Opciones*: Opciones de configuración de la lista. Pueden indicarse en el momento de definir a la lista o, lo que probablemente sea preferible, cuando la lista sea invocada. Las principales opciones (que ya se han explicado) son `criterion` ([apartado 8.1.4](#) de la [sección 8.1.3](#)) y `alternative` (en el [apartado 8.1.5](#) de la misma sección).

Un efecto colateral de la creación de una lista combinada mediante `\definecombinedlist` es que se crea también un comando llamado `\placeNombreLista` que sirve para invocar a la lista, es decir: para incluirla en el fichero de salida. Así por ejemplo,

```
definecombinedlist[Sumario]
```

creará el comando `\placeSumario`; y

```
definecombinedlist[content]
```

creará el comando `\placecontent`

Un momento ¿`\placecontent`? ¿No es ese el comando que se usa para generar un índice de contenido *normal*? En efecto: eso significa que en realidad el índice de contenido estándar es creado por ConTeXt mediante el siguiente comando:

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

Una vez definida nuestra lista combinada, podemos configurarla (o reconfigurarla) con `\setupcombinedlist` que admite las opciones ya explicadas `criterion` (véase el [apartado 8.1.4](#) de la [sección 8.1.3](#)) y `alternative` (véase el [apartado 8.1.5](#) de la misma sección), así como la opción `list` para *cambiar* las listas incluidas en la lista combinada.

El listado oficial de comandos de ConT_EXt (véase [sección 3.6](#)) no menciona entre las opciones admisibles para `\setupcombinedlist` la opción `list` pero esta es utilizada en varios ejemplos de uso de este comando de la wiki (la cual, por otra parte, tampoco la menciona en la página que dedica a este comando). He comprobado, además, que la opción funciona.

8.3 Índices terminológicos

8.3.1 Generación del índice

Un índice terminológico, también llamado «analítico», consiste en un listado de voces significativas ubicado normalmente al final de un documento, en el que se indican las páginas en las que tal noción es tratada.

Cuando los libros se componían a mano, generar un índice terminológico era tarea compleja, además de tediosa. Cualquier cambio en la paginación podía afectar a todas las entradas del índice. Por ello no eran muy habituales. Hoy día los mecanismos informáticos de composición de textos hacen que, si bien probablemente la tarea siga siendo tediosa, ya no es tan compleja; pues para un sistema informático no es tan difícil mantener actualizados los datos asociados a una entrada del índice.

Para generar un índice terminológico necesitamos:

1. Determinar qué palabras voces o conceptos han de formar parte de él. Esto es tarea que sólo puede hacer el autor.
2. Comprobar para cada entrada del futuro índice, en qué puntos del documento aparece la misma. Aunque, para ser precisos, más que *comprobar* los lugares del fichero fuente en los que se habla del concepto o cuestión de que se trate, lo que hacemos cuando trabajamos con ConT_EXt es *marcar* dichos lugares, insertando un comando que luego servirá para que el índice pueda generarse automáticamente. Esta es la parte tediosa.
3. Por último generar y formatear el índice ubicándolo en el punto del documento de nuestra elección. Esto último es bastante sencillo con ConT_EXt y requiere de un sólo comando: `\placeindex`.

A. La previa definición de las entradas del índice y el marcado de los puntos del fichero fuente que se refieren a ellas

El trabajo fundamental está en el segundo paso. Es cierto que los sistemas informáticos también lo facilitan en el sentido de que podemos hacer una búsqueda global de texto para localizar los lugares del fichero fuente en donde se trata de una concreta cuestión. Pero tampoco debemos confiar ciegamente en esas búsquedas

de texto: un buen índice analítico ha de ser capaz de detectar todos los lugares en los que se habla de cierta cuestión, aunque ello se haga sin usar el término *estándar* para referirse a ella.

Para *marcar* un punto concreto del fichero fuente, asociándolo a una palabra, voz o noción que aparecerá en el índice terminológico se usa el comando `\index` cuyo formato es el siguiente:

```
\index[Alfabetización]{Entrada del índice}
```

donde *Alfabetización* es un argumento opcional que sirve para indicar un texto alternativo al de la propia entrada del índice para ordenarla alfabéticamente, y *Entrada del índice* es el texto que aparecerá en el índice, asociado a dicha marca. A dicho texto podemos, por otra parte, aplicarle las características de formato que deseemos, y si en él aparecen caracteres reservados, estos deberán escribirse del modo habitual en ConT_EXt.

La posibilidad de alfabeticar una entrada del índice de una manera distinta a como realmente se escribe, ofrece bastante utilidad. Piénsese, por ejemplo, en este documento, si quiero generar una entrada en el índice para todas las referencias al comando `\TeX`, por ejemplo, la secuencia `\index{\backslash TeX}` alfabeticará el comando, no en la «t» de «TeX», sino en el apartado de los símbolos pues el término que se envía al índice empieza por una barra invertida. Esto se solucionaría escribiendo `\index[tex]{\backslash TeX}`.

Las *entradas del índice* serán las que nosotros queramos. Para que un índice terminológico sea verdaderamente útil hay que esforzarse un poco preguntándose por qué conceptos es más probable que el lector de un documento busque; y así, por ejemplo, será en general mejor definir una entrada como «Hodgkin, Enfermedad de» que definirla como «enfermedad de Hodgkin».

Por convención las entradas de un índice analítico se escriben siempre en minúscula, salvo que se trate de nombres propios. Martínez de Sousa sugiere también emplear las mayúsculas cuando se ha usado una coma para colocar delante la palabra más significativa y ésta empieza por mayúsculas; como en el ejemplo anterior: «Hodgkin, Enfermedad de» (en *Ortografía y ortotipografía del español actual*, 2ª ed., ediciones Trea 2008, pág. 506).

Si el índice tiene varios niveles de profundidad (se admiten hasta tres niveles) para asociar una concreta entrada del índice con un nivel concreto se usa el carácter «+». Del siguiente modo:

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

En el primer caso habremos definido una entrada de segundo nivel llamada *Entrada 2* que será una subentrada de *Entrada 1*. En el segundo caso habremos definido una entrada de tercer nivel llamada *Entrada 3* que será una subentrada de *Entrada 2*, la cual será, a su vez, subentrada de *Entrada 1*. Por ejemplo

A mi `\index{perro}`perro, que es un `\index{perro+galgo}`galgo llamado Yuri. No le gustan los `\index{gato+callejero}`gatos callejeros.

Conviene observar varios detalles del fragmento anterior:

- El comando `\index` habitualmente se coloca *delante* de la palabra a la que se asocia y normalmente no se le separa de ella por un espacio en blanco. Eso es para asegurarse de que el comando estará exactamente en la misma página que la palabra a la que está vinculado:
 - Si hubiera un espacio de separación, cabría la posibilidad de que ConT_EXt eligiera exactamente dicho espacio para insertar un salto de línea y de que diera la casualidad de que ese salto de línea fuera también un salto de página, en cuyo caso el comando estaría en una página y la palabra a la que se asocia en la página siguiente.
 - Si el comando estuviera *detrás* de la palabra, cabría la posibilidad de que dicha palabra se partiera silábicamente y entre dos de sus sílabas se insertara un salto de línea que también fuera un salto de página, en cuyo caso el comando estaría apuntando a una página distinta de aquella en la que empieza la palabra a la que apunta.
- Véase como en las apariciones 2^a in 3^a del comando se introducen voces del segundo nivel.
- Compruébese, asimismo, como en el tercer uso del comando `\index`, aunque la palabra que aparece en el texto es «gatos», la voz que se enviará al índice es «gato».
- Por último: véase como en solo dos líneas se han escrito tres entradas para el índice terminológico. He dicho antes que marcar los lugares precisos del fichero fuente es tedioso. Ahora añadiré que marcarlo demasiado es contraproducente. Un índice demasiado extenso no es preferible a uno más conciso en el que toda la información sea relevante. Por ello dije antes que el decidir qué palabras generarán entrada en el índice y buscarlas para marcarlas, es algo que debe hacer el autor y que es mejor no hacerlo automáticamente, sino que cada entrada del índice debe ser fruto de una decisión consciente del autor.

Si queremos que nuestro índice sea verdaderamente útil, los términos que se usan como sinónimos, deben agruparse en el índice bajo una sola voz. Pero como es posible que el lector busque en el índice información por cualquiera de las otras voces, es corriente que en los índices haya entradas que remiten a otras entradas. Por ejemplo en el índice terminológico de un manual de Derecho civil podríamos encontrarnos perfectamente algo así como

invalidéz contractual
véase *nulidad*.

Ese efecto lo conseguiríamos, no con el comando `\index` sino con `\seeindex` cuyo formato es

```
\seeindex[Alfabetización] {Entrada1} {Entrada2}
```

donde *Entrada1* es la entrada del índice que se remitirá a otra; y *Entrada2* es el destino de la remisión. En nuestro ejemplo de antes habría que escribir:

```
\seeindex{invalidéz contractual}{nulidad}
```

En `\seeindex` puede usarse también el signo «+» para indicar subniveles para cualquiera de sus dos argumentos entre corchetes.

B. Generación del índice propiamente dicho

Una vez que hemos marcado en nuestro fichero fuente todas las entradas para el índice, la generación propiamente dicha del índice se realiza mediante los comandos `\placeindex` o `\completeindex`. Estos dos comandos recorren el fichero fuente en busca de los comandos `\index`, y generan una lista de todas las entradas que el índice debe tener, asociando dicha voz con el número de página correspondiente al lugar donde se encontraba el comando `\index`. Tras ello ordenan alfabéticamente la lista de voces que aparecerán en el índice y fusionan los casos en los que hay más de una aparición para la misma voz, y, finalmente, insertan en el documento final el resultado correctamente formateado.

La diferencia entre `\placeindex` y `\completeindex` es similar a la diferencia que hay entre `\content` y `\completecontent` (véase la [sección 8.1.2](#)): `\placeindex` se limita a generar el índice e insertarlo, mientras que `\completeindex` previamente inserta en el documento final un nuevo capítulo (titulado por defecto, en documentos en español, «Índice»), dentro del cual se escribirá nuestro índice.

8.3.2 Formateo del índice terminológico

Los índices terminológicos son una aplicación concreta de una estructura más general a la que ConT_EXt denomina «registro» («*register*»); por lo tanto el índice se formatea mediante el comando:

```
\setupregister[index] [Configuración]
```

Mediante este comando podemos:

- Determinar el aspecto que en el índice tendrán sus distintos elementos. A saber:

- Los encabezados del índice, que normalmente son los nombres de las letras. Se controlan con `style`, `color` y `command`. Por defecto los encabezados que se usan para cada letra van en minúscula. con `alternative=A` conseguiremos que vayan en mayúscula.
- Las entradas propiamente dichas, y su número de página, cuya apariencia depende de las opciones `textstyle`, `textcolor`, `textcommand` y `deeptextcommand` para la entrada propiamente dicha, y `pagestyle`, `pagecolor` y `pagecommand`, para el número de página. Con `pagenumber=no` podemos también generar un índice terminológico sin números de página (aunque no estoy muy seguro de cuál pueda ser su utilidad).
- La opción `distance` mide la anchura de la separación entre el nombre de una entrada y los números de página; pero también mide el tamaño del sangrado aplicable a las subentradas.

La denominación de las opciones `style`, `textstyle`, `pagestyle`, `color`, `textcolor`, y `pagecolor` creo que es suficientemente clara respecto de lo que cada una de ellas hace. Sobre `command`, `pagecommand`, `textcommand` y `deeptextcommand`, me remito a la explicación que para opciones de nombres similares se hace en la [sección 7.4.4](#), a propósito de la configuración de los comandos de seccionado.

- Configurar la apariencia general del índice, lo que incluye, entre otras, los comandos a ejecutar antes (`before`) o después (`after`) del índice, el número de columnas que éste ha de tener (`n`), si las columnas deben o no estar balanceadas (`balance`), la alineación de las entradas (`align`), etc.

8.3.3 Crear otros índices terminológicos

He explicado el índice terminológico como si en un documento sólo fuera posible un índice de este tipo; pero lo cierto es que los documentos pueden tener tantos índices analíticos como se desee. Los llamados índices onomásticos, por ejemplo, que recogen los nombres propios citados en un documento, con indicación del lugar donde se citan, no dejan de ser un tipo concreto de índice terminológico. También podríamos crear, en un libro jurídico, un índice especial para menciones del Código civil; o, en un documento como el presente, un índice de macros explicadas en él, etc.

Para crear en nuestro documento un índice analítico adicional se usa el comando `\defineregister` cuyo formato es:

```
\defineregister[NombreÍndice] [Configuración]
```

donde *NombreÍndice* es el nombre que tendrá nuestro nuevo índice, y *Configuración* controla su funcionamiento, siendo posible también configurar el índice más adelante mediante

```
\setupregister[NombreÍndice] [Configuración]
```

Una vez creado un nuevo índice de nombre *NombreÍndice* tendremos a nuestra disposición el comando `\NombreÍndice` para marcar con él las entradas que tendrá dicho índice de modo similar a como se marcan las entradas con `\index`. También dispondremos del comando `\seeNombreÍndice` para crear en el índice entradas que se remitan a otras entradas.

Por ejemplo: podría crear un índice de comandos de ConT_EXt en este documento mediante el comando:

```
\defineregister[macro]
```

que crearía el comando `\macro` que me permitiría marcar como entrada del índice todas las referencias a comandos de ConT_EXt, para luego generar un índice con `\placemacro` o con `\completemacro`.

Al crear un índice nuevo se habilita el comando `\NombreÍndice` para marcar sus entradas, y los comandos `\placeNombreÍndice` y `\completeNombreÍndice` para generar el índice. Pero estos dos últimos comandos en realidad son abreviaturas de dos comandos más generales, aplicados al índice en cuestión. Así, `\placeNombreÍndice` equivale a `\placeregister[NombreÍndice]` y `\completeNombreÍndice` equivale a `\completeregister[NombreÍndice]`.

Capítulo 9

Remisiones e hiperenlaces

Sumario: 9.1 Tipos de remisiones; 9.2 Remisiones internas; 9.2.1 La etiqueta en el destino de la remisión; 9.2.2 Comandos en el punto de origen de una remisión para recuperar datos del punto de destino; A Los comandos básicos para recuperar información de una etiqueta; B Recuperar la información asociada a una etiqueta con el comando `\ref`; C Detectar la dirección del enlace; 9.2.3 Generación automática de prefijos para evitar etiquetas duplicadas; 9.3 Documentos electrónicos interactivos; 9.3.1 Activación de la interactividad en los documentos; 9.3.2 Configuración básica de la interactividad; 9.4 Hiperenlaces a documentos externos; 9.4.1 Comandos que ayudan a escribir el hiperenlace, pero no lo crean ellos mismos; 9.4.2 Comandos que establecen el enlace; 9.5 Crear un índice de marcadores en el PDF final;

9.1 Tipos de remisiones

Los documentos científico-técnicos abundan en remisiones:

- A veces se remiten a otros documentos en los que se basa lo que se está diciendo, o que contradicen lo que se está explicando, o que desarrollan o matizan la idea objeto de tratamiento, etc. En estos casos la remisión se dice que es *externa* y, si el documento quiere ser riguroso, adopta la forma de *cita* bibliográfica.
- Pero también es corriente que un documento en alguno de sus apartados se remita a otro de sus apartados, en cuyo caso se dice que la remisión es *interna*. También hay una remisión interna cuando en un punto del documento se comenta algún aspecto de cierta imagen, tabla, nota, o elemento de naturaleza similar, refiriéndonos a él por su número o por la página en la que se encuentra.

La remisión interna para ser precisa debe hacerse a un punto exacto del documento que sea fácilmente identificable. Por ello este tipo de remisiones siempre se hace, bien a elementos numerados (como, por ejemplo, cuando se dice «véase la tabla 3.2», o «el capítulo 7»), bien a números de páginas. Las remisiones vagas del tipo «como ya se ha dicho» o «como se verá más adelante» no son verdaderas remisiones y no tienen ningún tipo de requerimiento de cara a la composición ni hay herramientas especiales para ellas. Yo personalmente además suelo desaconsejar a mis alumnos de doctorado o de TFM el uso habitual de las mismas.

En inglés a las remisiones internas se las suele denominar «cross references» lo que muchas veces se traduce al español como «referencias cruzadas», expresión esta que, en realidad no aclara nada a nadie que no sepa ya de antemano a qué se quiere referir. Por ello yo prefiero hablar aquí de «remisiones» en general y de «remisiones internas» en particular que es, por otra parte, la expresión que recomienda la Real Academia de la Lengua en el *Libro de Estilo de la Lengua Española*, Espasa 2018, págs. 188 y 144. [Esto, por cierto, es un ejemplo de *remisión externa*, me remito a un documento distinto de este].

Para precisar la terminología en materia de remisiones, llamaré *origen* al punto del documento en el que se hace la remisión, y *destino* al lugar al que la remisión apunta. Desde esta perspectiva diremos que una remisión es interna cuando el origen y el destino están en el mismo documento, y externa cuando el origen y destino están en documentos distintos.

Desde el punto de vista de la composición del documento:

- Las remisiones externas no plantean ningún problema especial y por lo tanto, en principio, no requieren de ninguna herramienta que ayude a realizarlas, ya que desde el documento origen no se puede influir de ninguna manera en el documento destino; y todos los datos que necesite del documento destino los tengo a mi disposición y puedo usarlos en la remisión. Pero si el documento de origen es un documento electrónico y el documento de destino también, y, además, está disponible en la Red, entonces es posible incluir en la remisión un hiperenlace que permita saltar directamente al destino. En estos casos se puede decir que el documento origen es *interactivo*.
- Por el contrario las remisiones internas si plantean un reto de cara a la composición del documento, pues cualquiera que tenga experiencia en la confección de documentos científico-técnicos medianamente extensos sabe que es casi inevitable que el número de página, sección, imagen, tabla, teorema o similar al que apunta la remisión, vaya cambiando durante la confección del documento, lo que dificulta mucho el mantenerla actualizada.

En los tiempos previos a la informática los autores huían de las remisiones internas; y las que eran inevitables como, por ejemplo, el índice de contenido (que, si va acompañado del número de página de cada sección, es un ejemplo de remisión interna) se redactaba al final.

Hasta los sistemas de composición más limitados, como, por ejemplo, los procesadores de texto permiten incluir algún tipo de remisiones internas como los índices de contenido. Pero eso no es nada comparado con el completo mecanismo de gestión de remisiones que incluye ConT_EXt, el cual, además, puede combinar el mecanismo de gestión de las remisiones internas dirigido a mantenerlas actualizados, con la utilidad de los hiperenlaces que no es, obviamente, exclusiva de las remisiones externas.

9.2 Remisiones internas

Para establecer una remisión interna se necesitan dos cosas:

1. Una etiqueta o nombre identificador en el punto de destino. ConT_EXt, durante la compilación, asociará ciertos datos a dicha etiqueta. Qué datos se asocian depende del tipo de etiqueta que sea; y así puede ser el número de sección, el número de nota, el número de imagen, el número asociado a un elemento concreto de una enumeración, el título de la sección, etc.
2. Un comando en el punto de origen que lea los datos asociados a la etiqueta vinculada al punto de destino y los inserte en el punto de origen. El comando varía dependiendo de qué dato de la etiqueta queremos insertar en el punto de origen.

Cuando pensamos en una remisión, lo hacemos en términos «origen → destino», por lo que, aparentemente, deberían explicarse primero los aspectos relativos al origen y después los atinentes al destino. Sin embargo creo que es más fácil entender la lógica de las remisiones si la explicación la hacemos al revés.

9.2.1 La etiqueta en el destino de la remisión

En este capítulo, por *etiqueta* entiendo una cadena de texto que quedará asociada al punto de destino de una remisión y que se usará internamente para recuperar cierta información relativa al punto de destino de una remisión tal como, por ejemplo, el número de página, el número de sección etc. En realidad la información que se asocia a cada etiqueta depende del procedimiento de creación de la misma. A estas «etiquetas» ConT_EXt las llama *referencias*, pero creo que este segundo término, como tiene un sentido mucho más amplio, resulta menos clarificador.

La etiqueta asociada al destino de una remisión:

- Debe ser única en el documento para cada posible destino, para identificarlo sin ningún género de dudas. Si se usa la misma etiqueta para dos destinos diferentes ConT_EXt no producirá ningún error de compilación, pero hará que todas las remisiones apunten a la primera etiqueta (en el fichero fuente) lo que, a su vez, tiene el efecto colateral de que parte de nuestras remisiones pueden estar mal, y, lo que es peor, que no nos demos cuenta. Por ello es importante asegurarse, en el momento de creación de la etiqueta, de que esa etiqueta nueva que estamos asignando, no ha sido ya asignada antes.
- Puede contener letras, dígitos, signos de puntuación, espacios en blanco, etc. En el caso de que tenga espacios en blanco, se le aplican las reglas generales de ConT_EXt respecto de este tipo de caracteres (véase [sección 4.2.1](#)), de manera que, por ejemplo «Mi etiqueta bonita» y «Mi etiqueta bonita»

se considerarán la misma etiqueta, aunque el número de espacios en blanco usados en una y otra sean distintos.

Dado que no hay limitación respecto de qué caracteres pueden formar parte de la etiqueta y cuántos pueden ser, mi consejo es usar nombres de etiquetas que sean claros, lo que nos ayudará a comprender el fichero fuente cuando, tal vez, lo leamos bastante tiempo después de su redacción original. Por ello el ejemplo que he puesto antes («Mi etiqueta bonita») no es un buen ejemplo, pues no nos dice nada del destino al que dicha etiqueta apunta. Para este epígrafe, por ejemplo, sería mejor etiqueta la de «sec:Etiquetas de destino»

Para asociar un concreto destino a una etiqueta hay básicamente dos procedimientos:

1. Mediante un argumento u opción del comando que se utilice para crear el elemento al que la etiqueta apuntará. Desde este punto de vista, todos los comandos que crean algún tipo de estructura o elemento de texto que es susceptible de ser destino de una remisión, incluyen una opción llamada «**reference**» que sirve para incluir la etiqueta. A veces en lugar de una *opción* la etiqueta es el contenido de todo el argumento.

Un buen ejemplo de lo que se quiere decir lo tenemos en los comandos de seccionado que, como sabemos ([sección 7.3](#)), admiten varias sintaxis. En la sintaxis clásica el comando se escribe como:

```
\section[Etiqueta]{Título}
```

y en la sintaxis más específica de ConT_EXt el comando se escribe como

```
\startsection[
  title=Título,
  reference=Etiqueta,
  ...
]
```

En uno y otro caso el comando prevé la introducción de una etiqueta que quedará asociada a la sección (o capítulo, subsección, etc) de que se trate.

He dicho que esta posibilidad está en *todos los comandos* que permiten crear un elemento de texto susceptible de ser destino de una remisión. Estos son todos los elementos de texto que admiten ser numerados, lo que incluye, entre otros, secciones, objetos flotantes de todo tipo (tablas, imágenes y similares), notas al pie o finales, citas, enumeraciones, descripciones, definiciones, etc.

Cuando la etiqueta se introduce directamente con un argumento, y no como una opción a la que se asigna un valor, ConT_EXt permite asociar varias etiquetas a un sólo destino. Por ejemplo:



```
\chapter[etiqueta1, etiqueta2, etiqueta3] {Mi capítulo de libro}
```

No tengo claro qué ventaja puede haber en tener varias etiquetas distintas para un sólo objetivo y sospecho que ello se hace, no porque ofrezca ventajas, sino como exigencia *interna* de ConT_EXt aplicable a cierto tipo de argumentos.

2. Mediante los comandos `\pagereference`, `\reference`, o `\textreference`, cuyas sintaxis son:

```
\pagereference[Etiqueta]
\reference[Etiqueta]{Texto}
\textreference[Etiqueta]{Texto}
```

- La etiqueta creada con `\pagereference` permite recuperar el número de página.
- Las etiquetas creadas con `\reference` y con `\textreference` permiten recuperar el número de página así como el texto asociado a ellas que se incluya como argumento.

Tanto en `\reference` como en `\textreference` el texto que se vincula a la etiqueta desaparece como tal del documento final en el punto en el que se encuentra el comando (destino de la remisión), pero puede ser recuperado y reaparecer en el punto de origen de la remisión.

He dicho antes que cada etiqueta queda asociada a cierta información relativa al punto de destino. Cuál sea esa información depende del tipo de etiqueta que sea:

- Todas las etiquetas *recuerdan* (en el sentido de que permiten recuperar) el número de página en el que se encuentra el comando que las ha creado. Para etiquetas vinculadas a secciones que tal vez tengan varias páginas, ese número será el de la página en donde empieza la sección de que se trate.
- Las etiquetas insertadas con el comando que crea un elemento de texto numerado (sección, nota, tabla, imagen, etc.) *recuerdan* también el número asociado a dicho elemento (número de sección, número de nota, etc.).
- Si dicho elemento tiene un *título* como ocurre, por ejemplo, en las secciones, pero también en las tablas si se han insertado mediante el comando `\placetable`, recordarán dicho título.
- Las etiquetas creadas mediante `\pagereference` sólo *recuerdan* el número de página.
- Las creadas mediante `\reference` o `\textreference` recuerdan también el texto asociado a ellas que dichos comandos reciben como argumento.



De hecho no estoy seguro de la diferencia real entre los comandos `\reference` y `\textreference`. Pienso que es posible que el diseño de los tres comandos que permiten crear

etiquetas intente ser paralelo a los tres comandos que permiten recuperar información de las etiquetas (que en seguida veremos); pero lo cierto es que, según mis pruebas, `\reference` y `\textreference` parecen ser comandos redundantes.

9.2.2 Comandos en el punto de origen de una remisión para recuperar datos del punto de destino

Los comandos que a continuación explicaré, recuperan información de las etiquetas y, además, si nuestro documento es interactivo, generan un salto al destino de la remisión. Pero lo importante de estos comandos es la información que se recupera de la etiqueta. Si sólo queremos generar el salto, sin recuperar ninguna información de la etiqueta, es preferible usar el comando `\goto` que se explica en la [sección 9.4.2](#).

A. Los comandos básicos para recuperar información de una etiqueta

Teniendo en cuenta que cada etiqueta asociada a un punto de destino puede almacenar informaciones distintas, es lógico que ConT_EXt contemple tres comandos distintos para recuperar tales informaciones: según qué información del punto de destino de una remisión queramos recuperar, se debe usar un comando u otro:

- El comando `\at` nos permite recuperar el número de página de una etiqueta.
- Para etiquetas que además del número de página recuerdan un número de elemento (número de sección, de nota, de item, de tabla, etc.), el comando `\in` nos permite recuperar dicho número.
- Por último, para etiquetas que recuerdan un texto asociado a una etiqueta (el título de una sección, el título de una imagen insertada con `\placefigure`, etc.) el comando `\about` nos permite recuperar dicho texto.

Los tres comandos tienen la misma sintaxis:

```
\at{Texto}[Etiqueta]
\in{Texto}[Etiqueta]
\about{Texto}[Etiqueta]
```

- Etiqueta es la etiqueta de la que queremos recuperar información.
- Texto es el texto que se escribirá justo antes de la información recuperada por el comando. Entre el texto y los datos de la etiqueta que el comando recupera se insertará un espacio de no separación y si está activada la función de interactividad de tal forma que el comando, además de recuperar la información, genera un enlace que permita saltar al punto del destino, el texto incluido como argumento formará parte del enlace (será texto clicable).

Así, en el siguiente ejemplo se puede ver como `\in` recupera el número de sección y `\at` el número de página.

En la in{sección}[sec:Etiquetas de destino], que empieza en la \at{página}[sec:Etiquetas de destino], se explican las características de las etiquetas usadas para las remisiones internas.

En la sección 9.2.1, que empieza en la página 179, se explican las características de las etiquetas usadas para las remisiones internas.

Obsérvese que ConT_EXt ha creado automáticamente hiperenlaces (véase la sección 9.3), y que el texto recibido como argumento por `\in` y por `\at` forma parte del enlace. Pero si hubiéramos escrito esto otro, resultaría:

En la sección \in{}[sec:Etiquetas de destino], que empieza en la página \at{}[sec:Etiquetas de destino], se explican las características de las etiquetas usadas para las remisiones internas.

En la sección 9.2.1, que empieza en la página 179, se explican las características de las etiquetas usadas para las remisiones internas.

el texto sigue siendo el mismo, pero las palabras *sección* y *página* que preceden a la remisión, al no formar ya parte del comando, no se incluyen en el enlace.

Si ConT_EXt no consigue encontrar la etiqueta a la que apuntan los comandos `\at`, `\in` o `\about`, no se produce ningún error de compilación, pero en el lugar del documento final en el que debería aparecer la información recuperada por estos comandos, se escribirá «??».

Que ConT_EXt no encuentre una etiqueta puede deberse a dos razones:

1. Que nos hayamos equivocado al escribirla.
2. Que, estemos compilando sólo una parte del documento, y la etiqueta apunta a la parte que no está siendo compilada (véanse las secciones 4.5.1 y 4.6).

En el primer caso conviene reparar el error. Por ello es buena idea, al terminar la compilación del documento completo (cuando ya no se puede dar el segundo caso), buscar en el fichero PDF todas las apariciones de la secuencia de caracteres «??» para asegurarnos de que no haya en el documento alguna remisión *rota*.

B. Recuperar la información asociada a una etiqueta con el comando `\ref`

`\at`, `\in` y `\about` recuperan, cada uno de ellos, algún elemento de una etiqueta. Hay disponible otro comando que permite rescatar cualquier elemento de la etiqueta que se le indique, se trata de `\ref`, cuya sintaxis es:

`\ref[Elemento a recuperar] [Etiqueta]`

donde el primer argumento puede ser:

- **text**: Devuelve el texto asociado a una etiqueta.
- **title**: Devuelve el título asociado a la etiqueta.
- **number**: Devuelve el número vinculado a una etiqueta. Por ejemplo, en las secciones, el número de sección.
- **page**: Devuelve el número de página.
- **realpage**: Devuelve el número real de página.
- **default**: Devuelve el que ConT_EXt considera que es el elemento *natural* de dicha etiqueta. En general éste coincide con el que devuelve **number**.

En realidad `\ref` es mucho más preciso que `\at`, `\in` o `\about`, y así, por ejemplo, diferencia entre el número de página y el número real de página. El número de página puede no coincidir con el real si, por ejemplo, se inició la numeración de páginas del documento en un 1500 (por ser este documento la continuación de otro anterior) o si las páginas del preámbulo se numeraron con números romanos y al acabar este se reinició la numeración. Asimismo `\ref` diferencia entre el *texto* y el *título* asociados a una referencia, cosa que, por ejemplo `\about` no hace.

Si se usa `\ref` para obtener de una etiqueta información de la que dicha etiqueta carece (por ejemplo, el título de una etiqueta asociada a una nota a pie de página), el comando devolverá una cadena vacía.

C. Detectar la dirección del enlace

ConT_EXt dispone también de dos comandos que son sensibles a *la dirección del enlace*. Con «dirección del enlace» me quiero referir a determinar si el destino de la remisión se encuentra, en el fichero fuente, antes del origen, o después. Por ejemplo: estamos redactando nuestro documento y nos queremos referir a una sección que aún no hemos decidido si en el índice final irá antes o después de la que estamos escribiendo. En tal circunstancia sería útil disponer de un comando que escriba una cosa u otra según, en el documento definitivo, el destino finalmente se encuentre antes o después del origen. Para tales necesidades ConT_EXt proporciona el comando `\somewhere` cuya sintaxis es:

`\somewhere{Texto si antes}{Texto si después}[Etiqueta]`.

Por ejemplo, en el siguiente texto:

La dirección de un hiperenlace puede también ser detectada por el comando `\type{\somewhere}`. De esta forma podemos encontrar capítulos u otros elementos de texto `\somewhere {anteriores} {posteriores}` `[sec:referencias]` y discutir en algún lugar `\somewhere {anterior} {posterior}` `[sec:interactividad]` sus descripciones.

La dirección de un hiperenlace puede también ser detectada por el comando `\somewhere`. De esta forma podemos encontrar capítulos u otros elementos de texto `anterioresanterior-resposterioresposterioressec:referencias` y discutir en algún lugar `anterioranteriorposterior-posteriorsec:interactividad` sus descripciones.

Para este ejemplo he usado dos etiquetas reales que en el fichero fuente de esta introducción existen en este capítulo.

Otro comando capaz de detectar si la etiqueta a la que apunta se encuentra antes o después, es `\atpage` cuya sintaxis es:

`\atpage[etiqueta]`

Este comando es bastante parecido al anterior, pero en lugar de permitirnos escribir nosotros mismos el texto a poner según la etiqueta resulte estar antes o después, `\atpage` inserta un texto predeterminado para cada uno de ambos casos y, si el documento es interactivo, también inserta un hiperenlace.

El texto que `\atpage` inserta es el asociado a las *etiquetas* «`precedingpage`» para el caso de que la *etiqueta* que recibe como argumento se encuentre *antes* que el comando, y «`hereafter`» para el caso contrario.

Al llegar aquí me traiciona una decisión previa: decidí llamar, en este capítulo, «etiqueta» a lo que ConT_EXt llama «referencia». Me parecía más claro. Pero también se llaman «etiquetas» (ahora en otro sentido) a los «rótulos» o fragmentos de texto que ciertos comandos de ConT_EXt, como `\atpage`, generan (véase la [sección 10.5.3](#)). Espero que ello no confunda al lector. Creo que el contexto permite diferenciar bien a cuál de los distintos sentidos de *etiqueta* me refiero en cada caso.

Por lo tanto podemos cambiar el texto insertado por `\atpage` del mismo modo que se cambia el texto de cualquier otra etiqueta:

`\setuplabeltext[es][precedingpage=Nuevo texto]`
`\setuplabeltext[es][hereafter=Nuevo texto]`

En este punto creo que hay un pequeño error en «ConT_EXt Standalone» (que es la distribución que estoy manejando). Examinando los nombres de las etiquetas predefinidas en ConT_EXt que se pueden cambiar con `\setuplabeltext` hay dos pares de etiquetas que son candidatas a ser usadas por `\atpage`:

- «`precedingpage`» y «`followingpage`».
- «`hencefore`» y «`hereafter`».

Podríamos presuponer que `\atpage` usará bien el primer par, bien el segundo. Pero de hecho usa, para los objetivos anteriores «`precedingpage`» y para los posteriores «`hereafter`», lo que creo que es una inconsistencia.

De otro lado la etiqueta «`precedingpage`» no tiene traducción al español en la distribución.

9.2.3 Generación automática de prefijos para evitar etiquetas duplicadas

En un documento grande no siempre es fácil evitar la duplicación de etiquetas. Por lo tanto, es aconsejable poner un poco de orden en la forma de elegir qué etiquetas utilizar. Una práctica que ayuda a ello es la de usar prefijos para las etiquetas que variarán según el tipo de etiqueta que sea. Por ejemplo «`sec:`» para las secciones, «`fig:`» para las figuras, «`tbl:`» para las tablas, etc.

Pensando en esta forma de actuar, ConT_EXt incluye un conjunto de herramientas que permiten:

- Que el propio ConT_EXt genere automáticamente etiquetas para todos los elementos que las admitan.
- Que toda etiqueta generada manualmente, reciba un prefijo, bien predeterminado por nosotros mismos, bien generado automáticamente por el propio ConT_EXt.

La explicación detallada de este mecanismo es larga y, aunque se trata, sin duda, de herramientas útiles, no me parecen imprescindibles, por ello, como no se pueden explicar en pocas palabras, prefiero no explicarlas y remitirme a lo que al respecto se dice en el capítulo de «Referencias» del manual de referencia de ConT_EXt, o a la [información de la wiki](#) sobre esta cuestión.

9.3 Documentos electrónicos interactivos

Sólo los documentos electrónicos pueden ser interactivos; pero no todos los documentos electrónicos lo son. Un documento *electrónico* es aquel que se almacena en un fichero informático y que puede ser abierto y leído directamente en la pantalla del mismo. Interactivo es, por otra parte, aquel documento electrónico que está dotado de utilidades que permiten al usuario *interactuar* con él; es decir: hacer algo más que limitarse a leerlo. Hay interactividad, por ejemplo, cuando el documento dispone de botones que realizan alguna acción, o de enlaces mediante los que se puede saltar a otro punto del documento, o a un documento externo; o cuando en el documento hay zonas en las que el usuario puede escribir, vídeos o clips de audio que puede reproducir, etc.

Todos los documentos que genera ConT_EXt son electrónicos (pues ConT_EXt lo que genera es un fichero PDF que es, por definición, un documento electrónico), pero no siempre son interactivos. Para dotarles de interactividad es preciso indicarlo expresamente como se muestra en la próxima sección.

Téngase en cuenta, por otra parte, que aunque ConT_EXt genere un fichero PDF interactivo, para apreciar la interactividad es preciso que el programa visor de PDF que utilicemos sea capaz de ello, pues no todos los visores de PDF existentes permiten hacer uso de hiperenlaces, botones y recursos similares propios de los documentos interactivos.

9.3.1 Activación de la interactividad en los documentos

ConT_EXt no utiliza por defecto funciones interactivas a no ser que expresamente se le indique cosa que, normalmente se hace en el preámbulo del documento. El comando que activa esta utilidad es:

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

Normalmente este comando se usará sólo en una vez en el preámbulo del documento, cuando deseemos generar un documento interactivo. Pero en realidad podemos usarlo tantas veces como queramos e ir alterando el «estado de interactividad» del documento. La opción «`state=start`» activa la interactividad, mientras que «`state=stop`» la desactiva, por lo que podemos desactivar la interactividad en aquellos capítulos, o *fragmentos* de nuestro documento en que queramos hacerlo.

No se me ocurre por qué razón íbamos a querer tener fragmentos no interactivos en documentos que tienen interactividad. Pero lo importante de la filosofía de ConT_EXt es que si algo es técnicamente posible, aunque sea poco probable que se quiera utilizar, se habilita un procedimiento para hacerlo. Es esa filosofía la que dota a ConT_EXt de tantísimas posibilidades, e impide que una simple introducción como la presente sea *breve*.

Una vez establecida la interacción:

- Ciertos comandos de ConT_EXt de modo predeterminado incluirán enlaces para hipersaltos. Así:
 - Los comandos de creación de índices de contenido, los cuales serán, en principio y salvo que se indique expresamente otra cosa, interactivos, es decir: haciendo clic sobre una entrada del índice saltaremos a la página en la que empieza la sección de que se trate.
 - Los comandos para remisiones internas que hemos visto en la primera parte de este mismo capítulo, en las que haciendo clic sobre ellas, se salta automáticamente al destino de la remisión.

- Las notas a pie de página y las notas finales en las que un clic sobre la llamada de la nota en el cuerpo principal del texto nos llevará a la página en donde se escribe la nota propiamente dicha, y un clic sobre la marca de la nota en el texto de la nota nos llevará al punto del texto principal donde se hizo la llamada.
- Etc.
- Se activa la posibilidad de usar otros comandos específicamente pensados para documentos interactivos como pueden ser las presentaciones, que utilizan numerosas herramientas asociadas a la interactividad tales como botones, menús, superposición de imágenes, sonido o vídeo incrustado, etc. La explicación de todo esto sería demasiado larga y además las presentaciones son un tipo muy particular de documentos. Por ello en las líneas que siguen tan sólo describiré una característica asociada a la interactividad: los hiperenlaces.

9.3.2 Configuración básica de la interactividad

`\setupinteraction` además de activar o desactivar la interacción, permite configurar algunas cuestiones relacionadas con ella; principalmente, pero no sólo, el color y estilo de los enlaces. Ello se hace mediante las siguientes opciones del comando:

- `color`: controla el color *normal* de los enlaces.
- `contrastcolor`: determina el color de aquellos enlaces en los que el destino se encuentra en la misma página que el origen. Recomento que esta opción se establezca siempre con el mismo contenido que la anterior.
- `style`: controla el estilo de los enlaces.
- `tittle`, `subtitle`, `author`, `date`, `keyword`: Los valores asignados a estas opciones se convertirán en metadatos del PDF generado por ConTeXt.
- `click`: Esta opción controla si el enlace se ha de resaltar cuando se hace clic en él.

9.4 Hiperenlaces a documentos externos

Distinguiré entre comandos que no crean el enlace, pero ayudan a escribir la URL del mismo y comandos que crean el hiperenlace. Veamoslos por separado:

9.4.1 Comandos que ayudan a escribir el hiperenlace, pero no lo crean ellos mismos

Las URLs tienden a ser muy largas, e incluyen caracteres de todo tipo, incluso caracteres que en ConTeXt son caracteres reservados y no se pueden usar directamente. Además, cuando la URL se debe mostrar en el documento, es muy difícil componer el párrafo, pues la URL puede llegar a superar la longitud de una línea y nunca incluye dentro de sí espacios en blanco que puedan servir para insertar un salto de línea. En una URL, además, no es razonable realizar partición silábica de palabras para insertar saltos de línea, pues el lector difícilmente podría saber si el guión de la partición silábica forma o no realmente parte de la URL.

Por ello ConTeXt ofrece dos utilidades para *escribir* las URLs. La primera es, fundamentalmente, para URLs que serán usadas internamente, pero no se mostrarán realmente en el documento. La segunda está pensada para URLs que haya que escribir en el texto del documento. Veamoslas por separado:

`\useURL`

Este comando nos permite escribir en el preámbulo del documento una URL asociándola a un nombre, de tal manera que cuando queramos usarla en nuestro documento, podamos invocarla por el nombre asociado a ella. Es especialmente útil con URLs que serán utilizadas varias veces a lo largo del documento.

El comando admite dos formas de uso:

1. `\useURL[Nombre asociado][URL]`
 2. `\useURL[Nombre asociado][URL][] [Texto del enlace]`
- En la primera versión simplemente se asocia la URL al nombre por el que la invocaremos en nuestro documento; pero luego, para usar dicha URL, tendremos que, al invocarla, indicar de alguna manera qué texto clicable se mostrará en el documento.
 - En la segunda versión el último argumento incluye el texto clicable. El tercer argumento, existe por si deseamos dividir una URL en dos partes, de tal modo que la primera parte recoja la dirección de acceso y la segunda el nombre del documento o página concreta que se desea abrir. Por ejemplo: la dirección del documento explicativo de lo que es ConTeXt es <http://www.pragma-ade.com/general/manuals/what-is-context.pdf>. Esa dirección podemos escribirla toda en el segundo argumento, dejando vacío el tercero:

```
\useURL [QueEsCTX]
[http://www.pragma-ade.com/general/manuals/what-is-context.pdf]
[]
[¿Qué es \ConTeXt?]
```

pero también podemos repartirla entre los dos argumentos:

```
\useURL [QueEsCTX]
[http://www.pragma-ade.com/general/manuals/]
[what-is-context.pdf]
[¿Qué es \ConTeXt?]
```

En ambos casos habremos asociado dicha dirección a la palabra «QueEsCTX», de tal manera que para incluir un enlace a tal dirección, usemos el comando que usemos para crear el enlace, en lugar de la URL propiamente dicha, podremos escribir simplemente «QueEsCTX».

