

Не баш кратак увод у ConT_EXt Mark IV

Не баш кратак увод у ConT_EXt Mark IV

Верзија 1.6 [2. јануар 2021]

© 2020-2021, Joaquín Ataz-López

Наслов оригинала: Una introducción (no demasiado breve) a ConT_EXt Mark IV

Превод на српски: добар пријатељ који жели да остане анониман.

Аутор овог текста (и његов преводилац на енглески) дозвољава слободну дистрибуцију и употребу, укључујући и право на умножавање и редистрибуцију садржаја овог документа у дигиталном облику под условима да се потврђује ауторство и да се не укључује у било који софтверски пакет или скуп програма, или у документацију чији услови употребе или дистрибуције не укључују слободно право прималаца да је копирају и дистрибуирају. Исто тако, даје се дозвола за преводње овог документа, под условом да се наведе ауторство оригиналног документа и да се преведени текст дистрибуира под FDL лиценцом коју објављује *Free Software Foundation*, *Creative Commons* лиценцом која дозвољава копирање и редистрибуцију, или неком сличном лиценцом.

Уз све ово, издавање или оглашавање или преводње овог документа у папирном облику захтева писмену личну ауторову дозволу.

Историја верзија:

- 18. август 2020: Верзија 1.0 (само на шпанском): оригинални документ.
- 23. август 2020: Верзија 1.1 (само на шпанском): исправка мањих грешака, ауторових грешака у куцању и неразумевања.
- 3. септембар 2020: Верзија 1.15 (само на шпанском): још грешака, грешака у куцању и неразумевања.
- 5. септембар 2020: Верзија 1.16 (само на шпанском): још грешака, грешака у куцању и неразумевања, као и неке врло ситне измене које (надам се) чине текст јаснијим.
- 6. септембар 2020: Верзија 1.17 (само на шпанском): број ситних грешака које проналазим је невероватан. Једноставно би требало да престанем са поновним читањем ако желим да их више не проналазим!
- 21. октобар 2020: Верзија 1.5 (само на шпанском): унос сугестија и исправки грешака које су пријавили корисници NTG-context мејлинг листе.
- 2. јануар 2021: Верзија 1.6: исправке које су предложене након пажљивог читања документа, приликом његовог преводјења на енглески језик. Ово је прва верзија на енглеском.

Садржај

Предговор	6
I Шта је ConTeXt и како се користи	12
1 ConTeXt: општи преглед	13
1.1 Па шта је уопште ConTeXt?	13
1.2 Словослагање текстова	14
1.3 Језици за означавање	15
1.4 T _E X и језици изведени из њега	16
1.5 ConTeXt	19
2 Наш први изворни фајл	26
2.1 Припрема експеримента: неопходни алати	26
2.2 Сам експеримент	28
2.3 Структура фајла нашег примера	32
2.4 Неки додатни детаљи у вези са покретањем команде „context”	32
2.5 Управљање грешкама	33
3 Команде и остали фундаментални концепти система ConTeXt	36
3.1 ConTeXt резервисани карактери	36
3.2 Саме команде	39
3.3 Опсег важења команди	42
3.4 Опције рада команде	44
3.5 Резиме синтаксе команде и опција, употреба великих и витичастих заграда приликом позивања	47
3.6 Званична листа ConTeXt команди	48
3.7 Дефинисање нових команди	49
3.8 Остали основни концепти	52
3.9 Метода за самостално учење система ConTeXt	56
4 Изворни фајлови и пројекти	58
4.1 Кодирање изворних фајлова	58
4.2 Карактери у изворном фајлу које ConTeXt третира на посебан начин	60
4.3 Прости пројекти и пројекти са више фајлова	63
4.4 Структура изворног фајла у простим пројектима	63
4.5 Рад са више фајлова у T _E X стилу	65
4.6 ConTeXt пројекти у ужем смислу	67

II	Глобални аспекти документа	72
5	Странице и пагинација документа	73
5.1	Величина странице	73
5.2	Елементи на страници	77
5.3	Распоред странице (<code>\setuplayout</code>)	79
5.4	Нумерација страница	83
5.5	Форсирани или предложени преломи странице	85
5.6	Заглавља и подножја	87
5.7	Уметање текст елемената у ивице и маргине странице	90
6	Фонтови и боје у систему ConTeXt	92
6.1	Типографски фонтови укључени у „ConTeXt Standalone”	92
6.2	Особине фонта	93
6.3	Постављање главног фонта документа	95
6.4	Промена неких особина фонта	97
6.5	Остале ствари везане за употребу неких алтернатива	102
6.6	Употреба и конфигурација боја	104
7	Структура документа	109
7.1	Структурна подела у документима	109
7.2	Типови одељака и њихова хијерархија	110
7.3	Заједничка синтакса команди поделе	111
7.4	Формат и конфигурација одељака и њихових наслова	113
7.5	Дефинисање нових команди поделе	121
7.6	Макроструктура документа	122
8	Садржаји, индекси, листе	124
8.1	Садржај	124
8.2	Листе, комбиноване листе и садржаји базирани на листи	133
8.3	Индекс	136
9	Референце и хиперлинкови	141
9.1	Типови референци	141
9.2	Интерне референце	142
9.3	Интерактивни електронски документи	148
9.4	Хиперлинкови на спољашње документе	150
9.5	Креирање маркера у финалном PDF фајлу	153
III	Специфична питања	155
10	Карактери, речи, текст и хоризонтални размак	156
10.1	Добијање карактера којима нормално не може да се приступи са тастатуре	156

10.2	Специјално форматирање карактера	164
10.3	Размак између карактера и речи	167
10.4	Сложенице	170
10.5	Језик текста	171
11	Пасуси, линије и вертикални размак	178
11.1	Пасуси и њихове карактеристике	178
11.2	Вертикални размак између пасуса	180
11.3	Како ConTeXt изграђује линије које формирају пасусе	184
11.4	Проред	188
11.5	Остале ствари у вези са линијама	189
11.6	Хоризонтално и вертикално поравнање	191
12	Специјалне конструкције и пасуси	195
12.1	Фусноте и белешке на крају	195
12.2	Пасуси у више колона	202
12.3	Уређене листе	206
12.4	Описи и набрајања	212
12.5	Линије и оквири	215
12.6	Остала интересантна окружења и конструкције	218
13	Слике, табеле и остали плутајући објекти	220
13.1	Шта су плутајући објекти и шта они раде?	220
13.2	Спољне слике	221
13.3	Табеле	227
13.4	Заједнички аспекти слика, табела и осталих плутајућих објеката	233
13.5	Дефинисање додатних плутајућих објеката	238
Додаци		240
A	Инсталација, конфигурација и ажурирање система ConTeXt	241
1	Инсталирање и конфигурисање „ConTeXt Standalone”	241
2	Инсталација LMTX	246
3	Употреба неколико верзија система ConTeXt на истом систему (само за Unix системе)	249
B	Команде за генерисање математичких и осталих симбола	250
B	Индекс команди	253

Предговор*

Поштовани читаоче, овај документ описује ConTeXt, словослагачки систем изведен из система T_EX, такође словослагачког система, који је креирао ДОНАЛД Е. КНУТ са универзитета Стенфорд између 1977. и 1982. године.

ConTeXt је осмишљен за израду докумената врло високог типографског квалитета – било папирних докумената или оних предвиђених да се приказују на екрану рачунара. То није текст процесор или текст едитор, већ, као што сам раније поменуо, *сисџем*, или другим речима скуп алата намењен за словослагање докумената, то јест графички распоред и визуелизацију различитих елемената документа на страници или на екрану. Укратко, ConTeXt тежи да обезбеди све алате неопходне да документи добију најбољи могући изглед. Идеја је да budete у могућности да генеришете документе који, осим тога што су лепо написани, такође и лепо „изгледају”. У том смислу, овде можемо поменути шта је ДОНАЛД Е. КНУТ написао када је представљао T_EX (систем на којем се заснива ConTeXt):

Ако само желиш да направиш документ довољно добар да прође—нешто прихватљиво и у основи читљиво, али не и заиста лепо—обично ће вам бити довољан једноставнији сисџем. Циљ сисџема T_EX је да произведе најфинији квалитет; то захтева више пажње на дејалве, а ипак вам неће бити много теже да дођеш до тог циља, а биће и посебно поносни на завршени производ.

Када припремамо рукопис системом ConTeXt, ми прецизно наводимо како он треба да се трансформише у странице (или екране) чији је типографски квалитет такав да се може упоредити са оним што могу да произведу најбоље штампарије на свету. Једном када научимо систем, ово постижемо само са мало више рада у односу на рад потребан да се документ откуца у било ком текст процесору или текст едитору. Уствари, када једном достигнумо одговарајућу лакоћу у руковању системом ConTeXt, укупан посао је вероватно мањи ако имамо на уму да се у систему ConTeXt већина детаља форматирања документа описује глобално и да се ради са текст фајловима који су – једном када се навикнемо на њих – много природнији начин за рад на креирању и уређивању докумената; а чињеница је и да је ова врста фајлова много мања и једноставнија за обраду од комплексних бинарних фајлова које користе текст процесори.

Постоји поприлична количина документације о систему ConTeXt, и скоро сва је на енглеском језику. Оно што бисмо могли сматрати да је *званична* дистрибуција система ConTeXt – под називом „ConTeXt Standalone”² – на пример, садржи неких 180 PDF фајлова документације (од којих је већина на енглеском језику, мада су неки и на холандском и немачком)

* Писање овог предговора је започето са намером да буде превод/адаптација на ConTeXt предговора књиге „The T_EXBook”, документа који објашњава *све што је потребно да знаш о програму T_EX*. На крају сам морао да одступим од тога; ипак, задржао сам неке фрагменте тог предговора који ће онима који га познају, надам се, пружити неке његове *одјекве*.

² У време када је писана прва верзија овог текста, оно што је овде речено је било поткрепљено чињеницама; али у пролеће 2020. године, ConTeXt вики је ажуриран и од тада морамо да претпоставимо да је „званична” дистрибуција система ConTeXt постала LMTX. Ипак, за оне који по први пут улазе у ConTeXt свет, још увек бих препоручио да користе „ConTeXt Standalone” јер је то стабилнија дистрибуција. [Додатак А](#) објашњава како да инсталирате било коју од ових дистрибуција.

укључујући упутства, примере и техничке чланке; а на Pragma ADE веб сајту (компанија која је изнедрила ConTeXt) постоји (на дан када сам пребројао у мају 2020. године) 224 документа које можете преузети, од којих се већина дистрибуира уз „ConTeXt Standalone”, али и неких других. Свакако, ова огромна документација није заиста од користи за учење система ConTeXt јер, у општем случају, ови документи нису намењени читаоцу који не зна ништа о систему, али жели да га научи. Од 56 PDF фајлова које „ConTeXt Standalone” назива „упутства”, постоји само један који претпоставља да читалац не зна ништа о систему ConTeXt. То је документ под именом „ConTeXt Mark IV, an Excursion”. Међутим, овај документ, као што му само име говори, ограничава свој садржај на представљање система и објашњавање начина на који се раде одређене ствари које могу да се ураде системом ConTeXt. Он би био добар увод ако би се наставио мало боље уређеним и систематским референтним приручником. Такво упутство не постоји, па је празнина између ConTeXt „Excursion” и остатка документације сувише велика.

2001. године је написан референтни приручник који може да се пронађе на [Pragma ADE веб сајту](#); али упркос наслову, с једне стране он није дизајниран тако да буде *комплетно уџучиштво*, док је с друге стране био (и јесте) текст намењен претходној верзији система ConTeXt (под називом Mark II), па је стога прилично застарео.

Упутство је 2013. године делимично ажурирано, али многи његови одељци нису поново написани, тако да садржи информације које се тичу и система ConTeXt Mark II и ConTeXt Mark IV (текуће верзије), без потпуно јасног назначавња која информације се односи на коју од верзија. Вероватно је ово разлог што се ово упутство не налази међу документима који су део „ConTeXt Standalone”. Без обзира на ове мане, упутство и даље представља најбољи документ за почетак учења система ConTeXt након што се прочита уводни „ConTeXt Mark IV, an Excursion”. За прве кораке у систему ConTeXt такође су корисне информације које могу да се пронађу на његовом [вики](#) веб сајту који се, у време писања овог текста, редизајнира тако да добија много јаснију структуру, мада и он меша објашњења која функционишу само у Mark II са осталима за Mark IV, или за обе верзије. Овај недостатак диференцијације се такође проналази у званичној листи ConTeXt команди¹ која не наводи које команде раде само у једној од две верзије.

У основи, овај увод је написан користећи информације из четири извора који се овде наводе: упутство ConTeXt „Excursion”, упутство из 2013. године, садржај вики веб сајта и званична листа команди у којој се налази, за сваку од њих, допуштена конфигурација опција; уз то, наравно, и моји сопствени тестови и закључци. Дакле, овај увод у суштини представља резултат истраживачког рада, па сам се неко време био премишљао да ли да га назовем „Шта знам о ConTeXt Mark IV” или „Шта сам научио о ConTeXt Mark IV”. Коначно, одбацио сам ове наслове јер, колико год да су истинити, осећао сам да бих њима вероватно одвратио некога од упознавања са системом ConTeXt; и оно што је сигурно је то да мада документација има (према мом мишљењу) неке недостатке, имамо заиста користан и свестран алат за који се труд потребан да се научи несумњиво исплати. Користећи ConTeXt можемо да манипулишемо и конфигуришемо текст документа тако да постигнемо ствари које неко ко не познаје систем не може ни да замисли.

¹ За листу погледајте одељак 3.6.

Због природе своје личности, не могу спречити да се у овом документу с времена на време појаве притужбе у вези са недостатком информација. Не бих желео да се ово схвати погрешно: ја сам веома захвалан творци-ма система ConTeXt што су дизајнирали тако моћан алат и учинили га јавно доступним. Једноставно не могу да се одупрем мишљењу да би овај алат био много популарнији ако би се унапредила његова документација: потребно је уложити много времена да се систем научи, не због његове сложености (која је неоспорна, али није већа од неких сличних алата – уствари чак напротив), већ услед недостатка јасних, комплетних и добро организованих информација које праве разлику између две верзије система ConTeXt, објашњавајући функције у свакој од њих, и првенствено, јасно наводећи шта свака команда, аргумент и опција раде.

Тачно је да ова врста информација захтева улагање доста времена. Али ако се има у виду да многе команде деле опције са сличним именима, можда би могла да се обезбеди нека врста *речника* опција што би такође помогло да се открију неке недоследности које се јављају када две опције са истим именом раде различите ствари, или када је потребно да се за исту ствар у две различите команде користе опције са другачијим именом.

Као читаоца који по први пут приступа систему ConTeXt, нека вас моје притужбе не одврате од учења система, јер мада може бити тачно да недостатак информација продужава време потребно за учење система, бар што се тиче материјала о коме се говори у овом уводу, ја сам већ уложио то време тако да читалац нема потребе поново то да ради. И само са оним што науче из овог увода, читаоци ће на располагању имати алат који им омогућава да једноставно произведу документе какве никада нису очекивали да ће бити у стању да произведу.

Пошто оно што је објашњено у овом документу у највећој мери долази из мојих сопствених закључака, врло је могуће да, упркос томе што сам лично тестирао већину онога о чему говорим, неке изјаве или мишљења можда нису у потпуности исправна нити уобичајена. Ценићу, наравно, било какву исправку, побољшање или разјашњење које читаоци могу да ми понуде, и оне могу да се пошаљу на адресу joaquin@ataz.org. Међутим, да би се смањио број ситуација у којима је вероватно да нисам у праву, покушао сам да не залазим у материју о којој нисам пронашао никакве информације и коју нисам могао (или желео) лично да испробам. Понекад је ово случај јер резултати мојих тестова нису били убедљиви, а понекад јер нисам успео све да тестирам: број команди и опција које има систем ConTeXt је импресиван, па ако бих морао све да испробам, никада не бих завршио овај увод. Међутим, постоје прилике у којима не могу да избегнем *прећућост* представљање нечега, нпр. исказ да нешто видим као вероватно, али да нисам у потпуности сигуран у то. У тим случајевима је на леву маргину пасуса у којем наводим такву претпоставку постављена слика 'нагађања'. Сврха слике је да графички прикаже претпоставку.¹ У другим ситуацијама нисам имао избора осим да признам како нешто не знам и да немам разумну претпоставку о томе: у том случају, слика која се приказује непосредно лево у маргини служи да покаже више од нагађања или незнања.² Али пошто никада нисам био добар са графичким представљањем, нисам сигуран да изабране слике успевају да пренесу толико финеса.



С друге стране, овај увод је написан из угла читаоца који не зна ништа о системима T_EX или ConTeXt, мада се надам да такође може бити користан и некоме ко долази са система T_EX или L^AT_EX (најпопуларнијим од система изведених из T_EX) и који по први пут приступају систему ConTeXt. Исто тако, свестан сам да тиме што покушавам да задовољим сваког читаоца, постоји ризик да нико не буде задовољан. Стога, у случају да постоји сумња, увек сам јасно стављао до знања да је књига углавном намењена некоме ко се по први пут упознаје са системом ConTeXt, некоме ко је тек наишао на овај очаравајући екосистем.

То што је неко почетник у систему ConTeXt не повлачи да је он такође и почетник у употреби компјутерских алата; и мада у овом уводу не претпостављам да читаоци поседују било

¹ Сliku нисам ја нацртао, већ сам је преузео са интернета (<https://es.dreamstime.com/>), где се каже да је то бесплатна слика.

² Такође је пронађена на интернету (<https://www.freepik.es/>) где се одобрава слободна употреба.