Si en algún punto del texto queremos reproducir una URL que hemos asociado a un nombre mediante `\useURL`, podemos usar el comando `\url [Nombre asociado]` que inserta en el documento la URL asociada a dicho nombre. Pero este comando, aunque escribe la URL, no crea ningún enlace.

El formato en el que se muestran las URLs escritas mediante `\url` no es el establecido de modo general mediante `\setupinteraction`, sino el fijado específicamente para este comando mediante `\setupurl` que permite configurar el estilo (opción `style`) y el color (opción `color`).

`\hyphenatedurl`

Este comando está pensado para URLs que se escribirán en el texto de nuestro documento, y hace que ConTeXt incluya dentro de la URL saltos de línea, si es preciso, para componer correctamente el párrafo. Su formato es:

```
\hyphenatedurl{Dirección URL}
```

A pesar del nombre del comando, `\hyphenatedurl` no parte silábicamente el nombre de la URL. Lo que hace es considerar que ciertos caracteres, habituales en las URLs son buenos puntos para insertar antes o después de ellos saltos de línea. Podemos añadir los caracteres que queramos a la lista de caracteres donde se autoriza un salto de línea. Para ello disponemos de tres comandos:

```
\sethyphenatedurlnormal{Caracteres}
\sethyphenatedurlbefore{Caracteres}
\sethyphenatedurlafter{Caracteres}
```

Estos comandos añaden los caracteres que reciban como argumento a, respectivamente, la lista de caracteres que admiten saltos de línea antes y después, la de los que sólo admiten saltos de línea anteriores, y la de los que sólo admiten saltos de líneas posteriores.

`\hyphenatedurl` puede usarse siempre que se deba escribir una URL que vaya a aparecer en el documento final tal cual. Incluso se puede usar como último argumento de `\useURL` en la versión de dicho comando en donde el último

argumento recoge el texto clicable que se mostrará en el documento final. Por ejemplo:

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

En el argumento de `\hyphenatedurl` se pueden usar todos los caracteres reservados salvo tres que se deben sustituir por comandos:

- % se debe sustituir por `\letterpercent`
- # se debe sustituir por `\letterhash`
- \ se debe sustituir por `\letterescape` o `\letterbackslash`.

Cada vez que `\hyphenatedurl` inserta un salto de línea ejecuta el comando `\hyphenatedurlseparator`, el cual, por defecto, no hace nada. Pero si lo redefinimos conseguiremos que, de modo similar a como ocurre en las palabras normales, en las que se inserta un guión indicador de que la palabra sigue en la línea siguiente, se inserte, en la URL un carácter representativo de ello. Por ejemplo:

```
\def\hyphenatedurlseparator{\curvearrowright}
```

mostrará así la siguiente dirección web especialmente larga:

<https://support.microsoft.com/?scid=http://support.microsoft.com:80~support/kb/articles/Q208/4/27.ASP&NoWebContent=1>.

9.4.2 Comandos que establecen el enlace

Para establecer enlaces a URLs predefinidas mediante `\useURL` podemos utilizar el comando `\from`, el cual se limita a establecer el enlace, pero no escribe ningún texto clicable. Se usará como texto del enlace el predefinido en `\useURL`. Su sintaxis es:

```
\from[Nombre]
```

donde *Nombre* es el nombre asociado previamente a una URL mediante `\useURL`.

Para crear enlaces y asociarlos a un texto clicable no previamente predefinido disponemos del comando `\goto`, que sirve tanto para generar saltos internos como para saltos externos. Su sintaxis es:

```
\goto{Texto clicable}[Destino]
```

donde *Texto clicable* es el texto que se mostrará en el documento final y en el que podremos hacer clic con el ratón para generar el salto, y *Destino* puede ser:

- Una etiqueta de nuestro documento. En tal caso `\goto` generará el salto de modo similar a como lo hacen, por ejemplo, los comandos ya examinados `\in` o `\at`. Pero a diferencia de dichos comandos, no se recuperará ninguna información asociada a la etiqueta.
- Una URL propiamente dicha. En tal caso hay que indicar expresamente que se trata de una URL escribiendo el comando de la siguiente manera:

```
\goto{Texto clicable}[url(URL)]
```

donde URL, a su vez, podrá ser el nombre previamente asociado a una URL mediante `\useURL`, o la URL propiamente dicha, caso este en el que al escribir la URL debemos asegurarnos de que los caracteres reservados de ConTeXt que en ella haya se escriban del modo correcto en ConTeXt. Esto, escribir la URL según las reglas de ConTeXt, no afectará a la funcionalidad del enlace.

9.5 Crear un índice de marcadores en el PDF final

Los ficheros PDF pueden tener un índice de marcadores interno que permite al lector ver en una ventana especial del programa visor del PDF el índice del documento, y desplazarse por él simplemente haciendo clic sobre cada una de las secciones y subsecciones.

Por defecto ConTeXt no dota al PDF de salida de un índice de marcadores, aunque conseguir que lo haga es tan sencillo como incluir en nuestro documento el comando `\placebookmarks` cuya sintaxis es:

```
\placebookmarks[Listado de secciones]
```

donde *Listado de secciones* es una lista, separada por comas, de los niveles de seccionado que deben aparecer en el índice.

Respecto de este comando, ténganse en cuenta las siguientes observaciones:

- De acuerdo con mis pruebas `\placebookmarks` no funciona si se encuentra en el preámbulo del documento. Pero, dentro de lo que es cuerpo del documento (entre `\starttext` y `\stoptext`, o entre `\startproduct` y `\stopproduct`), da igual en qué lugar se escriba: el índice de marcadores incluirá también las secciones o subsecciones anteriores al comando. No obstante creo que lo más razonable para un fichero fuente comprensible es ubicar el comando al principio.
- Los tipos de sección definidos por el usuario (con `\definehead`) no siempre se ubican, en el índice de marcadores, en el lugar correcto. Es preferible excluirllos.

- Si en alguna sección el título de la sección incluía una nota final o a pie de página, el texto de la nota se considerará que forma parte del marcador.
- Como argumento, en lugar de una lista de secciones, podemos indicar simplemente la palabra simbólica «all» que, como su propio nombre indica, incluirá todas las secciones; sin embargo, de acuerdo con mis pruebas, esta palabra, además de lo que son ciertamente secciones, incluye en el índice textos incluidos con algunos comandos que no son de seccionado, por lo que el resultado del índice es algo imprevisible.

No todos los programas visores de PDF permiten ver el índice de marcadores; y muchos que sí lo permiten, por defecto no tienen activada tal posibilidad. Por lo tanto, para comprobar el resultado de esta función debemos asegurarnos de que nuestro programa lector de PDF soporta dicha función y la tiene activada. Acrobat, por ejemplo, creo recordar que por defecto no muestra el índice de marcadores, aunque hay algún botón en su barra de herramientas para mostrarlos.

III

Cuestiones particulares

Capítulo 10

Caracteres, palabras, texto y espacio horizontal

Sumario: **10.1 Obtener caracteres no accesibles normalmente desde el teclado;** 10.1.1 Signos diacríticos y letras especiales; 10.1.2 Ligaduras tradicionales; 10.1.3 Letras griegas; 10.1.4 Símbolos variados; 10.1.5 Definición de caracteres; 10.1.6 Utilización de conjuntos de símbolos predefinidos; **10.2 Formatos especiales de los caracteres;** 10.2.1 Letras mayúsculas, minúsculas y pseudoversalitas; 10.2.2 Texto en subíndice o superíndice; 10.2.3 Texto verbatim; **10.3 Espacios de separación entre caracteres y palabras;** 10.3.1 Fijación automática del espacio horizontal; 10.3.2 Alteración del espacio entre caracteres dentro de una palabra; 10.3.3 Comandos para añadir espacio horizontal entre palabras; **10.4 Palabras compuestas;** **10.5 El idioma del texto;** 10.5.1 Fijación y cambio del idioma; 10.5.2 Configuración del idioma; 10.5.3 Etiquetas asociadas a los concretos idiomas; 10.5.4 Algunos comandos vinculados al idioma; A Comandos relacionados con la fecha; B El comando `\translate`; C Los comandos `\quote` y `\quotation`;

El elemento nuclear básico de todo documento de texto es el carácter: los caracteres se agrupan en palabras, éstas a su vez forman líneas que componen párrafos con los que se van formando las páginas.

El presente capítulo, partiendo del «*carácter*» explica algunas utilidades de ConTeXt relacionadas con los caracteres, las palabras y el texto.

10.1 Obtener caracteres no accesibles normalmente desde el teclado

En un fichero de texto codificado como UTF-8 (véase [sección 4.1](#)) cabe casi cualquier carácter o símbolo, tanto de lenguas vivas como de muchas lenguas ya extintas. Pero, como las posibilidades de un teclado son limitadas, gran parte de los caracteres y símbolos admisibles en UTF-8 normalmente no se podrán obtener directamente desde el teclado. Ello ocurre, en particular, con numerosos signos diacríticos, es decir: los que se sitúan sobre (o bajo) ciertas letras dotándoles de un valor especial; pero también con muchos otros caracteres como símbolos matemáticos, ligaduras tradicionales, etc. Muchos de estos caracteres los podemos obtener en ConTeXt mediante comandos.

10.1.1 Signos diacríticos y letras especiales

Casi todos los idiomas occidentales tienen signos diacríticos (con la importante salvedad del inglés) y en general los teclados están preparados para generar los diacríticos correspondientes al o a los idiomas regionales. Así, un teclado español puede generar todos los diacríticos necesarios para el idioma español (básicamente acentos agudos y diéresis) así como algunos diacríticos que se usan en otros idiomas próximos como el catalán (acentos graves y cedillas) o el francés (cedillas, acentos graves y circunflejos); pero no, por ejemplo, algunos que se usan en portugués, como la tilde sobre vocal de palabras como «navegação».

TeX fue diseñado en Estados Unidos donde los teclados en general no permiten obtener ningún diacrítico; por ello su autor le dotó de un conjunto de comandos que permitían obtener casi todos los símbolos diacríticos conocidos (al menos en los idiomas que usan el alfabeto latino). Si usamos un teclado español, como supongo que será el caso de la mayor parte de los lectores de este texto, no tiene demasiado sentido usar esos comandos para obtener aquellos diacríticos que se pueden obtener directamente desde el teclado. Aún así conviene saber que estos comandos existen, y cuáles son, pues los teclados españoles no permiten generar todos los diacríticos posibles.

Nombre	Carácter	Abreviatura	Comando
Acento agudo	ú	\'u	\uacute
Acento grave	ù	\`u	\ugrave
Acento circunflejo	û	\^u	\ucircumflex
Diéresis o umlaut	ü	\"u	\udiaeresis, \uumlaut
Tilde	ũ	\~u	\utilde
Macrón o acento largo	ū	\=u	\umacron
Acento breve	ǔ	\u u	\ubreve

Tabla 10.1 Acentos y otros diacríticos

En la [tabla 10.1](#) se contienen los comandos y las abreviaturas que permiten obtener estos signos. Con un teclado español nunca necesitaremos usar los cuatro primeros comandos; pero los tres últimos tal vez sí debamos usarlos en alguna ocasión. En todos ellos es indiferente usar el comando o la abreviatura. En la tabla he puesto como ejemplo la letra «u» pero estos comandos funcionan con cualquier vocal (la mayoría¹) y también con algunas consonantes o semivocales (algunos de ellos).

- Como la mayor parte de los comandos abreviados son *símbolos de control* (véase la [sección 3.2](#)), la letra sobre la que ha de recaer el diacrítico puede

¹ De los comandos recogidos en la [tabla 10.1](#) la tilde no funciona con la letra «e», no conozco la razón.

escribirse inmediatamente detrás del comando, o separarse de él con uno o varios espacios en blanco, y así, por ejemplo: Para conseguir la «ã» portuguesa podremos escribir `\=a` o `\=_a`¹. Pero en el caso del acento breve (`\u`), al tratarse de una *palabra de control*, el espacio en blanco de separación es obligatorio.

- Tratándose de la versión larga del comando, la letra sobre la que debe recaer el diacrítico será la primera letra del nombre del comando, y así, por ejemplo `\emacron` pondrá un macrón sobre una «e» minúscula (ē), `\Emacron` lo hará sobre una «E» mayúscula (Ē), mientras que `\Amacron` lo hará sobre una «A» mayúscula (Ă).

Si los comandos de la [tabla 10.1](#) funcionan sobre cualquier vocal y sobre algunas consonantes, hay otros comandos para generar algunos diacríticos y letras especiales, que sólo funcionan sobre una o varias letras. Se muestran en la [tabla 10.2](#).

Nombre	Carácter	Abreviatura	Comando
O escandinava	ø, Ø	<code>\o</code> , <code>\O</code>	
A escandinava	å, Å	<code>\aa</code> , <code>\AA</code> , <code>{\r a}</code> , <code>{\r A}</code>	<code>\aring</code> , <code>\Aring</code>
L polaca	ł	<code>\l</code> , <code>\L</code>	
Eszett alemana	ß	<code>\ss</code> , <code>\SS</code>	
«i» y «j» sin punto	ı, j	<code>\i</code> , <code>\j</code>	
Umlaut húngaro	ű, Ű	<code>\H u</code> , <code>\H U</code>	
Cedilla	ç, Ç	<code>\c c</code> , <code>\c C</code>	<code>\ccedilla</code> , <code>\Ccedilla</code>

Tabla 10.2 Más diacríticos y letras especiales

De la tabla anterior quiero señalar que algunos de los comandos generan ellos mismos el carácter a partir de otros caracteres, mientras que otros comandos sólo funcionan si la fuente que estamos usando ha previsto expresamente el carácter en cuestión. Por ello a propósito de la Eszett alemana (ß), en la tabla se muestran dos comandos pero un solo carácter; porque la fuente con la que estoy escribiendo este texto sólo ha previsto la Eszett alemana en mayúsculas (cosa, por otra parte, bastante corriente). Probablemente sea esa la razón de que tampoco consigo que `\AA` genere la A escandinava en mayúsculas, aunque «`{\r A}`» y `\Aring` funcionan correctamente.

El umlaut húngaro, además, funciona también con la letra «o», y la cedilla con las letras «k», «l», «n», «r», «s» y «t», en minúsculas o en mayúsculas; para ello los comandos a usar son, respectivamente, `\kcedilla`, `\lcedilla`, `\ncedilla`, etc.

10.1.2 Ligaduras tradicionales

Una ligadura es un símbolo formado por la unión de dos o más grafemas que suelen escribirse por separado. Esta «fusión» entre dos caracteres en muchas ocasiones

¹ Recuérdese que en este documento representamos los espacios en blanco, cuando es importante que se les vea, con el carácter «`_`».

empezó haciéndose para abreviar la escritura, cuando esta se hacía a mano; hasta que finalmente alcanzaron cierta independencia tipográfica. Algunas de ellas se incluyeron incluso entre los caracteres que se suelen definir en una fuente tipográfica como es el caso de «&» que empezó siendo una contracción de la conjunción copulativa latina «et», o la Eszett alemana (ß), que, como su propio nombre indica, empezó siendo una combinación de una «s» y una «z». En algunos diseños de fuentes, todavía hoy, se pueden rastrear esos orígenes de estos dos caracteres; o tal vez yo los veo porque sé que están ahí. En particular, con la fuente Pagella para «&» y con Bookman para «ß». Como ejercicio sugiero que (tras leer el capítulo ??, donde se explica cómo hacerlo) se intente representar esos caracteres, con dichas fuentes, a un tamaño lo suficientemente grande (por ejemplo, 30 pt) como para que se puedan “intuir” los componentes originales.

Otras de estas ligaduras tradicionales, que no llegaron a popularizarse tanto, pero que todavía hoy día son usadas ocasionalmente, son las terminaciones latinas «oe» y «ae» que ocasionalmente se escribían como «œ» o «æ» para indicar que en latín formaban diptongo. En ConT_EXt estas ligaduras se obtienen con los comandos recogidos en la [tabla 10.3](#)

Ligadura	Abreviatura	Comando
æ, Æ	\ae, \AE	\aeligature, \AEligature
œ, Œ	\oe, \OE	\oeligature, \OEligature

Tabla 10.3 Ligaduras tradicionales

Una ligadura que era tradicional en la escritura castellana y que no suele estar representada en las fuentes tipográficas, es «Ð»: una contracción entre la «D» y la «E». No hay (que yo sepa) ningún comando en ConT_EXt que permita representarla¹, pero podemos crear uno, tal y como se explica en la [sección 10.1.5](#).

Junto con las anteriores ligaduras, a las que he llamado *tradicionales*, por proceder de la escritura a mano, tras la invención de la imprenta se fueron desarrollando ciertas ligaduras propias de textos impresos, a las que llamaré «ligaduras tipográficas» que en ConT_EXt se consideran «utilidades» de la fuente y son gestionadas automáticamente por el programa, aunque podemos influir en cómo se gestionarán las utilidades de las fuentes (incluyendo las ligaduras) mediante `\definefontfeature` (que no se explica en esta introducción).

10.1.3 Letras griegas

En fórmulas matemáticas y físicas es corriente usar caracteres griegos. Por ello T_EX incluyó la posibilidad de generar todo el alfabeto griego, en mayúsculas y

¹ En L^AT_EX, por el contrario, puede usarse el comando `\DH` implementado por el paquete «fontenc».

en minúsculas. Aquí el comando se construye a partir del nombre en inglés de la letra griega en cuestión. Si el primer carácter de la misma se escribe en minúscula tendremos la letra griega minúscula y si se escribe en mayúsculas obtendremos la letra griega mayúscula. Por ejemplo, en inglés a la letra griega que en español se llama «my», se la llama «mu». Por tanto el comando `\mu` generará dicha letra en minúscula (μ) y `\Mu` Generará la versión en mayúsculas (M). En la [tabla 10.4](#) se puede ver con qué comando se genera cada una de las letras del alfabeto griego, en mayúsculas o en minúsculas.

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

Tabla 10.4 Alfabeto griego

Obsérvese como para la versión en minúscula de algunos caracteres (epsilon, kappa, zeta, pi, rho, sigma y fi) hay dos posibles variantes.

10.1.4 Símbolos variados

Junto a los caracteres ya vistos, \TeX (y, por lo tanto, también \ConTeXt), ofrece comandos para generar numerosos símbolos. Estos comandos son muchos. En el [Apéndice B](#) he recogido un listado amplio, aunque incompleto.

10.1.5 Definición de caracteres

Si necesitamos usar algún carácter que no es accesible desde nuestro teclado, podemos buscar alguna página web en donde dicho carácter se represente y copiarlo

en nuestro fichero fuente. Si estamos usando la codificación UTF-8 (tal y como se recomienda) eso funcionará casi siempre. A estos efectos, en la wiki de ConT_EXt existe una página con multitud de símbolos para que se puedan simplemente copiar y pegar en nuestro documento. Para acceder a ella pulse [en este enlace](#).

Pero si debemos usar más de una vez el carácter en cuestión, irlo copiando y pegando no es el procedimiento más eficiente. Es preferible definir dicho carácter de tal modo que el mismo quede asociado a un comando que lo generará en lo sucesivo. Para ello se usa `\definecharacter` cuya sintaxis es

```
\definecharacter Nombre Carácter
```

donde

- **Nombre** es el nombre al que se asociará el nuevo carácter. No ha de ser el nombre de un comando ya existente, pues ello sobrescribiría dicho comando.
- **Carácter** es el carácter que se generará cada vez que se ejecute `\Nombre`. Este carácter lo podemos indicar de tres modos:
 - Simplemente escribiéndolo o pegándolo en nuestro fichero fuente (si lo hemos copiado de algún documento electrónico o página web).
 - Indicando el número asociado a dicho carácter en la fuente que estamos usando en este momento. Para poder ver los caracteres incluidos en la fuente, y los números que se les asocian, podemos usar el comando `\showfont[Nombre de la fuente]`.
 - Construyendo el nuevo carácter con alguno de los comandos de construcción de caracteres compuestos que veremos inmediatamente.

Como ejemplo del primer uso, retrocedamos un instante a la sección relativa a las ligaduras (10.1.2). En ella he hablado de una ligadura tradicional de la escritura castellana que no suele existir en las fuentes: «Ð». Podemos asociar ese carácter a, por ejemplo, el comando `\decontrac` de tal modo que cuando escribamos `\decontrac` se genere dicho carácter. Ese se conseguiría con:

```
\definecharacter decontrac Ð
```

Para construir un nuevo carácter que no está en nuestra fuente, y no se puede conseguir desde el teclado, como es el caso del ejemplo que acabo de poner, primero hay que buscar algún texto en el que dicho carácter se represente, para copiarlo y poderlo pegar en nuestra definición. En el ejemplo concreto que acabo de poner, el carácter «Ð» lo copié originalmente de la wikipedia.

ConT_EXt también incluye algunos comandos que permiten crear caracteres compuestos y que se pueden usar en combinación con `\definecharacter`. Por caracteres compuestos quiero decir caracteres con algún signo diacrítico. Estos comandos son los siguientes:


```

\buildmathaccent Acento Carácter
\buildtextaccent Acento Carácter
\buildtextbottomcomma Carácter
\buildtextbottomdot Carácter
\buildtextcedilla Carácter
\buildtextgrave Carácter
\buildtextmacron Carácter
\buildtextognek Carácter

```

Por ejemplo: Como ya sabemos, por defecto ConT_EXt sólo tiene comandos para escribir con cedilla algunas letras (c, k, l, n, r, s y t), que son, por otra parte, las letras que habitualmente se incorporan a la fuente con la cedilla. Si quisiéramos poder usar en nuestro documento una «b» con cedilla, podríamos usar el comando `\buildtextcedilla` del siguiente modo:

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

Este comando creará el nuevo comando `\bcedilla` que generará una «b» con cedilla («*ḃ*»). Estos comandos literalmente «construyen» el nuevo carácter, por lo que éste se generará aunque nuestra fuente no lo defina. Ello es porque, en realidad, lo que estos comandos hacen es superponer un carácter sobre otro y asociar un nombre a dicha superposición.

En mis pruebas no he conseguido hacer funcionar `\buildmathaccent` ni `\buildtextognek`. Por lo tanto a partir de aquí no los mencionaré más.

`\buildtextaccent` recibe dos caracteres como argumentos y superpone uno sobre el otro, elevando, ligeramente, uno de ellos. Aunque se denomina «`buildtextaccent`» no es imprescindible que alguno de los caracteres recibidos como argumento sea un acento; pero la superposición dará mejores resultados si lo es, pues en tal caso al superponer el acento sobre el carácter es menos probable que el acento sobrescriba al carácter. Por otra parte, la superposición de dos caracteres que en condiciones normales tienen la misma línea base, se ve afectada por el hecho de que el comando eleva ligeramente uno de los caracteres sobre el otro. Por eso no podemos usar este comando para, por ejemplo, obtener con él la contracción «*Đ*» de la que antes se ha hablado, pues si escribimos en nuestro fichero fuente

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

la ligera elevación sobre la línea base de la «D» que este comando produce, hace que el efecto no sea muy bueno («*Đ*»). Pero si las alturas de los caracteres lo permiten, podemos realizar cualquier combinación. Por ejemplo

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

definirá el carácter «*_*» que quedará asociado al comando `\caralarga`.

Los restantes comandos de construcción reciben como único argumento el carácter al que se le añadirá el diacrítico que cada comando genera. A continuación mostraré un ejemplo de cada uno de ellos, construido sobre la letra «z»:

- `\buildtextbottomcomma` añade una coma bajo el carácter que recibe como argumento («z»).
- `\buildtextbottomdot` añade un punto bajo el carácter que recibe como argumento («z»).
- `\buildtextcedilla` añade una cedilla bajo el carácter que recibe como argumento («z»).
- `\buildtextgrave` añade un acento grave sobre el carácter que recibe como argumento («z»).
- `\buildtextmacron` añade una pequeña barra bajo el carácter que recibe como argumento («z»).

`\buildtextgrave` parece a primera vista redundante respecto a `\buildtextaccent`; sin embargo si se comprueba el acento grave generado con el primero de estos dos comandos queda algo mejor. En el siguiente ejemplo se muestra el resultado de ambos comandos, a un tamaño de letra suficiente como para apreciar la diferencia:

$\grave{z} - \grave{z}$

10.1.6 Utilización de conjuntos de símbolos predefinidos

En «ConT_EXt Standalone» se incluye, junto con ConT_EXt propiamente dicho, varios conjuntos predefinidos de símbolos que podemos usar en nuestros documentos. Esos conjuntos se denominan «cc», «cow», «fontawesome», «jmn», «mvs» y «nav». Cada uno de esos conjuntos, a su vez incluye varios subconjuntos:

- **cc** incluye «cc».
- **cow** incluye «cownormal» y «cowcontour».
- **fontawesome** incluye «fontawesome».
- **jmn** incluye «navigation 1», «navigation 2», «navigation 3» y «navigation 4».
- **mvs** incluye «astronomic», «zodiac», «europe», «martinvogel 1», «martinvogel 2» y «martinvogel 3».
- **nav** incluye «navigation 1», «navigation 2» y «navigation 3».

La wiki menciona también un conjunto llamado `was` que incluye «wasy general», «wasy music», «wasy astronomy», «wasy astrology», «wasy geometry», «wasy physics» y «wasy apl». Pero en mi distribución no he conseguido localizarlo, y mis pruebas para intentar ver los símbolos que contiene han fracasado.

Para ver los símbolos concretos que contiene cada uno de estos conjuntos, se usa la siguiente sintaxis:

```
\usesymbols[Conjunto]
\showsymbolset[Subconjunto]
```

Por ejemplo: si queremos ver el conjunto de símbolos incluidos en «mvs/zodiac» deberíamos escribir en el fichero fuente:

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

y obtendríamos el siguiente resultado:

Aquarius	♒	♒
Aries	♈	♈
Cancer	♋	♋
Capricorn	♐	♐
Gemini	♊	♊
Leo	♌	♌
Libra	♎	♎
Pisces	♓	♓
Sagittarius	♐	♐
Scorpio	♏	♏
Taurus	♉	♉
Virgo	♍	♍

Obsérvese que junto con cada símbolo se indica el nombre del mismo. El comando `\symbol` nos permitirá usar alguno de los símbolos concretos. Su sintaxis es:

```
\symbol[Subconjunto][NombreSímbolo]
```

donde el subconjunto ha de ser uno de los subconjuntos asociados a algún conjunto que hayamos cargado previamente con `\usesymbols`. Así, por ejemplo, si queremos usar el símbolo astrológico asociado a Acuario (que se encuentran en mvs/zodiac) deberíamos escribir

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

con lo que obtendríamos el carácter «♒», el cual será considerado, a todos los efectos, un «carácter», siendo, en consecuencia, afectado por el tamaño de la fuente

que esté activo cuando se imprima. También podemos usar `\definecharacter` para asociar el símbolo de que se trate con un comando. Por ejemplo

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

creará un nuevo comando llamado `\Aries` que generará el carácter «♈».

También podemos usar estos símbolos, por ejemplo, en un entorno `itemize`. Por ejemplo:

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

producirá

```
☞ item
☞ item
  ☑ item
  ☑ item
☞ item
```

10.2 Formatos especiales de los caracteres

En sentido estricto, son comandos de *formato* los que afectan a la fuente utilizada, a su tamaño, estilo o variante. Estos comandos se explican en el capítulo ???. Pero desde un punto de vista *amplio* también podemos considerar comandos de formato a los que transforman de alguna manera los caracteres que reciben como argumento, alterando así su apariencia. En esta sección examinaremos algunos de estos comandos; otros como es el caso del texto subrayado o rayado, que se construye con líneas sobre o bajo el texto, se verán en la [sección 12.5](#).

10.2.1 Letras mayúsculas, minúsculas y pseudoversalitas

Las letras propiamente dichas pueden ir en mayúsculas o en minúsculas. Para ConTEXt las mayúsculas y las minúsculas son caracteres distintos, por lo tanto,

en principio, procesará las letras tal y como se hayan escrito. No obstante existe un grupo de comandos que nos permiten asegurarnos de que el texto que reciban como argumento se escriba siempre en mayúsculas o en minúsculas:

- `\word{texto}`: Convierte a letras minúsculas el texto recibido como argumento.
- `\Word{texto}`: Convierte a mayúsculas la primera letra del texto recibido como argumento.
- `\Words{texto}`: Convierte a mayúsculas la primera letra de cada una de las palabras que componen el texto recibido como argumento; el resto se escribirá con minúsculas.
- `\WORD{texto}` o `\WORDS{texto}`: Escriben con mayúsculas el texto recibido como argumento.

Muy parecidos a estos comandos son `\cap` y `\Cap`: también convierten a mayúsculas el texto que reciben como argumento, pero después le aplican un factor de escalado igual al que aplica el sufijo «x» en los comandos de cambio de fuente (véase la [sección 6.4.2](#)) con lo que, en la mayor parte de las fuentes, se obtiene la misma altura de las letras minúsculas, consiguiéndose así una especie de *pseudoversalitas* que, frente a las auténticas versalitas (véase [sección 6.5.2](#)) tienen las siguientes dos ventajas:

1. `\cap` y `\Cap` funcionarán con cualquier fuente, a diferencia de las auténticas versalitas, que sólo funcionan para aquellas fuentes y estilos en cuyo diseño se hayan incluido expresamente.
2. Las verdaderas versalitas son, por otra parte, una variante de la fuente que, como tal, es incompatible con cualquier otra variante como la negrita, la cursiva, o la letra inclinada. Sin embargo `\cap` y `\Cap` son plenamente compatibles con cualquier variante de la fuente.

La diferencia entre `\cap` y `\Cap` está en que mientras la primera aplica el factor de escalado a todas las letras de las palabras que constituyen su argumento, `\Cap` no aplica escalado alguno a la primera letra de cada palabra, consiguiendo así un efecto similar al que se obtiene cuando en un texto en versalitas se usan las verdaderas mayúsculas. Si el texto recibido como argumento en `\Cap` consta de varias palabras, se mantendrá el tamaño de la mayúscula en la primera letra de cada una de ellas.

Así en el siguiente ejemplo

La ONU, cuyo <code>\Cap{presidente}</code> tiene el despacho en la sede de la <code>\cap{oNu}</code> ...		La ONU, cuyo PRESIDENTE tiene el despacho en la sede de la ONU...
--	--	---

debe observarse, en primer lugar, la diferencia de tamaño entre la primera vez que se escribe «ONU» (con mayúsculas) y la segunda vez (con pseudoversalitas, «ONU»). En el ejemplo, la segunda vez he escrito en la fuente `\cap{oNu}` para que se compruebe que da igual que el argumento de `\cap` se escriba con mayúsculas o con minúsculas: el comando convierte todas las letras a mayúsculas y luego aplica el factor de reducción; a diferencia de `\Cap` que no reduce la primera letra.

Estos comandos, por otra parte, pueden *anidarse*, en cuyo caso se volvería a aplicar el factor de escalado, obteniéndose una mayor reducción, como en el siguiente ejemplo en el que la palabra «capital» de la primera línea aparece doblemente reducida:

```
\cap{La gente que ha reunido su
\cap{capital} a costa de otras
personas es con frecuencia
{\bf decapitada} en tiempos
revolucionarios}.
```

```
LA GENTE QUE HA REUNIDO SU CAPITAL A COSTA
DE OTRAS PERSONAS ES CON FRECUENCIA DE-
CAPITADA EN TIEMPOS REVOLUCIONARIOS.
```

El comando `\nocap` aplicado a un texto al que se está aplicando `\cap` anula, para el texto que constituya su argumento, el efecto de `\cap`. Por ejemplo:

```
\cap{Amaya tiene un gallo que no calla,
siempre está este gayo \nocap{apoyado}
en una valla}.
```

```
AMAYA TIENE UN GALLO QUE NO CALLA, SIEMPRE
ESTÁ ESTE GAYO apoyado EN UNA VALLA.
```

El funcionamiento de `\cap` podemos configurarlo mediante `\setupcapitals` y también podemos definir distintas versiones del comando, cada una de ellas con su propio nombre y configuración específica. Esto último se logra mediante `\definecapitals`.

Ambos comandos funcionan de un modo parecido:

```
\definecapitals[Nombre] [Configuración]
\setupcapitals[Nombre] [Configuración]
```

El parámetro «Nombre» en `\setupcapitals` es opcional, si no se usa la configuración afectará al comando `\cap` propiamente dicho. Si se usa, hay que poner el nombre que previamente hayamos asignado en `\definecapitals` a alguna configuración concreta.

La configuración, en cualquiera de los dos comandos, admite tres opciones: «title», «sc» y «style» la primera y la segunda admiten los valores «yes» y «no», mediante «title» indicamos si la capitalización afectará también a los títulos (por defecto sí lo hace) y mediante «sc» indicamos si el comando debe usar auténticas

versalitas («yes»), o pseudoversalitas («no»). Por defecto usa pseudoversalitas lo que tiene la ventaja de que el comando funciona aunque se esté usando una fuente que no haya implementado las versalitas. El tercer valor que se puede indicar «`style`» permite indicar un comando de estilo que se aplicará al texto afectado por `\cap`.

10.2.2 Texto en subíndice o superíndice

Ya sabemos (véase la [sección 3.1](#)) que, en modo matemático, los caracteres reservados «`_`» y «`^`» sirven para convertir en subíndice o superíndice al carácter o grupo inmediatamente posterior. Para lograr ese mismo efecto fuera del modo matemático ConT_EXt incluye los siguientes comandos:

- `\high{Texto}`: Escribe el texto recibido como argumento en formato de superíndice.
- `\low{Texto}`: Escribe el texto recibido como argumento en formato de subíndice.
- `\lohi{SubIndice}{SuperIndice}`: Escribe sus dos argumentos uno encima del otro: en la parte inferior el primer argumento, y en la superior el segundo, lo que constituye un efecto curioso:

`\lohi{abajo}{arriba}`

| arriba
| abajo

10.2.3 Texto verbatim

La expresión latina *verbatim* (de *verbum* = *palabra* + el sufijo *atim*), que podría traducirse por «literalmente» o «palabra por palabra», se usa en sistemas de procesamiento de texto como ConT_EXt para referirse a fragmentos de texto que no deben procesarse en absoluto, sino que deben volcarse, tal cual se escribieron, al fichero final. ConT_EXt dispone para ello del comando `\type`, pensado para textos cortos que no ocupen más de una línea y del entorno `typing` pensado para textos de más de una línea. Estos comandos son muy usados en libros que hablan de informática para recoger fragmentos de Código; y por ello ConT_EXt formatea, por defecto, estos textos con letra monoespaciada del tipo de máquina de escribir o terminal informática. En ambos casos el texto se envía al documento final sin *procesar*, lo que significa que en él pueden usarse caracteres reservados o caracteres especiales que serán transcritos *tal cual* en el fichero final. Asimismo, si como argumento de `\type`, o como contenido de `\starttyping`, se incluye un comando, este será *escrito* en el documento final, pero no ejecutado.

El comando `\type` tiene, además, la siguiente peculiaridad: su argumento se *puede* delimitar con llaves (como es normal en ConT_EXt), pero también se puede usar como delimitador del argumento cualquier otro carácter.

Cuando ConT_EXt lee el comando `\type` asume que el carácter inmediatamente posterior al nombre del comando que no sea un espacio en blanco, actuará como delimitador de su argumento; por lo que considera que el contenido del argumento empieza en el carácter siguiente, y acaba en el carácter anterior a la próxima aparición del *delimitador*.

Esto con algunos ejemplos se comprende mejor:

```
\type 1Tararí que te vi, Mariví1
\type |Tararí que te vi, Mariví|
\type zTararí que te vi, Marivíz
\type (Tararí que te vi, Mariví(
```

Obsérvese que en el primer ejemplo el primer carácter tras el nombre del comando es un «1», en el segundo un «|» y en el tercero una «z»; pues bien: en cada uno de esos casos ConT_EXt considerará que es argumento de `\type` todo lo que haya entre dicho carácter y la siguiente aparición del mismo carácter. Lo mismo ocurre en el último ejemplo, el cual, además, es muy aleccionador, porque en principio podríamos asumir que si el delimitador inicial del argumento es un «(», el final debería ser un «)», pero no es así ya que «(» y «)» son caracteres diferentes y `\type`, como he dicho, busca como delimitador de cierre el mismo carácter que se haya usado para delimitar el inicio del argumento.

Solo hay dos casos en los `\type` admite que el delimitador inicial y el final del argumento sean distintos:

- Si el delimitador inicial es el carácter «{», se considera que el delimitador final será «}».
- Si el delimitador inicial es «<<», se considera que el delimitador final será «>>». Este caso, además, es el único en que se usan dos caracteres consecutivos como delimitadores.

Ahora bien: que `\type` admita cualquier delimitador no significa que nosotros debamos usar delimitadores “raros”. Desde el punto de vista de la *legibilidad* y *comprensibilidad* del fichero fuente, lo mejor es delimitar el argumento de `\type` siempre que se pueda con llaves, tal y como es normal en ConT_EXt; y cuando ello no sea posible, porque en el argumento de `\type` hay llaves, usar algún símbolo; preferentemente uno que no constituya un carácter reservado de ConT_EXt. Por ejemplo: `\type *Esto es una llave de cierre: <}*>`.

Tanto `\type` como `\starttyping` se pueden configurar mediante `\setuptype` y `\setuptyping`. También podemos crear una versión personalizada de ellos mediante `\definetyping` y `\definetype`. Respecto de las concretas opciones de configuración de estos comandos, me remito a «[setup-en.pdf](#)» (en el directorio `tex/texmf-context/doc/context/documents/general/qrcs`).

Dos comandos muy parecidos a `\type` son:

- `\typ`: Funciona de modo similar a `\type`, pero no inhabilita la partición silábica de palabras.
- `\tex`: Comando pensado para escribir textos sobre T_EX o sobre ConT_EXt: añade una barra invertida delante del texto que recibe como argumento. Por lo demás,

este comando, se diferencia de `\type` en que procesa algunos de los caracteres reservados que encuentre dentro del texto que recibe como argumento. En particular, dentro de `\tex` las llaves serán tratadas del modo habitual en ConT_EXt.

10.3 Espacios de separación entre caracteres y palabras

10.3.1 Fijación automática del espacio horizontal

El espacio entre los distintos caracteres y palabras (llamado en T_EX *espacio horizontal*) normalmente es fijado automáticamente por ConT_EXt:

- El espacio de separación entre los caracteres que forman una palabra viene definido por la propia fuente, que, salvo en fuentes de anchura fija, suele utilizar una cantidad mayor o menor de espacio en blanco dependiendo de los caracteres que hay que separar, y así, por ejemplo, el espacio entre una «A» y una «V» («AV») suele ser menor que el existente entre una «A» y una «X» («AX»). Pero, aparte de esas posibles variaciones, que dependen de la combinación de letras de que se trate y que vienen ya predefinidas por la fuente, el espacio entre los caracteres que componen una palabra es, en general, una medida fija e invariable.
- Por el contrario el espacio de separación entre palabras dentro de una misma línea tiende a ser más elástico.
 - Tratándose de palabras en una línea cuya anchura ha de ser igual que la del resto de líneas del párrafo, la variación del espaciado entre palabras es uno de los mecanismos que ConT_EXt utiliza para obtener líneas de igual anchura, tal y como se explica con algo más de detalle en la [sección 11.3](#). En estos casos ConT_EXt establecerá exactamente el mismo espacio horizontal entre todas las palabras de la línea (salvo las reglas que más adelante se indican), al tiempo que procurará que la separación entre palabras en las distintas líneas del párrafo sea lo más parecida posible.
 - Pero, además de las necesidades de estiramiento o encogimiento del espaciado entre palabras para poder justificar las líneas, ConT_EXt, dependiendo del idioma activo, toma en consideración determinadas reglas tipográficas en virtud de las cuales en ciertos lugares la tradición tipográfica asociada a dicho idioma añadía algo de espacio en blanco extra, como ocurre, por ejemplo, en la tradición tipográfica inglesa, que añade espacio en blanco adicional tras un punto y seguido.

Estos espacios blancos extra funcionan para el idioma inglés, y posiblemente para algunos otros; pero no para el español, donde la tradición tipográfica es diferente. Podemos activar temporalmente esta función mediante `\setupspacing[broad]` y desactivarla mediante `\setupspacing[packed]`. También podemos cambiar la configuración por defecto del idioma español, como se explica en la [sección 10.5.2](#).

10.3.2 Alteración del espacio entre caracteres dentro de una palabra

La alteración del espacio predeterminado para los caracteres que componen una palabra está considerada, desde el punto de vista tipográfico, una muy mala práctica, salvo en títulos y encabezados. No obstante ConT_EXt proporciona un comando para alterar este espacio de separación entre los caracteres de la palabra¹: `\stretched`, cuya sintaxis es la siguiente:

`\stretched[Configuración]{Texto}`

donde *Configuración* admite cualquiera de las siguientes opciones:

- **factor**: un número entero o decimal representativo del espaciado que se pretende obtener. No debe ser un número demasiado alto. Un factor de 0.05 ya es observable a simple vista.
- **width**: Indica la anchura que ha de tener, en total, el texto sometido al comando, de tal manera que el propio comando calculará el espaciado necesario para repartir los caracteres en dicho espacio.



De acuerdo con mis pruebas, cuando la anchura establecida con la opción `width` es inferior a la necesaria para representar el texto con un *factor* igual a 0.25, se ignora la opción *anchura* y se aplica dicho factor. Supongo que eso es porque `\stretched` permite sólo *aumentar* el espacio de separación entre los caracteres de una palabra, no reducirlo. Pero no termino de entender por qué se utiliza como medida mínima para la opción `width` la anchura necesaria para representar el texto con un factor de 0.25, y no la *anchura natural* del texto (con un factor 0).

- **style**: Comando o comandos de estilo a aplicar al texto recibido como argumento.
- **color**: Color en el que se escribirá el texto recibido como argumento.

Así en el siguiente ejemplo puede verse gráficamente cómo funcionaría el comando aplicado a la misma frase, pero con diferentes anchuras:

¹ Es muy propio de la filosofía de ConT_EXt incluir un comando para hacer algo que la documentación del propio ConT_EXt desaconseja hacer. Y es que aunque se busca la perfección tipográfica, también se pretende conceder al autor el control absoluto sobre la apariencia de su documento: que esta sea mejor o peor, es, en definitiva, su responsabilidad.

```
\stretched[width=4cm]{\bf texto de prueba}
\stretched[width=6cm]{\bf texto de prueba}
\stretched[width=8cm]{\bf texto de prueba}
\stretched[width=9cm]{\bf texto de prueba}
```

```
t e x t o   d e   p r u e b a
t e x t o   d e   p r u e b a
t e x t o   d e   p r u e b a
t e x t o   d e   p r u e b a
```



En este ejemplo se puede observar que el reparto del espacio horizontal entre los distintos caracteres no es uniforme. La «x» y la «t» de «texto» y la «e» y la «b» de «prueba», aparecen siempre mucho más próximos entre sí que el resto de los caracteres. No he conseguido averiguar por qué ocurre esto.

Aplicado sin argumentos, el comando usará toda la anchura disponible en la línea. De otro lado, dentro del texto que es argumento de este comando, el comando `\` se redefine y en lugar de un salto de línea, inserta espacio horizontal. Por ejemplo:

```
\stretched{texto\\de\\prueba}          t e x t o           d e           p r u e b a
```

Podemos personalizar la configuración por defecto del comando mediante `\setupstretched`.