какав одређени ниво компјутерске писмености, ипак претпостављам одређену „разумну писменост” која на пример значи да се поседује опште разумевање разлике између текст процесора и текст едитора, познавање начина да се креира, отвори и манипулише текст фајл, познавање начина да се инсталира програм, познавање начина да се отвори терминал и изврши команда... и још по нешто.

Читајући претходне делове увода док пишем ове линије, схватам да се понекад занесем у компјутерске проблеме који нису неопходни за учење система ConTeXt и да би то могло да уплаши почетника, док сам у другим ситуацијама заузет објашњавањем прилично очигледних ствари које би могле бити досадне искусном читаоцу. Молим за попустљивост и од једног и од другог. Наравно да разумем како је почетнику у компјутеризованом управљању текстом веома тешко да уопште зна за постојање система ConTeXt, али ја сам у свом професионалном окружењу окружен људима који се константно боре са текстовима док користе текст процесоре и то чине прилично успешно, али пошто никада нису радили са текст фајловима, они игноришу неке основне ствари као што су, на пример, кодирање које се користи у фајлу или шта је разлика између текст процесора и текст едитора.

Чињеница да је овај приручник дизајниран за људе који не знају ништа о систему ConTeXt или T_EX, повлачи да сам навео и информације које очигледно нису везане за ConTeXt већ за T_EX; али сам схватио да није неопходно оптеретити читаоце информацијама које им нису битне, што би могао да буде случај када нека команда која у *суштини* функционише, заиста ConTeXt команда, или припада систему T_EX; тако да само у неким ситуацијама, када ми се чини да је то корисно, разјашњавам да одређена команда уствари припада систему T_EX.

У погледу организације овог документа, материјал је груписан у три блока:

- **Први део**, који се састоји од прва четири поглавља, даје уопштени преглед система ConTeXt, објашњава шта је он и како се ради са њим, приказује први пример трансформисања документа тако да би касније могли да се објасне неки фундаментални концепти система ConTeXt заједно са одређеним питањима о ConTeXt изворним фајловима.

Као целина, ова поглавља су намењена читаоцима који до сада познају само рад са текст процесорима. Читалац који већ зна да ради са означавајућим језицима би могао да прескочи ова рана поглавља; а ако читалац већ познаје T_EX или L^AT_EX, могли би такође да прескоче и већину садржаја поглавља 3 и 4. Исто тако, препоручио бих да се прочита барем:

- Информације у вези са ConTeXt командама (поглавље 3), а поготово начин на који команда функционише, како се конфигурише, јер ту лежи основна разлика између концепције и синтаксе система L^AT_EX и ConTeXt. Пошто се овај увод односи само на овај други, те разлике се не изражавају директно, али неко ко чита ово поглавље и зна начин на који функционише L^AT_EX ће одмах разумети разлику у синтакси ова два језика, а и начин на који нам ConTeXt дозвољава да конфигуришемо и прилагодимо функционисање скоро свих његових команди.
- Информације о ConTeXt пројектима са више фајлова (поглавље 4), што се прилично разликује од начина рада са осталим системима заснованим на систему T_EX.
- **Други део**, који чине поглавља 5 до 9, фокусира се на оно што сматрамо да су основни општи аспекти ConTeXt документа:

- Два аспекта која углавном утичу на изглед документа су величина и распоред његових страна, као и фонт који се користи. Поглавља 5 и 6 су посвећена овим стварима.
 - ★ Прво се посвећује страницама: величином, елементима који чине страницу, распоред тих елемената (што значи начин на који су распоређени елементи) итд. Из разлога систематизације, овде се говори и о одређенијим аспектима као што су они који се тичу нумерације страница и механизма који омогућавају да се на утиче на нумерацију.
 - ★ Поглавље 6 објашњава команде у вези са фонтовима и управљањем фонтовима. Ту се такође налази основно упутство за употребу и управљање бојама, мада оне нису стриктно *карактеристика* фонтова, ипак у истој мери утичу на спољашњи изглед документа.
 - Поглавља 7 и 8 се баве структуром документа и алатима које ConTeXt нуди ауторима као помоћ у писању добро структурираних докумената. Поглавље 7 се фокусира на структуру документа у ужем смислу (структурну поделу документа), а поглавље 8 на начин којим се та структура огледа у садржају; мада уз ово објашњење користимо прилику да такође објаснимо како се генеришу различите врсте индекса системом ConTeXt, јер све то у систему ConTeXt потпада под појам „листи”.
 - Коначно, поглавље 9 се бави референцама, важном глобалном аспекту било ког документа онда када је потребно да укажемо на нешто што се налази у неком другом делу документа (интерне референце) или у другим документима (спољашње референце). У случају ових других, тренутно нас интересују референце (линкови) који воде на спољашњи документ. Ови *линкови* (који такође могу да се јаве и у интерним референцама) чине да наш документ буде *интерактиван*, и у овом поглављу показујемо неке од могућности система ConTeXt које служе за креирање таквих докумената.
- Није потребно да се ова поглавља читају у било ком одређеном редоследу, осим поглавље 8, које би могло лакше да се разуме ако се прво прочита поглавље 7. У сваком случају, покушао сам да осигурам да када у поглављу или одељку искрсне питање о којем се говори на неком другом месту у овом уводу, текст садржи опаску о томе заједно са хиперлинком на место где се питање обрађује. Ипак, нисам у позицији да гарантујем како ће ово увек бити случај.
- Коначно, **трећи део** (поглавља 10 и наредна) се фокусира на детаљније аспекте. Они су независни, не само једни од других, већ чак и од својих одељака (осим можда у последњем поглављу). Имајући у виду велики број алата који су део система ConTeXt, овај део би требало да буде прилично опширан; али како сматрам да док читаоци стигну овде, већ ће бити припремљени да се и сами уроне у ConTeXt документацију, укључио сам само следећа поглавља:
 - Поглавља 10 и 11 се баве оним што бисмо могли да назовемо *основни елементи* било ког текст документа: текст је исписан карактерима који чине речи груписане у линије, које затим чине пасусе раздвојене један од другог вертикалним простором... Јасно је да би све ове ствари могле да се сместе у једно поглавље, али пошто би оно

било предугачко, материју сам поделио у два поглавља, једно које се бави карактерима, речима и хоризонталним размацама и друго које се бави линијама, пасусима и вертикалним размацама.

- Поглавље 12 је нека врста *мешавине* која се бави елементима и конструкцијама које се обично срећу у документима; углавном у академским и техничким документима; највећим делом у академским, научним или техничким документима: фусноте, структурне листе, набрајања, итд.
- Коначно, поглавље 13 се фокусира на пливајуће објекте, посебно оне који се најчешће користе: слике које се умећу у документе и табеле.
- Увод се завршава са три **додатка**. Један је у вези са инсталацијом система ConTeXt, а други додатак садржи неколико десетина команди које генеришу различите симболе – углавном, мада не само за математичку употребу и трећи садржи азбучну листу ConTeXt команди које су поменуте или објашњене раније у тексту.

Постоји много тема које још увек треба да се објасне: рад са цитатима и библиографским референцама, писање специјализованих текстова (математичких, хемијских...), веза са XML, интерфејс са Lua кодом, режими и обрада заснована на режимима, рад са MetaPost језиком за дизајнирање графике, итд. То је разлог због кога сам, како не дајем комплетно објашњење система ConTeXt, нити се правим да то чиним, назвао овај документ „Увод у ConTeXt Mark IV”; и додао сам чињеницу да увод није баш кратак, јер очигледно је тако: текст који је оставио још доста тога недовршеног, али је већ стигао до више од 300 страница никако не може да се назове кратак увод. Желим да читалац разуме логику система ConTeXt, или барем онако како сам је ја разумео. Не покушава да се представи као референтно упутство, већ пре водич за самостално учење који припрема читаоца да прави документе средње сложености (а то значи већину вероватних докумената) и да првенствено учи читаоца да *замисли* шта може да се уради овим моћним алатом, као и да у доступној документацији сазна како то да изврши. Овај документ није ни *туторијал*. Туторијали су дизајнирани тако да прогресивно повећавају ниво сложености, тако да се корак по корак прелази оно што треба да се научи; када се ово има на уму, ја сам хтео да почнем са другим делом уместо да поређам материјал по степену сложености, тако да будем систематичнији. Али како ово није туторијал, представио сам велики број примера.

Можда ће наслов овог документа неке читаоце да подсећа на текст доступан на интернету који су написали ЕТИКЕР, ПАРТЛ, ХИНА и ШЛЕГЛ, један од бољих докумената који читаоца уводи у L^AT_EX свет. Мислим на „*The Not So Short Introduction to L^AT_EX 2_ε*”. То није случајност, већ одавање почasti и уважавање: захваљујући онима који пишу текстове као што је овај, многи људи могу почети да раде са корисним и моћним алатима као што су L^AT_EX и ConTeXt. Ови аутори су ми помогли да почнем са системом L^AT_EX; ја се надам да могу урадити исто за некога ко жели да почне са системом ConTeXt, мада сам оригиналну шпанску верзију текста наменио само делу света који говори шпански језик и којем недостаје много документације на њиховом језику. Надам се да овај документ испуњава очекивања и у међувремену, други су несебично понудили помоћ да се он преведе на остале језике. Отуда и ово енглеско издање. Хвала вам.

Хоакин Атас-Лопес
Лето 2020

I

Шта је ConT_EXt и како се користи

Глава 1

ConTeXt: општи преглед

Садржај: 1.1 Па шта је уопште ConTeXt?; 1.2 Словослагање текстова; 1.3 Језици за означавање; 1.4 TeX и језици изведени из њега; 1.4.1 TeX машине; 1.4.2 Формати који су изведени из TeX; 1.5 ConTeXt; 1.5.1 Кратка историја система ConTeXt; 1.5.2 ConTeXt у односу на LaTeX; 1.5.3 Добро разумевање динамике рада у систему ConTeXt; 1.5.4 Добијање помоћи за систем ConTeXt;

1.1 Па шта је уопште ConTeXt?

ConTeXt је *словослајачки сисџем*, или другим речима: опсежан скуп алата креиран тако да кориснику пружи апсолутну и потпуну контролу над изгледом и презентацијом специфичних електронских докумената намењених за штампање на папиру или за приказ на екрану. Ово поглавље објашњава шта то све значи. Али најпре, хајде да истакнемо неке од карактеристика система ConTeXt.

- Постоје две *врсије* система ConTeXt познате као Mark II и Mark IV. ConTeXt Mark II је замрзнут, тј. сматра се да је потпуно развијен језик који се убудуће неће мењати, нити ће добијати нове могућности. Нова верзија би се појавила само у случају да постоји грешка која мора да се исправи. С друге стране, ConTeXt Mark IV наставља да се развија и с времена на време се појављују нове верзије које доносе нека побољшања или додатне могућности. Али, мада се још увек развија, он је прилично зрео језик у који нове верзије уносе прилично суптилне измене које скоро искључиво утичу на функционисање система на ниском нивоу. За обичног корисника су ове измене потпуно транспарентне; као да се уопште нису ни унеле. Мада обе *врсије* имају доста тога заједничког, оне имају и неке некомпатибилне могућности. Из тог разлога се овај увод фокусира само на ConTeXt Mark IV.
- ConTeXt је софтвер *libre* (или слободан софтвер, али не у смислу да је *десјлајан*). Тачније, програм (односно комплекс компјутерских алата који чине ConTeXt), се дистрибуира под ГНУ *Опшћом Јавном Лиценцом*. Документација се доставља под „Creative Commons” лиценцом која дозвољава да се слободно копира и дистрибуира.
- ConTeXt није ни текст процесор ни програм за уређивање текста, већ колекција алата намењених *трансформисању* текста који је написан омиљеним текст едитором. Стога, када радимо са системом ConTeXt:
 - Почињемо тако што пишемо један или више текст фајлова било којим текст едитором.
 - Ови фајлови, заједно са текстом који чини садржај документа, садрже и низ инструкција које систему ConTeXt описују жељени изглед финалног документа генерисаног

из оригиналних текст фајлова. Уствари, комплетан скуп ConTeXt инструкција је *језик*; и пошто овај језик омогућава *програмирање* типографске трансформације текста, можемо рећи да је ConTeXt *типиграфски програмски језик*.

- Када напишемо изворне фајлове, програм (који се такође зове „context”¹) ће од њих да генерише PDF фајл који је спреман за слање у штампарију или приказ на екран.
- Дакле, у систему ConTeXt морамо да направимо разлику између документа који пишемо и документа који генерише ConTeXt. Да би се спречиле било какве недоумице, документ који садржи инструкције за форматирање ћу у овом уводу да зовем *изворни фајл*, а PDF документ који из изворног фајла генерише ConTeXt ћу да зовем *финални докуменџ*.

О овим основним стварима ћемо још расправљати мало касније.

1.2 Словослагање текстова

Писање документа (књиге, чланка, поглавља, проспекта, рада...) и типографско састављање свега су две потпуно различите активности. Писање документа је скоро исто као и писање оловком; то ради аутор који одлучује о његовом садржају и структури. Документ који прави директно аутор, онако као да га је он или она написао, назива се *рукопис*. Природно је да само аутор, или они који имају право да га читају, могу да приступе рукопису. Дељење рукописа ван ове мале групе захтева да се рукопис *објави*. Данас је то – у етимолошком смислу омогућавања „јавног приступа” – просто као постављање на интернет, тако да је рукопис доступан сваком ко га пронађе и жели да га прочита. Али све до релативно скоро, објављивање је било прилично скуп процес који зависи од одређених професионалаца специјализованих за то, вољних да приступе рукопису који сматрају довољно значајним, било због његовог садржаја, било због његових аутора. Па чак и данас тежимо да реч *публикација* резервишемо за ову врсту *професионалних издања* код којих рукопис пролази кроз низ трансформација свог изгледа, чији циљ је да се унапреди *читљивост* документа. Овај низ трансформација је оно што називамо *словослагањем*.

Циљ словослагања је – у општем случају, остављајући по страни текстове маркетиншког типа који покушавају да привуку пажњу читаоца – да произведе документе највеће *читљивости*, што подразумева квалитет штампаног текста који позива на читање или га омогућава, и осигурава да је читаоцу удобно да чита. Овome доприноси много ствари; неке се, наравно, тичу *садржаја* документа: (квалитет, јасноћа, организација...), али остале зависе од ствари као што су врста и величина употребљеног фонта, употреба празног простора у документу, визуелно раздвајање пасуса, итд. Уз то, постоје и друге врсте ресурса које нису толико графичког или визуелног типа, као што су присуство одређених помоћних средстава читаоцу – заглавља и подножја страница, индекси, речници, употреба црног слога, наслова у маргини, итд. Знање и исправна употреба свих ресурса који су доступни словослагачу би могло да се назове „уметност словослагања” или „уметност штампања”.

¹ ConTeXt је у исто време и језик и програм (а и још неке друге ствари). У тексту као што је овај, та чињеница прави проблем, јер понекад морамо направити разлику између ова два аспекта. Због тога сам усвојио типографску конвенцију да када говорим о језику „ConTeXt”, или и о језику и о програму, његово име пишем користећи логотип (ConTeXt). Међутим, када желим да говорим само о програму, онда име пишем „context” користећи сва мала слова и фонт фиксне ширине, типичан за компјутерске термине и писање машине. Овај фонт ћу такође да користим и за примере и помињања команди које су део овог језика.

У прошлости, све до појаве компјутера, задаци и улоге писца и словослагача су били прилично раздвојени. Аутор је писао руком, или у XIX веку писаћом машином која је имала ограничене типографске ресурсе, чак и више ограничене него што су могућности оних који су ручно писали; тада је писац прослеђивао оригинале издавачу или штампару, који их је трансформисао тако да се добије штампани документ.

Данас је компјутерска наука аутору олакшала одлучивање о композицији све до најситнијих детаља. Међутим, то не умањује важност чињенице да квалитети потребни за доброг аутора нису исти као они који су потребни за доброг словослагача. Зависно од врсте документа, аутору је потребно разумевање материје о којој пише, јасноћа излагања, добро организовано размишљање које води до добро организованог текста, креативност, осећај за ритам, итд. Али словослагач мора да комбинује добро знање концептуалних и графичких ресурса који су му на располагању и довољно добар укус како би могао складно да их употреби.

Добрим програмом за обраду текста¹ је могуће постићи типографски разумно добро припремљен документ. Али текст процесори, у општем случају, нису дизајнирани за словослагање, па резултати, мада могу бити коректни, не могу да се пореде са резултатима који се добију помоћу других алата дизајнираних за контролу композиције документа. Уствари, текст процесори су еволуирали из писаћих машина, па њихова употреба, услед тога што маскирају разлику између ауторства текста и словослагања текста, води ка типографски неадекватним текстовима којима недостаје структура. С друге стране, алати као што је ConTeXt су еволуирали из штампарске пресе; они нуде много више композиционих могућности, а изнад свега, начин њихове употребе не може да се научи без успутног овладавања многим појмовима у вези са словослагањем. То је и разлика у односу на текст процесоре које неко може годинама да користи без потребе да научи било шта о типографији.

1.3 Језици за означавање

Као што сам већ поменуо, у данима пре компјутера, аутор је рукопис припремао ручно или писаћом машином и предавао га је издавачу или штампару који је био одговоран да га трансформише у финални штампани текст. Мада је аутор сасвим мало био умешан у трансформацију, он или она је утицао на указивање да су, на пример, одређене линије рукописа наслови његових разних делова (поглавља, одељака...), или да одређене ствари треба типографски истаћи на неки начин. Ове ознаке је аутор стављао на сам рукопис, понекад директно, а понекад користећи одређене конвенције које су се временом развијале. На пример, поглавља су увек почињала на новој страници уметањем неколико празних линија испред наслова, подвлачећи наслов, исписујући га верзалом, или постављајући текст који треба да се истакне између две подвлаке, повећавајући увлачење пасуса, итд.

Укратко, аутор је *означавао* текст како би навео начин на који треба да се сложи. Касније би уредник руком исписао у текст остале ознаке намењене штампару, као што су на пример, фонт који треба да се користи и његова величина.

¹ Према прилично старој конвенцији, правимо разлику између *TeX* и *TeX* процесора. Рани програми за уређивање текста су обрађивали неформатиране текст фајлове, док су друге врсте програма радиле са бинарним фајловима форматираног текста.

Данас, у компјутеризованом свету, ово настављамо да радимо кад генеришемо електронске документе користећи нешто што се назива *језик за означавање*. Ове врсте језика користе низ *ознака* или индикација које програм за обраду фајлова који их садрже зна како да интерпретира. Тренутно је HTML вероватно најпознатији језик за означавање, јер је већина веб страница базирана на њему. HTML страница садржи текст веб странице заједно са низом ознака које програму за преглед који је учитава говоре како би требало да је прикаже. HTML означавање које разумеју веб прегледачи, заједно са инструкцијама о томе где да их користе, се назива „HTML језик”, и то је *језик за означавање*. Али уз HTML постоји много других језика за означавање; они уствари цветају као печурке после кише, тако да се XML, језик за означавање *par excellence*, налази свуда и користи се практично за све: за дизајн база података, за креирање специфичних језика, пренос структурираних података, фајлове конфигурације апликација, итд. Такође постоје језици за означавање намењени графичком дизајну (SVG, TikZ или MetaPost), математичким формулама (MathML), музици (Lilypond и MusicXML), финансијама, геоматици, итд. А такође постоје и језици за означавање који су намењени типографској трансформацији текста, међу којима се истичу T_EX и језици изведени из њега.

Када се говори о *типиграфском* означавању које указује на то како би текст требало да изгледа, разликујемо две врсте означавања: *чисто типиграфско означавање* и *концептуално означавање* или, ако вам тако више одговара, *логичко означавање*. Чисто типографско означавање је ограничено на прецизно указивање типографског ресурса који би требало да се употреби за приказ одређеног текста; као када, на пример, наведемо да би неки текст требало да се испише црним слогом или курзивом. С друге стране, концептуално означавање наводи функцију коју одређени текст има у документу као целини, као када наведемо да је нешто наслов, поднаслов или цитат. У општем случају, документи у којима се користи ова друга врста означавања су конзистентнији и једноставнији за састављање, јер указују на разлику између ауторства и композиције: аутор наводи да је та и и та линија наслов, или тај и тај фрагмент упозорење, или цитат; па словослагач одлучује како да типографски истакне све наслове, упозорења или цитате; дакле, с једне стране је гарантована конзистенција, пошто ће сви фрагменти који имају одређену функцију изгледати исто, а са друге стране, тако се штеди време јер је потребно да се формат сваке врсте фрагмента наведе само једном.

1.4 T_EX и језици изведени из њега

T_EX је крајем 70их година развио ДОНАЛД Е. КНУТ, професор (сада емеритус професор) теоретског компјутерског програмирања на Универзитету Стенфорд. Он је имплементирао програм да би произвео сопствене публикације и као пример систематски развијеног и коментарисаног програма. Уз T_EX, КНУТ је развио још један програмски језик под именом METAFONT, креиран за дизајнирање типографских фонтова и користио га је да дизајнира фронт који је крстио *Computer Modern*, који, уз уобичајене карактере сваког фонта, такође укључује и комплетан скуп „словних ликова”¹ дизајнираних за писање математике. Свему овоме је додао неколико додатних алата и тако је рођен словослагачки систем под именом T_EX, који се, захваљујући својој снази, квалитету резултата, флексибилности употребе и

¹ У типографији, словни лик је графичка интерпретација карактера, већег броја карактера, или дела неког карактера, и то је савремени еквивалент реза (ствари у коју је урезано слово или покретно слово).

широким могућностима, сматра за један од најбољих компјутеризованих система за композицију текста. Дизајниран је за текстове у којима има доста математике, али је ускоро било јасно да га системске могућности чине погодним за све врсте текстова.

Интерно, \TeX функционише на исти начин као што су радили стари слагачи у штампаријама. За \TeX , све представља *кутију*: слова се налазе у кутијама, празни простори су такође кутије, неколико слова (кутије које садрже неколико слова) чине нову кутију која садржи реч, а неколико речи, заједно са празним размаком између њих, чини кутију која садржи целу линију, неколико линија постају кутија која садржи пасус... и тако даље. А све ово уз невероватну прецизност при обради растојања. Имајте на уму да је најмања јединица коју \TeX може да обради 65.536 пута мања од типографске тачке којом се мере карактери и линије, што је обично и најмања јединица коју могу да обраде већина текст процесор програма. Ово значи да је најмања јединица са којом \TeX може да ради приближно 0,00005356 милиметра.

Име \TeX је изведено из корена грчке речи τέχνη, исписане верзалом (ΤΕΧΝΗ). Стога, последње слово речи \TeX није латинично 'X', већ грчко 'χ', које се изговара као „х”. Дакле, \TeX би требало да се изговара као *џех*. С друге стране, ова грчка реч је значила и „уметност” и „технологија”, па је то разлог што је Кнут баш њу изабрао као име свог система. Сврха овог имена – написао је – „је да вас подсети како се \TeX првенствено тиче техничких рукописа високог квалитета. Његов акценат је на уметности и технологији, као што је то и грчка реч по којој је добио име”.

Користећи конвенцију коју је установио Кнут, \TeX би требало да се напише:

- У типографски форматираним текстовима као што је овај, користећи логотип који сам до сада користио: три слова у верзалу, са средњим 'E' помереним мало наниже, тако да се омогући мањи размак између 'T' и 'X'; или другим речима: „ \TeX ”.

Да би омогућио писање овог логотипа, Кнут је направио инструкцију у језику \TeX којом се он исписује у финални документ: `\TeX`.

- У неформатираним текстовима (као што је имејл, или текст фајл), са 'T' и 'X' у верзалу, а средњим 'e' у куренту; дакле: „TeX”.

Ова конвенција наставља да се користи у свим системима који се заснивају на систему \TeX тако да укључе своје исправно име, као што је случај са системом ConTeXt. Када се пише у текст режиму требало би да напишемо „ConTeXt”.

1.4.1 \TeX машине

Програм \TeX је слободан *libre* софтвер: његов изворни код је јавно доступан и свако може да га користи или мења како год жели, само уз услов да ако се направе измене, резултат више не сме да се назива „ \TeX ”. Ово је разлог што су се током времена појавиле извесне адаптације програма које су у њега уводиле различита побољшања, а која се уопштено називају *\TeX машине*. Уз оригинални \TeX програм, главне машине су, по хронолошком редоследу појављивања, pdf \TeX , ε - \TeX , $\text{\textsf{Xe}}\TeX$ и Lua \TeX . Свака од њих би требало да уводи побољшања у односу на претходну. С друге стране, ова побољшања, све до појаве Lua \TeX машине, нису

утицала на сам језик, већ само на улазне фајлове, излазне фајлове, управљање изворима и рад макроа на ниском нивоу.

Питање употребе T_EX машине је једно од питања о којем се у T_EX свету највише расправља. Ово питање нећу да разматрам овде јер ConTeXt Mark IV функционише само са LuaT_EX. У суштини, у ConTeXt свету, дискусија о T_EX машинама постаје дискусија о томе да ли да се користи Mark II (који ради са PdfT_EX и XeT_EX) или Mark IV (који ради само са LuaT_EX).

1.4.2 Формати који су изведени из T_EX

Језгро или срце система T_EX разуме само скуп од око 300 веома основних инструкција, које се називају *примитиве*, и које су погодне за словослагачке операције и функције програмирања. Већина ових функција су веома *ниској нивоа*, што у компјутерској терминологији значи да их компјутери лакше разумеју него људи, јер се тичу елементарних операција типа „помери овај карактер 0,000725 милиметара навише”. Тако да је КНУТ увидео да би T_EX требало да има могућност проширивања, што значи да би требало да постоји механизам који омогућава да се дефинишу инструкције на вишем нивоу, инструкције које људи лакше разумеју. Ове инструкције, које се у време извршавања раздијају на једноставније инструкције, називају се *макрои*. На пример, T_EX инструкција која штампа (`\TeX`) логотип, се у време извршавања разлаже на следећи начин:

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

Али за људско биће је много једноставније да разуме и упамти како једноставна команда „`\TeX`” извршава типографске операције неопходне да се испише логотип.

У суштини је разлика између онога шта је *макро* и онога шта је *примитива* битна само из перспективе T_EX програмера. Из перспективе корисника, то су све *инструкције* или, ако вам више одговара, *команде*. Кнут их је називао *контролни низови*.

Ова могућност проширивања језика помоћу *макроа* је једна од карактеристика које су учиниле T_EX тако моћним алатом. У суштини, сам КНУТ је сачинио око 600 макроа који, заједно са 300 примитива, чине формат који се назива „Plain T_EX”. Врло је уобичајено да се T_EX замени са Plain T_EX, а заправо се све што је написано или речено о T_EX, уствари односи на Plain T_EX. Књиге које тврде да су о T_EX (укључујући и базичну „*The T_EXBook*”), у суштини говоре о Plain T_EX; и они који верују да раде директно са T_EX у суштини раде са Plain T_EX.

Plain T_EX је оно што се у T_EX терминологији назива *формат*, а састоји се од опсежног скупа макроа, заједно са одређеним правилима синтаксе који одређују како и када треба да се користе. Током времена су уз Plain T_EX развијени и остали *формати*, од којих вреди поминути L^AT_EX, детаљни скуп макроа за T_EX који је 1985. године направио ЛЕСЛИ ЛАМПОРТ и који представља систем изведен из T_EX који се вероватно најчешће користи у академском, технолошком и математичком свету. ConTeXt је (или је почео да буде), на сличном нивоу као и L^AT_EX, формат настао из T_EX.

Обично уз ове *формате* долази програм који у меморију учитава макрое који их сачињавају пре позивања програма „tex” (или машине која се користи за обраду) да обради изворни

фајл. Али мада сви ови формати уствари извршавају T_EX, пошто сваки од њих, посматрано из угла корисника, поседује различите инструкције и различита синтаксна правила, сваки од њих можемо схватити као *различити језик*. Сви они узимају инспирацију из T_EX, али се разликују од језика T_EX и једни од других.

1.5 ConT_EXt

Мада је у суштини ConT_EXt започет као T_EX *формат*, он је данас много више од тога. ConT_EXt садржи:

1. Веома опширан скуп T_EX макроа. Ако Plain T_EX садржи око 900 инструкција, ConT_EXt их има око 3500; а ако додамо и имена различитих опција које поседују ове команде, говоримо о речнику од око 4000 речи. Речник је тако велики због ConT_EXt стратегије да олакшавање његовог учења значи увођење било ког броја синонима за команде и опције.

Ако је потребно постићи неки ефекат, намера је да онда за сваки од начина на који би енглески говорник могао да назове тај ефекат постоји команда или опција која га постиже – што би требало да олакша употребу језика. На пример, ако желите истовремено да добијете црни слог и курзив, ConT_EXt вам обезбеђује три инструкције које постижу исти резултат: `\bi`, `\italicbold` и `\bolditalic`.

2. Слично опширан скуп макроа за METAPOST, графички програмски језик изведен из језика METAFONT, језика за дизајн словних ликова који је КНУТ развио заједно са програмом T_EX.
3. Разне *скрипте* развијене за језик PERL (најстарије), RUBY (неке такође старе, неке баш и не) и LUA (најсвежије).
4. Интерфејс који интегрише T_EX, METAPOST, LUA и XML, који омогућава писање докумената користећи било који ид ових језика, или мешање елемената из неког од њих.

Да ли сте разумели претходно објашњење? Не брините о томе. У њему сам употребио доста компјутерског жаргона и поменуо многе програме и језике. Да бисте користили ConT_EXt, није неопходно да познајете све његове разне компоненте. Битна ствар у овој фази учења је да имате на уму да систем ConT_EXt интегрише многе алате из различитих извора тако да сви заједно чине *словослагачки систем*.

Ова последња особина интеграције алата различитог порекла је разлог што кажемо да је ConT_EXt „хибридна технологија” намењена словослагању докумената. Ја сматрам да то ConT_EXt претвара у изузетно напредан и моћан систем.

Мада је ConT_EXt много више од колекције T_EX макроа, он је још увек заснован на систему T_EX, па је то разлог што се овај документ, за који не тврдим да је нешто више од *увода*, фокусира на ту чињеницу.

С друге стране, ConT_EXt је модернији од T_EX система. Када је креиран T_EX, компјутери су тек почели да се појављују и били смо далеко од тога да схватимо како ће изгледати (постати) свет интернета и мултимедије. У овом погледу ConT_EXt природно интегрише неке од ствари које су одувек биле нешто као страно тело у систему T_EX, као што је укључивање графике, управљање бојама, хиперлинкови у електронским документима, подразумевање величине папира која је погодна за документ намењен приказу на екран, итд.

1.5.1 Кратка историја система ConT_EXt

ConT_EXt је рођен отприлике у 1991. години. Направили су га ХАНС ХАГЕН и ТОН ОТЕН у холандској компанији за дизајнирање и обраду докумената која се назива „*Pragma Advanced Document Engineering*”, што се обично скраћује на Pragma ADE. Почео је као колекција T_EX макроа који су имали холандска имена и незванично се звао *Pragmatex*. Био је намењен запосленима у компанији који немају техничко образовање, а који су морали да управљају многим детаљима уређивања словослагачких докумената и који нису навикли на употребу језика за означавање или интерфејса на неком језику који није холандски. Због тога је прва верзија система ConT_EXt написана на холандском. Идеја је била да се креира довољан број макроа са униформним и конзистентним интерфејсом. *Пакеџ* је отприлике у 1994. години постао довољно стабилан да се напише корисничко упутство на холандском, па је током 1996. године, на иницијативу ХАНСА ХАГЕНА, почео да узима име „ConT_EXt”. Тврди се да ово име значи „Текст са T_EX” (употребом латинског предлога 'con' који значи 'са'), али је у исто време и игра речи на енглеском (и холандском) од речи „контекст”. Дакле, иза имена стоји трострука игра речи у којој се налазе „T_EX”, „text” и „context”.

Дакле, пошто је његово име засновано на игри речи, ConT_EXt би требало да се изговара 'контекст' а не 'контект', јер би се на тај начин изгубио смисао игре са речима.

Превод интерфејса на енглески је почео отприлике у 2005. години, па је тако настала верзија позната под именом ConT_EXt Mark II, где се 'II' објашњава тиме што је у главама програмера претходна верзија на холандском била Верзија 'I', мада се званично никада није тако звала. Када се интерфејс превео на енглески, употреба система је почела да се шири ван Холандије, па је интерфејс преведен на и на друге европске језике, као што су француски, немачки, италијански и румунски. Ипак, „званична” документација система ConT_EXt је обично базирана на енглеској верзији, и то је верзија којом се бави овај документ.

У својој почетној верзији, ConT_EXt Mark II је радио са pdfT_EX T_EX машином. Али касније, појавом X_YT_EX машине, ConT_EXt Mark II је измењен тако да се дозволи употреба ове нове машине којом је уведено више предности у односу на pdfT_EX. Али када је неколико година касније стигла LuaT_EX, донета је одлука да се ConT_EXt интерно реконфигурише тако да integriше све додатне могућности које је нудила ова нова машина. И тако је рођен ConT_EXt Mark IV, представљен 2007. године, непосредно након представљања LuaT_EX машине. Врло је вероватно да је један од одлучујућих фактора за одлуку да се ConT_EXt реконфигурише тако да се прилагоди за LuaT_EX био тај што су два од три главна програмера система ConT_EXt, ХАНС ХАГЕН и ТАКО ХЕКВАТЕР, такође били део главног тима који је развијао LuaT_EX. То је разлог што су ConT_EXt Mark IV и LuaT_EX били рођени у исто време и заједно развијани. Између ConT_EXt и LuaT_EX постоји синергија која не постоји ни у једном другом деривату система T_EX; сумњам да било који други може боље да искористи предности LuaT_EX машине од система ConT_EXt.

Постоји много разлика између Mark II и Mark IV, мада је већа њих *инићерна*, то јест, оне се тичу начина на који макрои функционишу на ниском нивоу, тако да се те разлике из угла корисника ни не примећују: име и параметри макроа остају исти. Међутим, неке разлике утичу и на интерфејс, па приморавају да се ствари у једној верзији раде другачије него у другој. Постоји релативно мало таквих разлика, али оне утичу на важне аспекте, као на пример, кодирање улазног фајла, или обрада фонтова инсталираних на систему.



Било би заиста корисно када би негде постојао документ који објашњава (или барем наводи) битне разлике између Mark II и Mark IV. На пример, у ConTeXt викију, за сваку ConTeXt команду постоје *две врсте синтаксе* (које су често идентичне). Претпостављам да једна важи за Mark II а друга за Mark IV; па према овој претпоставци, такође претпостављам да је *прва верзија* за Mark II. Али истина је да нам вики не говори ништа о овоме.