No hay un comando `\definestretched` que permita establecer configuraciones personalizadas asociadas a un nombre de comando, pero en el listado oficial de comandos (véase [sección 3.6](#)) se dice que `\setupstretched` deriva de `\setupcharacterkerning` y sí hay un comando `\definecharacterkerning`. En mis pruebas, sin embargo, no he llegado a conseguir establecer, mediante este último, una configuración personalizada para `\stretched`, aunque he de admitir que tampoco he dedicado demasiado tiempo a intentarlo.

10.3.3 Comandos para añadir espacio horizontal entre palabras

Ya sabemos que para aumentar el espacio de separación entre palabras, no sirve de nada escribir en el fichero fuente dos o más espacios en blanco consecutivos, pues ConTeXt absorbe todos los espacios en blanco consecutivos, tal y como ya se explicó en la [sección 4.2.1](#). Si queremos aumentar la separación entre dos palabras, debemos acudir a alguno de los comandos que permiten hacerlo:

- `\,` inserta en el documento un muy pequeño espacio en blanco. Se usa, por ejemplo, para separar los millares en una cifra, o para separar una comilla simple de unas dobles comillas. Por ejemplo: «`1\,473\,451`» producirá «1 473 451».
- `\space` o «`_`» (una barra invertida seguida de un espacio en blanco que, como es un carácter invisible, he representado como «`_`») introducen un espacio en blanco adicional.
- `\enskip`, `\quad` y `\qquad` insertan en el documento un espacio en blanco de, respectivamente, medio *em*, 1 *em* o 2 *ems*. Recordemos que el *em* es una medida dependiente del tamaño de la fuente y equivale a la anchura de una «m», la

cual normalmente coincide con el tamaño en puntos de la fuente. Así, usando una fuente de 12 puntos, `\enskip` introduce un espacio de 6 puntos, `\quad` introduce 12 puntos y `\qquad` 24 puntos.

Junto a estos comandos, que introducen espacio en blanco de medida exacta, los comandos `\hskip` y `\hfill` introducen un espacio horizontal de dimensiones variables:

`\hskip` permite indicar exactamente la cantidad de espacio en blanco que queremos añadir. Así

<code>Esto es \hskip 1cm 1 centímetro\\</code>		Esto es	1 centímetro
<code>Esto son \hskip 2cm 2 centímetros\\</code>		Esto son	2 centímetros
<code>Esto son \hskip 2.5cm 2.5 centímetros\\</code>		Esto son	2.5 centímetros

El espacio indicado puede ser negativo, lo que hará que un texto se superponga sobre otro. Así

<code>Esto es una farsa más que una \hskip -1cm comedia</code>		Esto es una farsa más que comedia
--	--	--

`\hfill`, por su parte, introduce tanto espacio en blanco como sea necesario para ocupar toda la línea, lo que nos permite crear efectos interesantes tales como texto alineado a la derecha, texto centrado o texto a ambos lados de la línea tal y como se muestra en el siguiente ejemplo

<code>\hfill A la derecha\\</code>		A la derecha
<code>A ambos\hfill lados</code>		A ambos

10.4 Palabras compuestas

A efectos de esta sección por «palabras compuestas» me refiero, no a palabras que resultan de la unión de otras pero que formalmente son una sola palabra, como por ejemplo, «paraguas», «parabrisas» o «francófono»; sino a palabras distintas que se unen entre sí mediante algún elemento: normalmente un guión, pero, en ocasiones, algún otro elemento (principalmente guiones bajos («_») barras («\» o «/») o signos de apertura o cierre de paréntesis, corchetes o llaves; como en, por ejemplo, «franco–canadiense» o en «(inter)comunicación».

Las palabras compuestas plantean a ConTeXt algunos problemas principalmente relacionados con su posible partición silábica al final de una línea. Sobre todo si el

elemento separador es un guión, pues entonces, desde una perspectiva tipográfica, no habría problemas en permitir que el primer elemento de la unión de palabras se partiera silábicamente al final de una línea, pero debería evitarse que fuera el segundo elemento el que se partiera, pues ello dejaría en la palabra dos guiones muy seguidos, cada uno de ellos con una significación diferente.

Para indicar a ConT_EXt que dos palabras constituyen una palabra compuesta, está disponible el comando «| |» que, excepcionalmente, no empieza por una barra invertida, y admite dos usos diferentes:

- Se pueden usar las dos barras consecutivas y escribir, por ejemplo «his-pano||argentino».
- Se pueden usar encerrando entre ellas el elemento de separación/unión de ambas palabras como, por ejemplo, en «separación|/|unión».

En ambos casos ConT_EXt sabrá que se encuentra ante una palabra compuesta, y aplicará las reglas de partición silábica adecuadas para este tipo de palabras. La diferencia entre usar las dos barras consecutivas, o hacerlo enmarcando con ellas el elemento separador de las palabras está en que, en el primer caso, ConT_EXt usará como elemento separador el que tenga preconfigurado en `\setuphyphenmark` que, por defecto, es un guión intermedio («--»); y así si en nuestro documento fuente escribimos «escuela|taller», ConT_EXt generará «Escuela-taller».

Mediante `\setuphyphenmark` podemos cambiar el elemento de separación por defecto (para el caso de que se escriban las dos barras consecutivas). Como valores para este comando se admiten «--», «---», «-», «.», «(», «)», «=», «/». Téngase en cuenta, no obstante, que el valor «=» se traduce en un guión largo (igual que «---»).

El uso normal de «| |» es con guiones, pues son éstos los que habitualmente se utilizan para unir palabras compuestas. Pero en ocasiones el separador de las palabras compuestas puede ser un paréntesis, como, por ejemplo, en «(inter)espacio», o una barra inclinada, como en «contador/partidor». En estos casos, si queremos que se apliquen las reglas de división silábica de las palabras compuestas, podemos escribir «(inter|)|espacio» o «contador|/|partidor». Como ya se ha dicho antes, «|=|» se considera una abreviatura de «|---|» e inserta como elemento separador un guión largo (—).

10.5 El idioma del texto

Los caracteres forman palabras, las cuales normalmente pertenecerán a algún idioma. Para ConT_EXt es importante saber en qué idioma se está escribiendo, pues de cuál sea éste dependen varios aspectos importantes. Principalmente:

- El particionado silábico de las palabras.
- El formato de la salida de ciertos comandos.
- Ciertos aspectos de la composición del texto que van asociados a la tradición tipográfica del idioma de que se trate.

10.5.1 Fijación y cambio del idioma

En ConT_EXt se asume que el idioma será el inglés. Eso puede cambiarse por dos procedimientos:

- Mediante el comando `\mainlanguage`, concebido para fijar el idioma principal del documento en el preámbulo.
- Mediante el comando `\language`, pensado para cambiar el idioma activo en cualquier punto del documento.

Ambos comandos esperan un argumento consistente en el identificador del idioma que sea. Para identificar al idioma puede usarse, bien el código internacional de identificación del mismo de dos letras establecido en ISO 639-1 que es el mismo que se usa, por ejemplo, en la web, bien el nombre en inglés del idioma de que se trate, bien, en ocasiones, alguna abreviatura del nombre en inglés.

En la [tabla 10.5](#) se contiene el listado completo de los idiomas soportados por ConT_EXt, junto con los identificadores admisibles para cada uno de los idiomas en cuestión así como, en su caso, los identificadores de ciertas variantes del idioma expresamente previstas¹,

Así, por ejemplo, para establecer el español (castellano) como idioma principal del documento podemos usar cualquiera de las siguientes tres sentencias.

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

Para activar un concreto lenguaje *dentro* del documento, podemos usar, bien el comando `\language[Identificador del idioma]`, bien un comando específico que active dicho idioma. Así, por ejemplo, `\en` activa el idioma inglés, `\fr` activa el francés, `\es` el español o `\ca` el catalán. Una vez activado un idioma concreto, se mantendrá activado hasta que expresamente se active otro idioma, o se cierre

¹ La [tabla 10.5](#) es un resumen del listado que se obtiene con los siguientes comandos:

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

Si usted está leyendo este documento mucho después de la fecha de su confección (2020) es posible que se hayan incorporado algunos idiomas adicionales a ConT_EXt, por lo que puede ser buena idea ejecutar esos comandos para ver un listado de idiomas actualizado.

Idioma	Identificadores	Variantes (del idioma)
Afrikáans	af, afrikaans	
Alemán	de, deu, german	de-at, de-ch, de-de
Árabe	ar, arabic	ar-ae, ar-bh, ar-dz, ar-eg, ar-in, ar-ir, ar-jo, ar-kw, ar-lb, ar-ly, ar-ma, ar-om, ar-qa, ar-sa, ar-sd, ar-sy, ar-tn, ar-ye
Catalán	ca, catalan	
Checo	cs, cz, czech	
Coreano	kr, korean	
Croata	hr, croatian	
Danés	da, danish	
Eslovaco	sk, slovak	
Esloveno	sl, slovene, slovenian	
Español	es, sp, spanish	es-es, es-la
Estonio	et, estonian	
Finés	fi, finnish	
Francés	fr, fra, french	
Griego	gr, greek	
Griego antiguo	agr, ancientgreek	
Hebreo	he, hebrew	
Holandés	nl, nld, dutch	
Húngaro	hu, hungarian	
Inglés	en, eng, english	en-gb, uk, ukenglish, en-us, usenglish
Italiano	it, italian	
Japonés	ja, japanese	
Latín	la, latin	
Lituano	lt, lithuanian	
Malayo	ml, malayalam	
Noruego	nb, bokmal, no, norwegian	nn, nynorsk
Persa	pe, fa, persian	
Polaco	pl, polish	
Portugués	pt, portuguese	pt-br
Rumano	ro, romanian	
Ruso	ru, russian	
Sueco	sv, swedish	
Tailandés	th, thai	
Turco	tr, turkish	tk, turkmen
Ucraniano	ua, ukrainian	
Vietnamita	vi, vietnamese	

Tabla 10.5 Idiomas soportados en ConTeXt

el grupo dentro del cual el idioma se activó. Los idiomas funcionan, por lo tanto, igual que los comandos de cambio de fuente. Téngase en cuenta, no obstante, que el lenguaje establecido por el comando `\language` o por alguna de sus abreviaturas (`\en`, `\fr`, `\de`, etc.) no afecta al lenguaje en el que se imprimirán las etiquetas (véase la [sección 10.5.3](#)).

Aunque pueda ser trabajoso marcar el idioma de todas las palabras y expresiones que usemos en nuestro documento y que no pertenezcan al idioma principal del mismo, es importante hacerlo si queremos obtener un documento final bien compuesto. Sobre todo en trabajos profesionales. No hay que marcar todo el texto, sino sólo el que no vaya en el idioma principal. A veces es posible automatizar el marcado del idioma mediante alguna macro. Yo, por ejemplo, para este documento en el que se citan continuamente comandos de ConTeXt cuyo idioma original es el inglés, he diseñado una macro que, además de escribir el comando en el formato y color adecuado, lo marca como palabra inglesa. En mi trabajo profesional, donde necesito citar mucha

bibliografía francesa e italiana, tengo incorporado un campo en mi base de datos bibliográfica para recoger el idioma de la obra, de tal manera que puedo automatizar, en las citas y listas de referencias bibliográficas, la indicación del idioma.

Si en un mismo documento conviven dos idiomas que usan alfabetos diferentes (por ejemplo, español y griego, o español y ruso cirílico), hay un truco que nos evitará tener que marcar el idioma de las expresiones construidas con el alfabeto alternativo: modificar la configuración del idioma principal (véase la próxima sección) para que cargue por defecto también los patrones de particionado silábico correspondientes al idioma que usa un alfabeto distinto. Por ejemplo, si queremos usar simultáneamente el español y el griego, el siguiente comando nos ahorraría tener que marcar el idioma de los textos en griego:

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

Esto funciona sólo porque español y griego usan un alfabeto distinto, de tal modo que no es posible que se plantee ningún conflicto en los patrones de particionado de ambos idiomas, por lo que podemos cargarlos ambos simultáneamente. Pero en dos idiomas que usen el mismo alfabeto, cargar simultáneamente los patrones de particionado llevará necesariamente a particiones silábicas inadecuadas.

10.5.2 Configuración del idioma

ConT_EXt asocia el funcionamiento de ciertas utilidades al concreto idioma activo en cada momento. Las asociaciones por defecto podemos cambiarlas mediante `\setuplanguage` cuya sintaxis es:

```
\setuplanguage[Idioma][Configuración]
```

donde *Idioma* es el identificador del idioma que pretendemos configurar, y *Configuración* contiene la concreta configuración que deseamos establecer (o cambiar) para dicho idioma. En concreto se admiten hasta 32 opciones de configuración distinta de las que aquí trataré sólo las que me parecen adecuadas para un texto introductorio como este:

- **date**: Permite configurar el formato por defecto de la fecha. Véase, más adelante en la [página 218](#).
- **lefthyphenmin**, **righthyphenmin**: el número mínimo de caracteres que deben quedar a la izquierda o derecha para que se admita la partición silábica de una palabra. Por ejemplo `\setuplanguage[es][lefthyphenmin=4]` no partirá silábicamente ninguna palabra si en el lado izquierdo del guión de partición quedan menos de 4 caracteres.
- **spacing**: Esta opción admite dos valores posibles «**broad**» o «**packed**», en el primer caso se aplicarán en dicho idioma las reglas de separación entre palabras propias del inglés, que implican que detrás de los puntos y seguido y de otros caracteres se añada cierto espacio en blanco extra. Por el contrario «**spacing=packed**» inhabilitará estas reglas. Para el idioma español por defecto se encuentran inhabilitadas.

- **leftquote, rightquote:** Indican, respectivamente, los caracteres (o comandos) que usará el comando `\quote` a la izquierda y derecha del texto que constituye su argumento (véase, sobre este comando, la [página 220](#)).
- **leftquotation, rightquotation:** Indican, respectivamente, los caracteres (o comandos) que usará el comando `\quotation` a la izquierda y derecha del texto que constituye su argumento (véase, sobre este comando, la [página 220](#)).

10.5.3 Etiquetas asociadas a los concretos idiomas

Muchos comandos de ConT_EXt generan automáticamente ciertos textos (o *etiquetas*), como, por ejemplo, el comando `\placetable` que escribe, bajo la tabla que se inserta, la etiqueta “Tabla xx”, o `\placefigure` que inserta la etiqueta “Figura xx”.

Estas *etiquetas* son sensibles al idioma establecido con `\mainlanguage` (pero no al establecido con `\language`) y podemos cambiarlas mediante

```
\setuplabeltext[Idioma][Clave=Etiqueta]
```

donde *Clave* es la denominación por la que ConT_EXt conoce a dicha etiqueta y *Etiqueta* es el texto que queremos que genere ConT_EXt. Así, por ejemplo

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

hará que, cuando esté activo el idioma español, las imágenes insertadas con `\placefigure` no se titulen “Figura x”, sino “Imagen x”. Obsérvese que tras el texto de la etiqueta propiamente dicha hay que dejar un espacio en blanco para asegurarse de que ConT_EXt no pega la etiqueta al siguiente carácter. En el ejemplo he usado el carácter reservado «~»; podría también haber escrito «`[figure=Imagen{ }]`» encerrando el espacio en blanco entre llaves para asegurarme de que ConT_EXt no lo eliminará.

¿Qué etiquetas podemos redefinir mediante `\setuplabeltext`? En este punto la documentación de ConT_EXt no es lo completa que cabría esperar. El manual de referencia de 2013 (que es el que más cosas explica sobre este comando) menciona «chapter», «table», «figure», «appendix»... y, añade, “otros elementos textuales comparables”. Podemos suponer que los nombres serán los nombres en inglés del elemento en cuestión.



Una de las ventajas del *software libre* es la de que los ficheros fuente están disponibles para el usuario; de modo que podemos indagar en ellos. Yo lo he hecho, y *husmeando* entre los ficheros fuente de ConT_EXt, he descubierto el fichero «`lang-txt.lua`», disponible en `tex/texmf-context/tex/context/base/mkiv` que creo que es el que contiene las etiquetas predeterminadas y sus distintas traducciones; de modo que si en algún momento ConT_EXt genera un texto predeterminado que queremos cambiar, para ver cómo se llama la etiqueta a que dicho texto está

asociado, podemos abrir el fichero en cuestión y buscar en él el texto que queremos cambiar, para ver a qué nombre de etiqueta se le asocia.

Si queremos insertar en algún punto del documento el texto asociado a cierta etiqueta, podemos hacerlo con el comando `\labeltext`. Y así, por ejemplo, si quiero referirme a una tabla, para asegurarme de que la denomino de la misma forma en que ConT_EXt la llamará en el comando `\placetable`, puede escribir: «Tal y como se muestra en la `\labeltext{table}` de la próxima página.» Dicho texto, en un documento donde `\mainlanguage` sea el español, producirá: «Tal y como se muestra en la Tabla de la próxima página.»

Algunas de las etiquetas redefinibles mediante `\setuplabeltext`, por defecto vienen vacías; como, por ejemplo «chapter» o «section». Esto es así porque, por defecto, ConT_EXt no añade etiquetas a los comandos de seccionado. Si queremos cambiar ese funcionamiento por defecto, basta con redefinir dichas etiquetas en el preámbulo de nuestro documento y así, por ejemplo, `\setuplabeltext[chapter=Capítulo~]` hará que los capítulos sean precedidos de la palabra “Capítulo”.

Por último, es importante indicar que si bien en general, en ConT_EXt, los comandos que admiten como argumento varias opciones separadas por comas, la última opción puede terminar con una coma y no pasa nada, en `\setuplabeltext` eso generaría un error de compilación.

10.5.4 Algunos comandos vinculados al idioma

A. Comandos relacionados con la fecha

ConT_EXt dispone de tres comandos relacionados con la fecha que producen su salida en el idioma activo en el momento de su ejecución. Se trata de:

- `\currentdate`: Ejecutado sin argumentos en un documento en el que el idioma principal sea el español, devuelve la fecha del sistema en el formato “NumDía de NombreMes de NumAño”. Por ejemplo: «28 de julio de 2020». Pero también podemos indicarle que use un formato distinto, o que incluya también el nombre del día de la semana (`weekday`), o que sólo incluya algunos elementos de la fecha (`day`, `month`, `year`).

Para indicar un formato de fecha diferente, «dd» o «day» representan los días, «mm» los meses (en formato numérico), «month» los meses en formato alfabético con minúsculas y «MONTH» con mayúsculas. Respecto del año «yy» escribirá sólo las dos últimas cifras, mientras que «year» o «y» escribirán las cuatro cifras. Si queremos algún elemento separador entre los componentes de la fecha, debemos escribirlo expresamente. Por ejemplo

```
\currentdate[weekday,{,~}, dd, {~de~}, month]
```

ejecutado el 28 de julio de 2020 escribirá en nuestro documento «martes, 28 de julio». Obsérvese que para representar el espacio en blanco he usado el carácter reservado «~». Podría haber usado espacios en blanco normales, pero éstos deben necesariamente encerrarse entre llaves.

- `\date`: Este comando, ejecutado sin ningún argumento, produce exactamente la misma salida que `\currentdate`, es decir: la fecha actual en el formato estándar. Pero puede indicársele, como argumento, una fecha concreta. Para ello recibe dos argumentos: Con el primer argumento podemos indicar el día («d»), mes («m») y año («y») correspondientes a la fecha que queremos representar, mientras que con el segundo argumento (opcional) podemos indicar el formato de la fecha a representar. Por ejemplo, si queremos saber qué día de la semana era el día en que se conocieron John Lennon y Paul McCartney, suceso que, según la wikipedia, tuvo lugar el día 6 de julio de 1957, podríamos escribir

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

y así averiguaríamos que tal evento histórico sucedió un sábado.

- `\month` recibe como argumento un número, y devuelve el nombre del mes correspondiente a dicho número.

B. El comando `\translate`

El comando `translate` admite una serie de frases asociadas a un concreto idioma, de tal manera que en el documento final se insertará una u otra dependiendo del idioma activo en cada momento. Así en el siguiente ejemplo se usa el comando `translate` para asociar cuatro frases al español y al inglés, las cuales se guardan en un buffer de memoria (sobre el entorno `buffer` véase en la [sección 12.6](#)):

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

de tal modo que si insertamos el *buffer* en un punto del documento en el que esté activado el idioma español, se reproducirán las frases en español, pero si el punto del documento donde se inserta el buffer tiene activado el idioma inglés, se insertarán las frases en inglés. Así:

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

generaría

Su carta de fecha

Su referencia

Nuestra referencia

Fecha

mientras que

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

generaría

Your letter dated

Your reference

Our reference

Date

C. Los comandos `\quote` y `\quotation`

Uno de los errores tipográficos más corrientes en los documentos de texto se produce cuando en ellos se abren comillas (simples o dobles) que no se cierran expresamente. Para evitar que esto ocurra ConTeXt proporciona los comandos `\quote` y `\quotation` que entrecomillarán el texto que reciban como argumento; `\quote` usará comillas simples y `\quotation` comillas dobles.

Estos comandos son sensibles al idioma en el sentido de que usan para abrir y cerrar las comillas el carácter o comando que se haya establecido por defecto para el idioma en cuestión (véase [sección 10.5.2](#)); y así, por ejemplo, si queremos que al utilizar español se usen, como comillas dobles por defecto, las angulares propias de nuestra tradición tipográfica, podemos escribir:

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

Estos comandos no gestionan, sin embargo, las comillas anidadas; aunque podemos crear nosotros la utilidad que si lo haga, aprovechando que, en realidad, `\quote` y `\quotation` son aplicaciones concretas de lo que ConTeXt llama *delimitedtext* (= *texto delimitado*), y que es posible definir otras aplicaciones mediante `\definedelimitedtext`. Así el siguiente ejemplo:

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

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

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

creará tres comandos que permitirán hasta tres niveles diferentes de entrecomillado. El primer nivel con comillas laterales, el segundo con comillas dobles altas y el tercero con comillas simples altas.

Capítulo 11

Párrafos, líneas y espaciado vertical

Sumario: 11.1 Formación y características de los párrafos; 11.1.1 Sangría automática de la primera línea de los párrafos; 11.1.2 Sangrado especial de párrafos; 11.2 Espaciado vertical entre párrafos; 11.2.1 `\setupwhitespace`; 11.2.2 Párrafos sin espacio vertical extra entre ellos; 11.2.3 Añadir espacio vertical adicional en un punto concreto del documento; 11.2.4 `\setupblank` y `\defineblank`; 11.2.5 Otros procedimientos para conseguir espacio vertical extra; 11.3 Cómo construye ConTeXt las líneas que forman los párrafos; 11.3.1 Uso del carácter reservado «~»; 11.3.2 Partición silábica de las palabras; 11.3.3 Nivel de tolerancia en la ruptura de líneas; 11.3.4 Forzar un salto de línea en cierto punto; 11.4 Interlineado; 11.5 Otras cuestiones relacionadas con las líneas; 11.5.1 Convertir los saltos de línea del fichero fuente en saltos de línea en el documento final; 11.5.2 Numeración de líneas; 11.6 Alineación horizontal y vertical; 11.6.1 Alineación horizontal; 11.6.2 Alineación vertical;

El aspecto general de un documento está principalmente determinado por el tamaño y diseño de las páginas, que se ha visto en el [capítulo 5](#), por la fuente elegida, que se ha tratado en el [capítulo ??](#) y por otros aspectos tales como el interlineado, la alineación de los párrafos, el espaciado entre ellos, etc. Este capítulo se centra en estos otros aspectos.

11.1 Formación y características de los párrafos

Para ConTeXt el párrafo es la unidad de texto fundamental. Para iniciar un párrafo se dispone de dos procedimientos:

1. Insertar en el fichero fuente una o más líneas en blanco consecutivas.
2. Los comandos `\par` o `\endgraf`.

Normalmente se utiliza el primer procedimiento, pues es más simple, y produce ficheros fuente más fáciles de leer y comprender. Insertar saltos de párrafo mediante un comando explícito es algo que habitualmente sólo se hace dentro de la

definición de una macro (véase la [sección 3.7.1](#)) o en una celda de una tabla (véase la [sección 13.3](#)).

En un documento bien compuesto, desde el punto de vista tipográfico, conviene que los párrafos se destaquen visualmente unos de otros con claridad. Ello se suele hacer por dos procedimientos: Sangrando ligeramente la primera línea de cada párrafo o aumentando ligeramente el espacio en blanco entre los párrafos. En ocasiones se usa una combinación de ambos procedimientos aunque en algunos lugares se desrecomienda hacerlo por considerarlo tipográficamente redundante.

Así lo hace, por ejemplo, el *Libro de Estilo de la lengua española* (Espasa 2018). No estoy totalmente de acuerdo. La simple sangría de la primera línea no siempre destaca visualmente lo bastante la separación entre párrafos; pero el aumento del espaciado no acompañado de la sangría, plantea problemas en el caso de párrafos que empiezan en una página sin que, a veces, sea fácil saber si se trata de un párrafo nuevo o del mismo párrafo de la página anterior. Una combinación de ambos procedimientos elimina las dudas.

Veremos, en primer lugar, cómo funciona en ConT_EXt el sangrado de líneas o párrafos.

Nota terminológica: A lo que en la tradición tipográfica española se conoce como *sangría* o *sangrado* de líneas, hoy se le llama también, a veces, *indentación*; palabra esta que no está reconocida por el Diccionario de la RAE y que procede del mundo de la informática, no del de la tipografía.

Los nombres de los comandos de ConT_EXt que se refieren a esta práctica, parten, sin embargo, del verbo *indentar*, y por ello, aunque en las líneas que siguen usaré preferentemente la terminología tipográfica, también usaré ocasionalmente el verbo *indentar* o el sustantivo *indentación*.

11.1.1 Sangría automática de la primera línea de los párrafos

La introducción automática de una pequeña sangría en la primera línea de los párrafos, se encuentra desactivada por defecto. Podemos activarla, volverla a desactivar, e indicar, cuando está activada, el tamaño de la sangría, mediante `\set-upindenting` que admite los siguientes valores para indicar si la indentación debe o no estar activada:

- **always:** Indentará absolutamente todos los párrafos.
- **yes:** Activa el sangrado de los párrafos *normales*. No se indentarán ciertos párrafos que van precedidos de un espaciado vertical extra como, por ejemplo, el primer párrafo de las secciones, o los párrafos que siguen a ciertos entornos.
- **no, not, never, none:** Desactiva el sangrado automático de la primera línea de los párrafos.

Para el caso de que hayamos activado el sangrado automático, podemos también indicar, mediante el mismo comando, cuánto sangrado debe haber. Para ello

podemos usar expresamente una dimensión (por ejemplo 1.5cm) o las palabras simbólicas «`small`», «`medium`» y «`big`» que indican, respectivamente, que se desea un sangrado pequeño, mediano o grande.

En la tradición tipográfica española el sangrado estándar era de dos cuadratines. El «cuadratín» es, en tipografía, un cuadrado cuya altura y anchura es igual al cuerpo de la letra que se esté usando. Así, con una letra de 12 puntos, el cuadratín mediría 12 puntos de ancho por doce de alto. Como su uso fundamental es para separar elementos de texto, también se llama cuadratín al espacio en blanco de esa misma medida; en inglés a un espacio en blanco de ese tamaño se le denomina *quad*, y en ConT_EXt existen los comandos `\quad`, que genera un espacio en blanco de exactamente un cuadratín, y `\qqquad`, que genera un espacio en blanco de dos cuadratines. Un sangrado de dos cuadratines, con una letra de 11 puntos, mediría 22 puntos, y con una letra de 12 puntos, 24 puntos.

Si, estando activada la indentación, deseamos indicar que en cierto párrafo no se aplique, debemos utilizar el comando `\noindentation`.

Yo, en líneas generales, activo el sangrado automático en mis documentos, estableciendo `\set-upindenting[yes, big]`. En este documento, sin embargo, no lo he hecho pues la gran cantidad de frases cortas y ejemplos que en él hay, produciría, si se activara la indentación, una apariencia de las páginas visualmente muy desordenada.

11.1.2 Sangrado especial de párrafos

Un procedimiento gráfico que destaca bien un párrafo es el de aumentar el sangrado derecho o izquierdo (o ambos) de todo el párrafo. Esto se hace, por ejemplo, para representar las citas textuales lo suficientemente largas como para ocupar todo un párrafo.

En ConT_EXt existe un entorno que permite alterar el sangrado de los párrafos, destacándolos así del texto que les rodea. Se trata de «`narrower`»:

`\startnarrower[Opciones] ... \stopnarrower`

donde *Opciones* pueden ser:

- **left**: Sangrar el margen izquierdo.
- **Num*left**: Sangrar el margen izquierdo, multiplicando el sangrado *normal* por *Num* (por ejemplo `2*left`).
- **right**: Sangrar el margen derecho.
- **Num*right**: Sangrar el margen derecho, multiplicando el sangrado *normal* por *Num* (por ejemplo `2*right`).
- **middle**: Sangrar ambos márgenes. Este es el funcionamiento por defecto.
- **Num*middle**: Sangrar por ambos lados, multiplicando el sangrado *normal* por *Num*.

Al explicar las opciones he mencionado un *sangrado normal*; con tal expresión me refiero a la cantidad de sangrado izquierdo y derecho que, por defecto, aplica «**narrower**». Esta *cantidad* se puede configurar mediante `\setupnarrower` que admite las siguientes opciones de configuración:

- **left**: Cantidad de sangrado que se aplicará en el margen izquierdo.
- **right**: Cantidad de sangrado que se aplicará en el margen derecho.
- **middle**: Cantidad de sangrado que se aplicará en ambos márgenes.
- **before**: Comando que se ejecutará antes de entrar en el entorno.
- **after**: Comando que se ejecutará al salir del entorno.

Si en nuestro documento queremos utilizar diferentes configuraciones del entorno narrower, podemos asignar cada una de ellas a un nombre diferente mediante

```
\definennarrower [Nombre] [Configuración]
```

donde *Nombre* es el nombre que se vinculará a esta configuración, y *Configuración* admite los mismos valores que `\setupnarrower`.

11.2 Espaciado vertical entre párrafos

11.2.1 `\setupwhitespace`

Como ya sabemos (sección 4.2.2), para ConT_EXt es indiferente cuántas líneas en blanco consecutivas haya en el fichero fuente: una o más líneas en blanco insertarán en el documento final exclusivamente un salto de párrafo. Para aumentar el espaciado entre párrafos no sirve de nada añadir un salto de línea extra en el fichero fuente, sino que dicha función es controlada por el comando `\setupwhitespace` que admite los siguientes valores:

- **none**: Significa que no se establecerá ningún espacio vertical adicional entre párrafos.
- **small**, **medium**, **big**: Insertan, respectivamente, un espacio adicional pequeño, mediano o grande. El concreto tamaño del espacio insertado por estos valores depende del tamaño de la fuente.
- **line**, **halfline**, **quarterline**: Mide el espacio en blanco adicional en relación con la altura de las líneas e inserta, respectivamente, una línea extra de separación, media línea, o un cuarto de línea.
- **DIMENSIÓN**: Establece una concreta dimensión como espacio de separación entre párrafos. Por ejemplo `\setupwhitespace[5pt]`.

Con carácter general se desaconseja establecer como valor para `\setupwhitespace` una dimensión exacta, siendo preferible utilizar los valores simbólicos `small`, `medium`, `big`, `line`, `halfline` o `quarterline`. Esto es así por dos razones:

- Los valores simbólicos son dimensiones elásticas (véase la [sección 3.8.2](#)) es decir: tienen un valor *normal* pero se admite una cierta disminución o aumento de dicho valor, lo que ayuda a ConT_EXt a componer páginas en las que estéticamente la separación entre párrafos sea similar. Por el contrario una medida fija de separación entre párrafos hace más difícil conseguir una buena paginación del documento.
- Los valores simbólicos `small`, `medium`, `big`, etc., se calculan a partir del tamaño de la fuente, por lo que si este cambia en ciertos fragmentos, también cambiará la cantidad de espaciado vertical entre párrafos, de tal modo que el resultado será siempre armónico. Por el contrario, un valor fijo para el espaciado vertical no se verá afectado por cambios en el tamaño de la fuente, lo que se traducirá, normalmente, en un documento con el espacio en blanco mal repartido (desde el punto de vista estético) y no acorde con las reglas de la corrección tipográfica.

Cuando se ha establecido algún valor para el espaciado vertical entre párrafos, tenemos disponibles dos comandos adicionales: `\nowhitespace`, que elimina el espaciado extra entre dos párrafos concretos, y `\whitespace` que hace lo contrario. Pero estos comandos raramente serán necesarios, pues lo cierto es que ConT_EXt gestiona por sí solo bastante bien el espaciado vertical entre párrafos; sobre todo si como valor para él se ha insertado alguna de las dimensiones predefinidas que se calculan a partir del tamaño de letra e interlineado activos en cada momento.



El sentido de `\nowhitespace` es obvio. No tanto el de `\whitespace` porque ¿qué sentido tiene ordenar, para dos párrafos concretos, un espaciado vertical que está establecido ya con carácter general para todos los párrafos? Sin embargo en la escritura de macros avanzadas, `\whitespace` puede ser útil en el contexto de un bucle que tiene que tomar una decisión a partir del valor de cierta condición. Esto es programación más o menos avanzada, y no entraré en ello.

11.2.2 Párrafos sin espacio vertical extra entre ellos

Si deseamos que en un fragmento concreto de nuestro documento los párrafos no estén separados por un espacio vertical extra, podemos, por supuesto, modificar la configuración general de `\setupwhitespace`, pero eso es, en cierto modo, contrario a la filosofía de ConT_EXt en la que los comandos de configuración general deben ubicarse exclusivamente en el preámbulo del fichero fuente, para así conseguir documentos coherentes y en los que sea fácil modificar la apariencia general. Por ello se proporciona el entorno «`packed`» cuya sintaxis general es

```
\startpacked[Espacio] ... \stoppacked
```

donde *Espacio* es un argumento opcional que indica qué cantidad de separación vertical se desea entre los párrafos del entorno. Si se omite no se aplicará ninguna separación vertical extra.

11.2.3 Añadir espacio vertical adicional en un punto concreto del documento

Para el caso de que en un punto concreto del documento el espaciado vertical normal entre párrafos no sea suficiente, podemos usar el comando `\blank`. Usado sin argumentos, `\blank` insertará la misma cantidad de espacio vertical que se haya establecido con `\setupwhitespace`. Pero se le puede indicar entre corchetes, bien una dimensión concreta, bien alguno de los valores simbólicos calculados a partir del tamaño de la fuente: `small`, `medium` o `big`. También podemos multiplicar esos tamaños por algún número entero y así, por ejemplo, `\blank[3*medium]` insertará el equivalente a tres saltos medianos. También podemos juntar dos medidas. Por ejemplo `\blank[2*big, medium]` insertará dos saltos grandes y un salto mediano.

Como `\blank` está pensado para aumentar el espacio vertical entre párrafos no produce efectos si entre los dos párrafos cuyo espaciado vertical se debe aumentar, se inserta un salto de página; y si se insertan, seguidos, dos o más comandos `\blank`, sólo se aplicará uno de ellos (el que mayor espacio haya de insertar). Tampoco produce ningún efecto un comando `\blank` ubicado tras un salto de página. No obstante en estos casos podemos forzar la inserción del espaciado vertical indicando, como opción del comando, la palabra simbólica «`force`». Y así, por ejemplo, si queremos que en nuestro documento el título de los capítulos aparezca ligeramente desplazado hacia abajo en la página, de tal modo que la longitud total de la misma sea inferior a la del resto de las páginas, lo que es, por otra parte, una práctica tipográfica relativamente frecuente, deberemos escribir en la configuración del comando `\chapter`, por ejemplo:

```
\setuphead
[chapter]
[
  page=yes,
  before={\blank[4cm, force]},
  after={\blank[3*medium]}
]
```

Esta secuencia de comandos logrará que los capítulos empiecen siempre una página y que el rótulo del capítulo se desplace cuatro centímetros hacia abajo. Sin usar la opción «`force`» esto no funcionaría.

11.2.4 `\setupblank` y `\defineblank`

Antes he dicho que `\blank`, usado sin argumentos, equivale a `\blank[big]`. Pero eso podemos cambiarlo mediante `\setupblank` estableciendo, por ejemplo, `\setupblank[0.5cm]`, o `\setupblank[medium]`. Usado sin argumento, `\setupblank` ajustará el valor al tamaño actual de la fuente.

Al igual que ocurre con `\setupwhitespace` el espacio en blanco que inserta `\blank` cuando su valor es alguno de los valores simbólicos predefinidos, es una dimensión elástica que admite cierta oscilación. Esto lo podemos cambiar mediante la opción «fixed», siendo posible, más tarde, restaurar el valor por defecto («flexible»). Así, por ejemplo, en el texto a doble columna es recomendable establecer `\setupblank[fixed, line]`, y al volver al texto a una sola columna establecer `\setupblank[flexible, default]`.

Con `\defineblank` podemos asociar una determinada configuración a un nombre. El formato general de este comando es:

```
\defineblank[Nombre] [Configuración]
```

Una vez definido nuestra configuración de espacio en blanco, podemos usarla con `\blank[NombreConfiguración]`.

11.2.5 Otros procedimientos para conseguir espacio vertical extra

En T_EX el comando que inserta espacio vertical extra es `\vskip`. Este comando, como casi todos los comandos de T_EX, funciona también en ConT_EXt pero está fuertemente desaconsejado su uso pues interfiere en el funcionamiento interno de algunas macros de ConT_EXt. En su lugar se propone el uso de `\godown` cuya sintaxis es:

```
\godown[Dimensión]
```

donde *Dimensión* ha de ser un número con o sin decimales, seguido de una unidad de medida. Por ejemplo `\godown[5cm]` producirá un salto de cinco centímetros en la página actual; aunque si el cambio de página estaba a menos de cinco centímetros, `\godown` sólo saltará hasta la próxima página. Asimismo, `\godown` al principio de una página no produce efectos, aunque podemos *engañarlo* escribiendo, por ejemplo «`_godown[3cm]`»¹ que primero insertará un espacio en blanco que hará que ya no estemos al principio de la página, y después un salto de tres centímetros.

¹ Recuerdese que en este documento se usa el carácter «`_`» para representar a un espacio en blanco cuando la visibilidad del mismo sea importante.

Como sabemos `\blank` también admite como argumento una dimensión concreta. Por lo que, desde el punto de vista del usuario, escribir `\blank[3cm]` o `\godown[3cm]` es prácticamente lo mismo. No obstante hay algunas diferencias sutiles entre ellos; y así, por ejemplo, dos comandos `\blank` consecutivos no pueden acumularse y cuando ello sucede, se aplica sólo el que impone un salto mayor. Dos o más comandos `\godown`, por el contrario, pueden perfectamente acumularse.

Otro comando de \TeX bastante útil, y cuyo uso en \ConTeXt no plantea ningún problema, es `\vfill`. Este comando inserta un espacio en blanco vertical flexible que llega hasta la parte inferior de la página. Es como si el comando *empujara* hacia abajo lo que se escriba detrás de él. Ello nos permite efectos interesantes tales como ubicar en la parte inferior de la página cierto párrafo, para lo que simplemente hay que precederlo de `\vfill`. Ahora bien: el efecto de `\vfill` es difícil de apreciar si su uso no se combina con saltos de página forzados, porque de poco sirve empujar hacia abajo un párrafo o línea de texto, si luego ese párrafo, conforme vaya creciendo, crece hacia arriba.

Así por ejemplo, para asegurarnos de que una línea se ubique al fondo de la página, deberíamos escribir:

```
\vfill
Línea al fondo
\page[yes]
```

Al igual que el resto de los comandos que insertan espacio vertical, `\vfill` no produce ningún efecto al principio de una página. Pero podemos *engañarlo*, precediéndolo de un espacio en blanco forzado. Y así, por ejemplo:

```
\page[yes]
\ \vfill
Línea central
\vfill
\page[yes]
```

centrará verticalmente en la página la frase “Línea central”.

11.3 Cómo construye \ConTeXt las líneas que forman los párrafos

Uno de los principales deberes de un sistema de composición tipográfica es tomar una larga secuencia de palabras y dividirla en líneas individuales del tamaño apropiado. Por ejemplo, cada párrafo de este texto se ha dividido en líneas de 15 centímetros de ancho, pero el autor no ha tenido que preocuparse por tales detalles, pues \ConTeXt elige los puntos de ruptura tras considerar cada párrafo en

su totalidad; de tal modo que las palabras finales de un párrafo pueden realmente influir en la división de la primera línea. Como resultado, el espacio entre las palabras de todo el párrafo es tan uniforme como sea posible.

Es este un aspecto en el que se puede observar mejor que en otros la diferente forma de trabajar de los procesadores de texto, y la mayor calidad que se obtiene con sistemas como ConT_EXt. Porque un procesador de textos, cuando llega al final de la línea y salta a la siguiente ajusta los espacios en blanco de la línea que se acaba de terminar para conseguir justificarla a la derecha. Con cada línea irá haciendo eso, y al final, en un mismo párrafo, cada línea tendrá un espacio de separación entre palabras diferente. Eso puede causar un muy mal efecto. ConT_EXt por el contrario, procesa el párrafo en su totalidad y calcula para cada línea cuántos puntos de ruptura admisibles hay, y la medida del espacio en blanco entre palabras que cada posible ruptura de línea provocaría. Como el punto de ruptura de una línea afecta a los posibles puntos de ruptura de las próximas líneas, el número total de posibilidades puede llegar a ser muy alto; pero eso no es problema para ConT_EXt. Finalmente, se tomará una decisión basada en la totalidad del párrafo, procurando que el espacio en blanco entre palabras de cada línea sea *lo más parecido posible*, lo que provoca unos párrafos mucho mejor compuestos; visualmente más compactos.

Para hacer su tarea, ConT_EXt ensaya distintas alternativas y, asigna a cada una de ellas un valor de “inconveniencia” (*badness*, en inglés) basado en sus parámetros, los cuales, a su vez, se establecieron tras estudiar a fondo el arte de la tipografía. Finalmente, tras haber explorado todas las posibilidades, ConT_EXt elige la opción menos inadecuada (la que tenga un valor más bajo de inconveniencia). Este procedimiento funciona bastante bien en general, pero inevitablemente habrá casos en los que se escojan puntos de ruptura de líneas que no sean los mejores, o que a nosotros no nos los parezca. Por lo tanto, a veces querremos indicarle al programa que ciertos lugares no son buenos puntos de ruptura. Por el contrario, en otras ocasiones querremos forzar una ruptura en un punto en particular.

11.3.1 Uso del carácter reservado «~»

Los principales candidatos a ser puntos de ruptura de línea son, obviamente, los espacios en blanco de separación entre palabras. Para indicar que cierto espacio en blanco no debe nunca ser sustituido por un salto de línea, se usa, como ya sabemos, el carácter reservado «~» al que T_EX denomina «lazo» (*tie*, en inglés) puesto que *enlaza* dos palabras.