Чињеница да има мало разлика на нивоу језика значи да у већини случајева, уместо да говоримо о две верзије, говоримо о две „варијанте” система ConTeXt. Без обзира на то да ли их зовете овако или онако, чињеница је да у општем случају документ припремљен за Mark II не може глатко да се компајлира са Mark IV и обрнуто; а ако документ меша обе верзије, највероватније се неће исправно компајлирати ни на једној верзији; па то повлачи да аутор изворног фајла мора донети одлуку да ли жели да пише за Mark II или за Mark IV.

Ако радимо са различитим верзијама система ConTeXt, добар трик за прављење разлике на први поглед између фајлова намењених за Mark II и оних намењених за Mark IV је да се употреби различита екстензија за име фајла. Тако, на пример, свим фајловима које пишем за Mark II, стављам „.mki” као екстензију, а „.mkiv” за оне написане за Mark IV. Истина је да ConTeXt очекује да сви изворни фајлови имају „.tex” екстензију, али екстензију фајла можемо да променимо ако је експлицитно наведемо када позивамо ConTeXt над фајлом.

ConTeXt дистрибуција инсталирана на викију, „ConTeXt Standalone”, садржи обе верзије, па да би се спречила забуна – претпостављам – користи различиту команду за компајлирање фајла сваке верзије. Mark II компајлира командом „texexes”, а Mark IV командом „context”.

Уствари су обе команде „context” и „texexes”, *скрипте* са различитим опцијама које извршавају Lua скрипту „mtxrun”.

Данас је Mark II замрзнут, а Mark IV наставља да се развија, што значи да се нове верзије Mark II објављују само када се пронађу грешке или проблеми који морају да се исправе, док се нове верзије Mark IV редовно објављују; понекада два или три пута месечно, мада у већини случајева „нове верзије” не уносе уочљиве измене језика, већ су ограничене на унапређење имплементације команде на ниском нивоу, или на ажурирање неких од многих упутстава која се испоручују у дистрибуцији. Свеједно, ако инсталирамо развојну верзију – што топло препоручујем, а то је и оно што се подразумевано инсталира са „ConTeXt Standalone” – има смисла да с времена на време ажурирамо своју верзију (погледајте [Додатак А](#) у којем се говори како ажурирати „ConTeXt Standalone” верзију).

LMTX и остале алтернативне Mark IV имплементације

Програмери система ConTeXt су природно неуморни, тако да нису обуставили развој система ConTeXt са Mark IV; још увек се тестирају нове верзије и експериментише се са њима, мада се у општем случају оне врло мало разликују од Mark IV и не испољавају некомпатибилност компајлирања која постоји између Mark IV and Mark II.

Тако да је развијено неколико малих варијанти Mark IV под називима Mark VI, Mark IX и Mark XI. Од свих њих, на ConTeXt викију сам успео да пронађем само малу референцу на Mark VI, где се каже да је разлика у односу на Mark IV само у могућности дефинисања команди доделом именованих параметара а не бројева, као у традиционалном систему T_EX, дакле на начин како се то обично ради у скоро свим програмским језицима.

Верујем да је у ConTeXt свету појава нове верзије под називом LMTX много важнија од ових малих варијација. LMTX је акроним од luametaT_EX: нове T_EX *машине* која представља поједностављену верзију LuaT_EX, развијена са циљем уштеде компјутерских ресурса; што значи да LMTX захтева мање меморије и мање процесорске снаге од ConTeXt Mark IV.

LMTX је представљен у пролеће 2019. године и може да се претпостави да неће захтевати било какву спољашњу измену Mark IV језика. За аутора овог документа није било никакве разлике док је радио на њему; али када се компајлира, потребно је да се изабере да ли желите то да радите са LuaT_EX, или са luametaT_EX. У [додатку А](#), који

се тиче инсталације система ConTeXt, приказана је процедура за доделу другачијег имена команде за сваку од инсталација (одељак 3).

1.5.2 ConTeXt у односу на L^AT_EX

Пошто је L^AT_EX најпопуларнији формат изведен из T_EX, неизбежно је поређење између њега и система ConTeXt. Јасно је да говоримо о различитим језицима који су донекле у вези, јер и један и други проистичу из T_EX језика; однос је сличан ономе између, рецимо, шпанског и француског језика: језика који имају заједничко порекло (латински) што значи да су њихове синтаксе *сличне* и да за многе речи у сваком од ових језика постоји врло слична у оном другом. Али за разлику од ове *ћородичне сличности*, L^AT_EX и ConTeXt се разликују у својој филозофији и имплементацији, пошто су почетни циљеви сваког од њих у некој мери супротни. L^AT_EX тврди да олакшава употребу T_EX, издвајајући аутора од конкретних типографских детаља тако да може да се концентрише на садржај, остављајући систему L^AT_EX типографске детаље. Ово значи да се по цену поједностављења употребе система T_EX ограничава огромна флексибилност самог T_EX тако што се унапред дефинишу основни формати и ограничава број типографских проблема које аутор мора да реши. Супротно од ове филозофије, ConTeXt је рођен у компанији која је посвећена словослагању докумената. Стога, нипошто се не жели изоловање аутора од типографских детаља, већ је циљ да се аутору омогући апсолутна и комплетна контрола над њима. Да би се то постигло, ConTeXt обезбеђује униформни, конзистентан интерфејс који је много ближи оригиналном духу система T_EX него што је то L^AT_EX.

Ова разлика у филозофији и основним циљевима се затим преноси на разлику у имплементацији. L^AT_EX тежи да упрости ствари колико год је то могуће, тако да нема потребе да користи све ресурсе система T_EX. На неки начин, његово језгро је прилично једноставно. Па када се јави потреба да се његове могућности прошире, неопходно је да се експлицитно напише *ћакет* који то имплементира. Ова *ћакетизација* придружена систему L^AT_EX је у исто време и врлина и мана: врлина је, јер услед огромне популарности система L^AT_EX, уз великодушност његових корисника, то значи да је скоро све што можемо пожелети да урадимо неко пре нас већ имплементирао, па постоји пакет за то; али је такође и мана, јер често ови пакети нису компатибилни међусобно и синтакса коју користе није увек униформна. Ово значи да рад са системом L^AT_EX захтева константну претрагу кроз хиљаде постојећих пакета који могу да задовоље неку потребу, уз обезбеђивање услова да сви заједно раде онако како се очекује.

За разлику од једноставности L^AT_EX језгра, коју допуњава његова проширивост кроз пакете, ConTeXt је дизајниран тако да су у њега укључене све – или скоро све – типографске могућности система T_EX, тако да је његова концепција више монолитна, мада у исто време и више модуларна. ConTeXt језгро нам дозвољава да урадимо скоро све, уз гаранцију да неће бити некомпатибилности између различитих алата, неће бити потребно да се истраже проширења за испуњавање неке потребе, а синтакса језика се не мења само из разлога што нам је потребан одређени алат.

Тачно је да ConTeXt поседује нешто што се назива *модули* проширења, за које неко може да каже како обављају функцију сличну L^AT_EX пакетима, али чињеница је да они функционишу на различите начине: ConTeXt модули су дизајнирани тако да само укључе додатне могућности које, из разлога што су још у експерименталној фази, још увек нису постале

део језгра, или да се дозволи приступ проширењима које је направио неко ван ConTeXt развојног тима.

Ја лично не сматрам да је било која од ове две *филозофије* боља од оне друге. Питање зависи од корисниковог профила и тога шта он или она жели. Ако корисник не жели да се носи са типографским проблемима, већ једноставно да прави стандардизоване документе високог квалитета, вероватно би било боље да се определи за систем као што је L^AT_EX; с друге стране, корисник који воли да експериментирате, или којем је потребна контрола све до најситнијег детаља документа, или неко ко је осмислио специјалан распоред за документ, вероватно треба да користи систем као што је ConTeXt, у којем аутор има потпуну контролу; наравно, уз ризик да не зна како исправно да користи ту контролу.

1.5.3 Добро разумевање динамике рада у систему ConTeXt

Када радимо са системом ConTeXt, увек почињемо тако што пишемо текст фајл (који називамо *изворни фајл*), у којем се, заједно са садржајем финалног документа, налазе и инструкции (у ConTeXt-жаргону) које наводе како тачно желимо да се документ форматира: општи изглед његових страница и пасуса, маргине које желимо да доделимо одређеним пасусима, фонт којим желимо да се прикаже текст, фрагменти које желимо да се прикажу употребом другачијег фонта, итд. Када напишемо изворни фајл, у терминалу над њим извршавамо програм „context”, који ће га обрадити и генерисати из њега другачији фајл у којем ће садржај нашег изворног документа бити формиран сагласно са инструкцијама које смо навели у изворном фајлу. Овај излазни фајл би могао да се пошаље у (комерцијалну) штампарију, прикаже на екрану, постави на интернет или да се дистрибуира контакти-ма, пријатељима, клијентима, учитељима, ученицима... или другим речима, свима којима је намењен.

Ово значи да када ради у систему ConTeXt аутор ради са фајлом чији изглед нема везе са изгледом финалног документа: фајл са којим аутор непосредно ради је текст фајл који није типографски формиран. Тако да ConTeXt функционише на другачији начин од програма познатих под именом *текстни процесори*, који још у време писања приказују коначни изглед документа који се уређује. У почетку ће онима који су навикли на текст процесоре начин рада у систему ConTeXt бити чудан, па ће чак бити потребно и извесно време да се навикну на то. Међутим, када се неко навикне на такав рад, он схвата да је уствари овај другачији начин рада који прави разлику између радног фајла и коначног резултата, из многих разлога у суштини предност. Ја ћу овде, без икаквог одређеног редоследа да истакнем неке од тих предности:

1. Текст фајлови су 'лакши' за обраду од бинарних фајлова текст процесора, за уређивање је потребно мање компјутерске меморије, мања је вероватноћа да ће се искварити и не постају нечитљиви када се промени верзија програма који их је креирао. Компатибилни су са било којим оперативним системом и могу да се уређују многим текст едиторима, тако да за рад са њима није потребно да компјутерски систем има инсталиран програм који је креирао фајл: биће довољан било који други програм за уређивање; а сваки компјутерски систем увек поседује неки програм за уређивање текста.
2. Прављење разлике између радног документа и финалног документа помаже да се одвоји стварни садржај документа од онога што одређује његов изглед, па се аутору омо-

гућава да се у фази креирања концентрише на садржај, а да се на изглед фокусира у словослагачкој фази.

3. Омогућава брзу и прецизну измену изгледа документа, пошто се то ради ConTeXt командама које се једноставно проналазе у фајлу.
4. С друге стране, ова могућност измене изгледа дозвољава да се из једног садржаја једноставно креирају две (или више) различите верзије: можда је једна верзија оптимизована за штампу на папиру, а друга дизајнирана за приказ на екрану, подешена на његову величину и садржи хиперлинкове који немају смисла на папирном документу.
5. Могу лако да се спрече типографске грешке које су честе у текст процесорима, као што је продужавање курзива иза последњег карактера речи.
6. Пошто се радни фајл не дистрибуира и намењен је 'само за наше очи', могуће је да се унесу белешке и запажања, коментари, без бојазни да ће се појавити у коначном форматираном фајлу који се дистрибуира.
7. Квалитет који може да се постигне истовременом обрадом комплетног документа је много већи од квалитета који се постиже програмом који типографске одлуке мора да донесе док се документ уноси.
8. И тако даље.

Све горе наведено значи, с једне стране, да када се навикнемо на рад у систему ConTeXt, постајемо много ефикаснији и продуктивнији, а с друге стране, да је типографски квалитет који можемо постићи много већи од онога који може да се постигне такозваним *џексџи процесорима*. И мада је тачно да се они лакше користе, чињеница је да се не користе *много* лакше. Тачно је да систем ConTeXt, као што смо раније поменули, садржи 3500 инструкција, обичан корисник система ConTeXt не мора све да их зна. Да бисмо урадили оно што се обично ради текст процесорима, потребно је да знамо само инструкције које нам омогућавају да назначимо структуру документа, неколико инструкција које се тичу уобичајених типографских ресурса, као што су црни слог и курзив, и можда како да генеришемо листу, или фусноту. Све у свему, не више од 15 или 20 инструкција ће нам омогућити да урадимо скоро све ствари које се раде текст процесором. Остале инструкције нам омогућавају да урадимо ствари које обично не можемо да урадимо текст процесором, или које су компликоване за извођење. Можемо рећи да иако је теже научити коришћење система ConTeXt него коришћење текст процесора, то је зато што системом ConTeXt можемо да урадимо много више.

1.5.4 Добијање помоћи за систем ConTeXt

Док смо још увек почетници, [вики](#) је несумњиво најбоље место да се добије помоћ у вези са системом ConTeXt. Он је пун примера и има одличан систем претраге, посебно ако добро разумете енглески језик. Помоћ такође можемо да пронађемо и на интернету, али овде ћемо имати проблем са игром речи у имену ConTeXt, јер ће претрага на реч „context” да врати милионе резултата од којих већина нема никакве везе са оним што нас интересује. Да бисте пронашли информације о систему ConTeXt морате да додате нешто уз реч „context”; на пример, „tex”, или „Mark IV” или „Hans Hagen” (један од креатора система ConTeXt)

или „Pragma ADE”, или нешто слично. Такође би могло бити од користи да информације тражите употребом имена викија: „contextgarden”.

Када научимо нешто више о систему ConT_EXt, можемо да консултујемо неке од многих докумената који су део „ConT_EXt Standalone”, или да чак потражимо помоћ на [TeX – LaTeX Stack Exchange](#), или на мејлинг листи за ConT_EXt ([NTG-context](#)). Ову листу прате људи који знају већину ствари које се тичу система ConT_EXt, али правила добре сајбер етикеције захтевају да смо се пре постављања питања и сами заиста потрудили да дођемо до одговора.

Глава 2

Наш први изворни фајл

Садржај: 2.1 Припрема експеримента: неопходни алати; 2.2 Сам експеримент; 2.3 Структура фајла нашег примера; 2.4 Неки додатни детаљи у вези са покретањем команде „context”; 2.5 Управљање грешкама;

Ово поглавље је посвећено нашем првом експерименту и објасниће основну структуру ConTeXt документа, као и најбоље стратегије за исправљање потенцијалних грешака.

2.1 Припрема експеримента: неопходни алати

Да бисмо написали и компајлирали први изворни фајл, потребно је да на систему имамо инсталиране следеће алате.

1. **Текст едитор** за писање тест фајла. Постоји много текст едитора и тешко је замислити оперативни систем који већ нема неки преинсталиран. Можемо користити било који од њих: постоје једноставни, сложенији, моћнији, неки које морате да платите, неки бесплатни (тј. *ipaisic*), неки слободни (као *libre*), неки који су специјализовани за T_EX системе, а неки служе за општу примену, итд. Ако смо навикли да радимо у одређеном едитору, најбоље би било да наставимо да га користимо; ако у овом тренутку нисте навикли да радите ни са једним, мој савет је да најпре пронађете једноставни едитор, тако да уз комплексност учења система ConT_EXt не морате учити и како да користите текст едитор. Мада је тачно да су често програми који су најкомпликованији за учење они који су најмоћнији.

Овај текст сам написао користећи GNU Emacs, један од најмоћнијих и свестраних едитора опште намене који постоји; тачно је да има одређене специфичности и особине које га одвраћају од потенцијалних корисника, али у суштини има више „Emacs-љубитеља” него „Emacs-хејшера”. Постоји GNU Emacs екстензија под називом AucTeX која служи за рад са T_EX фајловима или неким од његових деривата. Она едитору обезбеђује неке врло интересантне додатне алате, мада је AucTeX у суштини боље припремљен за рад са L^AT_EX него са ConT_EXt фајловима. GNU Emacs у комбинацији са AucTeX би могао да буде одлична опција ако не знате који едитор да изаберете; и један и други су *libre* програми, тако да постоје верзије са све оперативне системе. Уствари, изјављивање како је GNU Emacs *софтвер libre* је потцењивање, јер овај програм боље него било који други отеловљује дух и значење *слободної софтвера*. На крају, његов главни програмер је био РИЧАРД СТОЛМЕН оснивач и идеолог GNU пројекта и *Фондације слободної софтвера*.

Слично као и GNU Emacs + AucTeX, остале добре опције, у случају да не знате шта да изаберете, су *Scite* и *TexWorks*. Овај први, мада је едитор опште намене који није посебно дизајниран за рад са ConT_EXt фајловима, једноставно се прилагођава и, како је то едитор који у општем случају користе ConT_EXt програмери, „ConT_EXt Standalone” садржи конфигурационе фајлове за овај едитор које је написао сам ХАНС ХАГЕН. С друге стране, *TexWorks* је брз едитор који је специјализован за обраду T_EX фајлова и фајлова

језика изведених из њега. Веома је једноставно да се подеси за рад са ConTeXt и „ConTeXt Standalone” такође доставља његову конфигурацију.

Који год едитор да изаберемо, једина ствар коју не смемо да користимо као текст едитор је *текстни процесор* као што је, на пример, OpenOffice Writer или Microsoft Word. Ови програми, по мом мишљењу преспори и тешки, могу, ако се то експлицитно назначи, да сачувају фајл као 'чисти текст (txt)', али они нису дизајнирани за тако нешто, па ћемо највероватније завршити са фајлом сачуваним у неком бинарном формату који није компатибилан са системом ConTeXt.

2. **ConTeXt** дистрибуција за обраду нашег тест фајла. Ако на вашем систему већ постоји T_EX (или L^AT_EX) инсталација, врло је вероватно да је инсталирана и верзија система ConTeXt. Да бисте то проверили, довољно је да покренете терминал и откуцате

```
$> context -version
```

НАПОМЕНА за оне који нису раније користили терминале, прва два карактера која сам написао („\$>”) се не пишу у терминал. Једноставно сам приказао оно што се назива терминалски одзив; мали трепћући знак који означава да терминал очекује ваше инструкције.