El *Libro de estilo de la lengua española*, de la Real Academia de la Lengua (Espasa, 2018) recomienda el uso de estos espacios de no separación en los siguientes casos:

- Entre los elementos de las abreviaturas complejas. Por ejemplo, EE~UU.
- Entre las abreviaturas y el término a que se refieren. Por ejemplo, D^a~Ana Ruiz o pág.~45.
- Entre los números y el término al que acompañan. Por ejemplo Carlos~III, 45~volúmenes.
- Entre las cifras y los símbolos pospuestos, siempre que no sean superíndices. Por ejemplo, 73~km, 58~€; pero 35'.

- En los porcentajes expresados con palabras. Por ejemplo `veinte~por~ciento`.
- En los grupos numéricos separados por espacios en blanco. Por ejemplo, `5~357~891`. Aunque en estos casos es preferible usar el llamado *espacio fino* que con ConTeXt se consigue con el comando `\,` y, por lo tanto, escribir `5\,357\,891`.
- Para evitar que una abreviatura sea el único elemento en su línea. Por ejemplo:

```
Existen secciones como entretenimiento, medios de comunicación,
comercio,~etc.
```

A estos casos, KNUTH (padre de T_EX) añade las siguientes recomendaciones:

- Detrás de una abreviatura que no esté al final de una frase.
- En referencias a partes del documento como capítulos, apéndices, figuras, etc. Por ejemplo `Capítulo~12`.
- Entre el primer nombre y la inicial del segundo nombre de una persona, o entre la inicial del nombre y el apellido. Por ejemplo, `Donald~E. Knuth, A.~Einstein`.
- Entre símbolos matemáticos en aposición de nombres. Por ejemplo `dimensión~d, anchura~w`.
- Entre símbolos en series. Por ejemplo `{1,~2, \dots,~n}`.
- Cuando un número está estrechamente ligado a una preposición. Por ejemplo de `0 a~1`.
- Cuando se expresan con palabras símbolos matemáticos. Por ejemplo, `igual~a~n`.
- En las enumeraciones de casos hechas dentro de un mismo párrafo. Por ejemplo: `(1)~verde, (2)~rojo, (3)~azul`.

¿Muchos casos? Sin duda la perfección tipográfica tiene un coste en sobreesfuerzo. Está claro que si no queremos, no tenemos por qué aplicar estas reglas, pero no está de más conocerlas. Además —y aquí hablo por experiencia— una vez que nos acostumbramos a aplicarlas (o a aplicar alguna de ellas) el hacerlo se automatiza. Es como poner tildes en las palabras mientras las escribimos: quienes así lo hacemos, si estamos tan acostumbrados como para escribirlas automáticamente, no tardamos en escribir una palabra con tilde más tiempo del que tardaría el que la escribiera sin tilde.

11.3.2 Partición silábica de las palabras

Salvo en un idioma compuesto básicamente de monosílabos, es bastante difícil que insertando puntos de ruptura de línea sólo en los espacios en blanco entre palabras se obtenga un resultado óptimo. Por ello ConTeXt también analiza la posibilidad de introducir un salto de línea entre dos sílabas de una palabra; y para hacerlo bien es imprescindible saber cuál es el idioma del texto, pues las reglas de partición de una palabra en sílabas son diferentes en cada idioma. De ahí la importancia del comando `\mainlanguage` en el preámbulo del documento.

Cuando se inserta un salto de línea entre dos sílabas de una palabra, se escribe automáticamente un guión en el punto de la ruptura de la palabra para indicar que el segmento que lo precede es una parte de una palabra dividida que continúa en la siguiente línea. Como en inglés al guión se le llama *hyphen*, a ese procedimiento se le denomina *hyphenation*. En español se usa ese mismo procedimiento pero, hasta donde yo se, no existe una palabra específica para nombrarlo¹.

Puede ocurrir que ConT_EXt no consiga dividir silábicamente una palabra de modo adecuado. A veces, porque sus propias reglas de particionado se lo impiden (por ejemplo, ConT_EXt nunca divide una palabra en dos segmentos si dichos segmentos no tienen un número mínimo de letras); o porque la palabra resulte ambigua, como ocurre, por ejemplo, con «sublime» que, se puede considerar que consta de las sílabas su-bli-me; pero que, como empieza por «sub» que es un prefijo relativamente corriente en español, es posible que el algoritmo de ConT_EXt considere que lo correcto sea dividirla como sub-li-me; o incluso es posible que, siendo correcta la división silábica, por el contexto produzca un resultado que se pretende evitar como, ocurriría, por ejemplo, si la última línea de una página acaba con la palabra “círculo” y esta se parte por la primera sílaba, haciendo que la página siguiente empiece (aparentemente) con la palabra “culo”.

Sean cuales sean las razones, si la división de una palabra no nos satisface, o no es correcta, podemos cambiarla indicando expresamente los puntos de ruptura admisibles para dicha palabra mediante el símbolo de control `\-`, y así, por ejemplo, si la palabra «sublime» nos hubiera dado algún problema, podríamos escribirla, en el fichero fuente como «su\ -bli\ -me»; o si el problema lo tenemos con «círculo», podríamos escribir «círcu\ -lo».

Si la palabra problemática es usada varias veces en nuestro documento, es preferible indicar su particionado correcto en el preámbulo, mediante el comando `\hyphenation`: este comando, que está pensado para ser incluido en el preámbulo del fichero fuente, recibe como argumento una o varias palabras (separadas por comas) en las que los puntos admitidos de ruptura se indican con un guión. Por ejemplo:

```
\hyphenation{su-bli-me, círcu-lo}
```

En el caso de que la palabra que es argumento de este comando no contenga ningún guión, el efecto será el de que dicha palabra no se partirá silábicamente nunca. Este mismo efecto podemos conseguirlo mediante los comandos `\hbox`, que crea una caja horizontal indivisible en torno a la palabra, o `\unhyphenated` que previene la partición silábica de la palabra o palabras que recibe como argumentos. Pero mientras `\hyphenation` actúa globalmente, `\hbox` y `\unhyphenated` actúan

¹ En el antes citado *Libro de estilo de la Lengua española* se describe este procedimiento (pág. 129) sin asignarle ningún nombre concreto.

localmente. Es decir: el comando `\hyphenation` afecta a todas las apariciones en el documento de las palabras incluidas en su argumento; a diferencia de `\hbox` o `\unhyphenated` que sólo actúan en el punto del fichero fuente en el que se encuentren.

Desde un punto de vista interno, el funcionamiento de la partición silábica de palabras, está controlado por las variables de T_EX `\pretolerance` y `\tolerance`. El primero de estos valores controla la admisibilidad de una partición hecha sólo sobre espacios en blanco. Por defecto vale 100, pero si alteramos su valor a, por ejemplo, 10 000, conseguiremos que ConT_EXt siempre considere aceptable una partición de líneas que no requiera partir silábicamente palabras, por lo que, *de facto*, estaremos anulando la partición silábica. Por el contrario, si fijamos, por ejemplo, el valor de `\pretolerance` a -1, forzaremos que ConT_EXt busque en todo caso la partición silábica de las palabras al final de la línea.

Podemos fijar, directamente, un valor arbitrario para `\pretolerance` en nuestro documento simplemente escribiendo en él la sentencia de asignación. Por ejemplo:

```
\pretolerance=10000
```

pero también podemos manipular este valor mediante los valores «lesshyphenation» y «morehyphenation» de `\setupalign`, véase al respecto la [sección 11.6.1](#).

11.3.3 Nivel de tolerancia en la ruptura de líneas

A la hora de buscar los posibles puntos de ruptura de línea, ConT_EXt es, normalmente, bastante estricto; lo que significa que prefiere permitir que una palabra sobrepase el margen derecho, por no haberla podido partir silábicamente, a insertar un salto de línea antes de la misma, si ello obliga a aumentar excesivamente el espacio entre palabras en la línea. Este comportamiento predeterminado da, normalmente, unos resultados óptimos, en los que muy excepcionalmente ciertas líneas sobresalen algo por el margen derecho. La idea es que el autor (o compositor), una vez terminado el documento, vaya revisando uno a uno estos casos excepcionales para tomar la decisión adecuada, que puede ser un comando `\break` antes de la palabra que sobresale, o una distinta redacción para el párrafo que haga que la palabra que causa problemas vaya a parar a otro lugar.

No obstante en algunos contextos la poca tolerancia de ConT_EXt puede ser un inconveniente. En estos casos podemos indicarle que debe ser más tolerante con los espacios en blanco en las líneas. Para ello contamos con el comando `\setup-tolerance` que permite alterar el nivel de tolerancia en el cálculo de los saltos de línea, a lo que ConT_EXt llama “tolerancia horizontal” (porque afecta al espacio horizontal) y en el cálculo de los saltos de página (“tolerancia vertical”) de la que hablaremos en la [sección 11.6.2](#).

La tolerancia horizontal (que es la que afecta a la ruptura entre líneas), por defecto está establecida con el valor «**verystrict**», es decir «muy estricto», podemos alterar este valor estableciendo, como posibles valores alternativos, «**strict**», «**tolerant**», «**verytolerant**» o «**stretch**». Así, por ejemplo

`\setuptolerance[horizontal, verytolerant]`

hará que sea casi imposible que una línea sobrepase el margen derecho, aunque para ello haya que establecer un muy grande espaciado entre palabras de la línea que resulte antiestético.

11.3.4 Forzar un salto de línea en cierto punto

Para forzar un salto de línea en un punto concreto se usan los comandos `\break`, `\crlf` o `\\`. El primero de ellos, `\break`, introduce, en el punto en el que se encuentre, un salto de línea. Ello, muy probablemente producirá que la línea en la que el comando se encuentra quede estéticamente deformada, con una inmensa cantidad de espacio en blanco entre las palabras que la componen. como se puede ver en el siguiente ejemplo en el que el comando `\break` en la tercera línea (del fragmento fuente) produce una segunda línea (del texto formateado) bastante fea.

Por la esquina del viejo barrio lo vi
pasar con el `\emph{tumbao}` que tienen
los`\break` guapos al caminar, las manos
siempre en los bolsillos de su gabán,
pa que no sepan en cuál de ellas lleva
el puñal.

Por la esquina del viejo barrio lo vi pasar con el
tumbao que tienen los
guapos al caminar, las manos siempre en los bol-
sillos de su gabán, pa que no sepan en cuál de
ellas lleva el puñal.

Para evitar este efecto podemos usar los comandos `\\` o `\crlf` que también insertan un salto forzado de línea, pero rellenan la línea original con el espacio en blanco suficiente como para que quede alineada a la izquierda:

Por la esquina del viejo barrio lo vi
pasar con el `{\em tumbao}` que tienen
los`\\` guapos al caminar, las manos
siempre en los bolsillos de su gabán,
pa que no sepan en cuál de ellas lleva
el puñal.

Por la esquina del viejo barrio lo vi pasar con el
tumbao que tienen los
guapos al caminar, las manos siempre en los bol-
sillos de su gabán, pa que no sepan en cuál de
ellas lleva el puñal.

En líneas *normales*, hasta donde yo se, no hay diferencias entre `\\` y `\crlf`; pero dentro del título de una sección sí las hay:

- `\\` genera un salto de línea en el cuerpo del documento, pero no en la transposición del título de la sección al índice.
- `\crlf` genera un salto de línea que se aplicará tanto en el cuerpo del documento como en la transposición al índice del título de la sección.

No se debe confundir un salto de línea con un salto de párrafo. El salto de línea simplemente hace que se termine la línea actual y se empiece la línea siguiente, pero manteniéndonos en el mismo párrafo, por lo que la separación entre la línea original y la nueva línea será la determinada por el interlineado normal dentro de un párrafo. Por ello sólo hay tres escenarios en los que puede estar recomendado forzar un salto de línea:

- En casos muy excepcionales, cuando ConT_EXt no ha podido encontrar un salto de línea adecuado, de tal modo que una línea sobresale por el lado derecho. En estos casos (que se dan en muy contadas ocasiones, principalmente cuando en la línea hay *cajas* indivisibles, o texto *verbatim* [véase la [sección 10.2.3](#)]), puede ser conveniente forzar un salto de línea con `\break` justo antes de la palabra que sobresale por el margen derecho.
- En párrafos que en realidad se componen de líneas sueltas, cada una de ellas con información independiente de la de las líneas anteriores como, por ejemplo, el encabezado de una carta en donde la primera línea puede contener el nombre del remitente, la segunda el del destinatario, y la tercera la fecha; o en el texto informativo de la autoría de un trabajo, en el que una línea informa del nombre del autor, otra de su cargo o posición académica y tal vez una tercera línea aclare la fecha, etc. En estos casos el salto de línea debe forzarse con `\\` o `\crlf`. Es corriente, por otra parte, que este tipo de párrafos se presenten alineados a la derecha.
- En la escritura de poemas y textos similares, para separar un verso de otro. Aunque en este último caso es preferible usar el entorno `lines` que se explica en la [sección 11.5.1](#).

11.4 Interlineado

El interlineado es la distancia que separa a las líneas que componen un párrafo. ConT_EXt lo calcula automáticamente a partir de la concreta fuente que se esté usando y, sobre todo, de su tamaño base, establecido con `\setupbodyfont` o `\switchtobodyfont`.

Podemos influir en el interlineado mediante el comando `\setupinterlinespace` que admite tres sintaxis diferentes:

- `\setupinterlinespace [...Interlineado...]`, donde *Interlineado* es un valor concreto o una palabra simbólica que asigna un interlineado predefinido:
 - Cuando es un valor concreto puede ser una dimensión (por ejemplo, 15pt), o un número simple, entero o decimal (por ejemplo, 1.2). En este último caso el número se interpreta como “número de líneas” partiendo del interlineado por defecto de ConT_EXt.

- Cuando es una palabra simbólica, esta puede ser «`small`», «`medium`» o «`big`», cada una de ellas aplicará, respectivamente, un interlineado pequeño, mediano o grande, partiendo siempre del interlineado por defecto que ConTeXt aplicaría.
- `\setupinterlinespace [...]=...]`. En esta modalidad el interlineado se fija alterando explícitamente las medidas con base en las cuales ConTeXt calcula el interlineado adecuado. Antes he dicho que el interlineado se calcula atendiendo a la fuente concreta y a su tamaño; pero eso era simplificar mucho: en realidad la fuente y su tamaño lo que hacen es establecer ciertas medidas a partir de las cuales se calcula el interlineado. Mediante esta modalidad de `\setupinterlinespace` se modifican esas medidas y, por lo tanto, el interlineado. Las concretas medidas y valores que se pueden manipular por este procedimiento (cuyo significado no voy a explicar por exceder ello el ámbito de una simple *introducción*), son: `line`, `height`, `depth`, `minheight`, `mindepth`, `distance`, `top`, `bottom`, `stretch` y `shrink`.
- `\setupinterlinespace [Nombre]`. Mediante esta modalidad, estableceremos o configuraremos un tipo concreto y personalizado de interlineado previamente definido con `\defineinterlinespace`.

Mediante

`\defineinterlinespace[Nombre] [Configuración]`

podemos asociar a un nombre concreto una determinada configuración de interlineado que luego podremos activar en un punto de nuestro documento simplemente con `\setupinterlinespace[Nombre]`. Para volver al interlineado normal deberíamos escribir `\setupinterlinespace[reset]`.

11.5 Otras cuestiones relacionadas con las líneas

11.5.1 Convertir los saltos de línea del fichero fuente en saltos de línea en el documento final

Como ya sabemos (véase [sección 4.2.2](#)) por defecto ConTeXt ignora los saltos de línea del fichero fuente que considera como simples espacios en blanco, salvo que sean dos o más saltos de línea consecutivos, en cuyo caso, se insertará un salto de párrafo. No obstante puede haber algunas situaciones en las que nos interese respetar los saltos de línea del fichero fuente original tal y como se insertaron,

como, por ejemplo, cuando se escribe poesía. Para ello ConT_EXt nos ofrece el entorno «`lines`», cuyo formato es

```
\startlines[Opciones] ... \stoplines
```

en donde las opciones pueden ser, entre otras, cualquiera de las siguientes:

- **space:** Cuando se establece esta opción con el valor «on», el entorno, además de respetar los saltos de línea del fichero fuente, respetará los espacios en blanco del fichero fuente, anulando temporalmente la regla de su absorción.
- **before:** Texto o comando a ejecutar antes de entrar en el entorno.
- **after:** Texto o comando a ejecutar al salir del entorno.
- **inbetween:** Texto o comando a ejecutar al entrar en el entorno.
- **indenting:** Valor que indica si se indentarán o no los párrafos dentro del entorno (véase, [sección 11.1.1](#)).
- **align:** Alineación de las líneas del entorno (véase [sección 11.6](#)).
- **style:** Comando de estilo a aplicar dentro del entorno.
- **color:** Color a aplicar al texto del entorno.

Así, por ejemplo,

<code>\startlines</code>	Un florín
<code>Un florín</code>	Saltarín
<code>Saltarín</code>	de Medellín
<code>de Medellín</code>	
<code>\stoplines</code>	

También podemos modificar el funcionamiento por defecto del entorno mediante `\setuplines` y, como en tantos comandos de ConT_EXt, es asimismo posible asignar a una configuración concreta de este entorno un nombre. Ello se hace mediante `\definelines` cuyo formato es:

```
\definelines[Nombre] [Configuración]
```

donde, como configuración, podemos incluir las mismas opciones que se han explicado con carácter general para el entorno. Una vez definido nuestro entorno de líneas personalizado, para insertarlo deberíamos escribir

```
\startlines[Nombre] ... \stoplines
```

11.5.2 Numeración de líneas

En cierto tipo de textos es corriente establecer algún tipo de numeración de líneas, como por ejemplo, en textos sobre programación informática, donde es relativamente corriente que en los fragmentos de código que se presentan como ejemplos aparezcan numeradas las líneas, o en poemas, ediciones críticas, etc. Para todas estas situaciones ConT_EXt ofrece el entorno `linenumbers` cuyo formato es

```
\startlinenumbers[Opciones] ... \stoplinenumbers
```

Las opciones admisibles son:

- **continue:** En el caso de que en nuestro documento haya más de un fragmento cuyas líneas queremos numerar, esta opción controla si en cada fragmento la numeración se reiniciará («`continue=no`», valor por defecto) o si, por el contrario, cada fragmento empezará la numeración en el punto en el que quedó la del fragmento anterior («`continue=yes`»).
- **start:** Indica el número de la primera línea, para el caso de que no queramos que este sea el «1» o el que corresponda continuando la última enumeración.
- **step:** Todas las líneas incluidas en el entorno se numerarán, pero, mediante esta opción, podemos indicar que sólo se imprima el número cada cierto intervalo. En la edición de poemas, por ejemplo, es corriente que sólo se muestre el número en los múltiplos de 5 (versos 5, 10, 15...).

Todas estas opciones se pueden indicar, con carácter general para todos los entornos `linenumbers` que haya en nuestro documento, mediante `\setuplinenumbers`. Este comando permite, además, configurar otros aspectos de la numeración de líneas:

- **conversion:** Tipo de numeración de líneas. Puede ser cualquiera de los explicados en la [página 146](#) a propósito de la numeración de capítulos y secciones.
- **style:** Comando (o comandos) que determinan el estilo que tendrá el número de línea (fuente, tamaño, variante...).
- **color:** Color con el que se imprimirá el número de línea.
- **location:** Lugar en donde se colocará el número de línea. Puede ser cualquiera de los siguientes: `text`, `begin`, `end`, `default`, `left`, `right`, `inner`, `outer`, `inleft`, `inright`, `margin`, `inmargin`.
- **distance:** Distancia entre el número de línea y la línea propiamente dicha.
- **align:** Alineación del número. Puede ser: `inner`, `outer`, `flushleft`, `flushright`, `left`, `right`, `middle` o `auto`.

- **command:** Comando al que se pasará como parámetro el número de la línea antes de imprimirlo.
- **width:** Anchura reservada para imprimir el número de la línea.
- **left, right, margin:**

También podemos crear diferentes configuraciones personalizadas de la numeración de líneas mediante `\definelinenumbering` de tal modo que la configuración se asocie a un nombre concreto:

```
\definelinenumbering[Nombre] [Configuración]
```

Una vez definida una configuración concreta y asociada a un nombre, podemos usarla con

```
\startlinenumbering[Nombre] ... \stoplinenumbering
```

11.6 Alineación horizontal y vertical

El comando que controla la alineación del texto con carácter general es `\setup-align`. Este comando se usa tanto para controlar la alineación horizontal como para controlar la alineación vertical.

11.6.1 Alineación horizontal

Cuando la anchura *exacta* de una línea de texto no ocupa toda su anchura posible, se plantea el problema de qué hacer con el espacio en blanco resultante¹. Al respecto podemos hacer básicamente tres cosas:

1. Acumularlo en uno de los dos lados de la línea: si lo acumulamos en el lado izquierdo, la línea se verá *empujada* hacia la derecha, mientras que si lo acumulamos en el lado derecho la línea queda en el lado izquierdo. Hablamos, en el primer caso, de *alineación derecha* y, en el segundo, de *alineación izquierda*. ConT_EXt aplica por defecto la alineación izquierda a la última línea de los párrafos.

Cuando varias líneas consecutivas se alinean a la izquierda, el lado derecho se ve irregular; por el contrario, cuando la alineación es a la derecha, el lado que se ve irregular es el izquierdo. ConT_EXt, para denominar a las opciones que alinean a uno u otro lado, se fija, no en el lado donde se alinean, sino en el lado que se ve desigual. Por ello la opción `flushright` (literalmente, *nivelación*

¹ Por anchura *exacta* quiero decir la anchura de la línea *antes* de que ConT_EXt ajuste el tamaño de los espacios en blanco entre palabras para conseguir su justificación.

derecha) produce una alineación izquierda y `flushleft` produce una alineación a la derecha, como abreviaturas de `flushright` y `flushleft` `\setupalign` admite también los valores `right` y `left`. Pero **atención**: aquí el significado de las palabras en inglés nos engañará, pues, a pesar de que la palabra *left* significa «izquierda» y la palabra *right* «derecha», `\setupalign[left]` alinea a la derecha y `\setupalign[right]` alinea a la izquierda.

En caso de que el lector se pregunte el por qué de este extraño funcionamiento, vale la pena citar la wiki de ConTEXt, donde se aclara que esto se debe a que desafortunadamente, cuando Hans estaba escribiendo esta parte de ConTeXt, estaba pensando en una alineación de "derecha desigual" e "izquierda desigual", en lugar de "izquierda nivelada" y "derecha nivelada". Y ahora es imposible cambiarlo, porque ello rompería la compatibilidad retroactiva con todos los documentos existentes que lo utilizan".

En documentos preparados para ser impresos a doble cara, además de margen derecho e izquierdo puede hablarse de margen interior y margen exterior. Los valores `flushinner` (o simplemente `inner`) y `flushouter` (o simplemente `outer`) establecen la correspondiente alineación en estos casos.

2. Repartirlo entre ambos márgenes. El resultado será que la línea se muestre centrada. La opción de `\setupalign` que consigue esto es `middle`.
3. Repartirlo entre todas las palabras que componen la línea, aumentando, si fuera necesario, el espacio entre las palabras, de tal modo que la línea pase a tener exactamente la misma anchura que el espacio disponible para ella. En estos casos se habla de *líneas justificadas*. Este es asimismo, el valor por defecto de ConTEXt y por ello no hay ninguna opción especial en `\setupalign` que lo establezca. No obstante, si hemos alterado la alineación justificada por defecto, podemos restaurarla mediante `\setupalign[reset]`.

Los valores para `\setupalign` que acabamos de ver (`right`, `flushright`, `left`, `flushleft`, `inner`, `flushinner`, `outer`, `flushouter` y `middle`) pueden combinarse con `broad`, lo que provoca alineaciones algo más toscas.

Otros dos valores posibles de `\setupalign` que afectan a la alineación horizontal, tienen que ver con la partición silábica de palabras al final de la línea, pues del hecho de que esta partición se haga o no se haga, dependerá que la medida *exacta* de la línea sea mayor o menor; lo que a su vez afecta al espacio en blanco sobrante.

En este sentido `\setupalign` admite el valor `morehyphenation` que hace que ConTEXt se esfuerce más en buscar puntos de ruptura basados en la partición silábica de palabras, y `lesshyphenation` que produce el efecto contrario. Con `\setupalign[horizontal, morehyphenation]`, el espacio en blanco sobrante de las líneas será más reducido y, por lo tanto, la alineación menos "ostentosa" o aparente. Por el contrario, con `\setupalign[horizontal, lesshyphenation]` habrá más espacio en blanco sobrante, y la alineación será más visible.

`\setupalign` está pensado para ser incluido en el preámbulo y afectar a todo el documento o, para ser incluido en un punto concreto y afectar desde ese punto

hasta el final. Si sólo queremos cambiar la alineación de una o varias líneas podemos usar:

- El entorno «`alignment`», pensado para afectar a varias líneas. Su formato general es

```
\startalignment[Opciones] ... \stopalignment
```

donde *Opciones* son cualquiera de las admisibles para `\setupalign`.

- Los comandos `\leftaligned`, `\midaligned` o `\rightaligned` que provocan, respectivamente, una alineación a la izquierda, centrada o a la derecha; y si lo que queremos es que la última palabra de un párrafo (pero sólo ella, no el resto de la línea) se alinee en el lado derecho podemos usar `\wordright`. Todos estos comandos han de recibir entre llaves el texto al que afectarán.

Obsérvese, por otra parte, que si las palabras «right» y «left», en `\setupalign` provocan una alineación inversa a lo que su nombre sugeriría, no ocurre lo mismo con los comandos `\leftaligned` y `\rightaligned`, que provocan exactamente el tipo de alineación que su nombre indica: left a la izquierda, y right a la derecha.

11.6.2 Alineación vertical

Si la alineación horizontal se plantea cuando la anchura de una línea no ocupa todo el espacio posible, la alineación vertical afecta a la altura de la página: Si la altura exacta del texto de una página no ocupa toda la altura disponible, ¿qué hacemos con el espacio en blanco sobrante?: Podemos amontonarlo en la parte superior («`height`»), lo que hará que el texto de la página se empuje hacia abajo; amontonarlo en la parte inferior («`bottom`») o repartirlo entre todos los párrafos («`line`»). Por defecto el valor para la alineación vertical es «`bottom`».

Nivel de tolerancia vertical

Del mismo modo que con `\setuptolerance` podemos alterar el nivel de tolerancia de ConTeXt de cara al tamaño del espacio horizontal admisible en una línea (tolerancia horizontal), podemos también alterar la tolerancia vertical, es decir: la tolerancia hacia espacios de separación vertical entre párrafos mayores de los que ConTeXt considera, por defecto razonables para una página bien compuesta. Como valores posibles para la tolerancia vertical se admiten los mismos que para la tolerancia horizontal: `verystrict`, `strict`, `tolerant` y `verytolerant`. El valor asignado por defecto es `\setuptolerance [vertical, strict]`.

Control de líneas viudas y huérfanas

Un aspecto que afecta indirectamente a la alineación vertical es el del control de líneas viudas y huérfanas. Ambos fenómenos implican que se introduzca un salto de página de tal modo que una línea de un párrafo quede aislada en una página

diferente de donde está el resto del párrafo, cosa esta que produce un efecto que tipográficamente no se considera adecuado. Si la línea que queda separada del resto del párrafo es la primera, se habla de *línea viuda*; si la que queda separada del párrafo es la última se habla de *línea huérfana*.

Por defecto ConT_EXt no lleva a cabo un control que impida que estas líneas aisladas se produzcan. Pero eso podemos cambiarlo alterando dos variables internas de ConT_EXt: `\widowpenalty`, que controla las líneas viudas y `\clubpenalty` que controla las líneas huérfanas. Así, las siguientes sentencias en el preámbulo de nuestro documento harán que dicho control se lleve a cabo:

```
\widowpenalty=10000  
\clubpenalty=10000
```

Llevar a cabo este control significa que ConT_EXt evitará insertar un salto de página que separe a la primera o a la última línea de un párrafo de la página en la que se encuentra el resto. Ese *evitar* será más o menos riguroso según el valor que asignemos a las variables. Con un valor de 10 000, como el que he utilizado en el ejemplo, el control será absoluto; con un valor de, por ejemplo, 150, el control no será tan riguroso y ocasionalmente podrá haber alguna línea viuda o huérfana cuando la alternativa sea, tipográficamente hablando, peor.

Capítulo 12

Construcciones y párrafos especiales

Sumario: **12.1 Notas al pie y notas finales;** 12.1.1 Tipos de notas en ConT_EXt y comandos asociados a ellas; 12.1.2 Examen particular de las notas al pie y de las notas finales; 12.1.3 Notas locales; 12.1.4 Creación y utilización de tipos personalizados de notas; 12.1.5 Configuración de las notas; 12.1.6 Excluir de la compilación, temporalmente, las notas; **12.2 Párrafos con múltiples columnas;** 12.2.1 El entorno `\startcolumns`; 12.2.2 Párrafos paralelos; **12.3 Listas estructuradas;** 12.3.1 Selección de tipo de lista y de separador entre *items*; A Listas no ordenadas; B Listas ordenadas; 12.3.2 Introducción de los elementos de la lista; 12.3.3 Configuración básica de las listas; 12.3.4 Configuración adicional de las listas; 12.3.5 Listas simples con el comando `\items`; 12.3.6 Predeterminación del comportamiento de las listas y creación de nuestros propios tipos de lista; **12.4 Descripciones y enumeraciones;** 12.4.1 Descripciones; 12.4.2 Enumeraciones; **12.5 Líneas y marcos;** 12.5.1 Líneas simples; 12.5.2 Líneas vinculadas a texto; 12.5.3 Palabras o textos enmarcados; **12.6 Otros entornos y construcciones de interés;**

12.1 Notas al pie y notas finales

Las notas son «elementos textuales secundarios que se emplean con diversos propósitos, como realizar una aclaración o extensión del texto principal, ofrecer la referencia bibliográfica de las fuentes citadas, incluir citas, remitir a otros documentos o fijar el sentido del texto» [Libro de Estilo de la Lengua española, pág. 195]. Son especialmente importantes en los textos de naturaleza académica. Pueden ubicarse en distintos puntos de la página o del documento. Hoy día las más extendidas son las que se ubican al pie de la página (llamadas, por ello, notas al pie); también se ubican a veces en alguno de los márgenes (notas marginales), al final de cada capítulo o sección, o al final del documento (notas finales). En documentos especialmente complejos, también puede haber distintos *juegos* de notas: Notas del autor, notas del traductor, notas del anotador o actualizador, etc. En particular en las ediciones críticas el aparato de las notas puede llegar a ser especialmente complejo y pocos sistemas de composición son capaces de soportarlo. ConT_EXt es uno de ellos. En él disponemos de numerosos comandos para establecer y configurar diferentes tipos de notas.

Para su explicación conviene empezar señalando los distintos elementos que pueden intervenir en una nota:

- *Marca o llamada* de la nota: El signo que se sitúa en el cuerpo del texto para indicar que existe una nota vinculada a él. No todos los tipos de notas tienen una *llamada* asociada a ellos, pero cuando existe, la *llamada* se escribe en dos lugares: en el punto del texto principal a que se refiere la nota, y al principio del texto de la nota propiamente dicha. La presencia de una misma marca en esos dos lugares es la que permite relacionar a la nota con el texto principal.
- *Identificador* de la nota: La letra, número o símbolo que identifica a una nota y la diferencia de otras notas. Algunas notas, como, por ejemplo, las notas marginales, pueden carecer de identificador. Cuando no es así, el identificador normalmente coincide con la *llamada*.

Si pensamos exclusivamente en las notas al pie de página, no veremos ninguna diferencia entre lo que acabo de llamar *marca* y el *identificador*. La diferencia se ve clara pensando en otros tipos de notas: Las notas lineales, por ejemplo, tienen identificador, pero no marca.

- *Texto o contenido* de la nota, que se sitúa siempre en un punto de la página o del documento distinto de aquel en el que se encuentra el comando que genera la nota e indica cuál es su contenido.
- *Etiqueta* asociada a la nota: Una etiqueta o nombre asociado a una nota que no se muestra en el documento final, pero nos permite referirnos a ella y recuperar su identificador en algún otro punto del documento.

12.1.1 Tipos de notas en ConT_EXt y comandos asociados a ellas

En ConT_EXt disponemos de varios tipos de notas. De momento sólo los enumeraré, describiéndolos en líneas generales e informando de los comandos que las generan. Mas adelante desarrollaré las dos primeras:

- **Notas al pie:** Sin duda las más populares; hasta el punto de que es corriente que para referirse genéricamente a todos los tipos de notas, se hable globalmente de *footnotes* (que es el nombre de este tipo de notas en inglés). Las notas a pie de página introducen una *marca* con el *identificador* de la nota en el punto del documento en el que se encuentra el comando, e insertan el texto de la nota propiamente dicho al pie de la página en donde aparece la marca. Se crean con el comando `\footnote`.
- **Notas al final:** Estas notas, que se crean con el comando `\endnote`, introducen en el punto del documento donde se encuentren una marca con el identificador de la nota; pero el contenido de la nota, se inserta en otro punto del documento, y la inserción la produce un comando distinto (`\placenames`).

- **Notas al margen:** Como su propio nombre indica se escriben en el margen del texto y en ellas no hay identificador ni marca o llamada generada automáticamente en el cuerpo del documento. Los dos comandos principales (pero no los únicos) que las crean son `\inmargin` y `\margintext`.
- **Notas a la línea:** Un tipo de notas propio de entornos en los que las líneas están numeradas como, por ejemplo en el caso de `\startlinenumbering ... \stoplinenumbering` (véase la [sección 11.5.2](#)). La nota, que se escribe habitualmente al pie, se refiere a un número concreto de línea. Se generan con el comando `\linenote` que, a su vez, se configura con `\setuplinenote`. Este comando no imprime ninguna *marca* en el cuerpo del texto, pero sí imprime en la nota propiamente dicha el número de línea sobre el que recae la nota (que se usa como *identificador*).

A continuación desarrollaré exclusivamente los dos primeros tipos de notas, pues:

- Las notas al margen se tratan en otro lugar ([sección 5.7](#)).
- Las notas lineales tienen un uso fuertemente especializado (sobre todo en ediciones críticas) y entiendo que en un documento introductorio como el presente, basta con que el lector sepa que existen.

No obstante, para el lector interesado recomiendo un vídeo (en español) acompañado de un texto (también en español) sobre ediciones críticas en ConT_EXt del que es autor Pablo Rodríguez y que está disponible en [este enlace](#). Es también bastante útil para comprender varias de las opciones de configuración generales de las notas en general.

12.1.2 Examen particular de las notas al pie y de las notas finales

Notas al pie y notas finales son bastante parecidas en la sintaxis del comando que las crea y en el mecanismo de configuración y personalización, pues, en realidad, ambos tipos de notas son instancias particulares de una construcción más general (las notas), de la que es posible establecer otras instancias mediante el comando `\definenote` (véase la [sección 12.1.4](#)).

La sintaxis de los comandos que crean cada uno de estos tipos de notas es la siguiente:

```
\footnote[Etiqueta]{Texto}  
\endnote[Etiqueta]{Texto}
```

donde

- *Etiqueta* es un argumento opcional que asigna a la nota una etiqueta que nos permitirá referirnos a ella en otros puntos del documento.

- *Texto* es el contenido de la nota. Puede ser todo lo largo que se desee, e incluir en él párrafos y entornos especiales, si bien hay que señalar que, cuando se trata de notas al pie, la correcta composición de las páginas es bastante difícil en documentos con notas abundante y excesivamente largas.

En principio, en el texto de una nota se puede usar cualquier comando que pudiera usarse en el texto principal. No obstante, he podido comprobar que ciertas construcciones y caracteres que en el texto principal no plantean ningún tipo de problemas, cuando tienen lugar en el texto de una nota generan un error de compilación. Estos casos me los he ido encontrando conforme hacía pruebas, pero no los he sistematizado.

Cuando se ha usado el argumento *Etiqueta*, estableciendo una etiqueta para la nota, el comando `\note` nos permite recuperar el identificador de la nota en cuestión. Este comando imprime en el documento el identificador de la nota asociada a la etiqueta que recibe como argumento. Así, por ejemplo:

```
Un día en que el Nota\footnote[lebowski]
{Personaje de la película «El Gran Lebowski»}
estaba dando la nota alguien dijo ¿No notas
al Nota\note[lebowski] dando la nota?
```

Un día en que el Nota¹ estaba dando la nota alguien dijo
¿No notas al Nota¹ dando la nota?

¹Personaje de la película «El Gran Lebowski»

La principal diferencia entre `\footnote` y `\endnote` estriba en el lugar donde se escribe el contenido de la nota:

`\footnote` como regla imprime el texto de la nota en la parte inferior de la página en la que se encuentra el comando, de tal forma que la marca de la nota y su texto (o el principio del texto, si este se debe repartir entre dos páginas) aparecerán en la misma página. Para ello ConT_EXt hará los ajustes necesarios para componer la página calculando el espacio que requiere la ubicación de la nota en la parte inferior de la página.

Pero en algunos entornos `\footnote` insertará el texto de la nota, no al pie de la página propiamente dicho, sino al pie del entorno. Así ocurre, por ejemplo, en las tablas, o en el entorno `columns`. En estos casos, si queremos que las notas dentro del entorno se ubiquen al pie de la página, debemos usar, en lugar de `\footnote` el comando `\footnotetext` en combinación con el comando `\note` que más arriba se ha mencionado. El primero, que también admite como argumento opcional una etiqueta, imprime sólo el texto de la nota, pero no la marca. Pero como `\note` imprime sólo la marca sin el texto, la combinación de ambos nos permite ubicar la nota en el punto en el que deseemos y así por ejemplo, podríamos escribir dentro de una tabla o de un entorno multicolumnas `\note[MiEtiqueta]{Texto de la nota}`, y luego, una vez fuera de dicho entorno `\footnotetext[MiEtiqueta]{Texto de la nota}`.

Otro ejemplo del uso de `\footnotetext` en combinación con `\note` sería el de las notas dentro de otras notas. Por ejemplo:

<pre>Esta% \footnote{o esta\note[notaB], si lo prefieres.}% \footnotetext[notaB]{o posiblemente incluso esta otra\note[notaC].}\footnotetext[notaC]{podría ser algo enteramente diferente.} es una frase con notas anidadas.</pre>	<p>Esta¹ es una frase con notas anidadas.</p> <hr/> <p>¹o esta², si lo prefieres.</p> <p>²o posiblemente incluso esta otra³.</p> <p>³podría ser algo enteramente diferente.</p>
--	---

\endnote sólo imprime, en el punto del fichero fuente en el que se encuentre, la marca de llamada a la nota. El contenido propiamente dicho de la nota se inserta en otro punto del documento con otro comando (**\placenotes[*endnote*]**) que insertará, en el punto en el que se encuentre, el contenido de *todas* las notas finales del documento (o del capítulo o sección de que se trate).

12.1.3 Notas locales

El entorno **\startlocalfootnotes** hace que las notas al pie incluidas dentro de él se consideren notas *locales*, lo que implica que se reiniciará la numeración de las mismas y que el contenido de las notas no se insertará automáticamente junto con el resto de las notas, sino sólo en el punto del documento en el que se encuentre el comando **\placelocalfootnotes**, el cual puede, o no, encontrarse dentro del entorno.

12.1.4 Creación y utilización de tipos personalizados de notas

Podemos crear tipos especiales de notas con el comando **\definernote**. Esto puede ser útil en documentos complejos en los que hay notas de distintos autores, o con diferentes finalidades, para diferenciar gráficamente, mediante un formato distinto y una numeración diferente, cada uno de los tipos de notas de nuestro documento.

La sintaxis de **\definernote** es la siguiente:

```
\definernote [Nombre] [Modelo] [Configuración]
```

donde

- *Nombre* es el nombre que asignaremos a nuestro nuevo tipo de nota.
- *Modelo* es el modelo de nota que se usará inicialmente. Puede ser **footnote** o **endnote**; en el primer caso nuestro modelo de nota funcionará como las notas a pie de página, y en el segundo como las notas finales, si bien para insertarlas en el documento no usaríamos **\placenotes[*endnote*]** sino **\placenotes[*Nombre*]** (el nombre que hayamos asignado a nuestro tipo de notas).

En teoría este argumento es opcional, aunque en mis pruebas algunas notas creadas sin él no eran visibles, y tampoco he tenido paciencia para indagar cuál era la causa.

- *Configuración* es un segundo argumento opcional que permite diferenciar nuestras nuevo tipo de notas de su modelo: bien estableciendo un formato distinto, bien un tipo de numeración diferente, bien ambas cosas.

Según el listado oficial de comandos de ConT_EXt (véase [sección 3.6](#)); la configuración que se puede hacer en el momento de la creación del nuevo tipo de nota se basa en la que se podría hacer después con `\setupnote`. Sin embargo, como veremos en seguida, en realidad hay dos comandos posibles para la configuración de las notas: `\setupnote` y `\setupnotation`. Por ello creo que es preferible omitir este argumento en el momento de la creación del tipo de nota, y luego configurar nuestras nuevas notas usando los comandos adecuados. Al menos eso es más fácil de explicar.

Por ejemplo: el siguiente fragmento creará un nuevo tipo de notas llamado «NotaAzul» que será similar a las notas a pie de página, pero imprimirá su contenido en negrita y en color azul:

```
\definernote [NotaAzul] [footnote]
\setupnotation
  [NotaAzul]
  [color=blue, style=bf]
```

Una vez que hemos creado un nuevo tipo de nota, por ejemplo *NotaAzul* estará disponible el comando que permita usarlas. En nuestro ejemplo este sería `\NotaAzul` cuya sintaxis sería similar a la de `\footnote`:

```
\NotaAzul[Etiqueta]{Texto}
```

12.1.5 Configuración de las notas

La configuración de las notas (al pie, finales, creadas con `\definernote` e incluso de las notas lineales, establecidas con `\linenote`) se realiza mediante dos comandos: `\setupnote` y `\setupnotation`¹. La sintaxis de ambos es similar:

¹ `\setupnote` dispone de 35 opciones *directas* de configuración y 45 opciones adicionales heredadas de `\setupframed`; `\setupnotation` cuenta con 45 opciones directas de configuración y otras 23 heredadas de `\setupcounter`. Dado que estas opciones no están documentadas y, aunque por el nombre de muchas de ellas puede intuirse su utilidad, hay que comprobar si nuestra intuición es o no cierta; y teniendo en cuenta también que muchas de esas opciones admiten varios valores y hay que probarlos todos... Se verá que para escribir esta explicación he tenido que hacer bastantes pruebas; y aunque hacer una prueba es rápido, hacer muchas pruebas es lento y aburrido. Por ello espero que el lector me disculpe si le digo que además de los dos comandos de configuración general de notas que menciono en el texto principal, y en los que se centra la explicación que sigue, existen otros cuatro posibles comandos de configuración que sin embargo omitiré en mi explicación:

- `\setupnotes` y `\setupnotations`: O sea, el mismo nombre pero en plural. La wiki dice que las versiones en singular y plural del comando son sinónimas, y yo me lo creo.

```
\setupnote[TipoNota][Configuración]
\setupnotation[TipoNota][Configuración]
```

donde *TipoNota* se refiere al tipo de nota que estamos configurando (`footnote`, `endnote` o el nombre de algún tipo de nota creado por nosotros mismos), y *configuración* contiene las concretas opciones de configuración del comando.

El problema es que el nombre de estos dos comandos no resulta muy clarificador para saber cuál es la diferencia entre ellos y qué aspectos configura cada uno; y el hecho de que gran parte de las opciones de estos comandos no estén documentadas no ayuda mucho. Yo tras muchas pruebas no he conseguido llegar a ninguna conclusión que permita comprender por qué ciertos aspectos se configuran con uno de ellos y ciertos aspectos se configuran con el otro¹, salvo acaso la de que, por las opciones que he conseguido hacer funcionar, `\setupnotation` siempre afecta al texto de la nota, o al identificador que se imprime con el texto de la nota, mientras que `\setupnote` tiene alguna opción que afecta a la marca de la nota que se inserta en el texto principal.

A continuación intento sistematizar lo que he sacado en claro tras hacer algunas pruebas con las diferentes opciones de ambos comandos. Dejo sin mencionar la mayor parte de las opciones de ambos, pues no están documentadas y no he podido sacar ninguna conclusión de para qué sirven o en qué condiciones deben utilizarse:

- **Identificador usado para la marca:**

Las notas se identifican siempre por un número. Lo que podemos aquí configurar es

-
- `\setupfootnotes` y `\setupendnotes`: Se supone que son aplicaciones concretas para, respectivamente, las notas a pie de página y las notas finales. Tal vez explicar la configuración de las notas a partir de estos comandos sería más sencillo, sin embargo, como la primera opción que probé en `\setupfootnotes`, que fue `numberconversion`, no conseguí hacerla funcionar, aunque me consta que otras opciones de estos comandos sí funcionan... me dio pereza añadir a las múltiples pruebas que he tenido que hacer para redactar lo que sigue, las pruebas necesarias para incluir en la explicación a estos dos comandos.

Lo que sí creo (por unas pocas pruebas aleatorias) es que todo lo que funciona en estos dos comandos cuya explicación omito, funciona también en los comandos cuya explicación se realiza.

¹ Hay una página en la [wiki de ConTExT](#), que descubrí por casualidad (pues no está específicamente dedicada a las notas), que cifra la diferencia en que `\setupnotation` controla el texto de la nota que se inserta, y `\setupnote` el entorno de la nota en el que ésta se situará (¿?). Pero esto es contradictorio con el hecho de que, por ejemplo, la anchura que haya de tener el texto de la nota (que tiene que ver con la *inserción* de la misma) se controle con la opción `width` de `\setupnote` y no con la opción del mismo nombre de `\setupnotation` que lo que controla es la anchura del espacio entre la marca y el texto de la nota.

- *El número inicial:* Lo controla la opción `start` de `\setupnotation`. Su valor ha de ser un número entero, a partir del cual se empezarán a contar las notas.
 - *El sistema de numeración,* el cual depende de la opción `numberconversion` de `\setupnotation`. Sus valores pueden ser:
 - ★ *Numeración con números arábigos:* `n`, `N` o `numbers`.
 - ★ *Numeración con números romanos:* `I`, `R`, `Romannumerals`, `i`, `r`, `romannumerals`. Las tres primeras usan números romanos en mayúsculas y las tres últimas en minúsculas.
 - ★ *Numeración con letras:* `A`, `Character`, `Characters`, `a`, `character`, `characters` según queramos que las letras sean en mayúsculas (las primeras opciones) o en minúsculas (las restantes).
 - ★ *Numeración con palabras.* Es decir: se escribe la palabra que designa al número y así, por ejemplo, «3» se convierte en «tres». Admite dos modalidades. La opción `Words` escribe las palabras en mayúsculas y `words` en minúsculas.
 - ★ *Numeración con símbolos:* Puede utilizar cuatro juegos de símbolos distintos según la opción elegida: `set 0`, `set 1`, `set 2` o `set 3`. En la [página 147](#) hay un ejemplo de los símbolos usados en cada una de estas opciones.
 - *El evento que determina que se reinicie la numeración de las notas:* Esto depende de la opción `way` de `\setupnotation`. Con el valor `bytext` todas las notas del documento se numerarán secuencialmente sin que la numeración se reinicie. Con `bychapter`, `bysection`, `bysubsection`, etc. se reiniciará el contador de notas cada vez que se cambie, respectivamente, de capítulo, sección o subsección, mientras que `byblock` reinicia la numeración cada vez que se cambia de bloque en la macroestructura del documento (véase la [sección 7.6](#)). El valor `bypage` hace que el contador de notas se reinicie cada vez que se cambia de página.
- **Configuración de la marca de la nota:**
 - Si se debe o no mostrar: opción `number` de `\setupnotation`.
 - Ubicación de la marca en relación con el texto de la nota: Opción `alternative` de `\setupnotation`: Puede asumir cualquiera de los siguientes valores: `left`, `inleft`, `leftmargin`, `right`, `inright`, `rightmargin`, `inmargin`, `margin`, `innermargin`, `outermargin`, `serried`, `hanging`, `top`, `command`.
 - Formato de la marca en la nota propiamente dicha: Opción `numbercommand` de `\setupnotation`.
 - Formato de la marca en el cuerpo del texto: Opción `textcommand` de `\setupnote`.

Las opciones `numbercommand` y `textcommand` han de consistir en un comando que reciba como argumento el contenido de la marca. Puede ser un comando definido por nosotros mismos. No obstante, he comprobado que los comandos simples de cambio de formato (`\bf`, `\it`, etc.), aunque no son comandos que hayan de recibir ningún argumento, funcionan.

- Distancia entre la marca y el texto (en la nota propiamente dicha): Opciones `distance` y `width` de `\setupnotation`. No he conseguido descubrir la diferencia (si la hay) entre usar una opción o la otra.
- Existencia o no de hipervínculo que permita saltar entre la marca en el texto principal y la marca en la nota propiamente dicha: Opción `interaction` de `\setupnote`. Con el valor `yes` habrá hipervínculo, y con `no` no lo habrá.

- **Configuración del texto de la nota propiamente dicho.**

Podemos influir en los siguientes aspectos:

- Ubicación: Esto depende de la opción `location` de `\setupnote`.

En principio ya sabemos que las notas al pie se ubican en la parte inferior de la página (`location=page`) y las notas finales en el punto en el que se encuentre el comando `\placenotes[endnote]` (`location=text`), sin embargo podemos alterar este funcionamiento y establecer, por ejemplo, para las notas a pie de página `location=text` lo que hará que las notas a pie funcionen de modo similar a las notas finales y se inserten en el punto del documento en el que aparezca el comando `\placenotes[footnote]`, o el comando, específico para las notas a pie de página `\placefootnotes`. Por este procedimiento podríamos, por ejemplo, imprimir las notas debajo del párrafo en el que se encuentran.

- Separación por párrafos entre notas: Por defecto cada nota se imprime en su propio párrafo, pero podemos hacer que todas las notas de una misma página se impriman en un mismo párrafo estableciendo la opción `paragraph` de `\setupnote` como «yes».
- Estilo en que se escribirá el texto de la nota propiamente dicho: Opción `style` de `\setupnotation`.
- Tamaño de la letra: Opción `bodyfont` de `\setupnote`.

Esta opción es sólo para el caso de que queramos establecer manualmente un tamaño de letra para las notas a pie de página. Casi nunca es buena idea hacerlo pues, por defecto, ConTeXt ajusta el tamaño de la letra de las notas a pie de página para que

sea más pequeño que el del texto principal, pero con un tamaño *proporcionado* al del tamaño de la letra en el cuerpo principal.

- Margen izquierdo que tendrá el texto de la nota: Opción `margin` de `\setupnotation`.
 - Anchura máxima: Opción `width` de `\setupnote`.
 - Número de columnas: Opción `n` de `\setupnote` determina que las notas se escriban en dos o más columnas. El valor de «n» ha de ser un número entero.
- **Separación entre las notas o separación entre las notas y el texto:**
Aquí disponemos de las siguientes opciones:
 - `rule`, de `\setupnote` establece si habrá o no una línea de separación entre la zona de notas y la zona de la página con el texto principal. Sus valores posibles son `yes`, `on`, `no` y `off`. Los dos primeros activan la línea y los dos últimos la desactivan.
 - `before`, de `\setupnotation`: comando o comandos que se deben ejecutar antes de insertar el texto de la nota. Sirve para insertar espacio adicional de separación, líneas separadoras entre notas, etc.
 - `after`, de `\setupnotation`: comando o comandos que se deben ejecutar tras insertar el texto de la nota.

12.1.6 Excluir de la compilación, temporalmente, las notas

Los comandos `\notesenabledfalse` y `\notesenabledtrue` indican a ConT_EXt, respectivamente, que inhabilite o rehabilite la compilación de las notas. Esta función puede ser útil si se desea obtener una versión sin notas de un documento con numerosas y extensas notas. En mi experiencia personal, por ejemplo, cuando corrijo una tesis doctoral, prefiero leerla la primera vez de un tirón, sin las notas, y luego hacer una segunda lectura con las notas incorporadas.

12.2 Párrafos con múltiples columnas

La composición del texto en más de una columna es una posibilidad que se puede establecer:

- a. Como característica general del diseño de la página.
- b. Como característica de ciertas construcciones tales como, por ejemplo, las listas estructuradas, o las notas a pie de página o finales.
- c. Como característica aplicada a ciertos párrafos concretos del documento.

En cualquiera de estos casos, la mayor parte de los comandos y entornos funcionarán perfectamente aunque se trabaje con más de una columna. Hay no obstante algunas limitaciones; principalmente en relación con los objetos flotantes en general (véase la [sección 13.1](#)) y con las tablas en particular ([sección 13.3](#)) aunque no sean flotantes.

Respecto al número máximo de columnas admisibles, en teoría ConTeXt no lo limita. No obstante existen dos limitaciones físicas que hay que tomar en consideración:

- La anchura del papel: Un número ilimitado de columnas exige una anchura ilimitada de papel (si el documento se va a imprimir) o de pantalla (si es un documento pensado para ser mostrado en pantalla). En la práctica, teniendo en cuenta la anchura *normal* de los papeles que se comercializan y con los que se componen libros, y de las pantallas de dispositivos informáticos, es difícil que un texto compuesto con más de cuatro o cinco columnas, se vea bien.
- El tamaño de la memoria del ordenador. Al respecto señala el manual de referencia de ConTeXt que, en sistemas *normales* (ni especialmente potentes ni especialmente limitados de recursos) se podrán manejar entre 20 y 40 columnas.

En la presente sección me centraré en el uso del mecanismo multicolumnas en párrafos o fragmentos especiales, pues

- De las múltiples columnas como opción de diseño de página ya se ha hablado (en el [apartado B](#) de la [sección 5.3.4](#)).
- De la posibilidad ofrecida en ciertas construcciones como listas estructuradas o notas a pie de página de componer texto en más de una columna, se habla a propósito de la construcción o entorno de que se trate.

12.2.1 El entorno `\startcolumns`

El procedimiento normal para insertar en el documento fragmentos compuestos con varias columnas consiste en usar el entorno `columns` cuyo formato es:

```
\startcolumns[Configuración] ... \stopcolumns
```

donde *Configuración* permite controlar numerosos aspectos del entorno. Podemos indicar la configuración deseada cada vez que invoquemos el entorno, o adaptar

el funcionamiento por defecto del entorno para todas las invocaciones al entorno, cosa ésta última que se conseguiría con

`\setupcolumns` [Configuración]

En uno y otro caso las opciones de configuración son las mismas. Las más importantes, ordenadas atendiendo a su función, son las siguientes:

- **Opciones que controlan el número de columnas y la separación entre ellas:**
 - **n:** Controla el número de columnas. Si se omite esta opción se generan dos columnas.
 - **nleft, nright:** Estas dos opciones se usan, en documentos diseñados a doble página (véase el [apartado A](#) de la [sección 5.3.4](#)), para establecer, respectivamente, el número de columnas en las páginas izquierdas (pares) y en las derechas (impares).
 - **distance:** Distancia de separación entre las columnas.
 - **separator:** Determina el elemento de separación entre las columnas. Puede ser **space** (valor por defecto) o **rule** en cuyo caso se generará una línea entre las columnas. En el caso de que se establezca una línea de separación entre las columnas, entonces ésta línea se puede a su vez configurar con las siguientes dos opciones:
 - ★ **rulecolor:** Color de la línea de separación.
 - ★ **rulethickness:** Grosor de la línea de separación.
 - **maxwidth:** Anchura máxima que pueden tener todas las columnas + el espacio de separación entre ellas.
- **Opciones que controlan la distribución del texto en las columnas:**
 - **balance:** Por defecto ConTeXt *balancea* las columnas, es decir, reparte el texto entre ellas para que todas tengan, más o menos, la misma cantidad de texto. Pero si se establece esta opción con el valor «no» el texto no empezará a introducirse en una columna hasta que la anterior esté totalmente llena.
 - **direction:** Determina en qué dirección se distribuye el texto entre las columnas. Por defecto, se sigue el orden natural de lectura (de izquierda a

derecha), pero dándole a esta opción el valor `reverse` se consigue el orden inverso.

- **Opciones que afectan a la composición del texto dentro del entorno:**

- **tolerance:** La escritura de texto en varias columnas determina que la anchura de la línea, dentro de la columna, sea más pequeña, lo que, tal y como se explicó al exponer el mecanismo de construcción de líneas de ConTeXt ([sección 11.3](#)), dificulta la localización de puntos óptimos para insertar saltos de línea. Esta opción permite alterar temporalmente, dentro del entorno, la tolerancia horizontal (véase [sección 11.3.3](#)), lo que puede facilitar la composición del texto.
- **align:** Controla la alineación horizontal de las líneas dentro del entorno. Puede asumir cualquiera de los siguientes valores: `right`, `flushright`, `left`, `flushleft`, `inner`, `flushinner`, `outer`, `flushouter`, `middle` o `broad`. Sobre el significado de estas opciones, véase la [sección 11.6.1](#).
- **color:** Especifica el nombre del color con que se escribirá el texto dentro del entorno.

12.2.2 Párrafos paralelos

Una versión concreta de la composición con múltiples columnas es la de los párrafos paralelos. En este tipo de construcción el texto se distribuye en dos columnas (normalmente, aunque a veces se usan más de dos), pero no se permite que fluya libremente entre ellas, sino que se mantiene un rígido control de lo que aparecerá en cada columna. Esto es muy útil, por ejemplo, para generar documentos en los que se puedan contrastar dos versiones de un texto, como, por ejemplo, la versión nueva y la versión vieja de una ley recientemente modificada, o para ediciones bilingües; o también para redactar glosarios de textos definidos, en donde el texto a definir aparece a la izquierda y la definición a la derecha, etc.

Normalmente para procesar este tipo de párrafos se usa el mecanismo de las tablas; pero ello es porque la mayoría de los sistemas de composición de textos no son tan potentes como ConTeXt donde se dispone de los comandos `\defineparagraphs` y `\setupparagraphs` que permiten construir este tipo de párrafos acudiendo al mecanismo de las columnas, el cual, aunque tiene limitaciones, es más flexible que el de las tablas.

Hasta donde yo sé, este tipo de párrafos no tiene un nombre especial. Los he llamado «párrafos paralelos» porque me ha parecido un nombre más descriptivo que el que usa ConTeXt para referirse a esta construcción: «*paragraphs*» que significa simplemente «párrafos».

El comando básico aquí es `\defineparagraphs` cuya sintaxis es:

```
\defineparagraphs [Nombre] [Configuración]
```

siendo *Nombre* el nombre que daremos a nuestra construcción, y *Configuración* las características que tendrá, las cuales también se pueden fijar en un momento ulterior mediante

```
\setupparagraphs[Nombre][Columna][Configuración]
```

donde *Nombre* es el nombre que se estableció en el momento de la creación, *Columna* es un argumento opcional que si se indica, permite configurar una columna concreta, y *Configuración* permite determinar el funcionamiento concreto.

Por ejemplo:

```
\defineparagraphs
  [HechosMurcianos]
  [n=3, before={\blank},after={\blank}]
```

```
\setupparagraphs
  [HechosMurcianos][1]
  [width=.1\textwidth, style=bold]
```

```
\setupparagraphs
  [HechosMurcianos][2]
  [width=.4\textwidth]
```

El anterior fragmento crearía un entorno de tres columnas llamado HechosMurcianos y luego configuraría la primera columna para que ocupe un 10% del ancho de la línea y se escriba con negrita, y configuraría la segunda para que ocupe un 40% del ancho de la línea. Como no se configura la tercera columna, está tendrá la anchura que queda, o sea, el 50%.

Una vez creado el entorno, podríamos escribir con él nuestra pequeña historia murciana:

```
\startHechosMurcianos
  825
\HechosMurcianos
  Se funda la ciudad de Murcia.
\HechosMurcianos
  Los orígenes de la ciudad de Murcia son inciertos, pero hay
  constancia de que fue mandada fundar con el nombre de Madina (o
  Medina) Mursiya en el año 825 por el emir de al-Ándalus Abderramán
  II, probablemente sobre un asentamiento muy anterior.
\stopHechosMurcianos
```

825 Se funda la ciudad de Murcia.

Los orígenes de la ciudad de Murcia son inciertos, pero hay constancia de que fue mandada fundar con el nombre de Madina (o Medina) Mursiya en el año 825 por el emir de al-Ándalus Abderramán II, probablemente sobre un asentamiento muy anterior.

Si quisiéramos seguir narrando la historia de Murcia, para el siguiente evento sería precisa una nueva instancia del entorno (`\startHechosMurcianos`), pues con este mecanismo no es posible incluir varias *filas*.

Del ejemplo que se acaba de poner, quisiera resaltar los siguientes detalles:

- Una vez creado el entorno con, por ejemplo, `\defineparagraphs[MariPoppins]`, este pasa a funcionar como un entorno normal, que se inicia con `\startMariPoppins` y se termina con `\stopMariPoppins`.
- Se crea también un comando `\MariPoppins` que se usa, dentro del entorno para indicar cuándo hay que cambiar de columna.

En cuanto a las opciones de configuración de los párrafos paralelos (`\setupparagraphs`), entiendo que, a estas alturas de la introducción, y teniendo en cuenta el ejemplo que se acaba de poner, el lector ya está preparado para deducir por sí mismo la finalidad de cada una de las opciones, por ello, a continuación, tan sólo indicaré el nombre y el tipo de las opciones así como, en su caso, los valores posibles. Recuérdese, eso sí, que `\setupparagraphs [Nombre] [Configuración]` establece configuraciones que afectan a todo el entorno, mientras que `\setupparagraphs [Nombre] [NumColumna] [Configuración]` aplica configuraciones exclusivamente a la columna que se le indique.

- | | | |
|-------------------------------------|--|--|
| • <code>n</code> : Número | • <code>align</code> : Derivado de <code>\setuptalign</code> | • <code>rulecolor</code> : Color |
| • <code>before</code> : Comando | • <code>inner</code> : Comando | • <code>style</code> : Comando de estilo |
| • <code>after</code> : Comando | • <code>rule</code> : on off | • <code>color</code> : Color |
| • <code>width</code> : Dimensión | • <code>rulethickness</code> : Dimensión | |
| • <code>distance</code> : Dimensión | | |

La anterior lista de opciones no es completa; he excluido de la lista de opciones las que yo normalmente no explicaría aquí. He aprovechado también que estamos en el epígrafe dedicado a las columnas para mostrar la lista de opciones a triple columna, aunque ello no le he realizado con ninguno de los comandos que se explican en esta sección, sino con la opción `columns` del entorno `itemize`, al que se dedica el próximo epígrafe.

12.3 Listas estructuradas

Cuando la información se presenta de manera ordenada es más fácilmente aprehensible por el lector. Pero, a su vez, si la ordenación de la misma es perceptible visualmente, se acentúa para el lector la sensación de estar ante un texto estructurado. Por ello existen ciertas *construcciones* o *mecanismos* que, procuran destacar gráfica y visualmente la ordenación del texto, contribuyendo así a su estructuración. De las herramientas que ConTeXt pone a disposición del autor para ello la más importante, objeto de esta sección, es el entorno `itemize` que sirve para elaborar lo que podríamos llamar *listas estructuradas*.

En español la construcción que se realiza mediante un entorno del tipo de `itemize` es llamada simplemente *lista*. Yo prefiero hablar de «listas estructuradas» porque ConT_EXt denomina «listas» a algunos elementos de los documentos a los que en español solemos llamar «índice» (como, entre otros, el índice de imágenes o el índice de tablas).

Las listas consisten en una sucesión de *elementos de texto* (a los que llamaré *items*¹) en la que cada uno de ellos va precedido de un carácter que contribuye a resaltarlo diferenciándolo de los restantes, y al que aquí llamaré «separador». El separador puede ser un número, una letra o un símbolo. Habitualmente (pero no siempre) los *items* constituyen párrafos, y la lista se formatea para garantizar la *visibilidad* del separador de cada elemento; normalmente mediante la aplicación de una sangría francesa que lo haga destacar². En el caso de listas anidadas la sangría de cada lista va aumentando progresivamente. Con terminología que creo que procede del lenguaje HTML se suele llamar *listas ordenadas* a aquellas en las que el separador es un número o carácter que sigue una secuencia numérica o alfabética, de tal modo que cada *item* de la lista tendrá un separador diferente que nos permitirá referirnos a cada elemento por su número o identificador; y *listas no ordenadas* a aquellas otras en las que se usa como separador un carácter o símbolo que será el mismo para todos los elementos de la lista.

En las listas ConT_EXt gestiona automáticamente la numeración o secuenciación alfabética del separador en listas numeradas, así como, en el caso de listas anidadas, la sangría que cada una de ellas debe tener; y, si se trata de anidamiento de listas no ordenadas, la selección de un carácter o símbolo diferente que permita diferenciar a simple vista el nivel de un *item* de la lista atendiendo al símbolo que le precede.

El manual de referencia dice que el nivel máximo de anidamiento en listas es de 4, pero supongo que eso era así en 2013, fecha de redacción del manual. De acuerdo con mis pruebas no hay, aparentemente, límite en el anidamiento de listas *ordenadas* (en mis pruebas he llegado hasta 15 niveles de anidamiento), mientras que para las listas no ordenadas, tampoco parece haber un límite en el sentido de que por más anidamientos que incluyamos, no se generará ningún tipo de error; pero, para listas no ordenadas ConT_EXt sólo contempla símbolos predeterminados para los nueve primeros niveles de anidamiento.

En todo caso hay que indicar que el uso excesivo del anidamiento en listas puede provocar el efecto contrario al que se pretende con ellas, y es que el lector se sienta perdido, incapaz de ubicar cada elemento en la estructura general del listado. Por ello personalmente opino que si bien las listas son una herramienta poderosa para estructurar un texto, casi nunca es buena idea ir más allá del tercer nivel de anidamiento; e incluso llegar al tercer nivel debe hacerse sólo en casos muy justificados.

¹ Entre los distintos significados que el Diccionario de la Lengua Española atribuye a *item* hay tres que encajan con este sentido. Principalmente el cuarto significado que es: «Cada uno de los elementos que forman parte de un conjunto de datos».

² En tipografía se llama *sangría francesa* a una sangría que se aplica a todas las líneas de un párrafo salvo a la primera, lo que hace que la primera palabra o carácter del párrafo sea fácilmente localizable.

La herramienta general para la escritura de listas en ConT_EXt es el entorno `\itemize` cuya sintaxis es la siguiente:

```
\startitemize[Opciones][Configuración] ... \stopitemize
```

donde los dos argumentos son opcionales. El primero admite como contenido nombres simbólicos que tienen asignada por ConT_EXt una significación precisa; el segundo argumento, que se utiliza muy raramente, permite asignar valores concretos a ciertas variables que afectan al funcionamiento del entorno.

12.3.1 Selección de tipo de lista y de separador entre *items*

A. Listas no ordenadas

Por defecto la lista generada por `itemize` es una lista no ordenada, en la que el separador se seleccionará automáticamente dependiendo del nivel de anidamiento:

- | | |
|---|---|
| • Para el primer nivel de anidamiento. | ○ Para el sexto nivel de anidamiento. |
| – Para el segundo nivel de anidamiento. | ○ Para el séptimo nivel de anidamiento. |
| ★ Para el tercer nivel de anidamiento. | □ Para el octavo nivel de anidamiento. |
| ▷ Para el cuarto nivel de anidamiento. | ✓ Para el noveno nivel de anidamiento. |
| ◦ Para el quinto nivel de anidamiento. | |

No obstante podemos indicar expresamente que queremos que se utilice el símbolo asociado a un nivel concreto simplemente pasando el número de nivel como argumento. Así `\startitemize[4]` generará una lista no ordenada en la que se usará como separador el carácter ▷, con independencia del nivel de anidamiento que tenga la lista.

También podemos modificar el símbolo predeterminada para cada nivel mediante `\definesymbol`:

```
\definesymbol[Nivel]{Símbolo asociado al nivel}
```

Por ejemplo

```
\definesymbol[1]{\diamond}
```

hará que para el primer nivel de las listas no ordenadas se utilice el símbolo ◊. Con este mismo comando podemos asignar algún símbolo a niveles de anidamiento superiores al noveno. Así, por ejemplo

```
\definesymbol[10]{\copyright}
```

Asignará al nivel 10 de anidamiento el símbolo internacional del *copyright*: ©.

B. Listas ordenadas

Para generar una lista ordenada debemos indicar a `\itemize` el tipo de ordenación que deseamos. Puede ser:

n	1, 2, 3, 4, ...	a	a, b, c, d, ...	r	i, ii, iii, iv, ...
m	1, 2, 3, 4, ...	A	A, B, C, D, ...	R	I, II, III, IV, ...
g	$\alpha, \beta, \gamma, \delta, \dots$	KA	A, B, C, D, ...	KR	I, II, III, IV, ...
G	A, B, Γ , Δ , ...				

La diferencia entre **n** y **m** está en la fuente que se usa para representar el número: **n** usa la fuente activa en cada momento, mientras que **m** usa una fuente diferente, más elegante y casi caligráfica.

No sé cuál es el nombre de la fuente que usa **m**, y por ello en la lista anterior no he conseguido representar con exactitud el tipo de números que esta opción genera. Sugiero al lector que haga una prueba para verlos por sí mismo.

12.3.2 Introducción de los elementos de la lista

Como regla, los elementos de una lista creada mediante `\startitemize` se introducen mediante el comando `\item` que también tiene una versión en forma de entorno, más adecuada al estilo de Mark IV: `\startitem ... \stopitem`. Así, el siguiente ejemplo:

<pre>\startitemize[a, packed] \startitem Primer elemento \stopitem \startitem Segundo elemento \stopitem \startitem Tercer elemento \stopitem \stopitemize</pre>	<pre>a. Primer elemento b. Segundo elemento c. Tercer elemento</pre>
--	--

produce exactamente el mismo resultado que

<pre>\startitemize[a, packed] \item Primer elemento \item Segundo elemento \item Tercer elemento \stopitemize</pre>	<pre>a. Primer elemento b. Segundo elemento c. Tercer elemento</pre>
---	--

`\item` o `\startitem` es el comando *general* para introducir un elemento en la lista. Junto con él existen los siguientes comandos adicionales para cuando se quiera conseguir un resultado especial:

\head Este comando debe usarse, en lugar de `\item` cuando queremos evitar que se inserte un salto de página después del elemento en cuestión.

Una construcción corriente es la de inmediatamente debajo de un elemento de una lista incluir una lista anidada, o un bloque de texto, de tal modo que el elemento de la lista, en cierto modo, funciona como *título* de la sublista o bloque de texto. En estos casos un salto de página entre dicho elemento y los párrafos posteriores estaría desaconsejado. Si usamos `\head` en lugar de `\item` para introducir estos elementos ConTeXt *procurará* (en la medida de lo posible) no separar a dicho elemento del bloque siguiente.

- `\sym` El comando `\sym{Texto}` introduce un elemento en el que como *separador* se utiliza, no un número o símbolo, sino el texto usado como argumento de `\sym`. Esta lista, por ejemplo, se construye con elementos introducidos mediante `\sym` en lugar de `\item`.
- `\sub` Este comando, que se debe usar sólo en las llamadas listas ordenadas (en las que cada elemento va precedido de un número o letra en secuencia alfabética) provoca que el elemento introducido con él conserve el número del elemento anterior, al tiempo que, para indicar que el número está repetido, y que ello no es un error, a su izquierda se imprime el signo «+». Esto puede ser útil si nos estamos refiriendo a una lista previa, sobre la que sugerimos modificaciones, pero en la que, para mayor claridad, debe mantenerse la numeración de la lista original.
- `\mar` Este comando mantiene la numeración de los items, pero añade en el margen una letra o carácter (que se le pasa como argumento, entre llaves). No estoy muy seguro de su utilidad.

Hay dos comandos adicionales para introducir elementos, cuya combinación produce efectos *interesantes* y, si se me permite, creo que es mejor explicarlos con un ejemplo. se trata de `\ran` (diminutivo de *rango*) e `\its`, diminutivo de *items*. El primero recibe un argumento (entre llaves) y el segundo repite el símbolo usado como separador en la lista x número de veces (por defecto 4 veces, pero podemos alterar eso mediante la opción `items`). En el siguiente ejemplo se muestra cómo pueden trabajar juntos estos dos comandos para crear una lista que imita a un cuestionario:

Tras la lectura de esta introducción, "contexte"\footnote{Llámase «contextar» a contestar preguntas sobre \ConTeXt.} a las siguientes cuestiones:

```
\startitemize[5, packed][width=8em, distance=2em, items=5]
\ran{No \hss Si}
\its No usaré nunca \ConTeXt, es muy difícil.
\its Sólo lo usaré para escribir libros largos.
\its Lo usaré siempre.
\its Me gusta tanto que llamaré a mi próximo hijo "Hans", como
      homenaje a Hans Hagen.
\stopitemize
```

Tras la lectura de esta introducción, *contexte*¹ a las siguientes cuestiones:

No	Si
◦ ◦ ◦ ◦ ◦	No usaré nunca ConT _E Xt, es muy difícil.
◦ ◦ ◦ ◦ ◦	Sólo lo usaré para escribir libros largos.
◦ ◦ ◦ ◦ ◦	Lo usaré siempre.
◦ ◦ ◦ ◦ ◦	Me gusta tanto que llamaré a mi próximo hijo “Hans”, como homenaje a Hans Hagen.

¹ Llámase «contextar» a contestar preguntas sobre ConT_EXt.

12.3.3 Configuración básica de las listas

Recordemos que «*itemize*» admite dos argumentos. Ya hemos visto como el primer argumento permite seleccionar el tipo de lista que deseamos. Pero también podemos indicar mediante él otras características de la lista, ello se hace mediante las siguientes opciones que se le indican a «*itemize*» en su primer argumento:

- **columns**: Esta opción determina que la lista se componga con dos o más columnas. Tras la opción **columns** hay que escribir con letras y separado por una coma, el número de columnas deseadas: *two*, *three*, *four*, *five*, *six*, *seven*, *eight* o *nine*. **Columns** no seguido de ningún número genera dos columnas.
- **intro**: Esta opción procura no separar mediante un salto de línea a la lista del párrafo que la precede.
- **continue**: En listas ordenadas (numérica o alfabéticamente) esta opción hace que la lista continúe la numeración de la última lista numerada. Si se usa la opción **continue** no es preciso indicar qué tipo de lista queremos, pues se asume que será igual a la última lista numerada.
- **packed**: Es una de las opciones más utilizadas. Su uso provoca que el espacio vertical entre los distintos *items* de la lista se reduzca al máximo.
- **nowhite**: Produce un efecto similar a **packed**, pero más drástico: no sólo reduce al máximo el espacio vertical entre los elementos, sino también el espacio vertical entre la lista y el texto que la circunda.
- **broad**: Aumenta el espacio horizontal entre el separador de elementos y el texto del elemento. El espacio de aumento puede incrementarse multiplicando un número por **broad** como en, por ejemplo `\startitemize[2*broad]`.
- **serried**: Elimina el espacio horizontal entre el separador de elementos y el texto.
- **intext**: Elimina de la lista la sangría francesa.

- **text**: Elimina de la lista la sangría francesa y reduce el espacio vertical entre elementos.
- **repeat**: En listas anidadas hace que la numeración de un nivel hijo *repita* la del nivel previo. Así tendríamos, en el primer nivel: 1, 2, 3, 4; en el segundo: 1.1, 1.2, 1.3, etc. La opción hay que indicarla en la lista interior, no en la exterior.
- **margin**, **inmargin**: Por defecto el separador de la lista se imprime a la izquierda, pero dentro de la zona de texto propiamente dicha (**atmargin**). Las opciones **margin** e **inmargin** trasladan el separador al margen.

12.3.4 Configuración adicional de las listas

El segundo argumento, también opcional, de `\startitemize` permite una configuración más detallada y pormenorizada de las listas.

- **before**, **after**: Comandos a ejecutar, respectivamente, antes de iniciar o después de cerrar, el entorno `itemize`.
- **inbetween**: Comando a ejecutar entre dos `items`.
- **beforehead**, **afterhead**: Comando a ejecutar antes o después de un elemento introducido con el comando `\head`.
- **left**, **right**: Carácter que se imprimirá a la izquierda o a la derecha del separador. Por ejemplo, para conseguir listas alfabéticas en las que las letras vayan rodeadas de paréntesis habría que escribir:

```
\startitemize[a][lef=(, right=)]
```

- **stopper**: Indica un carácter que se escribirá después del separador. Sólo funciona en listas ordenadas.
- **width**, **maxwidth**: Anchura del espacio reservado para el separador y, por lo tanto, de la sangría francesa.
- **factor**: Número representativo del factor de separación entre el separador y el texto.
- **distance**: Medida de la distancia entre el separador y el texto.
- **leftmargin**, **rightmargin**, **margin**: Margen que se añade a la izquierda (**leftmargin**) o derecha (**rightmargin**) de los elementos.
- **start**: Número por el que empezará la numeración de elementos.

- `symalign`, `itemalign`, `align`: alineación de los elementos. Admite los mismos valores que `\setupalign`. `symalign` controla la alineación del separador; `itemalign` la del texto del elemento, y `align` la de ambos.
- `indenting`: Sangría de la primera línea en los párrafos dentro del entorno. Véase la [sección 11.1.1](#)
- `indentnext`: Indica si el párrafo posterior al entorno debe o no indentarse. Admite los valores *yes*, *no* y *auto*.
- `items`: En los elementos introducidos con `\its` indica el número de veces que se debe reproducir el separador.
- `style`, `color`; `headstyle`, `headcolor`; `marstyle`, `marcolor`; `symstyle`, `symcolor`: Estas opciones controlan el estilo y el color de los elementos según se hayan introducido en el entorno con los comandos `\item`, `\head`, `\mar` o `\sym`.

12.3.5 Listas simples con el comando `\items`

Una alternativa al entorno `itemize` para listas no numeradas muy simples, en las que los elementos no sean excesivamente largos es el comando `\items` cuyo formato es:

`\items[Configuración]{Elemento 1, Elemento 2, ..., Elemento n}`

Los distintos elementos que tendrá la lista se separan unos de otros por comas. Por ejemplo:

Los ficheros gráficos pueden tener, entre otras, las siguientes extensiones:

```
\items{png, jpg, tiff, bmp}
```

Los ficheros gráficos pueden tener, entre otras, las siguientes extensiones:

- png
- jpg
- tiff
- bmp

Las opciones de configuración que admite este comando son un subconjunto de las de `itemize`, salvo dos opciones específicas de este comando:

- `symbol`: Esta opción determina el tipo de lista que se generará. Admite los mismos valores que sirven en `itemize` para seleccionar algún tipo de lista.
- `n`: Esta opción indica a partir de qué número de elemento habrá separador.

12.3.6 Predeterminación del comportamiento de las listas y creación de nuestros propios tipos de lista

En las secciones anteriores hemos visto como indicar qué tipo de lista queremos y qué características debe tener. Pero hacer eso cada vez que se llama a una lista es poco eficiente y normalmente producirá un documento incoherente en el que cada lista tenga su propia apariencia, pero sin que las distintas apariencias respondan a algún criterio.

Resulta preferible por ello:

- Predeterminar, en el preámbulo del documento, el comportamiento *normal* de `itemize` y de `\items`.
- Crear nuestras propias listas personalizadas. Por ejemplo: una lista numerada alfabéticamente a la que podríamos llamar *ListaAlfa*, una lista numerada con números romanos (*ListaRomana*), etc.

Lo primero se consigue mediante los comandos `\setupitemize` y `\setupitems`. Lo segundo requiere usar, bien el comando `\defineitemgroup`, bien `\defineitems`. El primero creará un entorno de lista similar a `itemize` y el segundo un comando similar a `items`.

12.4 Descripciones y enumeraciones

Las descripciones y las enumeraciones son dos construcciones que permiten componer de manera consistente párrafos o grupos de párrafos que desarrollan, describen, o definen una frase o palabra.

12.4.1 Descripciones

En las descripciones se diferencia entre un *título* y su explicación o desarrollo. Podemos crear una nueva descripción mediante:

```
\definedescription[Nombre] [Configuración]
```

donde *Nombre* es el nombre por el que esta nueva construcción será conocida, y *Configuración* controla el aspecto que tendrá nuestra nueva estructura. Tras la anterior sentencia dispondremos de un nuevo comando y de un entorno con el nombre que hayamos elegido. Así:

```
\definedescription[Concepto]
```


creará los comandos:

```
\Concepto{Título}
\startConcepto {Título} ... \stopConcepto
```

Usaremos el comando para el caso en el que el texto explicativo del título conste de un sólo párrafo, y el entorno para títulos cuya descripción ocupe más de un párrafo. Cuando se usa el comando, el párrafo inmediatamente posterior a él es el que se considerará texto explicativo del título. Por el contrario cuando se usa el entorno, todo su contenido será formateado con el sangrado adecuado como para que quede clara su vinculación con el título.

Por ejemplo:

```
\definedescription
[Concepto]
[alternative=left, width=1cm, headstyle=bold]
```

```
\Concepto{Contextualizar}
```

Situar algo en un determinado contexto, o componer un texto con el sistema de composición de textos llamado `\ConTeXt`. La capacidad para contextualizar correctamente en cualquier situación viene siendo considerada signo de inteligencia y sensatez.

Esto generará el siguiente resultado:

Contextualizar Situar algo en un determinado contexto, o componer un texto con el sistema de composición de textos llamado `ConTeXt`. La capacidad para contextualizar correctamente en cualquier situación viene siendo considerada signo de inteligencia y sensatez.

Como resulta habitual en `ConTeXt` la apariencia que tendrá nuestra nueva construcción podemos indicarla en el momento de su creación, mediante el argumento *Configuración* o, más adelante, mediante `\setupdescription`:

```
\setupdescription[Nombre] [Configuración]
```

donde *Nombre* es el nombre de nuestra nueva descripción, y *Configuración* determina el aspecto de la misma. Entre las distintas opciones de configuración posibles destacaré:

- **alternative**: Esta opción es la que fundamentalmente controla la apariencia de la construcción. Determina la colocación del Título en relación con su descripción. Admite como valores posibles los de `left`, `right`, `inmargin`, `inleft`, `inright`, `margin`, `leftmargin`, `rightmargin`, `innermargin`, `outermargin`, `serried`, `hanging`, `top`, `empty`. La denominación de estos posibles valores creo que es lo bastante clara para (si sabemos algo de inglés) hacernos una idea

del resultado que tendrá la misma, aunque, en caso de duda, lo mejor es hacer una prueba para ver cómo queda.

- **width:** Controla la anchura de la caja en la que se escribirá el título. Dependiendo del valor de **alternative** esa distancia también formará parte del sangrado con el que se escriba el texto explicativo.
- **distance:** Controla la distancia entre el título y la explicación del mismo.
- **headstyle, headcolor, headcommand:** Afectan a la apariencia que tendrá el título propiamente dicho: Estilo (**headstyle**) y color (**headcolor**). Con **headcommand** podemos indicar un comando al que se le pasará como argumento el texto del título. Por ejemplo: **headcommand=\WORD** se asegurará de que el texto del título se escriba totalmente en mayúsculas.
- **style, color:** Controlan la apariencia del texto descriptivo del título.

12.4.2 Enumeraciones

Las enumeraciones son elementos de texto numerados y estructurados en varios niveles. Cada elemento se inicia con un título que consiste, por defecto, en el nombre de la estructura y su número, aunque podemos cambiar el título con la opción **text**. Se parecen bastante a las descripciones, aunque se diferencian en que:

- Todos los elementos de una enumeración comparten el mismo título.
- Por lo tanto se diferencian unos de otros por su numeración.

Esta estructura puede ser muy útil, por ejemplo, para ir escribiendo fórmulas, problemas o ejercicios en un libro de texto, garantizando que estos se numerarán correctamente y se formatearán de manera consistente.

Creamos una enumeración con

```
\defineenumeration[Nombre] [Configuración]
```

donde *Nombre* es el nombre de nuestra nueva construcción, y *Configuración* controla el aspecto que tendrá la misma.

Así, en el siguiente ejemplo:

```
\defineenumeration
[Ejercicio]
[alternative=top, before=\blank, after=\blank, between=\blank]
```

Habremos creado una nueva estructura llamada *Ejercicio* y, al igual que ocurre en las enumeraciones, tendremos disponibles los siguientes nuevos comandos:

```
\Ejercicio
\startEjercicio
```

El comando se usará para *ejercicios* de un sólo párrafo, y el entorno para *ejercicios* de varios párrafos. Pero como las enumeraciones pueden llegar a tener hasta cuatro niveles de profundidad, se crearán también los siguientes comandos y entornos:

```
\subEjercicio
\startsubEjercicio
\stopsubEjercicio
\subsubEjercicio
\startsubsubEjercicio
\stopsubsubEjercicio
\subsubsubEjercicio
\startsubsubsubEjercicio
\stopsubsubsubEjercicio
```

Y, para controlar la numeración, los siguientes comandos adicionales:

- `\setNombreEnumeración`: Establece el valor actual de la numeración.
- `\resetNombreEnumeración`: Pone a cero el contador de la enumeración.
- `\nextNombreEnumeración`: Incrementa en uno el contador de la enumeración.

La apariencia de las enumeraciones se puede determinar en el momento de su creación o más tarde con `\setupenumeration` cuyo formato es:

```
\setupenumeration[Nombre] [Configuración].
```

Para cada enumeración podemos configurar por separado cada uno de sus niveles. Así, por ejemplo `\setupenumeration [subEjercicio] [Configuración]` afectará al segundo nivel de la enumeración llamada «Ejercicio».

Las opciones y valores configurables con `\setupenumeration` son similares a las de `\setupdescription`.

12.5 Líneas y marcos

Dice el manual de referencia de ConT_EXt que T_EX tiene una enorme capacidad de manejo de texto, pero que es muy débil en el manejo de la información gráfica. Me permito discrepar: es cierto que para el manejo de líneas y marcos las posibilidades de ConT_EXt (en realidad T_EX) no son tan abrumadoras como cuando se trata de componer texto. Pero de ahí a decir que el sistema resulta débil en este aspecto creo que hay un gran trecho. No conozco ninguna función con líneas y marcos que puedan hacer otros sistemas de composición de documentos y ConT_EXt no sea capaz de generar. Y si combinamos ConT_EXt con MetaPost, o con T_iKZ (para cuyo

uso en ConT_EXt existe un módulo de expansión), entonces ya las posibilidades solo están limitadas por nuestra imaginación.