Ако је верзија ConTeXt система већ инсталирана, појавиће се нешто слично следећем:

```
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.mkx1
mtx-context | current version: 2020.04.30 11:15
```

Последња линија нас обавештава о датуму објављивања инсталиране верзије. Ако је то сувише старо, требало би или да је ажурирамо, или да инсталирамо нову верзију. Препоручујем да се инсталира дистрибуција под називом „ConTeXt Standalone” чија упутства за инсталацију можете да пронађете на [ConTeXt викију](#). Преглед свега овога можете пронаћи у [додатку А](#).

3. **Читац PDF фајлова**, тако да на екрану можемо да видимо резултат нашег експеримента. На Windows и Mac OS увек постоји Adobe Acrobat Reader. Ако подразумевано није инсталиран (или није када сам престао да користим Microsoft Windows пре више од 15 година), али се то уради први пут када покушате да отворите PDF фајл, тако да је највероватније преинсталиран. Linux/Unix системи немају тренутну верзију Acrobat Reader програма, али вам није ни потребна јер је доступно буквално на десетине бесплатних и веома добрих PDF читача. Осим тога, скоро увек је неки од њих подразумевано инсталиран на овим системима. Из разлога брзине и једноставности употребе, мој омиљени је MuPDF; мада има неке лоше стране ако користите друге језике осим енглеског са акцентованим карактерима, и не дозвољава вам да изаберете текст или да документ пошаљете на штампач; то је једноставно читач; али веома брз и лак за употребу. Када су ми потребне могућности које не MuPDF не поседује, обично користим или Okular, или qPdfView. Али још једном, то је ствар укуса: можете изабрати шта год вам одговара.

Можемо да изаберемо свој едитор, свој PDF читач, своју ConTeXt дистрибуцију... Добродошли у свет *слободної libre софтвера*!

2.2 Сам експеримент

Писање изворног фајла

Ако су вам алати поменути изнад већ доступни, потребно је да отворите свој текст едитор и да њиме креирате фајл који ћемо назвати „odmor.tex”. Као садржај фајла ћемо написати ово што следи:

```
% Прва линија документа

\mainlanguage[sr] % Језик = српски

\setuppapersize[S5] % величина папира

\setupbodyfont
  [dejavu,12pt] % Фонт = DejaVu Serif, 12 тачака

\setuphead      % Формат поглавља
  [chapter]
  [style=\bfc]

\starttext % Почетак садржаја документа

\startchapter
  [title=Годишњи одмор...]

Пепсико, Пепсико, хајдемо у Мексико.
Никада, никада, јер ја немам долара.
Ко има долара, купа се на плажи.
Ко нема долара, кући на гаражи!
Пепсико, Пепсико, хајдемо у Мексико.

\stopchapter

\stoptext % Крај документа
```

Док се пише, није битно ако се било шта промени, посебно ако се додаје празни простор или преломи линија. Оно што је битно је да се речи које следе након „\” напишу тачно како је приказано, као и садржај унутар витичастих заграда. Остатак може бити другачији.

Кодирање карактера фајла

Када напишемо ово изнад, фајл треба да сачувамо на диск, и обезбедимо да је кодирање карактера UTF-8. Данас је ово кодирање стандардно. У сваком случају, ако нисмо сигурни, сам едитор нам може показати кодирање, па ако је потребно, можемо и да га променимо. Начин како се то ради, очигледно зависи од самог едитора који се користи. На пример, у GNU Emacs се то ради притиском комбинације тастера CTRL-X, па затим Ентер након кога

слиди 'f, у последњој линији прозора (коју GNU Emacs назива *мини-бафер*) ће се појавити порука која од нас захтева ново кодирање и приказује нам које је текуће кодирање. У осталим едиторима се кодирање обично налази у „Save as” менију.

Када смо проверили да је кодирање исправно и сачували фајл на диск, затварамо едитор и фокусирамо се на анализирање онога што смо написали.

Поглед у садржај нашег првог изворног фајла написаног за ConTeXt

Прва линија почиње карактером „%”. Ово је резервисани карактер који систему ConTeXt говори да не обрађује текст који се налази између тог карактера и краја линије у којој се он налази. Ово је корисно када желимо да напишемо коментар на изворни фајл који само аутор може да чита, пошто он не постаје део финалног документа. У овом примеру сам употребио тај карактер да привучем пажњу на одређене линије, објашњавајући шта оне раде.

Линије које почињу са карактером „\”, још једним од резервисаних карактера у систему ConTeXt који означава да оно што следи представља име команде. Овај пример приказује низ команди које се налазе у ConTeXt изворном фајлу: језик на којем је написан документ, величина папира, фонт који ће се користити за приказ документа и начин на који ће се форматирали поглавља. Касније, у наредним поглављима ћемо видети детаље ових команди, али за сада је битно само да читалац види како изгледају ове команде: оне почињу са „\”, затим долази име команде, па онда између витичастих заграда или великих заграда, у зависности од ситуације, подаци који су команди потребни да произведе жељени ефекат. Између имена команде и великих или витичастих заграда може да стоји празан простор или преломи линија.

У деветој линији нашег примера (бројим само линије које у себи садрже неки текст) налази се важна `\starttext` команда: она систему ConTeXt говори да од те тачке почиње садржај документа; а у последњој линији нашег примера, видимо команду `\stoptext` која говори да је ово место ка којем се документ завршава. Ово су две веома важне команде о којима ћу ускоро имати још по нешто да кажем. Између њих се налази садржај документа, који у овом случају представља шаљиву дечију песмицу. Написао сам је у форми прозе да бисмо видели начин на који систем ConTeXt форматира пасус.

Обрада изворног фајла

За наредни корак, након што смо потврдили да је систем ConTeXt исправно инсталиран на нашем систему, морамо да отворимо терминал у оном директоријуму у који је сачуван фајл „odmor.tex”.

Многи текст едитори нам дозвољавају да компајлирамо документ на којем смо радили без потребе да отварамо терминал. Ипак, *канонска* процедура за обраду документа системом ConTeXt наводи да се то ради из терминала, директним извршавањем програма. Кроз цео овај документ ћу то радити на овај начин (или претпоставити да је урађено тако) из различитих разлога; први је што не могу да знам који едитор користи читалац. Али најважнији је онај што употреба терминала омогућава приступ екранском излазу програма „context” и могуће је да се виде поруке које исписује програм.

Ако је ConTeXt дистрибуција коју смо инсталирали „ConTeXt Standalone”, пре било чега морамо да извршимо *скрипти*у која терминалу говори путању и локацију фајлова који су потребни за извршавање програма ConTeXt. На Linux/Unix системима, то се ради тако што се изврши следећа команда:

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

под претпоставком да смо ConTeXt инсталирали у директоријум под именом „context”.

Везано за извршавање *скрипте* о којем смо управо говорили, погледајте шта у [додатку А](#) пише о инсталацији „ConTeXt Standalone”.

Када се у меморију учитају променљиве потребне да се покрене „context”, можемо да га извршимо. То радимо тако што у терминалу откуцамо

```
$> context odmor
```

Без обзира на то што се изворни фајл назива „odmor.tex”, приметите да смо приликом позивања „context” изоставили екстензију фајла. У случају да смо фајл назвали „odmor.mkiv”, на пример (што обично и радим како бих могао да знам да је тај фајл написан за Mark IV), морали бисмо експлицитно да наведемо екстензију, наводећи „context odmor.mkiv”.

Када у терминалу покренемо „context”, на екрану ће се појавити неколико десетина линија које нам говоре шта ConTeXt ради. Ове информације се појављују брзином коју човек не може да испрати, али то не треба да вас забрине, јер се исте информације чувају и у помоћном фајлу са екстензијом „.log”. Он се генерише у време обраде, па ако је неопходно, касније можемо на миру да га погледамо.

Неколико секунди касније, ако смо изворни фајл написали без неке озбиљне грешке, терминалске поруке ће се завршити. Последња порука нас обавештава колико је трајало компајлирање фајла. Први пут је за компајлирање потребно мало више времена, јер ConTeXt мора да учита у меморију одређене фајлове који му говоре који фонтови се користе, док су за накнадна компајлирања они већ учитани. Када се појави последња порука која обавештава о дужини компајлирања, обрада је завршена. Ако је све прошло како треба, сада ће се у директоријуму у којем смо извршили „context” појавити три додатна фајла:

- odmor.pdf
- odmor.log
- odmor.tuc

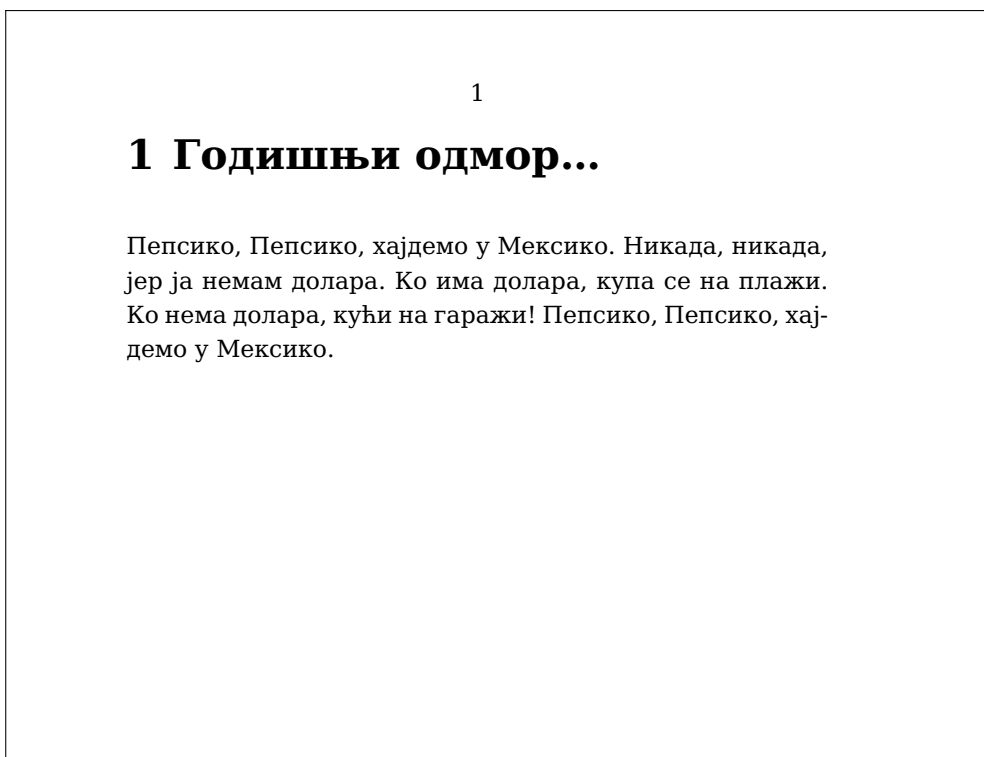
Први од ова три је резултат обраде, или другим речима, коначни форматирани PDF. Други је „.log” фајл у којем се налазе све информације приказане на екран током обраде изворног фајла; трећи је помоћни фајл који ConTeXt генерише током компајлирања и који користи за креирање индекса и унакрсних референци. За сада, ако је све прошло како треба, можемо да обришемо и (odmor.log и odmor.tuc). Ако је било проблема, садржај ових фајлова ће нам помоћи да пронађемо где је узрок и како да дођемо до решења.

Ако нисмо добили ове резултате, узрок је вероватно то што:

- или нисмо исправно инсталирали ConTeXt дистрибуцију, па том случају, када у терминалу покушамо да извршимо команду „context”, видимо поруку „command unknown”.
- или наш фајл није UTF-8 кодиран, па долази до грешке у обради.
- или је можда ConTeXt који је инсталиран на систему Mark II. У овој верзији не можемо да користимо UTF-8 кодирање ако се то експлицитно не наведе у изворном фајлу. Могли бисмо да преправимо изворни фајл тако да се исправно компајлира, али пошто се овај увод односи на Mark IV, нема смисла да се настави рад са Mark II: најбоље би било да се инсталира „ConTeXt Standalone”.
- или смо направили грешку у изворном фајлу док смо писали име команде или уносили податке потребне команди.

Ако након почетка извршавања „context” команде у терминалу крену да се емитију поруке, па се зауставе, а не појави се *одзив*, пре него што наставимо морамо да притиснемо CTRL-X да би прекинули извршавање ConTeXt које је зауставила грешка.

Затим морамо открити шта се догодило и решимо проблем, па наставимо то да радимо све док се компајлирање не обави успешно.



Слика 2.1 Годишњи одмор...

На [слици 2.1](#) видимо садржај фајла „odmor.pdf”. Такође можемо видети да је ConTeXt нумерисао страницу и поглавље, написао текст фонтом који смо тражили. ConTeXt ће подразумевано да прелама речи на крају реда сагласно правилима изабраног језика, а у нашем случају прва линија наводи (`\mainlanguage[sr]`).

Да сумирамо: ConTeXt је трансформисао изворни фајл и генерисао фајл у којем имамо документ форматиран сагласно инструкцијама у изворном фајлу. Нестали су сви евентуални

коментари, и што се тиче команди, оно што сада имамо није име команде, већ резултат извршавања команде.

2.3 Структура фајла нашег примера

У пројекту који се развија само у једном изворном фајлу, структура је веома једноставна и означена командама `\starttext ... \stoptext`. Све што се налази између прве линије фајла и команде `\starttext` се назива *преамбула*. Стварни садржај документа се умеће између команди `\starttext` и `\stoptext`. У нашем примеру се преамбула састоји из три глобалне конфигурационе команде: једна наводи језик нашег документа, (`\mainlanguage`), друга наводи величину страница (`\setuppapersize`) која је у нашем случају „S5” и представља димензије компјутерског екрана, а трећа команда (`\setuphead`) омогућава да конфигуришемо изглед наслова поглавља.

Тело документа се налази између команди `\starttext` и `\stoptext`. Ове команде наводе почетну и крајњу тачку текста који треба да се обради: између њих треба да поставимо сав текст који желимо да ConTeXt обради, уз команде које не треба да утичу на комплетан документ, већ само на одређене делове. За сада, претпоставимо да су `\starttext` и `\stoptext` команде обавезне у сваком ConTeXt документу, мада ћемо касније, када будемо говорили о пројектима у више фајлова (одељак 4.6), видети да постоје неки изузеци.

2.4 Неки додатни детаљи у вези са покретањем команде „context”

Команда „context” којом смо раније почели обраду нашег првог изворног фајла је уствари *LUA скрипта*, што значи да је то мали LUA програм који после обављања одређених провера позива LuaTeX, јер он обрађује изворни фајл.

Команду „context” можемо да позовемо уз разне опције. Оне се наводе непосредно након имена команде, тако што се откуцају две цртице. Ако желимо да наведемо више од једне опције, раздвајамо их размаком. Опција „help” исписује листу свих опција уз кратко објашњење сваке од њих:

```
$> context -help
```

Следе неке од интересантнијих опција:

interface: Као што сам већ напоменуо у уводном поглављу, ConTeXt интерфејс је преведен на разне језике. Подразумевани интерфејс је енглески, међутим, ова опција нам омогућава да користимо холандски (nl), француски (fr), италијански (it), немачки (de) или румунски (ro).

purge, purgeall: Брише помоћне фајлове који су генерисани током обраде.

result=Name: Наводи име које би требало да добије произведени PDF фајл. Подразумевано ће бити исто као и име фајла који се обрађује, са екстензијом .PDF.

usemodule=list: Учитава наведене модуле пре него што покрене ConTeXt (модул је проширење система ConTeXt које није део његовог језгра и које обезбеђује неку додатну могућност).

useenvironment=list: Учитава фајлове окружења пре него што покрене ConTeXt (фајл окружења је фајл са конфигурационим инструкцијама).

version: Приказује верзију система ConTeXt.

help: приказује помоћ у вези са опцијама програма.

noconsole: Спречава слање порука на екран током компајлирања. Међутим, ове поруке се ипак чувају у .log фајл.

nonstopmode: Извршава компајлирање без заустављања у случају грешака. Ово не значи да се грешка не генерише, већ да кад ConTeXt наиђе на грешку, чак и неку коју може да опорави, он наставља са компајлирањем све док не стигне до краја, или док не наиђе на грешку од које не може да се опорави.

batchmode: Комбинација претходне две опције. Извршава се без прекида и спречава испис порука на екран.

Сматрам да у првим корацима учења система ConTeXt није добра идеја да се користе последње три опције, јер када дође до грешке нећемо знати где се она налази или шта ју је изазвало. А верујте ми драги читаоци, пре или касније ћете имати грешке током обраде.

2.5 Управљање грешкама

Током рада у систему ConTeXt неизбежно је да се пре или касније појаве грешке током обраде. Грешке можемо да поделимо у следеће четири категорије:

1. **Грешке у писању.** Дешавају се када направимо грешку у куцању имена команде. У овом случају компајлеру шаљемо наредбу коју не разуме. На пример, када уместо да напишемо команду `\TeX` напишемо `\Tex` тако да је последње слово мало 'x', а како ConTeXt прави разлику између малих и великих слова, он види „TeX” и „Tex” као различите речи; или ако се опције функционисања команде поставе у велике заграде уместо у витичасте заграде, или ако покушамо да употребимо резервисане карактере као да су обични итд.
2. **Грешке изостављања.** ConTeXt има инструкције за почетак задатка чији крај морамо такође експлицитно да означимо; као што је резервисани карактер \$ за укључивање математичког режима који траје све док се не искључи, па заборавимо да га искључимо. Онда се генерише грешка када се наиђе на текст или инструкцију која нема смисла у математичком режиму. Исто је и ако започнемо блок текста резервисаним карактером '{' или командом `\startHешто` па се касније не пронађе експлицитна команда затварања '}' или `\stopHешто`.