En los próximos apartados, no obstante, me limitaré a explicar cómo generar líneas simples, horizontales y verticales y marcos en torno a palabras, frases o párrafos.

12.5.1 Líneas simples

La forma más sencilla de dibujar una línea horizontal es con el comando `\hairline`, que genera una línea horizontal que ocupa toda la anchura de una línea de texto normal.

En el renglón donde esté la línea generada por `\hairline` no puede haber texto de ningún tipo. Para generar una línea capaz de convivir con el texto de su renglón, necesitamos el comando `\thinrule`. Este segundo comando usará todo el ancho disponible de la línea. Por lo tanto, en un párrafo aislado, tendrá el mismo efecto que `\hairline`, pero en el caso contrario, `\thinrule` producirá la misma expansión horizontal que `\hfill` (véase la [sección 10.3.3](#)), pero llenando el espacio horizontal, no con espacio en blanco (como hace `\hfill`), sino con una línea.

```
A la izquierda\thinrule\\
\thinrule A la derecha\\
A ambos\thinrule lados\\
\thinrule centrado\thinrule
```

A la izquierda	_____	A la derecha

A ambos	_____	lados

	_____	centrado

Con el comando `\thinrules` podemos generar varias líneas. Por ejemplo `\thinrules[n=2]` generará dos líneas consecutivas, cada una de ellas de la anchura del renglón. Las líneas generadas con `\thinrules` pueden también ser configuradas, bien en una llamada concreta al comando, indicando la configuración como argumento del mismo, bien con carácter general mediante `\setupthinrules`. La configuración incluye el grosor de la línea (`rulethickness`), el color de la línea (`color`), el color de fondo (`background`), el interlineado (`interlinespace`), etc.

Dejo varias opciones sin explicar. El lector puede consultar cuáles son en `setup-en.pdf` (véase la [sección 3.6](#)). Algunas opciones se diferencian de otras en cuestiones muy de matiz y creo que es más rápido que el lector intente ver la diferencia, que esforzarme yo en transmitirlo con palabras. Por ejemplo: el grosor de la línea acabo de decir que depende de la opción `rulethickness`. Pero también le afectan las opciones `height` y `depth`.

Líneas de dimensiones más reducidas pueden generarse con los comandos `\hl` y `\vl`. El primero genera una línea horizontal y el segundo una línea vertical. Ambos reciben como parámetro un número que permite calcular la longitud de la línea. En `\hl` el número mide la longitud en *ems* (no hay que indicar en el comando la unidad de medida) y en `\vl` el argumento se refiere a la altura actual de la línea.

Así `\hl[3]` genera una línea horizontal de 3 ems y `\vl[3]` genera una línea vertical de la altura correspondiente a tres líneas. Recuérdese por otra parte que el indicador de medida de la línea debe introducirse entre corchetes, no entre llaves. En ambos comandos el argumento es opcional. De no introducirse se asume un valor de 1.

Otro comando para crear líneas horizontales de longitud precisa, y que admite una mayor configuración es `\fillinline`, en el que podemos indicar (o predefinir con `\setupfillinlines`) la anchura (opción `width`) además de algunas otras características.

Una peculiaridad de este comando es que el texto que se escriba a su derecha será ubicado a la izquierda de la línea; separando dicho texto de la línea por el espacio en blanco necesario para ocupar todo el renglón. Por ejemplo:

```
\fillinline[width=6cm] Nombre
```

generará

Nombre



Sospecho que este extraño funcionamiento se debe a que esta macro fue diseñada para escribir formularios en los que tras el texto hay una línea horizontal sobre el que debe escribirse algo. De hecho el propio nombre del comando `fillinline` significa eso: llenar en la línea.

Además de la anchura de la línea, podemos configurar el margen (`margin`), la distancia (`distance`), el grosor (`rulethickness`) y el color (`color`).

Casi idéntico a `\fillinline` es `\fillinrules`, si bien este comando permite insertar más de una línea (opción «`n`»).

```
\fillinrules[Configuración] {Texto} {Texto}
```

donde los tres argumentos son opcionales.

12.5.2 Líneas vinculadas a texto

Aunque algunos de los comandos que acabamos de ver pueden generar líneas que coexistan con el texto, en un mismo renglón, en realidad esos comandos se centran en el trazado de la línea. Para escribir líneas vinculadas a cierto texto ConTeXt dispone de comandos

- Que generan un texto entre líneas.
- Que generan líneas bajo el texto (subrayado), sobre el texto (sobrerayado) o atravesándolo (tachado).

Para generar un texto entre líneas el comando habitual es `\textrule`. Este comando traza una línea que atraviesa toda la anchura de la página y en el lado izquierdo (pero no en el extremo) escribe el texto que recibe como parámetro. Por ejemplo:

```
\textrule{Texto de ejemplo}
```

— Texto de ejemplo —



Se supone que `\textrule` admite un primer argumento opcional con tres valores posibles: `top`, `middle` y `bottom`. Pero, tras algunas pruebas, no he conseguido averiguar que efecto producen tales opciones.

Similar a `\textrule` es el entorno `\starttextrule` que, además de insertar la línea con texto al principio del entorno, inserta una línea horizontal al final. El formato de este comando es:

```
\starttextrule[Configuración]{Texto en la línea} ... \stoptextrule
```

Tanto `\textrule` como `\starttextrule` se pueden configurar con `\setuptextrule`.

Para trazar líneas bajo un texto, sobre él, o tachándolo, se usan los siguientes comandos:

```
\underbar{Texto}
\underbars{Texto}
\overbar{Texto}
\overbars{Texto}
\overstrike{Texto}
\overstrikes{Texto}
```

Como se ve para cada tipo de línea (bajo, sobre, o a través del texto) hay dos comandos. La versión del comando en singular traza una sola línea bajo, sobre o a través de todo el texto que se reciba como argumento, mientras que la versión en plural del comando sólo traza la línea sobre las palabras, pero no sobre los espacios en blanco.

Estos comandos no son compatibles entre sí, es decir: a un mismo texto no se le pueden aplicar dos de ellos. Si se intenta siempre prevalecerá el último. Por otra parte `\underbar` puede anidarse, subrayando lo ya subrayado.

Señala el manual de referencia que `\underbar` desactiva la partición silábica de palabras en el texto que constituye su argumento. No me queda claro si esa afirmación se refiere sólo a `\underbar` o a los seis comandos que estamos examinando.

12.5.3 Palabras o textos enmarcados

Para rodear un texto con un marco o cuadrícula se usa:

- Los comandos `\framed` o `\inframed` si el texto es relativamente corto y no ocupa más de una línea.
- El entorno `\startframedtext` para textos más largos.

La diferencia entre `\framed` e `\inframed` está en la el punto a partir del cual se traza el marco. En `\frame` el marco se traza hacia arriba a partir de una línea ideal, llamada línea base en la que se apoyan las letras, pero que ciertas letras traspasan hacia abajo. En `\inframed` el marco se traza, también hacia arriba, desde el punto más bajo posible de la línea. Por ejemplo

Aquí hay `\framed{dos}` buenos
`\inframed{marcos}`.

Aquí hay dos buenos marcos.

Tanto `\framed` como `\inframed` pueden personalizarse con `\setupframed`, y `\startframedtext` se personaliza con `\setupframedtext`. Las opciones de personalización de ambos comandos son bastante parecidas. Permiten indicar las medidas del marco (`height`, `width`, `depth`), la forma de las esquinas (`framecorner`), que puede ser `rectangular` o `redondeada` (`round`), el color del marco (`framecolor`), el grosor de la línea (`frametickness`), la alineación de su contenido (`align`), color del texto (`foregroundcolor`), color del fondo (`background` y `backgroundcolor`), etc.

Para `\startframedtext` también existe una propiedad aparentemente extraña: `frame=off` que hace que el marco no se dibuje (aunque sigue estando, pero invisible). Esta propiedad existe porque como el marco de un párrafo es indivisible, es corriente que para asegurarnos de que dentro de un párrafo no se insertará ningún salto de línea, se encierre todo el párrafo en un entorno `framedtext` con la opción de dibujo del marco desactivada.

También podemos crear una versión personalizada de estos comandos con `\definframed` y `\defineframedtext`.

12.6 Otros entornos y construcciones de interés

Quedan numerosos entornos en ConT_EXt que no he ni siquiera mencionado, o sólo lo he hecho muy de pasada. A título de ejemplo:

- **buffer** Los *buffers* son fragmentos de texto almacenados en la memoria para su posterior reutilización. Un *buffer* se define en algún punto del documento con `\startbuffer[NombreBuffer] ... \stopbuffer` y puede ser recuperado,

tantas veces como se desee, en algún otro punto del documento con `\getbuffer[NombreBuffer]`.

- **chemical** Este entorno permite ubicar dentro de él fórmulas químicas. Si \TeX destaca, entre otras muchas cosas, por su capacidad para componer bien textos con fórmulas matemáticas, \ConTeXt desde el principio quiso ampliar tal capacidad a las fórmulas químicas, y dispone de este entorno dentro del cual se habilitan comandos y estructuras pensadas para la escritura de la química.
- **combination** Este entorno permite combinar en una misma página varios elementos flotantes. Es particularmente útil para alinear imágenes externas distintas que quedan así vinculadas en nuestro documento.
- **formula** Se trata de un entorno pensado para introducir dentro de él una fórmula matemática.
- **hiding** El texto almacenado en este entorno no será compilado ni aparecerá, por lo tanto, en el documento final. Esto es útil para inhabilitar temporalmente la compilación de ciertos fragmentos del fichero fuente. Eso mismo se consigue marcando una o varias líneas como comentario. Pero cuando el fragmento que queremos inhabilitar es relativamente largo, más eficaz que marcar como comentario decenas o cientos de líneas del fichero fuente es insertar el comando `\starthiding` al principio del fragmento, y `\stophiding` al final.
- **legend** En un contexto matemático, \TeX aplica reglas diferentes, de tal modo que no se puede escribir texto normal. No obstante en ocasiones una fórmula va acompañada de una descripción de los elementos usados en ella. Para tal fin existe el entorno `\startlegend` que permite ubicar texto normal en un contexto matemático.
- **linecorrection** Habitualmente \ConTeXt gestiona correctamente el espacio vertical entre líneas, pero ocasionalmente una línea puede contener algo que hace que no se vea correctamente. Eso ocurre principalmente con líneas que tienen fragmentos enmarcados con `\framed`. En tales casos este entorno ajusta el interlineado para que el párrafo se vea de modo correcto.
- **mode** Este entorno está pensado para incluir en el fichero fuente fragmentos que sólo se compilarán si está activo el modo adecuado. El uso de *modos* no es objeto de esta introducción, pero es una herramienta muy interesante si se quiere, a partir de un sólo fichero fuente, poder generar varias versiones con formatos diferentes. Un entorno complementario a este es `\startnotmode`.
- **opposite** Este entorno sirve para componer textos en los que el contenido de las páginas página izquierda derecha esté relacionado.

- **quotation** Un entorno muy parecido a **narrower**, pensado para insertar citas literales moderadamente largas. El entorno se ocupa de entrecomillar el texto de su interior, y de aumentar los márgenes para que destaque visualmente en la página el párrafo con la cita. Pero hay que tener en cuenta que el entrecomillado se realiza de acuerdo con el estilo habitual de las comillas en inglés, sin que, aparentemente, esto se pueda cambiar, lo que limita la utilidad de este entorno.
- **standardmakeup** Este entorno está pensado para generar páginas con el título del documento, cosa relativamente habitual en los documentos académicos de cierta extensión como tesis doctorales, trabajos de fin de máster, etc.

Para aprender sobre cualquiera de estos entornos (u otros que no he mencionado), sugiero los siguientes pasos:

1. Buscar la información sobre el entorno en el manual de referencia de ConT_EXt. Este manual no menciona a todos los entornos; pero sí se dice algo de todos los de la lista anterior.
2. Escribir un documento de prueba en el que se utilice el entorno.
3. Buscar en el listado oficial de comandos de ConT_EXt (véase [sección 3.6](#)) las opciones de configuración del entorno en cuestión, e ir las probando para ver exactamente cómo se manejan.

Capítulo 13

Imágenes, tablas y otros objetos flotantes

Sumario: 13.1 Qué son los objetos flotantes y para qué sirven; 13.2 Imágenes externas; 13.2.1 Inserción directa de imágenes; 13.2.2 Inserción de una imagen con `\placefigure`; 13.2.3 Inserción de imágenes integradas en un bloque de texto; 13.2.4 Configuración y transformación de las imágenes que se insertan; A Opciones del comando de inserción que provocan cierta transformación en la imagen; B Comandos específicos para transformar una imagen; 13.3 Tablas; 13.3.1 Ideas generales en torno a las tablas y a su ubicación en el documento; 13.3.2 Tablas simples con el entorno `tabulate`; 13.4 Aspectos comunes a imágenes, tablas y otros objetos flotantes; 13.4.1 Opciones de inserción de los objetos flotantes; 13.4.2 Configuración de los títulos de los objetos flotantes; 13.4.3 Inserción combinada de dos o más objetos; 13.4.4 Configuración general de los objetos flotantes; 13.5 Definición de objetos flotantes adicionales;

Este capítulo trata, principalmente, sobre objetos flotantes. Pero al hilo de esta noción, se aprovecha para explicar dos tipos de objetos que, aunque muchas veces se configuran como objetos flotantes, no necesariamente lo son: las imágenes externas y las tablas. Quien consulte el índice del capítulo pensará que está muy desordenado: se empieza hablando de objetos flotantes, luego se habla de imágenes y de tablas, para terminar hablando, de nuevo, de objetos flotantes. Las razones de este orden son *pedagógicas*: imágenes y tablas se pueden explicar sin insistir demasiado en que suelen ser objetos flotantes; y sin embargo, al entrar en el examen de estos ayuda mucho el descubrir que ¡Sorpresa! ya conocemos dos objetos flotantes.

13.1 Qué son los objetos flotantes y para qué sirven

Si un documento contuviera exclusivamente texto *normal* paginarlo sería relativamente fácil: Conocida la altura máxima de la zona de texto de la página basta con medir la altura de los distintos párrafos para saber en qué puntos hay que insertar los saltos de página. El problema está en que en muchos documentos hay objetos, fragmentos o bloques de texto indivisibles como, por ejemplo, una imagen, una tabla, una fórmula, un párrafo enmarcado, etc. En ocasiones estos objetos pueden

llegar a ocupar un buen trozo de la página lo que, a su vez, plantea el problema de que si hay que insertarlo en un punto concreto del documento, es posible que, al no caber en la página en curso, esta tenga que interrumpirse abruptamente, dejando un amplio espacio en blanco en su parte inferior, para que el objeto en cuestión, y el texto que le sigue, se desplacen a la página siguiente. Las reglas de la buena composición tipográfica, sin embargo, indican que, salvo en la última página de un capítulo, la cantidad de texto en las páginas debe ser equivalente.

Es pues aconsejable evitar la aparición de grandes espacios verticales en blanco; y los objetos *flotantes* constituyen el principal mecanismo para conseguirlo. Se llama «objeto flotante» a aquel que no se tiene por qué ubicar en un punto exacto del documento, sino que puede *fluctuar* o *flotar* en torno a él. La idea es permitir que ConT_EXt decida el mejor lugar, desde el punto de vista de la paginación, para ubicar tales objetos, autorizando incluso a que se trasladen a otra página; pero procurando siempre que no se alejen demasiado del punto del fichero fuente en el que se ordenó su inclusión.

No hay, por lo tanto, objetos que siempre y necesariamente sean flotantes. Pero si hay objetos a los que es recomendable permitirles serlo. La decisión, en todo caso, corresponde al autor o a quien se encargue de la composición tipográfica, si es una persona distinta.

Que la ubicación exacta de un objeto indivisible pueda cambiar, sin duda facilita mucho la tarea de la composición de páginas equilibradas; pero, a cambio, tiene el inconveniente de que, como en el momento de escribir el original no sabemos exactamente en qué lugar aparecerá el objeto en cuestión, se hace difícil referirse a él. Y así, si yo, por ejemplo, justo después de incluir en mi documento la orden que inserta una imagen, en el párrafo siguiente pretendo describirla y escribo algo así como: «Como se puede ver en la figura anterior», pero al *flotar* la figura, esta se ubica *detrás* de dicho texto, se habrá producido una inconsistencia: el lector que busque una imagen *antes* del texto que se refiera a ella, no la encontrará, porque, al hacerla flotar, de hecho la imagen se ha insertado detrás.

Esto, a su vez, se arregla *numerando* los objetos flotantes (previa distribución de los mismos en categorías), de tal forma que en lugar de referirnos a una imagen como «la imagen anterior» o «la próxima imagen», nos referiremos a ella como «la imagen 1.3», pudiendo usar el mecanismo de las remisiones internas de ConT_EXt para asegurarnos de que el número de imagen se mantenga siempre actualizado (véase la [sección 9.2](#)). La numeración de este tipo de objetos, por otra parte, facilita el que se pueda crear con bastante facilidad un índice de los mismos (índice de tablas, índice de gráficos, índice de imágenes, índice de ecuaciones, etc., véase, sobre cómo generarlos, la [sección 8.2](#)).

El mecanismo de tratamiento de los objetos flotantes en ConT_EXt es bastante sofisticado y alcanza unos altos niveles de abstracción que, tal vez, no lo hagan

adecuado para los principiantes. Por ello en este capítulo, voy a empezar por explicarlo a partir de dos casos concretos: las imágenes y las tablas. Después intentaré generalizar algo para que se comprenda cómo podemos extender dicho mecanismo a otro tipo de objetos.

13.2 Imágenes externas

Como el lector a estas alturas sabe (pues se explicó en la [sección 1.5](#)), ConT_EXt está perfectamente integrado con MetaPost y puede generar imágenes y gráficos *programados* de modo similar a como se programan las transformaciones del texto. También hay un módulo de expansión de ConT_EXt¹ que le permite trabajar con TiKZ². Pero de ese tipo de imágenes no se trata en esta introducción (pues ello probablemente obligaría a duplicar su extensión). Me estoy refiriendo aquí al uso de imágenes externas, que residen en un fichero de nuestro disco duro o que sean descargadas directamente de Internet por ConT_EXt.

13.2.1 Inserción directa de imágenes

Para insertar una imagen directamente (no como objeto flotante) se usa el comando `\externalfigure` cuya sintaxis es

```
\externalfigure[Nombre] [Configuración]
```

donde

- *Nombre* puede ser, bien el nombre del fichero que contiene la imagen, bien la dirección web de una imagen ubicada en Internet, bien un nombre simbólico que previamente hayamos asociado a una imagen mediante el comando `\useexternalfigure` cuyo formato es similar al de `\externalfigure` aunque recibe un primer argumento con el nombre simbólico que se asociará a la imagen en cuestión.
- *Configuración* es un argumento opcional que nos permite aplicar ciertas transformaciones a la imagen antes de su inserción en nuestro documento. Examinaremos este argumento más detenidamente en el [epígrafe 13.2.4](#).

¹ Los módulos de expansión de ConT_EXt le dotan de utilidades adicionales; pero no están incluidos en esta introducción.

² Se trata de un lenguaje de programación de gráficos diseñado para trabajar en sistemas basados en T_EX. Su nombre está constituido por las siglas de la frase alemana «TiKZ ist keinen Zeichenprogramm» que traducido significa: «TiKZ no es un programa de dibujo», lo que es un *acrónimo recursivo* (igual que GNU); una broma propia de programadores. Dejando de lado MetaPost (que no sé manejar), creo que TiKZ es un gran sistema para el diseño de gráficos programados.


Los formatos de imagen admitidos son pdf, mps, jpg, png, jp2, jbig, jbig2, jb2, svg, eps, gif o tif. De ellos ConT_EXt realmente sólo soporta los ocho primeros, mientras que los restantes (svg, eps, gif o tif) antes de abrirlos necesita convertirlos mediante una herramienta externa que cambia según cual sea el formato y que, por lo tanto, debe estar instalada en el sistema para que ConT_EXt pueda manipular ese tipo de ficheros.

Entre los formatos soportados por `\externalfigure` también los hay de vídeo. En particular: QuickTime (extensión .mov), Flash Video (extensión .flv) y MPeg 4 (extensión .mp4). Pero la mayor parte de los reproductores de PDF no saben gestionar ficheros PDFs con vídeo incrustado en ellos. No puedo al respecto decir mucho, pues no he hecho pruebas de ningún tipo.

No es preciso, por otra parte, indicar la extensión del fichero: ConT_EXt buscará un fichero con el nombre especificado y alguna de las extensiones propias de los formatos de imagen conocidos. Si existen varios candidatos, en primer lugar se usa el formato PDF si lo hubiera, y en su defecto el formato MPS (gráficos generados por MetaPost). A falta de estos dos, se sigue el siguiente orden: jpeg, png, jpeg 2000, jbig y jbig2.

Si el formato real de la imagen no se corresponde con la extensión del fichero que la almacena, ConT_EXt no podrá abrirla, salvo que se le indique, mediante la opción `method` cuál es el formato real de la imagen.

Si la imagen no constituye un párrafo aislado, sino que se integra en un párrafo de texto, y su altura es superior a la del interlineado, éste se ajustará para evitar que la imagen se superponga sobre las líneas anteriores, como en el ejemplo que

acompaña a esta línea .

Por defecto ConT_EXt busca las imágenes en el directorio de trabajo, en su directorio padre y en el directorio padre de éste. Podemos indicar la ubicación de un directorio en el que se encuentren las imágenes con las que trabajaremos mediante la opción `directory` del comando `\setupexternalfigures`, con lo que se sumaría dicho directorio a la ruta de búsqueda. Si queremos que la búsqueda se realice sólo en el directorio de imágenes, tenemos que establecer también la opción `location`. Así, por ejemplo, para que nuestro documento busque todas las imágenes que necesite en el directorio «img» debemos escribir:

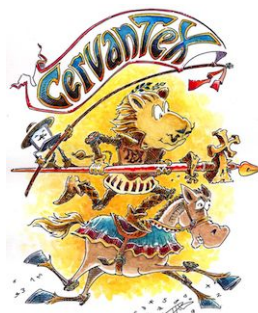
```
\setupexternalfigures  
[directory=img, location=global]
```

En la opción `directory` de `\setupexternalfigures` podemos introducir más de un directorio, separándolos por una coma; pero, en tal caso, es preciso encerrar entre corchetes todos los directorios. Por ejemplo «`directory={img, ~/imágenes}`».

En `directory` se usa siempre el carácter «/» como separador de directorios; incluso en Microsoft Windows cuyo sistema operativo usa el carácter «\» como separador de directorios.

`\externalfigure` también es capaz de usar imágenes alojadas en Internet. Y así, por ejemplo el siguiente fragmento insertará en el documento, directamente de Internet el logotipo de CervanTeX, el grupo de usuarios hispanoparlantes de T_EX¹:

```
\externalfigure  
[http://www.cervantex.es/files/  
cervantex/cervanTeXcolor-small.jpg]
```



Cuando se compila por primera vez un documento que contiene un archivo remoto, éste se descarga del servidor y se almacena en el directorio de caché de LuaTeX. Este archivo en caché se utiliza durante las compilaciones subsiguientes. Normalmente, la imagen remota se descarga de nuevo si la imagen en la caché es más antigua de 1 día. Para cambiar este umbral consulte la [wiki de ConT_EXt](#).

Si ConT_EXt no encuentra la imagen que debía insertar, no se genera ningún error, sino que en lugar de la imagen se insertará un bloque de texto con información sobre la imagen que debería ir allí. El tamaño de ese bloque será el de la imagen (si ConT_EXt lo conoce) o, en caso contrario, un tamaño estándar. Hay un ejemplo de esto en la [sección 13.4.3](#).

13.2.2 Inserción de una imagen con `\placefigure`

Las imágenes se pueden insertar directamente. Pero es preferible insertarlas mediante `\placefigure`. Este comando hace que ConT_EXt:

- Sepa que se está insertando una imagen que debe incorporarse a la lista de imágenes del documento, con la que podrá generarse, si lo deseamos, un índice de imágenes.
- Asigne un número a la imagen, facilitando así las remisiones internas a ella.
- Añada a la imagen un título, creando un bloque de texto inseparable entre la imagen y su título.

¹ Las direcciones de Internet son muy largas, y el espacio disponible para mostrar el ejemplo a doble columna es escaso. Por ello, para que cupiera bien la orden en la columna izquierda he insertado un salto de línea dentro de la dirección web. Si alguien desea copiar y pegar el ejemplo, no funcionará si no se suprime dicho salto.

- Establezca automáticamente el espacio en blanco (horizontal y vertical) necesario para que la imagen se vea correctamente.
- Ubique la imagen en el lugar que se le indique, haciendo fluir el texto en torno a ella si fuera preciso.
- Convierta a la imagen en objeto flotante si, teniendo en cuenta su tamaño y las especificaciones relativas a su ubicación, resultara posible¹.

La sintaxis de este comando es la siguiente:

```
\placefigure[Opciones][Etiqueta] {Título} {Imagen}
```

Los distintos argumentos significan lo siguiente:

- *Opciones* son un conjunto de indicaciones que en general se refieren a dónde ubicar la imagen. Dado que estas opciones son las mismas en este y en otros comandos, las explicaré de modo conjunto más adelante (en la [sección 13.4.1](#)). De momento, para los ejemplos, usaré la opción `here` que es una indicación a ConTeXt para que, en la medida de lo posible, ubique la imagen exactamente en el punto del documento en el que se encuentra el comando que la inserta.
- *Etiqueta* es una cadena de texto para referirse internamente a este objeto a efectos de poder realizar remisiones internas a él (véase la [sección 9.2](#)).
- *Título* es el texto del título que se añadirá a la imagen.
- *Imagen* es el comando que insertará la imagen.

Por ejemplo

```
\placefigure
[here]
[fig:texknuth]
{Logotipo de \TeX\ y fotografía de {\sc Knuth}}
{\externalfigure[https://i.ytimg.com/vi/8c5Rrfabr9w/maxresdefault.jpg]}
```



Figura 13.1 Logotipo de
TeX y fotografía de KNUTH

¹ Esto último es mi conclusión, a la vista de que entre las opciones de ubicación hay algunas (como `force` o `split`) que son contradictorias con la propia idea de objeto flotante.

Como se puede ver en el ejemplo, al insertar así la imagen (lo que, por cierto, se ha hecho directamente de una imagen alojada en Internet), se producen algunos cambios respecto de lo que ocurre cuando se usa directamente el comando `\externalfigure`. Así, se ha añadido espacio vertical, se ha centrado la imagen y se le ha añadido un título. Esos son cambios *externos*, apreciables a simple vista. Desde el punto de vista interno el comando también ha producido estos otros efectos no menos importantes:

- En primer lugar, la imagen se ha insertado en la «lista de imágenes» que ConTeXt mantiene internamente para los objetos que se insertan en el documento mediante `\placefigure`. Esto, a su vez, implica que la imagen aparecerá en el índice de imágenes que se puede generar con `\placelist[figure]` (véase la [sección 8.2](#)), aunque existen dos comandos específicos para generar el índice de imágenes que es `\placelistoffigures` o `\completelistoffigures`.
- En segundo lugar, la imagen ha quedado vinculada a la etiqueta que se le añadió como segundo argumento del comando `\placefigure` lo que significa que a partir de ahora podemos hacer remisiones internas a ella usando dicha etiqueta (véase la [sección 9.2](#)).
- Por último la imagen ha pasado a ser un objeto flotante, lo que significa que si fuera necesario por necesidades de composición de la página, ConTeXt desplazaría su ubicación.

En realidad `\placefigure`, a pesar de su nombre, no sirve sólo para insertar imágenes, podemos insertar con él cualquier material; incluso texto. Pero el texto o material que se inserte en el documento mediante `\placefigure`, será tratado *como si fuera una imagen*, aunque no lo sea; se añadirá a la lista de imágenes gestionada internamente por ConTeXt, y se le podrán aplicar transformaciones similares a las propias de las imágenes tales como escalar o rotar, enmarcar, etc. Así el siguiente ejemplo:



Figura 13.2 Uso de `\placefigure` para transformaciones de texto

que se consigue de la siguiente manera:

```
\placefigure
[here, force]
[fig:textoprueba]
{Uso de \backslash placefigure para transformaciones de texto}
{\rotate[rotation=180]{\framed{\tfd Texto de prueba}}}
```


13.2.3 Inserción de imágenes integradas en un bloque de texto

Salvo para imágenes muy pequeñas, que se pueden integrar en una línea sin desbaratar demasiado el interlineado del párrafo, normalmente las imágenes se insertan en un párrafo que sólo las contiene a ellas. Si la imagen se inserta con `\placefigure` y su tamaño lo permite, dependiendo de la indicación que hagamos respecto de dónde ubicarse (véase la [sección 13.4.1](#)), ConTeXt permitirá que el texto de los párrafos anterior y posterior fluya en torno a la imagen. No obstante, si queremos asegurarnos de que cierta imagen no se separará de cierto texto, podemos usar el entorno `figuretext` cuyo formato es el siguiente:

```
\startfiguretext
  [Opciones]
  [Etiqueta]
  {Título}
  {Imagen}

  ... Texto

\stopfiguretext
```

Los argumentos del entorno son exactamente los mismos que los de `\placefigure` y tienen el mismo significado. Pero aquí las opciones ya no son opciones de ubicación de un objeto flotante, sino indicaciones relativas a la integración de la imagen dentro del párrafo; y así, por ejemplo, «`left`» aquí significa que la imagen se situará en el lado izquierdo y el texto fluirá por el derecho, mientras que «`left, bottom`» significará que la imagen se debe situar en el lado inferior izquierdo del texto asociado a ella.

El texto escrito dentro del entorno es el que fluirá alrededor de la imagen.

13.2.4 Configuración y transformación de las imágenes que se insertan

A. Opciones del comando de inserción que provocan cierta transformación en la imagen

El último argumento del comando `\externalfigure` permite realizar ciertos ajustes en la imagen que se inserta. Estos ajustes podemos realizarlos:

- Con carácter general para todas las imágenes que se insertarán en el documento; o para todas las imágenes que se inserten a partir de cierto punto. En tal caso el ajuste lo haremos mediante el comando `\setuexternalfigures`.
- Para una imagen concreta que se pretende insertar varias veces en el documento. En tal caso el ajuste se hace en el último argumento del comando `\useexternalfigure` que asocia una imagen externa con un nombre simbólico.
- En el momento exacto en el que insertemos una concreta imagen. En este caso el ajuste se hace en el propio comando `\externalfigure`.

Los cambios en la imagen que pueden lograrse por esta vía son los siguientes:

Cambiar el tamaño de la imagen. Ello lo podemos lograr:

- *Asignándole una anchura o altura concretas*, cosa que se consigue, respectivamente, mediante las opciones `width` y `height`; si sólo se ajusta uno de los dos valores, el otro se adapta automáticamente para mantener la proporción.

Podemos asignar una altura o anchura exacta, o indicarla como un porcentaje de la altura de la página o anchura de la línea. Por ejemplo:

```
width=.4\textwidth
```

se asegurará de que la imagen tenga una anchura igual al 40% de la anchura de la línea.

- *Escalando la imagen*: La opción `xscale` escalará la imagen horizontalmente; `yscale` lo hará verticalmente y `scale` la escalará horizontal y verticalmente. Estas tres opciones esperan un número representativo del factor de escalado multiplicado por 1000. Es decir: `scale=1000` dejará la imagen en su tamaño original, mientras que `scale=500` la reducirá a la mitad, y `scale=2000` duplicará su tamaño.

Un escalado condicional, que se aplica sólo si la imagen supera ciertas dimensiones, se obtiene con las opciones `maxwidth` y `maxheight` que reciben una dimensión. Por ejemplo `maxwidth=.2\textwidth` escalará la imagen sólo si esta resulta tener una anchura superior al 20% de la anchura de la línea.

Rotar la imagen. Para rotar la imagen se usa la opción `orientation` que recibe un número representativo del número de grados de rotación que se aplicará. La rotación se hace en sentido inverso a las agujas del reloj.

La wiki da a entender que las rotaciones que se pueden conseguir con esta opción han de ser múltiplos de 90 (90, 180 o 270) pero que para conseguir una rotación diferente

habría que usar el comando `\rotate`. Yo, sin embargo, no he tenido ningún problema para aplicar a una imagen una rotación de 45 grados sólo con `orientation=45`, sin necesidad de acudir al comando `\rotate`.

Enmarcar la imagen. Podemos rodear también la imagen con un marco mediante la opción `frame=on`, y configurar su color (`framecolor`), la distancia entre el marco y la imagen (`frameoffset`), el grosor de la línea que dibuja el marco (`rulethickness`) o la forma de sus esquinas (`framecorner`) que puede ser redondeada (`round`) o rectangular.

Otros aspectos configurables de las imágenes. Además de los aspectos ya vistos, que implican una transformación en la imagen que se insertará, mediante `\setupexternalfigures` podemos configurar otros aspectos como, por ejemplo, el directorio donde hay que buscar la imagen (opción `directory`), si la imagen debe buscarse sólo en el directorio indicado (`location=global`) o se debe incluir también el directorio de trabajo y sus directorios antecesores (`location=local`), si la imagen debe o no ser interactiva (`interaction`), etc.

B. Comandos específicos para transformar una imagen

Hay tres comandos en ConTeXt que producen cierta transformación en una imagen y que se pueden usar en combinación con `\externalfigure`. Se trata de:

- *Reflejo especular:* que se consigue con el comando `\mirror`.
- *Recorte:* Esto se consigue con el comando `\clip` al que hay que indicar la anchura (`width`), altura (`height`), desplazamiento horizontal (`hoffset`) y desplazamiento vertical (`voffset`) del recorte a aplicar. Por ejemplo:

```
\clip
  [width=2cm, height=1cm, hoffset=3mm, voffset=5mm]
  {\externalfigure[logo.pdf]}
```

- *Rotación.* Un tercer comando capaz de aplicar transformaciones a una imagen es el comando `\rotate`. Puede usarse en conjunción con `\externalfigure` pero normalmente no será necesario dado que este último dispone, como acabamos de ver, de la opción `orientation` que produce ese mismo resultado.

El uso típico de estos comandos es con imágenes, pero en realidad actúan sobre *cajas*. Por ello podemos aplicarlos a cualquier texto simplemente encerrándolo en una caja (cosa que hace automáticamente el comando), lo que producirá efectos curiosos como los siguientes:

```
\mirror{Texto de prueba}\\
\rotate[rotation=20] {Texto de prueba}
```

sdəurɪq ɐb ɔɪxəT

Texto de prueba

13.3 Tablas

13.3.1 Ideas generales en torno a las tablas y a su ubicación en el documento

Las tablas son objetos estructurados que contienen texto, fórmulas o incluso imágenes, ordenadas en una serie de *celdas* que permiten ver gráficamente la relación entre los contenidos de cada celda. Para ello se organiza la información repartiéndola en filas y columnas: Todos los datos (o entradas) de una misma fila guardan cierta relación entre sí, al igual que todos los datos (o entradas) de una misma columna. Una celda es el cruce de una fila con una columna, tal y como se muestra en la [figura 13.3](#).

Las tablas son ideales para mostrar textos o datos que están relacionados unos con otros, pues al quedar cada uno encerrado en su propia celda, aunque cambie su contenido, o el contenido de las restantes celdas, la posición relativa de una de ellas con respecto a las otras no variará.

De todas las tareas que implica componer un texto, la creación de tablas es la única que, en mi opinión, es más sencilla de hacer en un programa gráfico (tipo procesador de textos) que en sistemas basados en textos

marcados como ConT_EXt. Porque es más fácil *dibujar* la tabla (que es lo que se hace en un programa de procesamiento de textos) que *describirla*, que es lo que hacemos cuando trabajamos con ConT_EXt. Cada cambio de fila y cada cambio de columna exige la presencia de un comando, y eso hace que se tarde bastante más



Figura 13.3 Imagen de una tabla simple

tiempo en implementar la tabla del que se tardaría simplemente diciendo cuántas filas y columnas deseamos.

ConT_EXt incorpora tres mecanismos distintos para producir tablas; el entorno `tabulate` que está recomendado para tablas simples y que es el que más directamente aparece inspirado en las tablas de T_EX, las llamadas *tablas naturales*, inspiradas en las tablas de HTML; adecuadas para tablas con necesidades de diseño especiales, en las que, por ejemplo, no todas las filas tengan el mismo número de columnas, y las llamadas *tablas extremas*, claramente basadas en XML y recomendadas para tablas de extensión mediana o larga, que ocupen más de una página. De los tres explicaré sólo el primero. Las tablas naturales están razonablemente bien explicadas en «Una excursión por ConT_EXt Mark IV», y para las *tablas extremas* se incluye un documento sobre ellas en la documentación de «ConT_EXt Standalone».

En las tablas ocurre algo parecido a lo que ocurría con las imágenes: podemos simplemente escribir en un punto del documento los comandos necesarios para generar una tabla, y ésta se insertará en ese punto exacto, o podemos usar el comando `\placetable` para insertar una tabla, con lo que obtendremos algunas ventajas:

- ConT_EXt numerará la tabla y la añadirá a la lista de tablas permitiendo remisiones internas a la tabla (a través de su numeración) e incluyéndola en un eventual índice de tablas.
- Se ganará flexibilidad en la ubicación de la tabla dentro del documento, facilitando así la tarea de la paginación del mismo.

El formato de `\placetable` es similar al ya visto de `\placefigure` (véase [sección 13.2.2](#)):

```
\placetable[Opciones][Etiqueta]{Título}{tabla}
```

Me remito a las secciones [13.4.1](#) y [13.4.2](#) respecto a las opciones relativas a la ubicación de la tabla y a la configuración del título. Entre las opciones hay una, no obstante, que parece pensada exclusivamente para las tablas. Se trata de la opción «`split`» que, cuando se establece, autoriza a ConT_EXt a extender la tabla a lo largo de dos o más páginas, caso este en el que la tabla ya no podrá ser un objeto flotante.

Podemos establecer con carácter general la configuración de las tablas mediante `\setuptables`. Asimismo, igual que ocurre con las imágenes, es posible generar un índice de tablas mediante `\placelistoftables` o `\completelistoftables`, véase, al respecto la [sección 8.2.2](#).

13.3.2 Tablas simples con el entorno `tabulate`

Las tablas más simples son las que se logran con el entorno `tabulate` cuyo formato es:

```
\starttabulate[Diseño de las columnas de la tabla]
... % Contenido de la tabla
...
...
\stoptabulate
```

Donde el argumento recibido entre corchetes describe (en clave) el número de columnas que tendrá la tabla, e indica (a veces de modo indirecto) la anchura de las mismas. Digo que el argumento describe el diseño *en clave*, porque a primera vista parece algo muy críptico: consiste en una secuencia de caracteres, cada uno de ellos con un significado especial. Lo iré explicando poco a poco y por pasos, pues creo que así es más fácil de comprender.

Este es el típico caso en el que la gran cantidad de aspectos que podemos configurar, hace que su descripción exija mucho texto y que parezca que esto es endiabladamente difícil. En realidad, para la mayor parte de las tablas que se construyen en la práctica, basta con los puntos 1 y 2. Lo demás son posibilidades extra que es útil saber que existen, pero que no es imprescindible conocer para componer una tabla.

1. **Delimitador de columnas:** Se usa el carácter «|» para delimitar las columnas de la tabla. Así, por ejemplo, «[|lT|rB|]» describiría una tabla con dos columnas, una de las cuales habría de tener las características asociadas a los indicadores «l» y «T» (que inmediatamente veremos) y la segunda columna tendría las características asociadas a «r» y «B». Una tabla sencilla de tres columnas alineadas a la izquierda, por ejemplo, se describiría como «[|l|l|l|]».
2. **Determinación de la naturaleza básica de las celdas de una columna:** Lo primero a determinar cuando construimos nuestra tabla es si queremos que el contenido de cada celda deba necesariamente escribirse en una sola línea, o si, por el contrario, deseamos que en nuestra tabla, si el texto de alguna columna es demasiado largo, éste se distribuya en dos o más líneas. En el entorno `tabulate` esa cuestión no se decide celda a celda, sino que se considera una característica de las columnas.
 - a. *Celdas de una sola línea:* Si el contenido de las celdas de una columna, sea cual sea su extensión, ha de escribirse en una sola línea, debemos especificar la alineación del texto en la columna, la cual puede ser izquierda («l», de *left*), derecha («r», de *right*) o centrado («c», de *center*).

En principio estas columnas tendrán la anchura que sea necesaria para que en ella quepa la celda más ancha. Pero podemos limitar la anchura de la columna con el especificador «w(Anchura)». Por ejemplo «[|rw(2cm)|c|c|]» estará describiendo

una tabla con dos columnas, la primera alineada a la derecha y con una anchura exacta de 2 centímetros, y las otras dos centradas y sin limitación de anchura.

Hay que señalar que la limitación de anchura en columnas de una sola línea, puede provocar que el texto de una columna se superponga sobre el de la siguiente columna. Por ello mi consejo es el de que cuando necesitemos columnas de anchura fija, usemos siempre columnas de celdas multilineales.

- b. *Celdas cuyo texto puede ocupar más de una línea si su extensión lo exige*: El especificador «p» genera columnas en las que el texto de cada celda ocupará tantas líneas como se necesiten. Si se indica simplemente «p» la anchura de la columna será toda la disponible. Pero también se puede indicar «p(Anchura)» en cuyo caso la anchura será la especificada expresamente. Así los siguientes ejemplos:

```
\starttabulate[|l|r|p|]
\starttabulate[|l|p(4cm)|]
\starttabulate[|r|p(.6\textwidth)|]
\starttabulate[|p|p|p|]
```

El primer ejemplo creará una tabla con tres columnas, la primera y la segunda, de una sola línea, alineadas, respectivamente a la izquierda y a la derecha, y la tercera, que ocupará la anchura restante, y la altura necesaria para albergar todo su contenido. En el segundo ejemplo, la segunda columna medirá exactamente cuatro centímetros de ancho, sea cual sea su contenido; pero si éste no cabe en ese espacio, ocupará más de una línea. En el tercer ejemplo se calcula la anchura de la segunda columna proporcionalmente a la anchura máxima de la línea, y en el último ejemplo, habrá tres columnas que se repartirán la anchura disponible a partes iguales.

Obsérvese que, en realidad, si una celda es un cuadrilátero, lo que hace el especificador «p» es autorizar una altura variable de las celdas de una columna, dependiendo de la extensión del texto.

3. **Añadir, a la descripción de la columna, indicaciones sobre el estilo y variante de la fuente a utilizar**: Una vez decidida la naturaleza básica de la columna (anchura y altura, automática o fija, de las celdas), todavía podemos añadir, en la descripción del contenido de la columna, un carácter representativo del *formato* en el que ha de escribirse la misma. Estos caracteres pueden ser «B» para negrita, «I» para cursiva, «S» para letra inclinada, «R» para letra de estilo roman o «T» para letra de estilo *typewriter*.
4. **Otros aspectos adicionales que se pueden especificar en la descripción de las columnas de la tabla**

:

- *Columnas con fórmulas matemáticas:* Los especificadores «**m**» y «**M**» activan en una columna el modo matemático sin necesidad de especificarlo en cada una de sus celdas. A cambio las celdas de esta columna no podrán albergar texto normal.

Aunque T_EX, el antecesor de ConT_EXt, nació para escribir sobre todo matemáticas, hasta ahora apenas he dicho nada de la escritura de las matemáticas. En el modo matemático (que no voy a explicar) ConT_EXt altera sus reglas normales e incluso utiliza unas fuentes diferentes. El modo matemático a su vez tiene dos variedades: el que podríamos llamar *en línea* en el que la fórmula se aloja dentro de una línea que contiene texto normal (indicador «**m**»), y el *modo matemático completo* que muestra fórmulas en un entorno en el que no hay texto normal. La diferencia principal de ambos modos, en una tabla, está fundamentalmente en el tamaño con que se escribirá la fórmula y el espacio horizontal y vertical con que se la rodeará.

- *Añadir espacio horizontal en blanco extra en torno al contenido de las celdas de una columna:* Mediante los indicadores «**in**», «**jn**» y «**kn**» podemos añadir espacio en blanco extra a la izquierda del contenido de la columna («**in**»), a la derecha («**jn**») o a ambos lados («**kn**»). En los tres casos «**n**» representa el número por el que se multiplicará el espacio en blanco que normalmente se dejaría de no haberse indicado alguno de estos especificadores (por defecto medio *em*). Así, por ejemplo «**|j2r|**» indicará que estamos ante una columna que se alineará a la derecha, y en la que queremos que tras el contenido de la columna se añada un espacio en blanco de 1 *em* de anchura.
- *Añadir texto antes o después del contenido de cada celda de una columna.* Los especificadores **b{Texto}** y **a{Texto}** hacen que el texto entre llaves se escriba antes («**b**», de *before*) o después («**a**», de *after*) del contenido de la celda.
- *Aplicar un comando de formato a toda la columna.* Los indicadores «**B**», «**I**», «**S**», «**R**» y «**T**» que hemos visto antes, no cubren todas las posibilidades de formato: No hay, por ejemplo, ningún indicador para versalitas, o para letra *sans serif*, o que afecte al tamaño de la fuente. Mediante el indicador «**f\Comando**» podremos especificar un comando de formato que se aplique automáticamente a todas las celdas de la columna. Por ejemplo «**|lf\sc|**» escribirá en versalitas el contenido de la columna.
- *Aplicar un comando cualquiera a todas las celdas de la columna.* Finalmente el indicador «**h\Comando**» aplicará el comando especificado al contenido de la columna.

En la [tabla 13.1](#) se muestran algunos ejemplos de cadenas de especificación de formato de tabla.

Especificador de formato	Significado
l	Genera una columna de anchura automática alineada a la izquierda.
rB	Genera una columna de anchura automática, alineada a la derecha, y escrita en negrita.
cIm	Genera una columna habilitada para contenido matemático. Este se centrará en la columna y se escribirá con cursivas.
j4cb{---}	Esta columna, tendrá el contenido centrado, empezará con un guión largo (—) y añadirá a la derecha 2 <i>ems</i> de espacio en blanco.
l p(.7\textwidth)	Se generan dos columnas: La primera se alinea a la izquierda y tiene anchura automática. La segunda ocupará un 70% de la anchura total de la línea.

Tabla 13.1 Algunos ejemplos de cómo especificar el formato de las columnas en `tabulate`

Una vez diseñada la tabla, hay que introducir su contenido. Para explicar cómo hacerlo empezaré por describir como habría que rellenar una tabla en la que se dibujaran las líneas separadoras de filas y columnas:

- Se empieza siempre dibujando una línea horizontal. Eso, dentro de una tabla, se hace con el comando `\HL` (de *Horizontal Line*).
- A continuación escribimos la primera línea: Al principio de cada celda hay que indicar que empieza una celda nueva y que debe dibujarse una línea vertical. Ello se hace con el comando `\VL` (de *Vertical Line*). Por tanto empezamos con dicho comando, y vamos escribiendo el contenido de cada celda. Cada vez que cambiemos de celda repetimos el comando `\VL`.
- Al terminar una fila, indicamos expresamente que se va a iniciar una fila nueva con el comando `\NR` (de *Next Row*). Tras él repetimos el comando `\HL` para dibujar una nueva línea horizontal.
- Y así, de una en una, vamos escribiendo todas las filas de la tabla. Al terminar añadimos, de propina, un comando `\NR` y otro `\HL` para cerrar la cuadrícula con la línea horizontal inferior.

Si no queremos dibujar la cuadrícula de la tabla, suprimimos los comandos `\HL` y sustituimos los comando `\VL` por `\NC` (de *New Column*).

No es especialmente difícil cuando uno le coge el tranquillo, aunque al ver el código fuente de una tabla resulta difícil hacerse una idea del aspecto que tendrá. En la [tabla 13.2](#) se recogen los comandos que se pueden (y deben) usar dentro de una

tabla. Hay algunos que no he explicado, pero creo que con la descripción que hago de ellos es suficiente.

Comando	Significado
<code>\HL</code>	Inserta una línea horizontal
<code>\NC</code>	Inicia una nueva columna
<code>\NR</code>	Inicia una nueva fila
<code>\VL</code>	Inserta una línea vertical delimitadora de una columna (se usa en lugar de <code>\NC</code>)
<code>\NN</code>	Inicia una columna en modo matemático (se usa en lugar de <code>\NC</code>)
<code>\TB</code>	Añade algo de espacio vertical extra entre dos filas
<code>\NB</code>	Indica que la fila que sigue inicia un bloque indivisible dentro del cual no puede haber un salto de página

Tabla 13.2 Comandos utilizables en el interior de una tabla

Y ahora, como ejemplo voy a transcribir el código con el que se ha escrito la [tabla 13.2](#).

```
\placetable
[here]
[tbl:comandostabla]
{Comandos utilizables en el interior de una tabla}
{\starttabulate[|l|p(.6\textwidth)|]
\HL
\NC {\bf Comando}
\NC {\bf Significado}
\NR
\HL
\NC \tex{HL}
\NC Inserta una línea horizontal
\NR
\NC \tex{NC}
\NC Inicia una nueva columna
\NR
\NC \tex{NR}
\NC Inicia una nueva fila
\NR
\NC \tex{VL}
\NC Inserta una línea vertical delimitadora de una columna
(se usa en lugar de \tex{NC})
\NR
\NC \tex{NN}
\NC Inicia una columna en modo matemático (se usa en lugar
de \tex{NC})
\NR}
```

```

\NC \tex{TB}
\NC Añade algo de espacio vertical extra entra dos filas
\NR
\NC \tex{NB}
\NC Indica que la fila que sigue inicia un bloque indivisible
dentro del cual no puede haber un salto de página
\NR
\HL
\stoptabulate}

```

El lector podrá observar que en general he usado una (o dos) líneas de texto para cada celda. En un fichero fuente real sólo habría usado una línea de texto para cada celda; en el ejemplo he partido las líneas demasiado largas. Usar una sola línea por celda me facilita mucho la escritura de la tabla pues lo que hago es ir escribiendo el contenido de cada celda, sin comandos de separación de filas o columnas. Cuando está todo escrito, selecciono el texto de la tabla y le pido a mi editor de textos que inserte, al principio de cada línea el texto «`\NC` ». Tras ello, cada dos líneas (porque la tabla tiene dos columnas) inserto una línea que añada el comando `\NR`, pues cada dos columnas empieza una fila nueva. Por último, a mano, inserto los comandos `\HL` en los puntos en los que quiero que aparezca una línea horizontal. Tardo casi más en describirlo que en hacerlo.

Véase, por otra parte, como en el interior de la tabla se pueden usar los comandos ordinarios de ConT_EXt. En particular en esta tabla se usa continuamente el comando `\tex` que se explicó en la [sección 10.2.3](#).

13.4 Aspectos comunes a imágenes, tablas y otros objetos flotantes

Ya sabemos que imágenes y tablas no tienen por qué ser objetos flotantes, pero son buenos candidatos a serlo si bien para ello hay que insertarlos en el documento mediante los comandos `\placefigure` o `\placetable`. Además de estos dos comandos, y con la misma estructura, tenemos en ConT_EXt los comandos `\placechemical` (para insertar fórmulas químicas), `\placegraphic` (para insertar gráficos) y `\placeintermezzo` para insertar una estructura a la que ConT_EXt llama *Intermezzo* y que sospecho que se refiere a fragmentos de texto enmarcados. Todos estos comandos son a su vez aplicación concreta de un comando más general que es `\placefloat` cuya sintaxis es la siguiente:

```
\placefloat [Nombre] [Opciones] [Etiqueta] {Título} {Contenido}
```

Obsérvese que `\placefloat` es idéntico a `\placefigure` y `\placetable` salvo por su primer argumento que en `\placefloat` recoge el nombre del objeto flotante. Ello es porque *cada tipo de objeto flotante se puede insertar en el documento con*



dos comandos distintos: `\placefloat[NombreTipo]` o `\placeNombreTipo`. O sea: `\placefloat[figure]` y `\placefigure` son exactamente el mismo comando, al igual que `\placefloat[table]` es el mismo comando que `\placetable`.

Hablaré, por lo tanto, a partir de ahora, de `\placefloat`, pero téngase en cuenta que todo lo que diga será aplicable también a `\placefigure` o `\placetable` que son aplicaciones concretas de dicho comando.

Los argumentos de `\placefloat` son:

- *Nombre*. Se refiere al objeto flotante de que se trate. Puede ser algún objeto flotante predeterminado (`figure`, `table`, `chemical`, `intermezzo`) o un objeto flotante creado por nosotros mismos mediante `\definefloat` (véase la [sección 13.5](#)).
- *Opciones*. Una serie de palabras simbólicas que indican a ConTeXt cómo debe insertar el objeto. La inmensa mayoría de ellas se refiere a *dónde* insertarlo. Las veremos en el próximo epígrafe.
- *Etiqueta*. Una etiqueta para futuras remisiones internas a este objeto.
- *Título*. Es el texto del título que se añadirá al objeto. Sobre su configuración, véase la [sección 13.4.2](#).
- *Contenido*. Este depende, claro está, del tipo de objeto que sea. Para imágenes suele ser un comando `\externalimage`; para tablas, los comandos que crearán la tabla; para *intermezzi*, un fragmento de texto enmarcado; etc.

Los tres primeros argumentos, que se introducen entre corchetes, son opcionales. Los dos últimos (que se introducen entre llaves) obligatorios, aunque pueden estar vacíos, y así, por ejemplo: `\placefloat{}{}` insertará en el documento:



Figura 13.4

Nota: Obsérvese que ConTeXt ha considerado que el objeto que se insertaba era una imagen, pues lo ha numerado como imagen e incluido en la lista de las mismas. Ello me hace suponer que las imágenes son los objetos flotantes predeterminados.



13.4.1 Opciones de inserción de los objetos flotantes

El argumento *Opciones* de `\placefigure`, `\placetable` y `\placefloat` controla diferentes aspectos de la inserción de este tipo de objetos. Principalmente el lugar de la página en donde se insertará el objeto. Aquí se admiten varios valores, de diferente naturaleza:

- Algunos de los lugares de inserción se establecen en relación con los elementos de la página (`top`, `bottom`, `inleft`, `inright`, `inmargin`, `margin`, `leftmargin`, `rightmargin`, `leftedge`, `rightedge`, `innermargin`, `inneredge`, `outeredge`, `inner`, `outer`). Tiene que ser, claro está, un objeto que pueda caber en la zona donde se le pretende ubicar y tiene además que haberse reservado espacio para ese elemento en el diseño de la página. Véase, al respecto, las secciones 5.2 y 5.3.
- Otros de los posibles lugares de inserción están más relacionados con el texto que circunda al objeto, y constituyen una indicación respecto de dónde se debe ubicar este para que el texto fluya a su alrededor. Fundamentalmente los valores `left` y `right`.
- La opción `here` se interpreta como una recomendación para que se mantenga el objeto en el punto del fichero fuente en el que se encuentre. Como tal *recomendación*, no será respetada si las exigencias de la paginación no lo permiten. Esta indicación se refuerza si se le añade la opción `force` que significa exactamente eso: que se fuerce la inserción del objeto en ese punto. Téngase en cuenta que al forzar la inserción en un punto concreto, el objeto dejará de ser flotante.
- Otras de las opciones posibles se refieren a la página en la que se ha de insertar el objeto: «`page`» lo inserta en una nueva página; «`opposit`» lo inserta en la página opuesta a la actual; «`leftpage`» en una página par; «`rightpage`» en una página impar.

Hay algunas opciones que no tienen que ver con la ubicación del objeto. Entre ellas:

- `none`: Esta opción suprime el título.
- `split`: Esta opción autoriza a que el objeto se extienda a lo largo de más de una página. Tiene que ser, claro está, un objeto que por naturaleza sea divisible como, por ejemplo, una tabla. Cuando se usa esta opción y el objeto se divide, ya no puede decirse que es flotante.

13.4.2 Configuración de los títulos de los objetos flotantes

Salvo que se use la opción «none» de `\placefloat`, por defecto los objetos flotantes llevan asociado un título que se compone de tres elementos:

- El nombre del tipo de objeto de que se trate. Este nombre es exactamente el del tipo de objeto; de manera que si, por ejemplo, definimos un nuevo objeto flotante llamado «secuencia» e insertamos una «secuencia» como objeto flotante, el título será “Secuencia 1”. Simplemente se pone en mayúsculas el nombre del objeto.

No obstante lo que se acaba de decir, el nombre en inglés de los objetos flotantes predefinidos, como, por ejemplo, el objeto «figure» o el objeto «table», si el idioma principal del documento no es el inglés se traducirá; y así, por ejemplo, el objeto «figure» en documentos en español se denomina «Figura», mientras que el objeto «table» se denomina «Tabla». Estas denominaciones en español para objetos predefinidos se pueden cambiar mediante `\setuplabeltext` tal y como se explica en la [sección 10.5.3](#).

- Su número. Por defecto los objetos se numeran por capítulos, y así la primera tabla del capítulo 3 será la tabla «3.1».
- Su contenido. Es este contenido el que se introduce como argumento de `\placefloat`.

Mediante `\setupcaptions` o `\setupcaption[Objeto]` podemos cambiar el sistema de numeración y la apariencia del título propiamente dicho. El primer comando afectará a todos los títulos de todos los objetos, y el segundo afectará sólo a los títulos de un tipo concreto de objetos:

- En cuanto al sistema de numeración, está controlado por las opciones `number`, `way`, `prefixsegments` y `numberconversion`:
 - `number` puede adoptar los valores `yes`, `no` o `none` y controla si habrá o no número.
 - `way` indica si la numeración será secuencial para todo el documento (`way=bytext`), o si se reiniciará al empezar cada capítulo (`way=bychapter`) o sección (`way=bysection`). En el caso de que se reinicie conviene coordinar el valor de esta opción con el de `prefixsegments`.
 - `prefixsegments` indica si el número tendrá un *prefijo*, y cuál será este. Así `prefixsegments=chapter` hace que el número de los objetos empiece

siempre por el número de capítulo, mientras que `prefixsegments=section` precederá al número de objeto por el número de sección.

- `numberconversion` controla el tipo de numeración. Los valores de esta opción pueden ser: numeración de tipo arábigo («`numbers`»), letras en minúsculas («`a`», «`characters`»), en mayúsculas («`A`», «`Characters`»), en versalitas «`KA`»), números romanos en mayúsculas («`I`», «`R`», «`Romannumerals`»), en minúsculas («`i`», «`r`», «`romannumerals`» o en versalitas («`KR`»)).
- La apariencia del título propiamente dicho se controla por numerosas opciones. Las enunciaré, pero para la explicación detallada del significado de cada una de ellas, me remito a la [sección 7.4.4](#) donde se explica el control de la apariencia de los comandos de seccionado, ya que las opciones son en gran medida las mismas. Las opciones en cuestión son:
 - Para controlar el formato de todos los elementos del título, `style`, `color`, `command`.
 - Para controlar el formato sólo del nombre del tipo de objeto: `headstyle`, `headcolor`, `headcommand`, `headseparator`.
 - Para controlar el formato sólo de la numeración: `numbercommad`.
 - Para controlar el formato sólo del título propiamente dicho: `textcommand`.
- Se pueden también controlar otros aspectos como la distancia entre los distintos elementos que componen el título, la anchura de éste, su colocación en relación con el objeto, etc. Me remito aquí a la información de la [wiki de ConTeXt](#) respecto de las opciones configurables con este comando.

13.4.3 Inserción combinada de dos o más objetos

Para insertar dos o más objetos distintos en el documento, de tal modo que ConTeXt los mantenga unidos y los trate como un solo objeto, se dispone del entorno `\startcombination` cuya sintaxis es:

```
\startcombination[Ordenación] ... \stopcombination
```

donde *Ordenación* indica cómo se ordenarán los objetos: si todos se han de ordenar horizontalmente, *Ordenación* sólo indica el número de objetos a combinar. Pero si queremos combinar los objetos en dos o más filas, habrá que indicar el número de objeto por fila, seguido del número de filas, y separando ambos números por el carácter *. Por ejemplo:

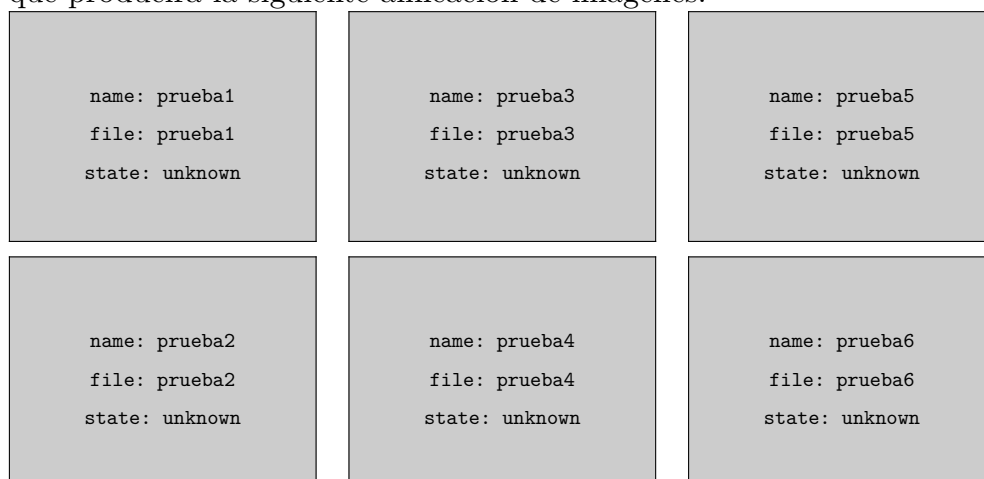
```
\startcombination[3*2]
  {\externalfigure[prueba1]}
  {\externalfigure[prueba2]}
```

```

{\externalfigure[prueba3]}
{\externalfigure[prueba4]}
{\externalfigure[prueba5]}
{\externalfigure[prueba6]}
\stopcombination

```

que producirá la siguiente alineación de imágenes.



En el anterior ejemplo, las imágenes que he combinado en realidad no existen, razón por la que ConT_EXt en lugar de las imágenes ha generado cajas de texto con información sobre las mismas.

Véase, por otro lado, cómo cada elemento a combinar dentro de `\startcombination`, va encerrado entre llaves.

En realidad `\startcombination` no sólo permite vincular y alinear imágenes, sino cualquier tipo de *caja* como pueden ser los textos ubicados dentro de un entorno `\startframedtext`, tablas, etc. Para configurar la combinación podemos usar el comando `\setupcombination` y, asimismo, podemos crear combinaciones preconfiguradas mediante `\definecombination`.

13.4.4 Configuración general de los objetos flotantes

Ya hemos visto que con `\placefloat` podemos controlar la ubicación del objeto flotante que se está insertando y algún otro detalle. Además es posible configurar:

- Las características globales de un concreto tipo de objeto flotante. Ello se hace mediante `\setupfloat[Nombre del tipo de objeto flotante]`.
- Las características globales de todos los objetos flotantes de nuestro documento. Ello se hace mediante `\setupfloats`.

Téngase en cuenta que del mismo modo que `\placefloat[figure]` equivale a `\placefigure`, `\setupfloat[figure]` equivale a `\setupfigures`, y `\setupfloat[table]` equivale a `\setuptables`.

Respecto a las opciones configurables para estos elementos, me remito al listado oficial de comandos de ConT_EXt (sección 3.6).

13.5 Definición de objetos flotantes adicionales

El comando `\definefloat` nos permite definir nuestros propios objetos flotantes. Su sintaxis es:

```
\definefloat[Nombre singular] [Nombre plural] [Configuración]
```

Donde el argumento *Configuración* es un argumento opcional que permite indicar ya la configuración de este nuevo objeto en el momento de su creación. Podemos también hacerlo más tarde mediante `\setupfloat[Nombre en singular]`.

Como con este epígrafe terminamos nuestra introducción, voy a aprovechar para, al hilo de `\definefloat` profundizar un poco en la aparente *selva* de comandos de ConT_EXt que, una vez que se comprende, no es tan *selvática* sino, de hecho, bastante racional.

Empecemos por preguntarnos qué es, en realidad, para ConT_EXt, un objeto flotante. La respuesta es que se trata de un objeto con las siguientes características:

- Que tiene cierto margen de libertad respecto de su ubicación en la página.
- Que lleva asociado a él una *lista* con su mismo nombre que permite numerar este tipo de objetos y, eventualmente, generar un índice de ellos.
- Que tiene un título.
- Que, cuando el objeto realmente puede flotar, debe ser tratado como unidad inseparable, es decir (en terminología de T_EX) *encerrado en una caja*.

O sea, que el objeto flotante, en realidad, se compone de tres elementos: el objeto propiamente dicho, la lista asociada a él y el título. Para controlar al objeto propiamente dicho se necesita, sólo, un comando que configure su ubicación y otro que permita insertar el objeto en el documento; para configurar los aspectos relativos a la lista, bastan los comandos generales de control de listas, y para configurar los aspectos relativos al título, bastan los comandos generales de control del título.

Y aquí es donde viene la genialidad de ConT_EXt: podría haberse diseñado un simple comando de control de los objetos flotantes (`\setupfloats`), y un simple comando de inserción de objetos flotantes: `\placefloat`; pero lo que ConT_EXt hace es:

1. Diseñar un comando que permita vincular un nombre a una configuración específica de objetos flotantes. Se trata de `\definefloat`, que en realidad no vincula un nombre, sino dos nombres uno en singular y otro en plural.
2. Crear, junto con el comando de configuración global de objetos flotantes, un comando que permita configurar sólo un tipo concreto de objetos: `\setupfloat[Objeto]`.
3. Añadir al comando de ubicación de objetos flotantes (`\placefloat`) un argumento que permita diferenciar entre unos tipos u otros (`\placefloat[Objeto]`).
4. Crear comandos, que incluyan el nombre del objeto, para todas las acciones propias de un objeto flotante. Algunos de esos comandos (que en realidad son clones de otros comandos más generales) usarán el nombre del objeto en singular y otros lo usarán en plural.

Por lo tanto, cuando creamos un nuevo objeto flotante y le decimos a ConT_EXt cuál es su nombre en singular y cuál en plural, ConT_EXt:

- Reserva un espacio de memoria para almacenar en él la configuración concreta de ese tipo de objetos.
- Crea una nueva lista con el nombre en singular de ese tipo de objetos, ya que los objetos flotantes se asocian a una lista.
- Crea un nuevo tipo de «título» vinculado a este nuevo tipo de objetos flotantes, para poder mantener una configuración personalizada de estos títulos.
- Y, por último, crea un grupo de comandos nuevos específicos para ese nuevo tipo de objeto, cuyo nombre es en realidad un sinónimo del comando más general.

En la [tabla 13.3](#) se pueden ver los comandos que se crean automáticamente cuando definimos un nuevo objeto flotante, así como el comando más general del que son sinónimos:

Comando	Sinónimo de	Ejemplo
<code>\completelistof<NombrePlural></code>	<code>\completelist[NombrePlural]</code>	<code>\completelistoffigures</code>
<code>\place<NombreSingular></code>	<code>\placefloat[NombreSingular]</code>	<code>\placefigure</code>
<code>\placelistof<NombrePlural></code>	<code>\placelist[NombrePlural]</code>	<code>\placelistoffigures</code>
<code>\setup<NombreSingular></code>	<code>\setupfloat[NombreSingular]</code>	<code>\setupfigure</code>

Tabla 13.3 Comandos que se crean automáticamente al crear un nuevo objeto flotante

En realidad se crean algunos comandos adicionales, que son sinónimos de los anteriores y que como no he incluido en la explicación del capítulo, he omitido de la [tabla 13.3](#): `\start<NombreSingular>`, `\start<NombreSingular>text` y `\startplace<NombreSingular>`.

He usado como ejemplo de los comandos creados al definir un nuevo objeto flotante el comando que se usa para las imágenes; y lo he hecho porque las imágenes, igual que las tablas y el resto de objetos flotantes predefinidos por ConT_EXt, son casos concretos de `\definefloat`:

```
\definefloat[chemical][chemicals]
\definefloat[figure][figures]
\definefloat[table][tables]
\definefloat[intermezzo][intermezzi]
\definefloat[graphic][graphics]
```

Por último, véase que, en realidad, ConT_EXt no controla de ninguna manera el tipo de material que se incluye en cada objeto flotante concreto; se supone que eso es tarea del autor. Esta es la razón de que con los comandos `\placefigure` o `\placetable` se pueda insertar también texto. Pero el texto insertado con `\placefigure` se incluya en la lista de imágenes, y el insertado con `\placetable`, en la lista de tablas.

Apéndice

Apéndice A

Instalación, configuración y actualización de ConT_EXt

Las principales distribuciones de T_EX (TeX Live, teTeX, MikTeX, MacTeX, etc.) incluyen una versión de ConT_EXt. Pero dicha versión normalmente no es la más actualizada. En el presente apéndice explicaré dos procedimientos para instalar dos versiones distintas de ConT_EXt; la primera incluye tanto ConT_EXt Mark II como Mark IV y la segunda incluye exclusivamente ConT_EXt Mark IV.

El procedimiento de instalación sigue los mismos pasos en cualquier sistema operativo; pero los detalles cambian de un sistema a otro. No obstante, podemos simplificar de tal modo que en las líneas que siguen distinguiré entre dos grandes grupos de sistemas:

- **Sistemas tipo Unix:** Aquí se incluye a Unix, propiamente dicho, así como a GNU Linux, Mac OS, FreeBSD, OpenBSD o Solaris. El procedimiento es básicamente el mismo en todos estos sistemas; hay algunas pequeñísimas diferencias que destacaré en el lugar adecuado.
- **Sistemas Windows,** lo que incluye las distintas versiones de dicho sistema operativo: Windows 10 (la última versión, según creo), Windows 8, Windows 7, Windows Vista, Windows XP, Windows NT, etc.

Nota importante sobre el proceso de instalación en sistemas Microsoft Windows:

ConT_EXt, como todos los sistemas T_EX, está pensado para trabajar desde una terminal; los programas y procedimientos para su instalación, también. Eso en Windows es perfectamente posible y no debería tener mayor dificultad. El problema está en que, de un lado, los usuarios de Windows no siempre tienen costumbre de hacer eso y, de otro, en que como Windows nació sobre la *ilusión* (falsa) de que en un sistema informático podía hacerse todo gráficamente, en general las versiones de dicho sistema operativo no *publicitan* demasiado cómo utilizar la terminal y, sobre todo, es corriente que en cada versión del sistema se cambie el nombre del programa que ejecuta la terminal, y la manera de iniciarlo. Hasta donde yo se, al programa de emulación de terminal de Windows se le ha llamado de muchas maneras: “Ventana de DOS”, “Símbolo del sistema”, “Command Prompt”, “cmd”, etc. La ubicación de este programa en el menú de aplicaciones de Windows también cambia según la versión de Windows de que se trate.

Yo dejé de usar sistemas basados en Windows en 2004, por lo que aquí poco puedo ayudar al lector. Tendrá él que descubrir, por su cuenta, cómo abrir una terminal en su concreta versión del sistema operativo; lo que no debería ser muy difícil.

1 Instalación y configuración de «ConT_EXt Standalone»

La distribución de ConT_EXt conocida como «Standalone», término inglés que podríamos traducir como «autónoma», «independiente» o «autosuficiente», conocida también como «ConT_EXt Suite», es una distribución completa y actualizada de ConT_EXt, que descarga de Internet los ficheros necesarios, no ocupa demasiado espacio en disco, es fácil de actualizar, y, sobre todo —de ahí el nombre de *Standalone*— se contiene en un solo directorio, que puede estar ubicado en el lugar del disco duro en que deseemos; de tal manera que incluso sería posible que en un mismo ordenador convivieran varias versiones de ConT_EXt, cada una en su propio directorio. Esta distribución incluye las fuentes, ficheros binarios y documentación necesaria para ejecutar ConT_EXt Mark II (lo que implica los motores de T_EX PdfLatex y XeT_EX), y ConT_EXt Mark IV (lo que implica el motor LuaT_EX).

Para información sobre lo que son los *motores* de T_EX, véase la [sección 1.4.1](#); y sobre los motores de T_EX en relación con ConT_EXt, así como sobre las versiones llamadas Mark II y Mark IV, la [sección 1.5.1](#).

A continuación se explica cómo instalar, ejecutar, actualizar y restaurar en nuestro sistema «ConT_EXt Standalone». Los datos y procedimientos que aquí se proporcionan son un resumen de la información mucho más extensa (pero en inglés) incluida en la [wiki de ConT_EXt](#), a la que he añadido algún detalle adicional extraído de un wikilibro sobre ConT_EXt alojado en [wikibooks](#). Si hubiera algún problema con la instalación, o se deseara ampliar algún detalle, se debe consultar directamente cualquiera de estas dos páginas (según prefiramos el idioma inglés o el francés).

1.1 Instalación

La instalación de «ConT_EXt Standalone» exige tener una conexión a Internet, e implica los siguientes pasos:

1. Crear el directorio en el que se instalará ConT_EXt.
2. Descargar en dicho directorio, el *script* de instalación.
3. Ejecutar dicho *script* con las opciones deseadas.
4. Realizar ciertos ajustes finales.

Paso 1: Crear el directorio de instalación

Esto, en realidad, no tiene nada que ver con ConT_EXt y hay que suponer que todo usuario sabrá hacerlo. En sistemas Windows lo normal es hacerlo desde el

administrador de archivos. En sistemas tipo Unix, se puede hacer desde algún administrador de archivos o desde una terminal. Es importante, eso sí, tener en cuenta que se desaconseja que el directorio de instalación contenga algún espacio en blanco en su ruta de acceso. Yo personalmente, además, tiendo a huir de usar en los nombres de directorios caracteres no anglosajones tales como vocales acentuadas.

En adelante asumiré que el directorio de instalación es, en sistemas tipo Unix, «~/context/» y en Windows, «C:\Programs\context».

Paso 2: Descargar, en el directorio de instalación, el *script* de instalación

El *script* de instalación es distinto según el sistema operativo en el que se vaya a realizar la instalación:

- En sistemas tipo Unix se puede descargar, con un navegador web, o, desde una terminal con «**wget**» o con «**rsync**»:

```
wget http://minimals.contextgarden.net/setup/first-setup.sh
rsync rsync://minimals.contextgarden.net/setup/first-setup.sh
```

- En sistemas tipo Windows, hasta donde yo se, no hay herramientas estándar para descargarlo desde consola. Ha de ser con un navegador web. La dirección de descarga puede ser cualquiera de las siguientes:

```
http://minimals.contextgarden.net/setup/context-setup-mswin.zip
http://minimals.contextgarden.net/setup/context-setup-win64.zip
```

Una vez descargado, en Windows, hay que descomprimir el fichero

Paso 3: Ejecutar el *script* de instalación

El *script* de instalación se debe ejecutar desde una terminal. En sistemas tipo Unix el nombre del *script* es «**first-setup.sh**» y se puede ejecutar con **bash** o con **sh**. En sistemas tipo Windows el *script* se denomina «**first-setup.bat**» y se ejecuta simplemente escribiendo su nombre, en la consola del sistema o ventana de MS-DOS desde el directorio de instalación.

El *script* de instalación admite las siguientes opciones:

- **-context:** Esta opción determina qué versión de ConT_EXt se instalará, si la última versión de desarrollo («--context=latest») o la última versión estable («--context=beta»). El valor por defecto es «beta».
- **-engine:** Nos permite indicar si queremos instalar Mark IV («--engine=luatex», valor por defecto) o si queremos instalar Mark II.
- **-modules:** Instala también los módulos de expansión de ConT_EXt que no pertenecen propiamente a la distribución, pero que proporcionan interesantes utilidades adicionales. Para ello hay que indicar «--modules=all».

Respecto a las opciones de instalación, creo que la información de la wiki está obsoleta. En ella se dice que para instalar solamente Mark IV hay que indicar explícitamente la opción «--engine=luatex» y que la opción «--context=latest» instala la última versión estable, no la versión de desarrollo. Sin embargo desde mediados del año 2020 cambió el contenido de first-setup.sh y ojeando en su interior he comprobado que para instalar la ultimísima versión hay que indicar expresamente «--context=latest», pero que «--engine=luatex» viene habilitada por defecto.

El wikilibro francés que mencioné al principio, a las opciones que acabo de mencionar (documentadas en la wiki de ConT_EXt) añade otras dos opciones posibles: «--fonts=all» y «goodies=all». Contextgarden no las menciona, pero incluirlas también en el comando de instalación no hace daño. Por lo tanto yo aconsejaría ejecutar el script de instalación con las siguientes opciones (según nos encontremos en un sistema tipo Unix o tipo Windows):

- Unix: `bash first-setup.sh --context=latest --modules=all --fonts=all --goodies=all`
- Windows: `first-setup.bat --context=latest --modules=all --fonts=all --goodies=all`

Esto, dependiendo de la velocidad de nuestra conexión a Internet, puede tomar cierto tiempo, aunque no demasiado.

Configuración del proxy

El script de instalación utiliza rsync para obtener los archivos necesarios. Así que, si estamos detrás de un servidor proxy, es preciso especificar los detalles del mismo a rsync. La forma más fácil de establecer esto es establecer la variable RSYNC_PROXY en la terminal o en su *script* de inicio (.bashrc o el archivo correspondiente para cada shell). Reemplace el nombre de usuario, contraseña, proxyhost y proxyport con la información correcta. Esto se hace, en sistemas tipo Unix con el comando «export» y en sistemas tipo Windows con «set». Por ejemplo:

```
export RSYNC_PROXY=nombredeusuario:contraseña@proxyhost:proxyport
```

A veces, cuando estamos detrás de un cortafuegos, el puerto 873 puede estar cerrado para las conexiones TCP salientes. Si el puerto 22 está abierto para conexiones ssh, un truco que puede utilizarse es conectarse a un ordenador situado en algún lugar fuera del cortafuegos y hacer un túnel en el puerto 873 (utilizando el programa nc).

```
export RSYNC_CONNECT_PROG='ssh tunelhost nc %H 873'
```


donde el "tunnelhost" es la máquina fuera del firewall a la que tenemos acceso. Por supuesto, esta máquina debe tener nc y el puerto 873 abierto para la conexión TCP saliente

Al terminar la ejecución de «**first-setup**» en el directorio de instalación habrán aparecido dos nuevos directorios llamados, respectivamente, «**bin**» y «**tex**».

Paso 4: Ajustes finales (Sólo en GNU Linux)

En sistemas GNU Linux son muchos los directorios en los que puede haber fuentes instaladas. Si queremos que ConT_EXt las pueda utilizar debemos informarle de dónde localizarlas. Para ello deberemos añadir la siguiente línea al fichero «**tex/setup_{tex}**» creado tras la instalación:

```
export OSFONTDIR=~/.fonts:/usr/share/fonts:/usr/share/texmf/fonts/opentype/"
```

con la que se carga la variable de entorno OSFONTDIR con los tres directorios en los que normalmente se pueden localizar las fuentes instaladas en el sistema

El directorio /usr/share/texmf/fonts/ sólo existirá si en nuestro sistema hay alguna otra instalación de T_EX o de sistemas basados en él; en tal caso conviene incluirlo en la ruta de OSFONTDIR pues así podremos usar las fuentes opentype que dicha instalación haya podido incluir. Si hemos adquirido alguna fuente comercial que queremos que ConT_EXt pueda utilizar, hay que asegurarse de que la ruta de acceso a la misma sea alguna de las incluidas en OSFONTDIR y en caso contrario, añadir a esta variable, la ruta que sea. Tengo entendido, por ejemplo, que algunas fuentes se instalan en /usr/local/fonts en lugar de en /usr/share/fonts.

Por último puede ser buena idea hacer que ConT_EXt genere una base de datos con los archivos necesarios para su ejecución. Ello se hará ejecutando las siguientes tres órdenes desde un terminal:

```
. ~/context/tex/setuptex
context --generate
context --make
```

La primera instrucción es un punto. Eso es una abreviatura para el comando interno de bash **source**. También podemos, claro, ejecutar *source* si nos es más cómodo.

1.2 Ejecución de «ConT_EXt Standalone»

«ConT_EXt Standalone» se ha diseñado para poder convivir con otras instalaciones de sistemas T_EX, lo que es una ventaja pues nos permite tener instaladas varias versiones distintas en un mismo sistema; pero para poder disfrutar de dicha ventaja es imprescindible que las variables de entorno que se precisan para la ejecución de ConT_EXt no se establezcan de modo permanente, lo que, a su vez, supone una servidumbre, pues cada vez que iniciemos una terminal para ejecutar

en ella «context» tendremos que empezar por cargar en memoria dichas variables de entorno, las cuales se contienen en el fichero «tex/setuptex» (Unix) o «tex/setuptex.bat» (Windows). Ello se hace:

- En sistemas tipo Unix, ejecutando, tras abrir la terminal en la que queremos usar «context», cualquiera de los siguientes dos comandos:

```
source ~/context/tex/setuptex
. ~/context/tex/setuptex
```

(asumiendo que el directorio donde se encuentra la versión de «context» que queremos utilizar sea sea «~/context»).

- En sistemas tipo Windows, en la terminal desde la que usaremos context, ejecutando el comando `tex\setuptex.bat` desde el directorio de instalación.

Si en nuestro sistema no hay ninguna otra instalación de T_EX o de alguno de sus derivados, podemos evitar dicha servidumbre automatizando la ejecución de esta orden cada vez que se abra una terminal:

- En sistemas tipo Unix ello se hace incluyéndola en el fichero donde se contiene el *script* general de arranque de las terminales (normalmente «.bashrc»).

El fichero de configuración de una terminal depende del programa de *shell* que dicha terminal utilice por defecto. Si esta es bash (que es la más usada en sistemas GNU Linux), el fichero que se lee al principio es .bashrc. Las *shell* sh y ksh utilizan un fichero llamado .profile, zsh utiliza .zshenv, y tcsh o csh leen el fichero .cshrc. Alguna implementación concreta puede cambiar los nombres de estos ficheros y así, por ejemplo, .bashrc a veces se denomina .bash_profile.

- En sistemas tipo Windows podemos crear un atajo en el escritorio que ejecute cmd.exe y luego editarlo, poniendo como orden a ejecutar cuando se haga doble click sobre él:

```
C:\WINDOWS\System32\cmd.exe /k C:\Programs\context\tex\setuptex.bat
```

Otra posibilidad, si no queremos ejecutar este script cada vez que vayamos a usar ConT_EXt, ni queremos tampoco establecer de modo permanente las variables de entorno necesarias para su ejecución, es la de, en lugar de ejecutar ConT_EXt desde una terminal, hacerlo desde el propio editor de textos. El cómo hacer esto depende del concreto editor de textos que se esté utilizando. En la wiki de ConT_EXt se informa de cómo configurar varios editores corrientes: LEd, Notepad++, Scite, TeXnicCenter, TeXworks, vim y algunos otros.

1.3 Actualizar la versión de «ConT_EXt Standalone» o volver a una versión anterior

Mark IV aún se encuentra en desarrollo, por lo que «ConT_EXt Standalone» se renueva con cierta frecuencia. Para actualizar nuestra instalación basta con repetir el proceso: descargamos una nueva versión de «`first-setup.sh`» y lo ejecutamos.

Si, por las razones que sean, queremos volver a una versión anterior de «ConT_EXt Standalone» basta con ejecutar «`first-setup`» con la opción «`--context=fecha`» donde *fecha* es la fecha correspondiente a la versión que queremos recuperar. Piénsese que la fecha ha de introducirse en el formato anglosajón: meses-días-años y no en el español: días-meses-años.

La lista completa de versiones de ConT_EXt y fechas asociadas a las mismas se encuentra en [este enlace](#).

Téngase en cuenta, por último, que tras reinstalar el sistema, tanto si es para actualizar como si es para regresar a una versión previa, en sistemas GNU Linux habrá que volver a ejecutar el paso 4 de la instalación, al que he denominado «Ajustes finales».

2 Instalación de LMTX

Si sólo pensamos usar ConT_EXt Mark IV, y queremos compilar nuestros proyectos no directamente con LuaT_EX, sino con LuaMetaT_EX, una versión simplificada de LuaT_EX que utiliza menos recursos del sistema y que, por lo tanto, puede funcionar en sistemas *menos potentes*, debemos instalar en nuestro sistema, en lugar de «ConT_EXt Standalone», LMTX que es la ultimísima versión de ConT_EXt, y cuya denominación es un acrónimo del nombre del *motor* T_EX que se utiliza: LuaMetaT_EX. Esta versión se lanzó en 2019 y, desde aproximadamente mayo de 2020 es la distribución de ConT_EXt cuya instalación se propone por defecto en la [wiki de ConT_EXt](#).

El desarrollo actual de LMTX es intenso, y la versión beta puede llegar a cambiar varias veces en una misma semana. Algunos de sus desarrollos, además, plantean, temporalmente, ciertas incompatibilidades con Mark IV, y así, por ejemplo, mientras escribo estas líneas la última versión de LMTX (del 4 de agosto de 2020) produce un error con el comando `\Caps`. Por ello yo aconsejaría a los novatos, de momento, trabajar más bien con «ConT_EXt Standalone».