3. **Грешке концепције.** Тако зовем грешке које се јаве када се позове команда којој су потребни одређени аргументи, али јој се не доставе, или када није исправна синтакса позива команде.
4. **Грешке ситуације.** Постоје команде које су дизајниране тако да раде само у одређеним контекстима или окружењима и уопште се не препознају ван њих. Ово се посебно дешава у математичком режиму: неке ConTeXt команде раде само док се пишу математичке команде и ако се позову из неког другог окружења, генерисаће грешку.

Шта да радимо када нас „context”, док обрађује, упозори да је дошло до грешке? Очигледно је прва ствар да одредимо шта је грешка. За то је потребно да анализирамо „.log” фајл који је био генерисан за време обраде; мада понекад то није неопходно, јер је грешка такве природе да је тренутно обуставила обраду, па је у том случају порука о грешки видљива у истом терминалу у којем смо и покренули „context”.

```

4
5      \setuppapersize[S5] % величина папира
6
7      \setupbodyfont
8      [dejavu,12pt] % Фонт = DejaVu, 12 тачака
9
10     \setuphead      % Формат поглавља
11     [chapter]
12     [style=\bfc]
13
14 >> \starttext % Почетак садржаја документа
15
16     \startchapter
17     [title=Годишњи одмор...]
18
19     Пепсико, Пепсико, хајдемо у Мексико.
20     Никада, никада, јер ја немам долара.
21     Ко има долара, купа се на плажи.
22     Ко нема долара, кући на гаражи!
23     Пепсико, Пепсико, хајдемо у Мексико.
24
mtx-context | fatal error: return code: 256

```

Слика 2.2 Излаз екрана у случају грешке при компајлирању

На пример, ако смо у нашем тест фајлу „odmor.tex”, грешком уместо `\starttext` написали `\startext` (са једним 't'), што је врло честа грешка, током извршавања команде „context odmor” обрада ће да се прекине и у терминалу ћемо видети информације приказане на [слици 2.2](#). Ту можемо видети да су линије нашег изворног фајла нумерисане и да је у једној од њих, у овом случају линији број 14, између броја и линије текста, компајлер додао „>>” да би означио да је у тој линији наишао на грешку. Фајл „odmor.log” ће нам пружити још трагова. У нашем примеру фајл није тако велики, јер се извор који се компајлира знатно скраћен; али у осталим случајевима може да садржи огромну количину информација. Али морамо да уронимо у њих. Ако текст едитором отворимо „odmor.log” видећемо да се у њему налази све што ConTeXt ради. Морамо да пронађемо линију у којој почиње упозорењем о грешки, а за то можемо употребити функцију претраге едитора. Потражићемо „tex error”, па ће нас то довести до следећих линија:

```

tex error      > tex error on line 14 in file |
                ./odmor.tex: Undefined control sequence \undefined

<line 3.14>
    \startext
    % Почетак садржаја документа

```

Напомена: прва линија која нам говори о грешки у фајлу „odmot.log” је доста дугачка. Да би лепше изгледала, имајући у виду ширину странице, поделио сам је на две линије. Карактер '|' приказује тачку поделе.

Ако обратимо пажњу на три линије поруке о грешки, видећемо да нам прво говоре у којој линији се налази грешка (линија 14) и које је врсте: „Undefined control sequence”, или, што је иста ствар: непознати контролни низ, то јест непозната команда. Наредне линије лог фајла приказују линију 14, подељену на тачки која је изазвала грешку. Тако да нема сумње да се грешка налази у `\starttext`. Читамо пажљиво и уз мало среће и искуства, схватамо да смо написали „starttext” уместо „starttext” (са два 't').

Имајте на уму да су компјутери веома успешни и врло брзи у извршавању инструкција, али веома споро читају наше мисли. А реч „starttext” није иста као „starttext”. Програм зна како да изврши ову другу, али не и прву. Не зна шта да уради са њом.

У другим случајевима грешка неће моћи тако лако да се открије. Посебно када је грешка у томе да је нешто започето, а није наведено место на којем треба да се заврши. Понекада би, уместо да у „.log” фајлу тражимо „tex error”, требало да тражимо звездицу. Овај карактер на почетку линије не означава фаталну грешку, већ пре упозорење. Међутим, упозорења могу бити од помоћи код проналажења грешака.

А ако информације у „.log” фајлу нису довољне, требало би да прођемо кроз главни фајл, део по део, и тражимо грешку. Добра стратегија за то је да се промени локација команде `\stoptext`. Упамтите да ConTeXt прекида обраду текста када наиђе на ову команду. Дакле, ако поставим команду `\stoptext` мање више на половину фајла, па га онда компајлирам, компајлираће се само прва половина фајла; ако се поново јави грешка, онда знам да се она налази у првој половини изворног фајла, а ко се не јави, онда то значи да је у другој половини... па тако, мало по мало, изменом локације команде `\stoptext`, бићемо у могућности да пронађемо место на коме се налази грешка. Када је пронађемо, можемо покушати да је разумемо и исправимо, или ако не разумемо зашто долази до грешке, онда пошто знамо где долази до грешке, можемо покушати да тај део напишемо на другачији начин и спречимо појаву грешке. Ово друго решење може да се примени само ако смо ми аутор. Ако једноставно радимо словослагање нечијег текста, не можемо да га изменимо и морамо наставити да истражујемо све док не откријемо узроке грешке и могући начин да је исправимо.

У пракси, када се системом ConTeXt прави релативно дугачак документ, он се обично повремено компајлира док се уноси, тако да када дође до грешке, биће мање више јасно да се она јавља у новом делу, јер се последњи пут документ успешно компајлирао. Такође ће бити јасније и зашто долази до грешке.

Глава 3

Команде и остали фундаментални концепти система ConTeXt

Садржај: 3.1 ConTeXt резервисани карактери; 3.2 Саме команде; 3.3 Опсег важења команди; 3.3.1 Комадне којима треба или не треба назначити опсег важења; 3.3.2 Команде којима је потребно изричито навођење где почињу и где се завршавају (окружења); 3.4 Опције рада команде; 3.4.1 Команде које могу да раде на неколико различитих начина; 3.4.2 Команде које конфигуришу начин на који раде друге команде (`\setupNesto`); 3.4.3 Подешавање прилагођених верзија подесивих команди (`\defineNesto`); 3.5 Резиме синтаксе команде и опција, употреба великих и витичастих заграда приликом позивања; 3.6 Званична листа ConTeXt команди; 3.7 Дефинисање нових команди; 3.7.1 Општи механизам за дефинисање нових команди; 3.7.2 Креирање нових окружења; 3.8 Остали основни концепти; 3.8.1 Групе; 3.8.2 Димензије; 3.9 Метода за самостално учење система ConTeXt;

Већ смо видели да се у изворном фајлу, као и у самом садржају нашег будућег форматираних документа, налазе инструкције које систему ConTeXt објашњавају како желимо да се наш рукопис трансформише. Ове инструкције можемо да назовемо „команде”, „макрои” или „контролни низови”.

Из угла интерног функционисања система ConTeXt (то јест функционисања система TeX), постоји разлика између *примитива* и *макроа*. Примитива је једноставна инструкција која не може да се разбије у друге једноставније инструкције. Макро је инструкција која се разбија у друге једноставније инструкције, које можда такође могу да се разбију у друге једноставније и тако редом. Већина ConTeXt инструкција су, у ствари, макрои. Из перспективе програмера, разлика између макроа и примитива је важна. Али из перспективе корисника ствар није тако битна: у оба случаја имамо инструкције које се извршавају без потребе да водимо рачуна о начину на који оне функционишу на ниском нивоу. Стога, ConTeXt документација обично говори о *команди* када посматра из угла корисника, а *макроу* када говори из угла програмера. Пошто у овом уводу говоримо само из угла корисника, ја ћу користити било који од ова два израза и посматраћу их као да су синоними.

Команде су налози којима се систему ConTeXt говори да нешто уради; њима ми *контролишемо* ефекат програма. Тако да кнут, отац система TeX, користи израз *контролни низови* када мисли и на примитиве и на макрое, и ја мислим да је то најпрецизнији од свих израза. Користићу га када верујем да је важно направити разлику између *контролних симбола* и *контролних речи*.

ConTeXt инструкције могу у основи да буду или резервисани карактери, или команде у ужем смислу.

3.1 ConTeXt резервисани карактери

Када ConTeXt чита изворни фајл састављен само од текст карактера, а пошто је то чист текст, потребно је на неки начин разликовати текст који треба да се форматира од инструкција које се извршавају над њим. ConTeXt резервисани карактери омогућавају то разликовање. У принципу, ConTeXt ће претпоставити да сваки карактер у фајлу треба да се обради, осим ако није један од 11 резервисаних карактера који треба да се третирају као *инструкција*.

Само 11 инструкција? Не. Постоји само 11 резервисаних карактера; али пошто један од њих, карактер „\”, има функцију конвертовања карактера или више њих који му непосредно следе у инструкцију, онда је у суштини потенцијални број команди практично неограничен. ConTeXt поседује око 3000 команди (кад се саберу команде које постоје само у Mark II, Mark IV и оне заједничке за обе верзије).

Ово су резервисани карактери:

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

ConTeXt их интерпретира на следећи начин:

- \ Ово је за нас најзначајнији карактер: он означава да све што следи непосредно након њега не сме да се интерпретира као текст већ као инструкција. Он се назива „Означавајући (Escape) карактер” или „Означавајући низ” (мада нема никакве везе са „Esc” тастером који се налази на већини тастатура).¹
- % Говори систему ConTeXt да је све оно што следи до краја линије коментар који не сме да се обрађује или укључи у финални форматирани фајл. Уметање коментара у изворни фајл је изузетно корисно. Коментар може помоћи да се објасни зашто је нешто урађено на одређени начин, а то је заиста корисно у завршеним изворним фајловима у смислу каснијих ревизија, када понекад не можемо да се сетимо зашто смо нешто урадили онако како смо урадили; или такође може да послужи као подсетник о нечему што би можда требало да прерадимо. Може чак да се употреби као помоћ за лоцирање одређене грешке у изворном фајлу, јер се постављањем знака коментара на почетак линије она не компајлира, па можемо видети да ли је та линија била узрок грешке; такође може да се користи за чување две различите верзије истог макроа, па да на тај начин добијемо различите резултате након компајлирања; или да спречи компајлирање фрагмента за који нисмо сигурни, а да га не обришемо из изворног фајла, у случају да касније желимо да се вратимо на њега... итд. Једном кад добијемо могућност да изворни фајл садржи текст који само ми видимо и нико други, употреба овог карактера је ограничена само нашом сопственом маштом. Признајем да је ово једна од могућности која ми највише недостаје онда када је једини начин за писање текста употреба текст процесора.
- { Овај карактер отвара групу. Групе су блокови текста на које утичу одређене могућности. О њима ћемо говорити у одељку 3.8.1.
- } Овај карактер затвара групу претходно отворену са {.
- # Овај карактер се користи за дефиницију макроа. Он указује на аргументе макроа. Погледајте одељак 3.7.1 у овом поглављу.

¹ У компјутерској терминологији се тастер који утиче на интерпретацију наредног карактера назива *escape (означавајући) карактер*. За разлику од тога, *escape шасијер* се на тастатурама тако назива јер генерише карактер 27 у ASCII коду, који се у овом кодирању употребљава као *escape* карактер. Данас се употреба *Escape* тастера повезује са идејом отказивања текуће операције.

- ~ Уноси размак у документ који спречава прелом линије, што значи да ће две речи раздвојене карактером ~ увек бити у истој линији. О овој инструкцији и местима на којима би требало да се употреби ћемо говорити у одељку 11.3.1.
- | Овај карактер се користи за означавање да две речи спојене раздвајајућим елементом праве сложену реч која може да се подели на слоге у прву компоненту, али не у другу компоненту. Погледајте одељак 10.4.
- \$ Овај карактер је прекидач за математички режим. Он га укључује ако је био искључен, или га искључује ако је био укључен. Када се налази у математичком режиму, ConTeXt примењује неке фонтове и правила која се разликују од уобичајених, чији је циљ оптимизација писања математичких формула. Мада је писање математике веома битна употреба система ConTeXt, ја о томе нећу детаљно говорити у овом уводу. Пошто сам писац, није ми до тога!
- _ Овај карактер се користи да у математичком режиму означи да је оно што следи исписано у индексу. На пример, да бисмо добили x_1 , морамо да напишемо `x_1`.
- ^ Овај карактер се користи да у математичком режиму означи да је оно што следи исписано у експоненту. На пример, да бисмо добили $(x+i)^{n^3}$ потребно је да напишемо `$(x+i)^{n^3}$`.
- & У ConTeXt документацији пише да је ово резервисан карактер, али се не каже зашто. У Plain TeX овај карактер има у суштини две употребе: користи се да поравна колоне у окружењима основне табеле, а у математичком контексту да оно што следи треба да се третира као обичан текст. У уводном упутству „ConTeXt Mark IV, an Excursion”, мада не пише за шта служи, постоје примери његове употребе у математичким формулама, додуше не онако како се користи у Plain TeX, већ да поравна колоне унутар комплексних функција. Пошто сам ја писац, мислим да не могу обавити додатна тестирања којима могу сазнати за шта се тачно користи овај резервисани карактер.



Може се претпоставити да се при избору карактера који би требало да буду резервисани водило рачуна да то буду карактери доступни на већини тастатура, али они који се обично не користе у писаном језику. Међутим, мада није тако уобичајено, увек постоји могућност да се неки од њих појави у нашим документима, као на пример, када желимо да напишемо да нешто кошта 100 долара (\$100), или да је у Шпанији 2018. године проценат возача преко 65 година старости био 16%. У овим случајевима не смемо директно да пишемо резервисани карактер, већ морамо да користимо команду која ће исправно исписати резервисани карактер у финални документ. Команда за сваки резервисани карактер може да се пронађе у табели 3.1.

Још један начин за добијање резервисаног карактера је командом `\type`. Ова команда шаље свој аргумент у финални документ без било какве обраде, дакле и без интерпретирања. Текст који се прими из `\type` ће у финалном документу да се прикаже фонтом фиксне ширине који је типичан за компјутерске терминале и писаће машине.

Обично бисмо текст који `\type` треба да прикаже окружили витичастим заградама. Међутим, када тај текст и сам садржи отварајуће или затварајуће витичасте заграде, уместо у њих, текст морамо да окружимо са два иста карактера која нису део текста који чини аргумент команде `\type`. На пример: `\type*{*}`, или `\type+{+}`.

Резервисани карактер	Команда која га генерише
\	<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>

Табела 3.1 Писање резервисаних карактера

Ако неки од ових карактера грешком употребимо директно, а не за оно за шта је предвиђен, него зато што смо заборавили да резервисани карактер не може да се користи као обичан, могу да се догоде три ствари:

1. У највећем броју случајева, грешка приликом компајлирања.
2. Добијемо неочекивани резултат. Ово се дешава посебно са „~” и „%”; у првом случају, уместо „~” који смо очекивали у финалном документу, уметнуће се размак; а у другом случају, све након карактера „%” у тој линији ће престати да се обрађује. Неправилна употреба „\” такође може да има неочекивани резултат ако он или карактери непосредно иза творе команду коју ConTeXt разуме. Међутим, неправилна употреба „\” много чешће доводи до грешке у компајлирању.
3. Не долази до проблема: ово се дешава са три резервисана карактера који се углавном користе у математици (`_` `^` `&`): ако се користе ван овог окружења, они се третирају као обични карактери.



Трећа тачка је мој закључак. Истина је да нигде у ConTeXt документацији нисам пронашао где ове резервисане карактере можемо директно да користимо; међутим, у мојим тестовима нисам наишао на грешку када се ово уради; за разлику од, на пример, система L^AT_EX.

3.2 Саме команде

Саме команде увек почињу карактером „\”. У зависности од онога што долази непосредно иза означавајућег низа, прави се разлика између:

- Контролних симбола.** Контролни симбол почиње означавајућим низом („\”) и састоји се само од карактера који нису слова, као на пример „\,”, „\1”, „\ ’” или „\%”. Контролни симбол може бити било који карактер или симбол који није слово у буквалном смислу, укључујући бројеве, знаке interpункције, симболе, па чак и размак. Када је потребно да се у овом документу истакне присуство размака (празног простора), користим симбол `_`. Уствари, као што ћемо ускоро да видимо, „_” (обрнута коса црта након које следи размак) је контролни симбол који се често користи, што ћемо ускоро видети.

Празнина или размак је „невидљиви” карактер који у документима као што је овај представља проблем када постоји потреба да се јасно наведе шта треба да се упише у изворни фајл. Кнут је и сам био свестан проблема, па је у својој књизи „The TeXBook” увео обичај представљања значајних размака симболом „`\`”. Дакле, када је хтео да покаже да су две речи у изворном фајлу раздвојени са два размака, онда је писао „реч1`\``\`реч2”.