2.1 Instalación propiamente dicha

La instalación propiamente dicha es tan sencilla como:

- **Paso 1:** Decidir en qué directorio queremos instalar ConT_EXt LMTX y, si es preciso, crearlo. Asumiré que la instalación se hace en un directorio llamado “context” ubicado en nuestro directorio de usuario.
- **Paso 2:** Descargar de la [wiki de ConT_EXt](#), en el directorio de instalación, el fichero zip que se corresponda con nuestro sistema operativo, y procesador. Puede ser cualquiera de los siguientes:
 - GNU/Linux
 - ★ Procesador X86
 - ▷ [Versión de 32 bits.](#)
 - ▷ [Versión de 64 bits.](#)
 - ★ Procesador ARM
 - ▷ [Versión de 32 bits.](#)
 - ▷ [Versión de 64 bits.](#)
 - Microsoft Windows
 - ★ [Versión de 32 bits](#)
 - ★ [Versión de 64 bits](#)
 - Mac OS, [versión de 64 bits](#)
 - FreeBSD
 - ★ [Versión de 32 bits.](#)
 - ★ [Versión de 64 bits.](#)
 - OpenBSD6.6
 - ★ [Versión de 32 bits.](#)
 - ★ [Versión de 64 bits.](#)
 - OpenBSD6.7
 - ★ [Versión de 32 bits.](#)
 - ★ [Versión de 64 bits.](#)

Si no sabe si su sistema es de 32 o 64 bits, lo más probable —salvo que su ordenador sea muy viejo—, es que sea de 64 bits. Si no sabe si su procesador es X86 o ARM, lo más probable es que sea X86.

- **Paso 3:** Descomprimir, en el directorio de instalación, el fichero descargado en el paso anterior. Se generará una carpeta llamada «bin» y dos ficheros, uno, llamado «`installation.pdf`», que contiene —en inglés— información más detallada sobre la instalación, y un segundo fichero que es el auténtico programa de instalación y que se denomina «`install.sh`» (en sistemas tipo Unix) o «`install.bat`» (en sistemas Windows).
- **Paso 4:** Ejecutar el programa de instalación («`install.sh`» o «`install.bat`»). Se requiere conexión con Internet, pues el programa de instalación busca en la web los ficheros que necesita.

- En sistemas tipo Unix el programa de instalación se ejecuta desde una terminal, ubicados en el directorio de instalación, bien con **bash**, bien con **sh**. Para ello no es preciso tener privilegios de administrador, salvo que el directorio de instalación se encuentre fuera del directorio «home» del usuario.
 - En sistemas tipo Windows hay que abrir una terminal, desplazarse al directorio de instalación, y, desde la terminal, ejecutar **install.bat**. No es tampoco aquí preciso que el programa de instalación se ejecute como administrador del sistema, pero se recomienda que así se haga para que se puedan usar enlaces simbólicos de los ficheros ahorrando así espacio en disco.
- **Paso 5** informar al sistema de la ruta de acceso a LMTX:

En sistemas Windows, el programa de instalación genera un fichero, llamado «**setpath.bat**» que actualiza todos los ficheros de configuración necesarios para que Windows sepa que se ha instalado LMTX en el sistema y dónde lo ha hecho. En Sistemas GNU Linux, FreeBSD o Mac OS no se genera ningún *script* que automatice la tarea, por lo que debemos incorporar, en la variable **PATH** del sistema, la dirección de los binarios de ConT_EXt lo que conseguiríamos ejecutando en el terminal, desde el directorio de instalación:

```
export PATH="DirInstalación/tex/texmf-Plataforma/bin:$PATH
```

donde *DirInstalación* representa el directorio de instalación (por ejemplo “/home/usuario/context”) y *texmf-Plataforma* variará según qué versión de LMTX hemos instalado. Por ejemplo en una instalación hecha en un sistema Linux de 64 bits, *texmf-Plataforma* será “texmf-linux-64”. Deberíamos, por lo tanto, ejecutar la siguiente orden desde un terminal:

```
export PATH="/home/usuario/context/texmf-linux-64/bin:$PATH
```

Dicha orden incluirá a LMTX en la ruta del sistema, sólo mientras se mantenga abierta la terminal desde la que se ha ejecutado. Si queremos que ello se realice automáticamente cada vez que se abra un terminal, debemos incluir dicha orden en el fichero de configuración del programa de *shell* que se use por defecto en el sistema. Cuál sea este fichero cambia según cuál sea el programa de *shell*: **bash**, **sh**, **zsh**, **ksh**, **tcsh**, **csh**... En la mayoría de los sistemas Linux, que usan **bash**, el fichero se llama «**.bashrc**» por lo que deberíamos ejecutar la siguiente orden desde nuestro directorio de usuario:

```
echo 'export PATH="/home/usuario/context/texmf-linux-64/bin:$PATH >> .bashrc
```

Nota Importante: Al ejecutar este paso, inhabilitaremos la posibilidad de utilizar otras versiones de ConT_EXt existentes en nuestro sistema como, por ejemplo, la incorporada en TeX Live, o «ConT_EXt Standalone». Si queremos

compatibilizar ambas versiones, es preferible usar el procedimiento descrito en la [sección 3](#).

2.2 Instalación en LMTX de los módulos de expansión

ConT_EXt LMTX no incorpora un procedimiento para instalar o actualizar los módulos de expansión de ConT_EXt. Sin embargo en la wiki de ConT_EXt se contiene un *script* que permite instalar y actualizar todos los módulos junto con el resto de la instalación.

Para ello debemos copiar el [mencionado script](#), pegarlo en un fichero de texto ubicado en el directorio principal de instalación de LMTX (el que contiene `install.sh` o `install.bat`) y ejecutarlo desde una terminal. He comprobado personalmente que esto funciona en un sistema GNU Linux. No estoy seguro de si funcionará en un sistema Windows, pues no dispongo de ninguna versión de dicho sistema operativo con la que comprobarlo.

2.3 Actualizar LMTX

Actualizar LMTX es tan simple como volver a ejecutar el programa de instalación: se comprobarán los ficheros instalados con los del servidor web y se actualizará lo que sea necesario.

Si el sitio web desde donde se obtienen los archivos ha cambiado, debemos adaptar la dirección del mismo en el *script* de instalación; aunque acaso sea más sencillo descargar una nueva versión de los ficheros de instalación en el mismo directorio y extraer de ella el nuevo «`install.sh`» o «`install.bat`»; o, más fácil aún, descomprimir el fichero con el programa de instalación y reinstalar sin necesidad de eliminar primero los ficheros antiguos.

2.4 Crear un fichero que cargue en memoria las variables necesarias para LMTX (sólo sistemas GNU/Linux)

«ConT_EXt Standalone» contiene, como ya sabemos, un fichero («`tex/setuptex`») que carga en memoria todas las variables necesarias para su ejecución, pero LMTX no incluye un fichero similar. Podemos, no obstante, crearlo nosotros con facilidad y almacenarlo, por ejemplo, como «`setuplmtx`» en el directorio «`tex`». Los comandos que podría tener dicho fichero serían:

```
export PATH="$HOME/.context/LMTX/tex/texmf-linux-64/bin:$PATH"
echo "Añadiendo ~/context/LMTX/tex/texmf-linux-64/bin a la ruta de PATH"
export TEXROOT="$HOME/.context/LMTX/tex"
```

```
echo "Estableciendo ~/context/LMTX/tex como TEXROOT"
export OSFONTDIR=~/.fonts:/usr/share/fonts:/usr/share/texmf/fonts/opentype/"
echo "Cargando en memoria directorios de fuentes"
alias lmtx=~/.context/LMTX/tex/texmf-linux-64/bin/context"
echo "Creando un alias para ejecutar lmtx"
```

Con esto, además de cargar en memoria las rutas y variables necesarias para ejecutar LMTX habilitaríamos el comando «`lmtx`» como sinónimo de «`context`».

Tras crear este fichero, antes de poder usar LMTX deberíamos ejecutar, en la terminal en la que pretendemos usarlo,

```
source ~/.context/LMTX/tex/setuplmtx
```

todo ello suponiendo que LMTX se haya instalado en «`/context/LMTX`» y que hayamos denominado a este fichero «`setuplmtx`» y lo hayamos almacenado en «`/context/LMTX/tex`».

Lo anterior es lo que yo hago, para trabajar con LMTX del mismo modo con que trabajaba con «ConT_EXt Standalone». No obstante no excluyo la posibilidad de que en LMTX no sea necesario, por ejemplo, cargar en memoria el la variable OSFONTDIR, pues es llamativo que en la wiki de ConT_EXt no se diga nada al respecto.

3 Usar en el mismo sistema varias versiones de ConT_EXt (sólo para sistemas tipo Unix)

La utilidad del sistema operativo llamada `alias` nos permite asociar nombres distintos a diferentes versiones de ConT_EXt. Así podemos usar, por ejemplo, la versión de ConT_EXt incluida en TeX Live y LMTX; o la versión *Standalone* y LMTX.

Por ejemplo, si almacenamos en directorios distintos las versiones de LMTX descargadas en enero y en agosto de 2020 podríamos escribir en «`.bashrc`» (o fichero equivalente que se lee por defecto al abrir una terminal), las siguientes dos instrucciones:

```
alias lmtx-01="/home/usuario/context/202001/tex/texmf-linux-64/bin/context"
alias lmtx-08="/home/usuario/context/202008/tex/texmf-linux-64/bin/context"
```

Estas instrucciones asociarán los nombres `lmtx-01` a la versión de LMTX instalada en el directorio «`context/202001`» y `lmtx-08` a la versión instalada en «`context/202008`».

Apéndice B

Comandos para generar símbolos, matemáticos y no matemáticos

En las próximas tablas se recogen los comandos que, según he podido comprobar uno a uno, generan diferentes símbolos; la mayoría de los cuales (pero no todos) son de uso preferentemente matemático.

Soy consciente de que la ordenación de los mismos es manifiestamente mejorable. El problema está en que se incluyen muchísimos símbolos de uso matemático que yo —que soy de letras— no se para qué sirven; muchas veces ni siquiera estoy seguro de que se trate realmente de símbolos de uso matemático. Por ello he hecho un grupo con los símbolos que estoy razonablemente seguro de que no son de uso matemático, y respecto del resto he ido agrupando los distintos símbolos atendiendo a ciertas formas reconocibles (triángulos, cuadrados, asteriscos, rombos, flechas, puntos). El resto de los símbolos *probablemente* de uso matemático, los he ordenado alfabéticamente (a partir del comando que los genera).

Las tablas no son, por otra parte, exhaustivas. Estoy seguro de que hay muchos más símbolos que se pueden generar con ConT_EXt, aunque yo no he encontrado algún documento o página web que los recoja todos. Los que aquí se recogen son, en su mayor parte, símbolos que funcionan en T_EX o en L^AT_EX, y que he comprobado que funcionan también en ConT_EXt. Gran parte de estos símbolos en L^AT_EX sólo funcionan en modo matemático (encerrados entre «\$»). En ConT_EXt, como es fácil de ver en las próximas tablas, eso sólo es necesario en muy pocos casos.

Monedas y símbolos de uso legal

©	<code>\copyright</code>	®	<code>\registered</code>	¢	<code>\textcent</code>
Ⓟ	<code>\textcircledP</code>	₡	<code>\textcurrency</code>	\$	<code>\textdollar</code>
₪	<code>\textdong</code>	€	<code>\texteuro</code>	f	<code>\textflorin</code>
£	<code>\textsterling</code>	¥	<code>\textyen</code>	™	<code>\trademark</code>

Triángulos, círculos, cuadrados y otras formas

\triangle	<code>\triangle</code>	\bigcirc	<code>\bigcirc</code>	\square	<code>\square</code>
\triangleleft	<code>\triangleleft</code>	\circ	<code>\circ</code>	\blacksquare	<code>\blacksquare</code>
\triangleright	<code>\triangleright</code>	\bullet	<code>\bullet</code>	\boxdot	<code>\boxdot</code>
\triangledown	<code>\triangledown</code>	\circledast	<code>\circledast</code>	\boxminus	<code>\boxminus</code>
\blacktriangledown	<code>\blacktriangledown</code>	\circledcirc	<code>\circledcirc</code>	\boxplus	<code>\boxplus</code>
\blacktriangleleft	<code>\blacktriangleleft</code>	\circleddash	<code>\circleddash</code>	\boxtimes	<code>\boxtimes</code>
\blacktriangleright	<code>\blacktriangleright</code>	\bigoplus	<code>\bigoplus</code>	\ast	<code>\ast</code>
\blacktriangle	<code>\blacktriangle</code>	\bigotimes	<code>\bigotimes</code>	\maltese	<code>\maltese</code>
\triangleq	<code>\triangleq</code>	\oplus	<code>\oplus</code>	\star	<code>\star</code>
\diamond	<code>\diamond</code>	\ominus	<code>\ominus</code>	\clubsuit	<code>\clubsuit</code>
\lozenge	<code>\lozenge</code>	\otimes	<code>\otimes</code>	\heartsuit	<code>\heartsuit</code>
\blacklozenge	<code>\blacklozenge</code>	\oslash	<code>\oslash</code>	\spadesuit	<code>\spadesuit</code>
\diamondsuit	<code>\diamondsuit</code>	\odot	<code>\odot</code>	\varnothing	<code>\varnothing</code>

Flechas:

\leftarrow	<code>\leftarrow</code> , <code>\gets</code>	\rightarrow	<code>\rightarrow</code> , <code>\to</code>	\leftrightarrow	<code>\leftrightarrow</code>
\nleftarrow	<code>\nleftarrow</code>	\nrightarrow	<code>\nrightarrow</code>	\Leftrightarrow	<code>\Leftrightarrow</code>
\longleftarrow	<code>\longleftarrow</code>	\longrightarrow	<code>\longrightarrow</code>	\longleftrightarrow	<code>\longleftrightarrow</code>
\Lleftarrow	<code>\Lleftarrow</code>	\Rrightarrow	<code>\Rrightarrow</code>	\Longleftrightarrow	<code>\Longleftrightarrow</code>
\nLleftarrow	<code>\nLleftarrow</code>	\nRrightarrow	<code>\nRrightarrow</code>	\leftrightsquigarrow	<code>\leftrightsquigarrow</code>
\Lsh	<code>\Lsh</code>	\Rsh	<code>\Rsh</code>	\leftrightsquigarrow	<code>\leftrightsquigarrow</code>
\mapsfrom	<code>\mapsfrom</code>	\mapsto	<code>\mapsto</code>	\nLeftrightarrow	<code>\nLeftrightarrow</code>
\longmapsfrom	<code>\longmapsfrom</code>	\longmapsto	<code>\longmapsto</code>	\nletrightarrow	<code>\nletrightarrow</code>
\Mapsfrom	<code>\Mapsfrom</code>	\Mapsto	<code>\Mapsto</code>	\rightleftarrows	<code>\rightleftarrows</code>
\Longmapsfrom	<code>\Longmapsfrom</code>	\Longmapsto	<code>\Longmapsto</code>	\rightleftharpoons	<code>\rightleftharpoons</code>
\leftarrowtail	<code>\leftarrowtail</code>	\rightarrowtail	<code>\rightarrowtail</code>	\updownarrow	<code>\updownarrow</code>
\twoheadleftarrow	<code>\twoheadleftarrow</code>	\twoheadrightarrow	<code>\twoheadrightarrow</code>	\Updownarrow	<code>\Updownarrow</code>
\circlearrowleft	<code>\circlearrowleft</code>	\circlearrowright	<code>\circlearrowright</code>	\updownarrows	<code>\updownarrows</code>
\curvearrowleft	<code>\curvearrowleft</code>	\curvearrowright	<code>\curvearrowright</code>	\uparrow	<code>\uparrow</code>
\hookleftarrow	<code>\hookleftarrow</code>	\hookrightarrow	<code>\hookrightarrow</code>	\Uparrow	<code>\Uparrow</code>
\leftharpoondown	<code>\leftharpoondown</code>	\rightharpoondown	<code>\rightharpoondown</code>	\upuparrows	<code>\upuparrows</code>
\leftharpoonup	<code>\leftharpoonup</code>	\rightharpoonup	<code>\rightharpoonup</code>	\twoheaduparrow	<code>\twoheaduparrow</code>
\leftleftarrows	<code>\leftleftarrows</code>	\rightrightarrows	<code>\rightrightarrows</code>	\upharpoonleft	<code>\upharpoonleft</code>
\looparrowleft	<code>\looparrowleft</code>	\looparrowright	<code>\looparrowright</code>	\upharpoonright	<code>\upharpoonright</code>
\swarrow	<code>\swarrow</code>	\searrow	<code>\searrow</code>	\downarrow	<code>\downarrow</code>
\nwarrow	<code>\nwarrow</code>	\nearrow	<code>\nearrow</code>	\Downarrow	<code>\Downarrow</code>
\leftsquigarrow	<code>\leftsquigarrow</code>	\leadsto , \rightsquigarrow	<code>\leadsto</code> , <code>\rightsquigarrow</code>	\downdownarrows	<code>\downdownarrows</code>
\iff	<code>\iff</code>	\twoheadrightarrow	<code>\twoheadrightarrow</code>	\downharpoonleft	<code>\downharpoonleft</code>
\implies	<code>\implies</code>			\downharpoonright	<code>\downharpoonright</code>

Puntos y comillas

\because	<code>\because</code>	\cdot	<code>\cdot</code>	\cdot	<code>\cdot</code>
\cdots	<code>\cdots</code>	\cdot	<code>\cdot</code>	\colon	<code>\colon</code>
\ddots	<code>\ddots</code>	\dots	<code>\dots</code>	\cdot	<code>\cdot</code>
\ldots	<code>\ldots</code>	ellipses	<code>\text{ellipses}</code>	\therefore	<code>\therefore</code>
\vdots	<code>\vdots</code>	quotedblbase	<code>\text{quotedblbase}</code>	quotedbl	<code>\text{quotedbl}</code>

Símbolos de uso principalmente matemático:

\aleph	<code>\aleph</code>	\amalg	<code>\amalg</code>	\angle	<code>\angle</code>
\approx	<code>\approx</code>	\approxeq	<code>\approxeq</code>	\asymp	<code>\asymp</code>
\backsimeq	<code>\backsimeq</code>	\backslash	<code>\backslash</code>	\barwedge	<code>\barwedge</code>
\between	<code>\between</code>	\bigcap	<code>\bigcap</code>	\bigcup	<code>\bigcup</code>
\bigsqcup	<code>\bigsqcup</code>	\biguplus	<code>\biguplus</code>	\bigvee	<code>\bigvee</code>
\bigwedge	<code>\bigwedge</code>	\bot	<code>\bot</code>	\bowtie	<code>\bowtie</code>
\bumpeq	<code>\bumpeq</code>	\cap	<code>\cap</code>	\Cap	<code>\Cap</code>
\circeq	<code>\circeq</code>	\complement	<code>\complement</code>	\cong	<code>\cong</code>
\coprod	<code>\coprod</code>	\cup	<code>\cup</code>	\Cup	<code>\Cup</code>
\curlyeqprec	<code>\curlyeqprec</code>	\curlyeqsucc	<code>\curlyeqsucc</code>	\curlyvee	<code>\curlyvee</code>
\curlywedge	<code>\curlywedge</code>	\dashv	<code>\dashv</code>	\dagger	<code>\dagger, \dag</code>
\ddagger	<code>\ddagger, ddag</code>	\diamondsuit	<code>\diamondsuit</code>	\div	<code>\div</code>
\divideontimes	<code>\divideontimes</code>	\doteq	<code>\doteq</code>	\doteqdot	<code>\doteqdot</code>
\dotplus	<code>\dotplus</code>	ℓ	<code>\ell</code>	\emptyset	<code>\emptyset</code>
\eqcirc	<code>\eqcirc</code>	\eqslantgtr	<code>\eqslantgtr</code>	\eqslantless	<code>\eqslantless</code>
\equiv	<code>\equiv</code>	\eth	<code>\eth</code>	\exists	<code>\exists</code>
$\exists!$	<code>\exists!</code>	\fallingdotseq	<code>\fallingdotseq</code>	\flat	<code>\flat</code>
\forall	<code>\forall</code>	\frown	<code>\frown</code>	\geq	<code>\geq, \ge</code>
\geqslant	<code>\geqslant</code>	\gg	<code>\gg</code>	\ggg	<code>\ggg</code>
\gapprox	<code>\gapprox</code>	\gneq	<code>\gneq</code>	\gnsim	<code>\gnsim</code>
\gtrapprox	<code>\gtrapprox</code>	\gtrdot	<code>\gtrdot</code>	\gtreqless	<code>\gtreqless</code>
\gtreqqless	<code>\gtreqqless</code>	\gtrless	<code>\gtrless</code>	\gtrsim	<code>\gtrsim</code>
\hbar	<code>\hbar</code>	\heartsuit	<code>\heartsuit</code>	\hslash	<code>\hslash</code>
\iiint	<code>\iiint</code>	\Im	<code>\Im</code>	\imath	<code>\imath</code>
\in	<code>\in</code>	∞	<code>\infty</code>	\int	<code>\int</code>
\intercal	<code>\intercal</code>	\jmath	<code>\jmath</code>	\land	<code>\land</code>
\leftthreetimes	<code>\leftthreetimes</code>	\leq	<code>\leq, \le</code>	\leqq	<code>\leqq</code>
\leqslant	<code>\leqslant</code>	\lessapprox	<code>\lessapprox</code>	\lessdot	<code>\lessdot</code>
\lesseqgtr	<code>\lesseqgtr</code>	\lesseqqgtr	<code>\lesseqqgtr</code>	\lessgtr	<code>\lessgtr</code>
\lessssim	<code>\lessssim</code>	\ll	<code>\ll</code>	\lll	<code>\lll</code>
\lnapprox	<code>\lnapprox</code>	\lneq	<code>\lneq</code>	\lneqq	<code>\lneqq</code>
\lnsim	<code>\lnsim</code>	\lor	<code>\lor</code>	\ltimes	<code>\ltimes</code>
\measuredangle	<code>\measuredangle</code>	\models	<code>\models</code>	\mp	<code>\mp</code>
\multimap	<code>\multimap</code>	\nVDash	<code>\nVDash</code>	∇	<code>\nabla</code>
\natural	<code>\natural</code>	\ncong	<code>\ncong</code>	\neq	<code>\neq</code>
$\neg \circ \neg$	<code>\neg \circ \neg</code>	\nexists	<code>\nexists</code>	\ngeq	<code>\ngeq</code>
\ngtr	<code>\ngtr</code>	\ni	<code>\ni</code>	\nleq	<code>\nleq</code>
\nless	<code>\nless</code>	\nmid	<code>\nmid</code>	$\not\approx$	<code>\not\approx</code>
$\not\equiv$	<code>\not\equiv</code>	$\not\sim$	<code>\not\sim</code>	$\not\sim$	<code>\not\sim</code>
\notin	<code>\notin</code>	\nparallel	<code>\nparallel</code>	\nprec	<code>\nprec</code>
\nsim	<code>\nsim</code>	\nsubseteq	<code>\nsubseteq</code>	\nsucc	<code>\nsucc</code>
\nsubseteq	<code>\nsubseteq</code>	\ntriangleleft	<code>\ntriangleleft</code>	\ntrianglelefteq	<code>\ntrianglelefteq</code>
\ntriangleright	<code>\ntriangleright</code>	\ntrianglerighteq	<code>\ntrianglerighteq</code>	\nvdash	<code>\nvdash</code>
\nvDash	<code>\nvDash</code>	\oint	<code>\oint</code>	\parallel	<code>\parallel</code>
∂	<code>\partial</code>	\perp	<code>\perp</code>	\permil	<code>\permil</code>
\pm	<code>\pm</code>	\prec	<code>\prec</code>	\preccurlyeq	<code>\preccurlyeq</code>
\preceq	<code>\preceq</code>	\precnsim	<code>\precnsim</code>	\precsim	<code>\precsim</code>
\prime	<code>\prime</code>	\prod	<code>\prod</code>	\propto	<code>\propto</code>
\Re	<code>\Re</code>	\rightthreetimes	<code>\rightthreetimes</code>	\risingdotseq	<code>\risingdotseq</code>

\rtimes	<code>\rtimes</code>	\sharp	<code>\sharp</code>	\sim	<code>\sim</code>
\simeq	<code>\simeq</code>	\smile	<code>\smile</code>	\sphericalangle	<code>\sphericalangle</code>
\sqcap	<code>\sqcap</code>	\sqcup	<code>\sqcup</code>	\sqsubset	<code>\sqsubset</code>
\sqsubseteq	<code>\sqsubseteq</code>	\sqsupset	<code>\sqsupset</code>	\sqsupseteq	<code>\sqsupseteq</code>
\subset	<code>\subset</code>	\Subset	<code>\Subset</code>	\subseteq	<code>\subseteq</code>
\subsetneq	<code>\subsetneq</code>	\succ	<code>\succ</code>	\succcurlyeq	<code>\succcurlyeq</code>
\succeq	<code>\succeq</code>	\succsim	<code>\succsim</code>	\succsim	<code>\succsim</code>
\sum	<code>\sum</code>	\supset	<code>\supset</code>	\Supset	<code>\Supset</code>
\supseteq	<code>\supseteq</code>	\supsetneq	<code>\supsetneq</code>	\surd	<code>\surd</code>
tpm	<code>\text{tpm}</code>	\times	<code>\times</code>	\top	<code>\top</code>
\triangle	<code>\triangle</code>	\uplus	<code>\uplus</code>	\vDash	<code>\vDash</code>
\Vdash	<code>\Vdash</code>	\vee	<code>\vee</code> o <code>\lor</code>	\veebar	<code>\veebar</code>
\Vvert	<code>\Vvert</code>	\Vvdash	<code>\Vvdash</code>	\wedge	<code>\wedge</code> o <code>\land</code>
\wp	<code>\wp</code>	\wr	<code>\wr</code>		

Otros símbolos

\P	<code>\P</code>	\S	<code>\S</code>	$^{\circ}\text{C}$	<code>\celsius</code>
\checkmark	<code>\checkmark</code>	\mho	<code>\mho</code>	Ω	<code>\ohm</code>
$^{\circ}$	<code>\textdegree</code>	N°	<code>\textnumero</code>	visiblespace	<code>\textvisiblespace</code>

Apéndice C

Índice de comandos

En el presente índice se recogen los comandos de los que se habla en esta introducción. Algunos simplemente se mencionan, casi de pasada, en cuyo caso la página que aparece en el índice indica el lugar en donde son mencionados. Pero otros comandos son objeto de cierta explicación más detallada. En tal caso en el índice sólo se recoge el lugar en el que empieza la explicación detallada, aunque, tal vez, el comando sea citado también en otros lugares de la introducción.

No se incluyen en el índice:

- Los comandos `\stopLoQueSea` que cierran una construcción que previamente se abrió con `\startLoQueSea`, salvo que en el texto se diga algo especial sobre el comando `\stop`, o se le trate en un lugar diferente a aquel en el que se trató el correspondiente comando `\start`.
- Los comandos dirigidos a generar símbolos, que se recogen en el [apéndice B](#).
- Tratándose de comandos que generan un diacrítico o letra, y que tienen una versión en mayúscula y otra en minúscula, para generar, respectivamente, la letra mayúscula o la minúscula, sólo se recoge la versión con letra minúscula.

a	<code>\aring</code> 197
<code>\aa</code> 197	<code>\at</code> 182
<code>\aacute</code> 196	<code>\atilde</code> 196
<code>\about</code> 182	<code>\atleftmargin</code> 112
<code>\abreve</code> 196	<code>\atpage</code> 185
<code>\acircumflex</code> 196	<code>\atrighmargin</code> 112
<code>\adaptlayout</code> 101	
<code>\adaptpapersize</code> 95	b
<code>\adiaeresis</code> 196	<code>\backslash</code> 48
<code>\ae</code> 198	<code>\ </code> 233
<code>\aeligature</code> 198	<code>\begingroup</code> 65, 66
<code>\agrave</code> 196	<code>\beta</code> 199
<code>\alpha</code> 199	<code>\bf</code> 123
<code>\amacron</code> 196	<code>\bfa</code> 124

- `\bfb` 124
- `\bfc` 124
- `\bfd` 124
- `\bfx` 124
- `\bfxx` 124
- `\bgroup` 65, 66
- `\bi` 123
- `\bia` 124
- `\bib` 124
- `\bic` 124
- `\bid` 124
- `\bigbodyfont` 126
- `\bix` 124
- `\bixx` 124
- `\blank` 77, 226
- `\bold` 123
- `\bolditalic` 123
- `\boldslanted` 123
- `\break` 233
- `\bs` 123
- `\bsa` 124
- `\bsb` 124
- `\bsc` 124
- `\bsd` 124
- `\bsx` 124
- `\bsxx` 124
- `\buildmathaccent` 201
- `\buildtextaccent` 201
- `\buildtextbootomcomma` 201
- `\buildtextbottomdot` 201
- `\buildtextcedilla` 201
- `\buildtextgrave` 201
- `\buildtextmacron` 201
- `\buildtexttognek` 201
- c**
- `\Cap` 205
- `\c` 197
- `\ca` 214
- `\calligraphic` 123
- `\cap` 205
- caracteres reservados
 - `\{` 48
 - `\}` 48
 - `\$` 48
 - `\backslash` 48
 - `\letterhat` 48
 - `\lettertilde` 48
 - `\#` 48
 - `\%` 48
 - `\&` 48
 - `_` 48
 - `\|` 48
 - `\ccedilla` 197
 - `\cf` 123
 - `\chapter` 138
 - `\chi` 199
 - `\clip` 283
 - `\clubpenalty` 241
 - `\color` 131
 - `\colored` 131
 - `\completecontent` 157
 - `\completelist` 168
 - `\completelistofchemicals` 169
 - `\completelistoffigures` 169, 280
 - `\completelistofgraphics` 169
 - `\completelistofintermezzi` 169
 - `\completelistoftables` 169, 285
 - `\compleindex` 174
 - `\crlf` 233
 - `\currentdate` 218
 - d**
 - `\date` 218
 - `\de` 215
 - `\define` 61
 - `\definealternativestyle` 128
 - `\defineblank` 227
 - `\definebodyfontenvironment` 125
 - `\definebodyfontswitch` 127
 - `\definecapitals` 206
 - `\definecharacter` 199
 - `\definecharacterkerning` 211
 - `\definecolor` 134
 - `\definecombination` 296
 - `\definecombinedlist` 170

`\defineconversionset` 105
`\definedelimitedtext` 220
`\definedescription` 264
`\defineenumeration` 266
`\definefloat` 297
`\definefontfeature` 198
`\definefontstyle` 127
`\defineframed` 271
`\defineframedtext` 271
`\definehead` 152
`\defineinterlinespace` 235
`\defineitemgroup` 264
`\defineitems` 264
`\definelayou` 101
`\definelinenumbering` 238
`\definelines` 236
`\definelist` 168
`\definemargindata` 114
`\definenarrower` 224
`\definernote` 246
`\definepapersize` 94
`\defineparagraphs` 254
`\defineregister` 175
`\definesectionblock` 154
`\definestartstop` 63
`\definestretched` 211
`\definesymbol` 258
`\definetext` 111
`\definetype` 208
`\definotyping` 208
`\delta` 199
`\dontleavehmode` 123

e

`\eacute` 196
`\ebreve` 196
`\ecircumflex` 196
`\ediaeresis` 196
`\egrave` 196
`\egroup` 65, 66
`\em` 128
`\emacron` 196
`\emdash` 78

`\en` 214
`\enableregime` 74
`\endash` 78
`\endgraf` 221
`\endgroup` 66
`\endnote` 244
`\enskip` 211
`\environment` 84
`\epsilon` 199
`\es` 214
`\eta` 199
`\etilde` 196
`\externalfigure` 276

f

`\fillinline` 269
`\footnote` 244
`\fr` 214
`\framed` 271
`\from` 191

g

`\gamma` 199
`\getbuffer` 271
`\getmarking` 110
`\godown` 227
`\goto` 191

h

`\H` 197
`\HL` 289
`\hairline` 268
`\handwritten` 123
`\hbox` 231
`\head` 259
`\hfill` 212
`\high` 207
`\hl` 268
`\hskip` 212
`\hw` 123
`\hyphen` 78
`\hyphenatedurl` 190
`\hyphenatedurlseparator` 191

`\hyphenation` 231

i

`\i` 197
`\iacute` 196
`\ibreve` 196
`\icircumflex` 196
`\idiaeresis` 196
`\igrave` 196
`\imacron` 196
`\in` 182
`\index` 172
`\inframed` 271
`\inner` 112
`\ininneredge` 112
`\innermargin` 112
`\inleft` 112
`\inleftedge` 112
`\inleftmargin` 112
`\inmargin` 112
`\inothet` 112
`\inouter` 112
`\inouteredge` 112
`\inoutermargin` 112
`\input` 81
`\inright` 112
`\inrightedge` 112
`\inrightmargin` 112
`\iota` 199
`\it` 123
`\ita` 124
`\italic` 123
`\italicbold` 123
`\itb` 124
`\itc` 124
`\itd` 124
`\item` 259
`\items` 263
`\itilde` 196
`\its` 260
`\itx` 124
`\itxx` 124

j

`\j` 197

k

`\kappa` 199
`\kcedilla` 197

l

`\l` 197
`\labeltext` 218
`\lambda` 199
`\language` 214
`\lastpagenumber` 106
`\lastrealpagenumber` 106
`\lastuserpagenumber` 106
`\lcedilla` 197
`\leftaligned` 240
`\letterbackslash` 191
`\letterescape` 191
`\letterhash` 191
`\letterhat` 48
`\letterpercent` 191
`\lettertilde` 48
`\linenote` 244
`\loadinstalledlanguages` 214
`\lohi` 207
`\low` 207

m

`\mainlanguage` 214
`\mar` 260
`\margintext` 112
`\mediaeval` 123
`\midaligned` 240
`\minus` 78
`\mirror` 283
`\mono` 123
`\month` 218
`\mu` 199

n

`\NB` 290

`\NC` 289
`\NN` 290
`\NR` 289
`\ncedilla` 197
`\noheaderandfooterlines` 110
`\noindentation` 223
`\nolist` 140, 144
`\nomarking` 140, 144
`\normal` 123
`\note` 245
`\notesenabledfalse` 251
`\notesenabledtrue` 251
`\nowhitespace` 225
`\nu` 199

o

`\o` 197
`\oacute` 196
`\obreve` 196
`\ocircumflex` 196
`\odiaeresis` 196
`\oe` 198
`\oeligature` 198
`\ograve` 196
`\omacron` 196
`\omega` 199
`\omicron` 199
`\os` 123
`\otilde` 196
`\overbar` 270
`\overbars` 270
`\overstrike` 270
`\overstrikes` 270

p

`\page` 106
`\pagenumber` 106, 110
`\pagereference` 181
`\par` 221
`\parindent` 69
`\parskip` 69
`\part` 138
`\phi` 199

`\pi` 199
`\placebookmarks` 192
`\placechemical` 291
`\placecontent` 157
`\placefigure` 278
`\placefloat` 291
`\placegraphic` 291
`\placeindex` 174
`\placeintermezzo` 291
`\placelist` 168
`\placelistofchemicals` 169
`\placelistoffigures` 169, 280
`\placelistofgraphics` 169
`\placelistofintermezzi` 169
`\placelistoftables` 169, 285
`\placelocalfootnotes` 246
`\placenotes` 246
`\placetable` 285
`\pretolerance` 232
`\product` 85
`\project` 87
`\psi` 199

q

`\qqquad` 211
`\quad` 211
`\quotation` 220
`\quote` 220

r

`\ReadFile` 83
`\r` 197
`\ran` 260
`\rcedilla` 197
`\readfile` 83
`\realpagenumber` 106
`\ref` 183
`\reference` 181
`\regular` 123
`\rho` 199
`\rightaligned` 240
`\rm` 123
`\rma` 124

- `\rmb` 124
- `\rmc` 124
- `\rmd` 124
- `\rmx` 124
- `\rmxx` 124
- `\roman` 123
- `\rotate` 283

- s**
- `\sans` 123
- `\sansserif` 123
- `\sc` 123
- `\scedilla` 197
- `\section` 138
- `\seeindex` 174
- `\serif` 123
- `\sethyphenatedurlafter` 190
- `\sethyphenatedurlbefore` 190
- `\sethyphenatedurlnormal` 190
- `\setupalign` 238
- `\setuparranging` 100
- `\setupbackgrounds` 131
- `\setupblank` 227
- `\setupbodyfont` 119, 122
- `\setupbottomtexts` 112
- `\setupcapitals` 206
- `\setupcaption` 294
- `\setupcaptions` 294
- `\setupcharacterkerning` 211
- `\setupcolors` 131
- `\setupcolumns` 253
- `\setupcombinedlist` 158
- `\setupcounter` 247
- `\setupdescription` 265
- `\setupendnotes` 248
- `\setupenumeration` 267
- `\setupexternalfigures` 281
- `\setupfillinlines` 269
- `\setupfloat` 296
- `\setupfloats` 296
- `\setupfooter` 110
- `\setupfootertexts` 108, 111
- `\setupfootnotes` 248
- `\setupframed` 271
- `\setupframedtext` 271
- `\setuphead` 141
- `\setupheader` 110
- `\setupheadertexts` 108, 111
- `\setupheadnumber` 145
- `\setupheads` 141
- `\setupheadtext` 158
- `\setuphyphenmark` 213
- `\setupindenting` 222
- `\setupinteraction` 187
- `\setupinterlinespace` 234
- `\setuplabeltext` 217
- `\setuplanguage` 216
- `\setuplayout` 99
- `\setuplinenote` 244
- `\setuplinenumbering` 237
- `\setuplines` 236
- `\setuplist` 161
- `\setupmargindata` 113
- `\setupnarrower` 224
- `\setupnotation` 247
- `\setupnotations` 247
- `\setupnote` 247
- `\setupnotes` 247
- `\setuppagenumbering` 103
- `\setuppapersize` 91
- `\setupparagraphs` 254
- `\setupregister` 174, 175
- `\setupsectionblock` 154
- `\setupspacing` 210
- `\setupstretched` 211
- `\setuptables` 285
- `\setuptextrule` 270
- `\setuptolerance` 232, 240
- `\setuptoptexts` 112
- `\setuptype` 208
- `\setuptyping` 208
- `\setupurl` 190
- `\setupuserpagenumber` 103
- `\setupwhitespace` 224
- `\showbodyfont` 121
- `\showbodyfontenvironment` 126

<code>\showcolor</code>	133	<code>\startcolumns</code>	252
<code>\showcolorcomponents</code>	133	<code>\startcombination</code>	272, 295
<code>\showfont</code>	122	<code>\startcomponent</code>	85
<code>\showframe</code>	98	<code>\startenvironment</code>	84
<code>\showlayout</code>	98	<code>\startfiguretext</code>	281
<code>\showsetups</code>	98	<code>\startformula</code>	272
<code>\showsymbolset</code>	203	<code>\startframedtext</code>	271
<code>\sigma</code>	199	<code>\startfrontmatter</code>	153
<code>\sl</code>	123	<code>\starthiding</code>	272
<code>\sla</code>	124	<code>\startitem</code>	259
<code>\slanted</code>	123	<code>\startitemize</code>	258
<code>\slantedbold</code>	123	<code>\startlegend</code>	272
<code>\slb</code>	124	<code>\startlinecorrection</code>	272
<code>\slc</code>	124	<code>\startlinenumbers</code>	237
<code>\sld</code>	124	<code>\startlines</code>	236
<code>\slx</code>	124	<code>\startlocalfootnotes</code>	246
<code>\slxx</code>	124	<code>\startMPpage</code>	95
<code>\smalcaps</code>	123	<code>\startmode</code>	272
<code>\smallbodyfont</code>	126	<code>\startnarrower</code>	223
<code>\smallbold</code>	126	<code>\startnotmode</code>	272
<code>\smallbolditalic</code>	126	<code>\startopposite</code>	272
<code>\smallboldslanted</code>	126	<code>\startpacked</code>	225
<code>\smallitalicbold</code>	126	<code>\startpagefigure</code>	95
<code>\smallslanted</code>	126	<code>\startpart</code>	138
<code>\smallslantedbold</code>	126	<code>\startproduct</code>	85
<code>\somewhere</code>	184	<code>\startproject</code>	87
<code>\space</code>	211	<code>\startquotation</code>	273
<code>\ss</code>	123, 197	<code>\startsection</code>	138
<code>\ssa</code>	124	<code>\startsetups</code>	64
<code>\ssb</code>	124	<code>\startstandardmakeup</code>	273
<code>\ssc</code>	124	<code>\startsubject</code>	138
<code>\ssd</code>	124	<code>\startsubsection</code>	138
<code>\ssx</code>	124	<code>\startsubsubsection</code>	138
<code>\ssxx</code>	124	<code>\startsubsubsubject</code>	138
<code>\start</code>	65	<code>\startsubsubsubsection</code>	138
<code>\startalignment</code>	240	<code>\startsubsubsubsubject</code>	138
<code>\startappendices</code>	153	<code>\startTEXpage</code>	95
<code>\startbackmatter</code>	153	<code>\starttabulate</code>	286
<code>\startbodymatter</code>	153	<code>\starttext</code>	79
<code>\startbuffer</code>	271	<code>\starttextrule</code>	270
<code>\startchapter</code>	138	<code>\starttitle</code>	138
<code>\startchemical</code>	272	<code>\starttyping</code>	207
<code>\startcolor</code>	133	<code>\stop</code>	66

- `\stoptext` 79
- `\stretched` 210
- `\structureuservariable` 141
- `\sub` 260
- `\subject` 138
- `\subsection` 138
- `\subsubject` 138
- `\subsubsection` 138
- `\subsubsubject` 138
- `\subsubsubsection` 138
- `\subsubsubsubject` 138
- `\support` 123
- `\swhoinstalledlanguages` 214
- `\switchtobodyfont` 122
- `\sym` 260
- `\symbol` 203
- t**
- `\TB` 290
- `\TeX` 20, 22
- `\tau` 199
- `\tcedilla` 197
- `\teletype` 123
- `\tex` 208
- `\textheight` 100
- `\textreference` 181
- `\textrule` 270
- `\sq` 211
- `\textwidth` 100
- `\tf` 123
- `\tfa` 124
- `\tfb` 124
- `\tfc` 124
- `\tfd` 124
- `\tfx` 124
- `\tfxx` 124
- `\theta` 199
- `\thinrule` 268
- `\thinrules` 268
- `\title` 138
- `\tolerance` 232
- `\translate` 219
- `\tt` 123
- `\tta` 124
- `\ttb` 124
- `\ttc` 124
- `\ttd` 124
- `\ttx` 124
- `\ttxx` 124
- `\tx` 124
- `\txx` 124
- `\typ` 208
- `\type` 207
- u**
- `\u` 196
- `\uacute` 196
- `\ubreve` 196
- `\ucircumflex` 196
- `\udiaeresis` 196
- `\ugrave` 196
- `\umacron` 196
- `\underbar` 270
- `\underbars` 270
- `\unhyphenated` 231
- `\upsilon` 199
- `\usecolors` 132
- `\useexternalfigure` 276
- `\usemodule` 214
- `\useregime` 75
- `\userpagenumber` 106
- `\usesymbols` 203
- `\useURL` 189
- `\utilde` 196
- v**
- `\VL` 289
- `\varepsilon` 199
- `\varkappa` 199
- `\varphi` 199
- `\varpi` 199
- `\varrho` 199
- `\varsigma` 199
- `\vartheta` 199
- `\vbox` 107
- `\vfill` 228

`\vl` 268
`\vskip` 227

w

`\WORD` 205
`\WORDS` 205
`\Word` 205
`\Words` 205
`\whitespace` 225
`\widowpenalty` 241

`\word` 205
`\wordright` 240
`\writebetweenlist` 166
`\writetolist` 165

x

`\xi` 199

z

`\zeta` 199