- б. **Контролних речи.** Ако је карактер који следи непосредно иза обрнуте косе црте слово у ужем смислу, команда ће представљати *Контролну реч*. Ова група команди је најбројнија од свих и њена особина је да имена команди смеју да се састоје само од слова; бројеви, знаци интерпункције или било који други симболи нису дозвољени. Само мала и велика слова. Имајте на уму да ConTeXt прави разлику између малих и великих слова, што значи да су `\mojakomanda` и `\MojaKomanda` две различите команде. Али би се сматрало да су `\MojaKomanda1` и `\MojaKomanda2` исте команде, јер како '1' и '2' нису слова, они нису део имена команде.



ConTeXt референтно упутство не садржи правила у вези са именима команди, као ни остала „упутства” која су део „ConTeXt Standalone”. Оно што сам навео у претходном пасусу је мој закључак заснован на ономе што се дешава у TeX (где се, на пример, акцентовани карактери који нису део енглеског алфабета не сматрају за „слова”). Ово правило омогућава да се на добар начин објасни нестајање размака након имена команде.

Када ConTeXt чита изворни фајл и наиђе на означавајући карактер („`\`”), он зна да следи команда. Затим чита први карактер након означавајућег низа. Ако он није слово, то значи да је команда контролни симбол и да се састоји само од тог првог симбола. Али с друге стране, ако је први карактер након означавајућег низа слово, ConTeXt ће онда наставити да чита сваки наредни карактер све док не наиђе на први који није слово, па онда зна да је име команде завршено. Ово је разлог што имена команди која су контролне речи не могу да садрже карактере који нису слова.

Када је „не-слово” на крају имена команде размак, претпоставља се да тај размак није део текста који треба да се обради, већ да је уметнут само да се назначи крај имена команде, тако да ConTeXt игнорише овај размак. Ово понашање изненађује ConTeXt почетнике, јер када је ефекат команде да се нешто пише у финални документ, писани излаз команде се везује за наредну реч. На пример, следеће две реченице у изворном фајлу

Познавање TeX помаже да се научи ConTeXt.

Познавање TeX, мада није обавезно, помаже да се научи ConTeXt

редом производе следеће резултате:

Познавање TeXпомаже да се научи ConTeXt.

Познавање TeX, мада није обавезно, помаже да се научи ConTeXt.¹

Приметите како је у првом случају реч „TeX” спојена са речи која јој следи, али у другом случају није. То је зато што је у првом случају прво „не-слово” након имена команде `\TeX` био размак, па је потиснут јер ConTeXt претпоставља да се ту налази само да укаже на крај имена команде, док се у другом случају ту налази запета, а пошто то није размак, не потискује се.

¹ Напомена: у случајевима када у овом уводу треба да се нешто илуструје, писаће се фрагмент кода као и резултат његовог компајлирања користећи једну од две конвенције: понекад су код и резултат његовог компајлирања постављени један уз други у пасусу који се састоји из две колоне; понекад се код исписује **тамно магента** нијансом којом се у овом документу углавном представљају ConTeXt команде, а резултат његовог компајлирања у црвеној боји.

С друге стране, овај проблем се не решава једноставним уметањем додатног размака и писањем, на пример,

Познавање `\TeX` помаже да се научи `\ConTeXt`¹.

неће решити проблем, јер ConTeXt правило (које ћемо видети у одељку 4.2.1) каже да размак апсорбује све празнине и табулаторе који се налазе иза њега. Дакле, када имамо овај проблем (који се на сву срећу не јавља доста често) морамо обезбедити да прво „не-слово” након имена команде није размак. За то постоје два кандидата:

- Резервисани карактери „{ }”. Резервисани карактер „{”, као што сам напоменуо, отвара групу, а „}” затвара групу, тако да низ „{ }” уводи празну групу. Празна група нема ефекта на финални документ, али помаже да ConTeXt зна да име команде испред ње завршило. Или би исто тако могли да креирамо групу око команде у питању, на пример тако што напишемо „{\TeX}”. У сваком случају, резултат ће бити да прво „не-слово” након `\TeX` није размак.
- Контролни симбол „\” (обрнута коса црта иза које следи размак, погледајте напомену на страни 40). Овај контролни симбол умеће размак у финални документ. Да би се исправно схватила логика система ConTeXt, требало би да се одвоји мало времена и да се види шта се дешава када ConTeXt наиђе на контролну реч (на пример `\TeX`) иза које следи контролни симбол (нпр. „\”):
 - ConTeXt наилази на карактер `\` иза којег следи „Т” и како зна да ово долази испред контролне речи, он наставља са читањем карактера све док не наиђе на „не-слово”, тј. када наиђе на `\` карактер који уводи наредни контролни симбол.
 - Једном када сазна да је име команде `\TeX`, покреће команду и штампа `\TeX` у финални документ. Затим се враћа на место где је прекинуо читање да провери који карактер следи непосредно након друге обрнуте косе црте.
 - Проверава да је то размак, тј. „не-слово”, што значи да је контролни низ тачно то, тако да може да га изврши. Он то ради и умеће размак.
 - На крају, још једном се враћа на место где је прекинуо читање (размак који је био контролни симбол) и наставља да обрађује остатак изворног фајла.

Овај механизам сам објаснио детаљније, јер елиминација размака често изненађује почетнике. Међутим, треба имати на уму да је проблем релативно мали, јер се контролне речи обично не пишу директно у финални документ, већ утичу на формати и изглед. За разлику од тога, прилично је често да контролни симболи штампају нешто у финални документ.

Постоји и трећа процедура за спречавање проблема размака, која се састоји у дефинисању (у `\TeX` стилу) сличне команде и укључивањем „не-слова” на крај имена команде. На пример, следећи низ:

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

би креирао команду под именом `\txt`, која би радила исто као и команда `\TeX` и функционисала би као треба само ако се позове са цртицом на крају `\txt-`. Технички, ова цртица није део имена команде, али команда неће радити ако се иза имена не стави цртица. Разлог зашто је то тако се тиче механизма за дефинисање `\TeX` макроа, а то је превише компликовано да се овде објасни. Али функционише: када се ова команда дефинише,

¹ У вези симбола „\”, присетите се напомене на страни 40.

сваки пут када употребимо `\txt-`, ConTeXt је замењује са `\TeX` тако што елиминише цртицу, али користећи је интерно да зна да се име команде завршило, тако да се размак непосредно након команде не брише.

Овај 'трик' неће радити како треба са `\define` командом, ConTeXt команда за дефинисање макроа.

3.3 Опсег важења команди

3.3.1 Комадне којима треба или не треба назначити опсег важења

Многе ConTeXt команде, посебно оне које утичу на форматирање особина фонтова (црни слог, курзив, капитал, итд.), укључују одређену особину која остаје активна све док се не наиђе на наредну команду која је искључује, или која укључује неку другу особину која није компатибилна са њом. На пример, команда `\bf` укључује црни слог, и он остаје активан све док се не наиђе на *некомпатибилну* команду, као што је `\tf`, или `\it`.

Пошто нису дизајниране да се примене само на одређени текст, није потребно да ове врсте команди узимају било какав аргумент. Исто је као да су ограничене само на укључивање некакве функције (црни слог, курзив, безсерифна слова, одређена величина фонта, итд.)

Када се ове команде изврше унутар *групе* (погледајте одељак 3.8.1), оне губе свој ефекат када се група у којој се изврше затвори. Стога, да би ове команде утицале само на одређени део текста, често се генерише група која садржи команду и текст на који желимо да она утиче. Група се креира тако што се њен садржај окружи витичастим заградама. Дакле, следећи текст

У `\it The \TeX Book`, `\sc Кнут` је објаснио све што треба да знате о `\TeX`.

У *The TeXBook*, КНУТ је објаснио све што треба да знате о `\TeX`.

креира две групе, једну која одређује опсег важења `\it` (курзив) команде и друге која одређује опсег `\sc` (капитал) команде.

За разлику од ове врсте команде, постоје и остале које услед ефекта који имају, или из неких других разлога захтевају директно навођење текста на који треба да се примене. У тим случајевима се текст на који команда треба да се примени поставља унутар витичастих заграда *непосредно након команде*. Као пример овога би могли да поменемо команду `\framed`: ова команда исцртава оквир око текста који јој се прослеђује као аргумент, тако да ће

`\framed{Татамата и Дедабрада}`

произвести

Татамата и Дедабрада

Приметите да иако се у првој групи команди (онима које захтевају аргумент) витичасте заграде понекад користе и за одређивање опсега дејства, то није неопходно да би команда функционисала. Команда је дизајнирана тако да се њен ефекат примењује од места на

којем се појави. Зато, када се заградама одређује поље њене примене, команда се поставља *унушар ових заграда*, за разлику од друге групе команди, код којих заграде које окружују текст над којим треба да се примени команда, долазе *након* команде.

У случају `\framed` команде, очигледно је да је за њен ефекат неопходан аргумент – текст на који треба да се примени. У осталим случајевима, од програмера зависи да ли је команда једног или другог типа. Тако, на пример, оно што раде команде `\it` и `\color` је прилично слично: оне на текст примењују особину (формат или боју). Али донета је одлука да се прва програмира без аргумената, а друга као команда са аргументом.

3.3.2 Команде којима је потребно изричито навођење где почињу и где се завршавају (окружења)

Постоје одређене команде које свој опсег важења одређују прецизним назначивањем места на којем треба да почну и места на којем њихово дејство престаје. Тако да ове команде долазе у паровима: једна означава када се команда укључује, а друга када њено дејство треба да престане. „start”, након којег долази име команде се користи да означи почетак акције, а „stop”, након којег такође долази име команде, назначава крај. Тако на пример команда „itemize” постаје `\startitemize` и означава почетак *набрајања*, а `\stopitemize` означава место где престаје набрајање.

За ове парове команди не постоји посебно име у званичној ConTeXt документацији. Референтно упутство и увод их просто зову „start ... stop”. Понекад се називају *окружења*, што је име које L^AT_EX даје сличној врсти конструкција, мада ово има недостатак јер се у систему ConTeXt израз „окружење” користи за нешто сасвим друго (посебну врсту фајла о којем ћемо говорити када у одељку 4.6 будемо причали о пројектима у више фајлова). Чак штавише, пошто је израз *окружење* јасан, а из контекста ће бити лако да се направи разлика да ли говоримо о *командама окружења* или *фајловима окружења*, ја ћу употребљавати овај израз.

Окружења се, дакле, састоје из команде која их отвара или започиње, и друге које их затвара или завршава. Ако изворни фајл садржи команду за отварање окружења које се касније не затвара, обично ће се генерисати грешка.¹ С друге стране, ове врсте грешака се теже проналазе, јер грешка може да се појави много даље иза места на којем се налази команда отварања. Понекад ће нам „.log” фајл приказати линију у којој почиње окружење које није правилно затворено; али у другим приликама ће недостатак затварања окружења значити да ConTeXt погрешно интерпретира одређени пасаж и не у том неисправном окружењу, што значи да „.log” фајл и није од неке помоћи у откривању места проблема.

Окружења могу да се угнезде, што значи да друго окружење може да се отвори унутар постојећег окружења, мада у случајевима када постоје угњеждена окружења, окружење мора да се затвори унутар окружења у којем је било отворено. Другим речима, редослед у којем се окружења затварају мора бити конзистентан са редоследом у којем су била отворена. Верујем да би ово требало да буде јасно из следећег примера:

¹ Мада не и увек; то зависи од врсте окружења и од ситуације у остатку документа. ConTeXt се по овом питању разликује од система L^AT_EX, који је много стриктнији.

```

\startNesto
...
\startNestoDrugo
...
\startJosNestoDrugo
...
\stopJosNestoDrugo
\stopNestoDrugo
\stopNesto

```

У примеру можете видети како је окружење „JosNestoDrugo” отворено унутар окружења „NestoDrugo”, па мора унутар њега и да се затвори. Ако се то уради на неки други начин, генерисаће се грешка приликом компајлирања.

У општем случају, команде дизајниране као *окружења* су оне које имплементирају неку измену која треба да се примени на јединице текста не мање од пасуса. На пример, окружење „narrower” које мења маргине има смисла само када се примени на нивоу пасуса; или окружење „framedtext” које исцртава оквир око једног или више пасуса. Ово последње окружење нам може помоћи да разумемо зашто су неке команде дизајниране као окружења, а неке као индивидуалне команде: ако желимо да уоквиримо једну или више речи у једној линији, употребићемо команду `\framed`, али ако желимо да уоквиримо цео пасус (или неколико пасуса) онда ћемо употребити „framedtext” окружење.

С друге стране, текст који се налази унутар одређеног окружења обично чини *групу* (погледајте [одељак 3.8.1](#)), што значи да ако се унутар окружења пронађе активациона команда, за разлику од свих команди које се примењују на сав текст који следи, ова команда ће се примењивати само до краја окружења у којем се пронађе; а ConTeXt уствари има неименовано *окружење* које почиње командом `\start` (без икаквог текста иза; просто *start*. Зато га и зовем *неименовано окружење*) и завршава се командом `\stop`. Чини ми се да је једина његова функција да креира групу.



Нигде у ConTeXt документацији нисам прочитао да је један од ефеката окружења да групишу свој садржај, али то је резултат мојих тестова са више предефинисаних окружења, мада морам признати да ти тестови нису били превише исцрпни. Просто сам проверио нека насумично изабрана окружења. Међутим, моји тестови показују да би таква тврдња, у случају да је истинита, важила само за нека предефинисана окружења: она креирана командом `\definestartstop` (која се објашњава у [одељку 3.7.2](#)) не креирају било какву групу, осим када током дефинисања новог окружења наведемо и команде које су потребне да се креира група (погледајте [одељак 3.8.1](#)).

Такође је само моја претпоставка да окружења која сам назвао *неименована* (`\start`) окружења постоје само да би се креирала група: она креирају групу, али не знам да ли имају још неку другу примену. Ово је једна од недокументованих команди у референтном упутству.

3.4 Опције рада команде

3.4.1 Команде које могу да раде на неколико различитих начина

Многе команде могу да раде на више од једног начина. У тим случајевима увек постоји предодређени начин функционисања који се може променити навођењем параметара који одговарају жељеном начину рада у заградама након имена команде.

Добар пример овога што сам рекао налазимо у команди `\framed` поменутој у претходном одељку. Ова команда црта оквир око текста који јој се прослеђује као аргумент. Подразумевано је да оквир има висину и ширину текста на који се примењује; али можемо да наведемо и неку другу висину и ширину. Тако да можемо видети разлику између функционисања подразумеване `\framed` команде:

```
\framed{Tatamata}
```



и прилагођене верзије функције:

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



У другом примеру смо унутар великих заграда навели одређену ширину и висину оквира који окружује текст који команда узима као аргумент. Унутар заграда се различите конфигурационе опције раздвајају запетама; размаци па чак и преломи линије (све док нису двоструки прелом линије) између две или више опција се не узимају у обзир, тако да на пример, наредне четири верзије команде дају потпуно исти резултат:

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

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

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

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

Очигледно је да се последња верзија најлакше чита: на први поглед можемо видети колико има опција и како се користе. У примеру као што је овај, са само неколико опција, то можда и не изгледа тако важно; али у случајевима када постоји дугачка листа опција, ако свака од њих заузима посебну линију у изворном фајлу, лакше ће се *разумети* шта изворни фајл тражи да ConTeXt уради, а такође, ако је то неопходно, лакше ће се пронаћи и потенцијална грешка. Зато је 'пожељно' да корисници употребљавају овај последњи формат (или неки сличан) писања команди.

Што се тиче синтаксе конфигурационих опција, погледајте даље у тексту ([одељак 3.5](#)).

3.4.2 Команде које конфигуришу начин на који раде друге команде (`\setupNesto`)

Већ смо видели да команде које имају различите начине функционисања увек поседују подразумевани начин на који раде. Ако се нека од ових команди позива више пута у изворном фајлу, а желимо да изменимо подразумевано понашање сваке од њих, уместо да приликом сваког позива мењамо те опције, много је zgodније и ефикасније да се промени подразумевано подешавање. За то је скоро увек доступна команда чије име почиње са `\setup`, након чега следи име команде чија подразумевана подешавања желимо да променимо.

Команда `\framed` коју смо као пример користили у овом одељку и даље служи као добар пример. Дакле, ако у свом документу имамо доста оквира, али сваки од њих захтева прецизне димензије, онда је најбоље да помоћу `\setupframed` реконфигуришемо начин на који функционише команда `\framed`. Тако ће

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

обезбедити да од сада сваки позив команде `\framed` подразумевано прави оквир ширине 3 и висине 1 центиметар, без потребе да се то наводи сваки пут.

Постоји неких 300 ConTeXt команди које нам омогућавају да конфигуришемо начин функционисања осталих команди. Тако да можемо конфигурисати подразумевано понашање оквира (`\framed`), листи („itemize”), наслова поглавља (`\chapter`), или наслова одељака (`\section`), итд.

3.4.3 Подешавање прилагођених верзија подесивих команди (`\defineNesto`)

Ако наставимо са `\framed` примером, постаје очигледно да ако наш документ користи неколико врста оквира, сваки са другачијим димензијама, онда би било идеално када би могли да *предефинишемо* различите конфигурације команде `\framed`, па да им придружимо одређено име којим сваку од њих можемо по потреби да користимо. У систему ConTeXt то можемо да урадимо `\defineframed` командом, која има следећу синтаксу:

```
\defineframed[Име][Конфигурација]
```

где *Име* представља име додељено одређеној врсти оквира који се конфигурише; а *Конфигурација* је одређена конфигурација придружена том имену.

Ефекат свега овога је да ће се наведена конфигурација придружити имену које смо јој доделили и које ће се за све намене и сврхе понашати као да је нека нова команда. Можемо да га користимо у било ком контексту у којем би могли да користимо и оригиналну команду (`\framed`).

Ова могућност не постоји само за овај конкретни случај команде `\framed`, већ и за многе команде које поседују `\setup` могућност. Комбинација `\defineNesto` + `\setupNesto` је механизам који систему ConTeXt даје његову екстремну снагу и флексибилност. Ако детаљно истражимо шта ради команда `\defineNesto`, видећемо да:

- Она најпре клонира одређену команду која подржава мноштво конфигурација.
- Тај клон придружује имену нове команде.
- Коначно, поставља предодређену конфигурацију клона која се разликује од начина на који је била конфигурисана оригинална команда.

У примеру који смо навели, конфигурисали смо наш специјални оквир у време када смо га креирали. Али исто тако прво можемо да га креирамо, па да га касније конфигуришемо, јер као што сам напоменуо, једном када се креира клон, он може да се користи свуда

где може да се користи и оригинал. Тако на пример, ако смо креирали оквир под именом „MojSpecijalniOkvir”, можемо да га конфигуришемо командом `\setupframed` уз навођење одређеног оквира који желимо да конфигуришемо. У овом случају ће `\setup` команда да узме нови аргумент који представља име оквира који се конфигурише:

```
\defineframed[MojSpecijalniOkvir]
\setupframed
[MojSpecijalniOkvir]
[ ... ]
```

3.5 Резиме синтаксе команде и опција, употреба великих и витичастих заграда приликом позивања

Ево резимеа онога што смо видели до сада. У систему ConTeXt

- Команде у ужем смислу увек почињу карактером „\”.
- Неке команде могу да узму један или неколико аргумената.
- Аргументи који команди говоре *како* да функционише или који на неки начин утичу на функционисање команде се наводе унутар великих заграда.
- Аргументи који команди говоре над којим делом текста треба да делује се наводе унутар витичастих заграда.

Када команда треба да делује само над једним словом, као што је на пример то случај са командом `\buildtextcedilla` (само као пример – ‘ç’ се често користи у каталонском и помало у француском језику), витичасте заграде око аргумента могу да се изоставе: команда ће се применити на први карактер који није размак.

- Неки аргументи нису обавезни, па у том случају можемо да их изоставимо. Али оно што не можемо никада да променимо јесте редослед аргумената који команда очекује.

Аргументи који се наводе унутар великих заграда могу бити разних врста. Углавном:

- Могу имати само једну вредност која је скоро увек нека реч или фраза.
- Могу имати разне опције, па у том случају могу:
 - Да се представе само једном речи која би могла да буде симболичко име (оно чије значење ConTeXt познаје), мера или димензија, број, име неке друге команде, итд.
 - Да се састоје од имена променљивих којима мора бити задата вредност. У овом случају нам званична дефиниција команде (погледајте [одељак 3.6](#)) увек говори коју врсту вредности очекује свака од опција.
- * Када је очекивана вредност опције текст, он може да садржи размаке, а такође и команде. У тим случајевима је понекад zgodno да се вредност постави унутар витичастих заграда.

- * Када је вредност коју очекује опција команда, обично можемо да наведемо више команди као ту вредност, мада је понекад потребно да све команде које представљају вредност опције поставимо у витичасте заграде. Вредност опције такође морамо да поставимо у витичасте заграде када било која од команди које су део вредности узимају опцију унутар великих заграда.

У оба случаја ће различите опције које узима исти аргумент бити раздвојене запетама. Размаци и преломи линија (сви осим двоструких прелома) између различитих опција се игноришу. Такође се игноришу и размаци и преломи линија између различитих аргумената.

- Коначно, у систему ConTeXt се никада не догађа да исти аргумент истовремено добија опције које са састоје од речи и опције које се састоје од променљиве којој се вредност експлицитно додељује. Другим речима, можемо имати овакву опцију

```
\komanda[Opcija, Opcija2, ...]
```

а неку другу као што је

```
\komanda[Promenljiva1=vrednost, Promenljiva2=vrednost, ...]
```

Али никада нећемо наићи на мешавину обе:

```
\komanda[Opcija1, Promenljiva1=vrednost, ...]
```

3.6 Званична листа ConTeXt команди

У ConTeXt документацији постоји један посебно важан документ који приказује листу свих команди и за сваку од њих наводи колико аргумената које врсте очекује, као и различите осмишљене опције и њихове дозвољене вредности. Овај документ се назива „setup-en.pdf” и аутоматски се генерише за сваку нову верзију система ConTeXt. Можете га пронаћи у директоријуму „tex/texmf-context/doc/context/documents/general/qrcs”.

Уствари постоји седам верзија „qrc” документа, по једна за сваки од језика који имају ConTeXt интерфејс: немачки, чешки, француски, холандски, енглески, италијански и румунски. За сваки од ових језика у директоријуму постоје два документа: један се назива „setup-КодЈезика” (где је КодЈезика двословни међународни идентификатор језика), а други се зове „setup-mapping-КодЈезика”. Овај други документ садржи листу команди у абecedном редоследу која наводи *прототип* команде, али без даљих информација омогућим вредностима за сваки аргумент.

Овај документ је од изузетног значаја за учење употребе система ConTeXt, јер у њему можемо да пронађемо да ли одређена команда постоји или не; ово је заиста корисно, имајући у виду комбинацију КОМАНДА (или ОКРУЖЕЊЕ) + setupКОМАНДА + defineКОМАНДА. На пример, ако знам да се празна линија уноси командом `\blank`, могу да пронађем да ли постоји команда под називом `\setupblank` која ми омогућава да је конфигуришем, и друга која ми дозвољава да подезим прилагођену конфигурацију за празне линије, (`\defineblank`).

„setup-en.pdf” је дакле, незаменљив код учења система ConTeXt. Али ја бих заиста више волео да нам он првенствено каже да ли команда ради само у верзији Mark II или Mark IV, а посебно да нам уместо набрајања типова аргумената које узима свака команда каже чему служе ти аргументи. То би у знатној мери умањило недостатке ConTeXt документације. Неке команде дозвољавају и необавезне аргументе које у овом уводу чак

ни не помињем јер не знам чему служе, а пошто нису обавезни нема било какве потребе да их помињем. Ово заиста изазива огромну фрустрацију.

3.7 Дефинисање нових команди

3.7.1 Општи механизам за дефинисање нових команди

Управо смо видели како помоћу `\defineNesto` можемо да клонирамо постојећу команду и развијемо њену нову верзију, која ће за све намене и сврхе функционисати као нова команда.

Уз ову могућност, која је доступна само за неке одређене команде (сигурно поприличан број њих, али не све), ConTeXt поседује општи механизам за дефинисање нових команди који је изузетно моћан, мада је у одређеним применама и прилично сложен. У тексту као што је овај, намењеном почетницима, мислим да је најбоље да се уведе тако што се почне од неких његових најједноставнијих употреба. Најједноставнија од свих је да се фрагменти текста придруже некој речи, па сваки пут када се та реч појави у изворном фајлу, она се замени са текстом повезаним са њом. С једне стране, ово ће нам омогућити да уштедимо доста времена за куцање, а с друге стране, додатна предност је што се умањује могућност грешака у куцању, а у исто време се обезбеђује да је тај текст увек исписан на исти начин.

Хајде да замислимо, на пример, да пишемо трактат о алитерацији у латинским текстовима, у којем често цитирамо латинску реченицу „*O Tite tute Tati tibi tanta tyranne tulisti*” (О Тите Татијусе, ти тиранине, толико тога си наукао на себе!). То је прилично дугачка реченица у којој су две речи личне именице које почињу великим словом и где је, морамо признати, колико год да волимо латинску поезију, лако да се „саплетемо” када је пишемо. У овом случају, могли бисмо да у преамбулу изворног фајла једноставно ставимо следеће:

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

Према оваквој дефиницији, сваки пут када се команда `\Tite` појави у нашем изворном фајлу, она ће се заменити за наведеном реченицом, а налазиће се и унутар знака навода као што их је имала и оригинална дефиниција, чиме смо обезбедили да ће реченица сваки пут имати исти изглед. Могли смо и да је испишемо курзивом, већом величином фонта... како год желимо. Битна ствар је што је потребно да је испишемо само једном, а кроз цео текст ће се исписивати на потпуно исти начин на који смо је написали, без обзира на то колико често се понавља. Такође бисмо могли да креирамо две верзије команде под називом `\Tite` и `\tite`, у зависности од тога да ли је потребно да се реченица испише великим словима или не. Текст замене може бити чист текст, а може да садржи и команде, или да формира математичке изразе у којима је (барем у мом случају) већа вероватноћа да дође до грешке у куцању. На пример, ако је потребно да се израз (x_1, \dots, x_n) често појављује у нашем тексту, могли би да креирамо команду која га представља. На пример

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

тако да кад год се у тексту појави `\xvec`, замениће се са изразом који је придружен тој команди.

У општем случају, синтакса `\define` команде је:

```
\define[БрАргумента]\ИмеКоманде{ТекстЗаЗамену}
```

где се

- **БрАргумента** односи на број аргумената које узима нова команда. Ако не узима ниједан, као у претходним примерима, ово може и да се изостави.
- **ИмеКоманде** односи на име које се даје новој команди. Овде важи опште правило које се тиче имена команди. Име би могло да буде и један карактер који није слово, или једно или више слова без употребе било каквог „не-слово” карактера.
- **ТекстЗаЗамену** састоји из текста који ће да замени име нове команде, сваки пут када се она појави у изворном фајлу.

Могућност дефинисања нових команди са аргументима у дефиницији даје механизму велику флексибилност, јер омогућава да се дефинише променљиви текст замене који ће одредити прослеђени аргументи.

На пример: замислимо да желимо написати команду која креира почетак пословног писма. Врло проста верзија тога би била:

```
\define\ZaglavljePisma{
  \rightaligned{Петар Петровић}\par
  \rightaligned{Консултант}\par
  Зајечар, \date\par
  Драги господине,\par
}
```

али би било добро да имамо верзију команде која би у заглављу исписала име примаоца. То би захтевало да употребимо параметар који новој команди преноси име примаоца писма. Команда би требало да се редифинише на следећи начин:

```
\define[1]\ZaglavljePisma{
  \rightaligned{Петар Петровић}\par
  \rightaligned{Консултант}\par
  Зајечар, \date\par
  Драги г. #1,\par
}
```

Приметите да смо у дефиницију унели две измене. Прво смо између кључне речи `\define` и новог имена команде, уметнули 1 у велике заграде ([1]). То систему ConTeXt говори да команда коју дефинишемо узима један аргумент. Даље, у последњој линији дефиниције команде, написали смо „Драги г. #1,“, користећи резервисани карактер „#“. То наводи да се у тексту замене на месту појављивања „#1“ умеће садржај првог аргумента. Да је било два параметра, „#1“ би се односило на први параметар, а „#2“ на други. Да би се (у изворном фајлу) позвала команда, након имена команде морају да се у витичастим заградама наведу аргументи, сваки у свом пару заграда. Тако да би команда коју смо управо дефинисали, требало да се у тексту изворног фајла позива на следећи начин:

```
\ZaglavljePisma{Име примаоца}
```

На пример: `\ZaglavljePisma{Марко Марковић}`.

Претходну функцију би још могли да унапредимо, јер претпоставља да ће се писмо слати мушкарцу (поставља „Драги господине“), тако да би требало да се наведе још један параметар којим се прави разлика између мушког и женског примаоца. На пример:

```
\define[2]\ZaglavljePisma{
  \rightaligned{Петар Петровић}\par
  \rightaligned{Консултант}\par
  Зајечар, \date\par
  #1\ #2,\par
}
```

тако да би функција могла да се позове, на пример, са

```
\ZaglavljePisma{Драга г.}{Марија Марјановић}
```

мада ово и није баш елегантно (са програмерске тачке гледишта). Боље би било да се симболичке вредности дефинишу као први аргумент (мушкарац/жена; 0/1; м/ж) тако да сам макро према тој вредности изабере одговарајући текст. Али да бисмо објаснили како се то постиже, потребно је да уђемо дубље у материју коју сматрам да читалац-почетник у овом тренутку не може да разуме.

3.7.2 Креирање нових окружења

За креирање новог окружења, ConTeXt обезбеђује `\definestartstop` команду која има следећу синтаксу:

```
\definestartstop[Име][Опције]
```



У званичној `\definestartstop` дефиницији (погледајте [одељак 3.6](#)) постоји додатни аргумент који овде нисам навео, и за који нисам успео да пронађем чему служи. Не објашњавају га ни уводни ConTeXt „Excursion”, ни недовршено референтно упутство. Претпоставио сам да би овај аргумент (који мора да се унесе између имена и конфигурације) могао да буде име неког постојећег окружења које би послужило као почетни модел за ново окружење, али моји тестови показују да је та претпоставка погрешна. Претражио сам ConTeXt мејлинг листу и нисам наишао на било какву употребу овог могућег аргумента.

где

- **Име** представља име које ће добити ново окружење.
- **Конфигурација** омогућава да конфигуришемо понашање новог окружења. Имамо на располагању следеће вредности којима можемо да га подесимо:
 - `before` – Команде које треба да се изврше пре уласка у окружење.
 - `after` – Команде које треба да се изврше након напуштања окружења.
 - `style` – Стил који мора да има текст новог окружења.
 - `setups` – Скуп команди креиран са `\startsetups ... \stopsetups`. Овај увод не објашњава ову команду и њену употребу.
 - `color, inbetween, left, right` – Недокументоване опције које нисам успео да употребим. Из њиховог имена можемо претпоставити шта неке од њих раде, на пример `color`, али из више тестова које сам извршио, дајући неку вредност тој опцији, нисам видео било какву промену унутар окружења.

Ево примера могуће дефиниције окружења:

```
\definestartstop
```

```

[TextWithBar]
[before=\startmarginrule\noindeentation,
 after=\stopmarginrule,
 style=\ss\sl
]

\starttext

Прва два основна закона људске глупости недвосмислено тврде да:

\startTextWithBar

\startitemize[n,broad]

\item Увек и неизбежно потцењујемо број глупих индивидуа на свету.

\item Вероватноћа да је дата особа глупа не зависи од било које
друге карактеристике те исте особе.

\stopitemize

\stopTextWithBar

\stoptext

```

Резултат би био:

Прва два основна закона људске глупости недвосмислено тврде да:

1. *Увек и неизбежно потцењујемо број глупих индивидуа на свету.*
2. *Вероватноћа да је дата особа глупа не зависи од било које друге карактеристике те исте особе.*

Ако желимо да наше ново окружење буде група (одељак 3.8.1), тако да било каква измена уобичајеног начина функционисања система ConT_EXt која се деси унутар окружења престане чим се напусти окружење, морамо да укључимо команду `\bgroup` у опцију „before” и команду `\egroup` у опцију „after”.

3.8 Остали основни концепти

Осим команди, постоје и други појмови који су фундаментални за разумевање логике која одређује функционисање система ConT_EXt. Због своје сложености, неки од њих нису погодни за увод, па у овом документу о њима нећемо говорити; али постоје два појма која би сада требало да испитамо: групе и димензије.

3.8.1 Групе

Група је добро дефинисан фрагмент изворног фајла који ConT_EXt користи као *радну јединицу* (ускоро ћу објаснити шта то значи). Свака група има почетак и крај који експлицитно морају да се назначе. Група почиње:

- Резервисаним карактером „{” или командом `\bgroup`.
- Командом `\begingroup`
- Командом `\start`
- Отварањем одређених окружења (командом `\startNesto`).
- Отварањем математичког окружења (резервисаним карактером „\$”).

а затвара се

- Резервисаним карактером „}” или командом `\egroup`.
- Командом `\endgroup`
- Командом `\stop`
- Затварањем окружења (командом `\stopNesto`).
- Напуштањем математичког окружења (резервисаним карактером „\$”).

Одређене команде такође аутоматски генеришу групу, на пример, `\hbox`, `\vbox` и, у општем случају, команде повезане са креирањем *куџија*¹. Осим ових последњих случајева (групе које одређене команде аутоматски генеришу), начин затварања групе мора бити конзистентан са начином на који је отворена. Ово значи да група која је започета са „{” мора да се затвори са „}”, а група започета са `\begingroup` мора да се затвори са `\endgroup`. Ово правило има само један изузетак, да група започета са „{” може да се затвори са `\egroup`, а да група започета са `\bgroup` може да се затвори са „}”; то у суштини значи да су „{” и `\bgroup` потпуно идентични и могу да се користе једно уместо другог, а слично је и са „}” и `\egroup`.

Команде `\bgroup` и `\egroup` су дизајниране да би били у могућности да отворимо и затворимо групу. Стога, из разлога који се тичу \TeX синтаксе, те групе нису могле да отворе и затворе витичастим заградама, јер би се тако у изворном фајлу генерисале неупарене витичасте заграде, а то би увек изазивало грешку током компјирања.

За разлику од њих, команде `\begingroup` и `\endgroup` не могу да се користе уместо витичастих заграда или `\bgroup ... \egroup` команди јер група која је започета са `\begingroup` мора да се затвори са `\endgroup`. Ове друге команде су дизајниране тако да се омогући детаљнија провера грешака. У општем случају, обични корисници немају потребу да их користе.

Можемо имати угњеждане групе (групу унутар друге групе), па у том случају редослед у којем се групе затварају мора бити конзистентан са редоследом у којем су биле отворене: било која подгрупа мора да се затвори унутар групе у којој је била отворена. Такође могу да постоје и празне групе, генерисане са „{}”. У принципу, празна група нема утицаја на финални документ, али може бити корисна, на пример, да се назначи крај имена команде.

Главна намена група је да обухвате свој садржај у једну целину: дефиниције, формати и доделе вредности које се ураде унутар групе се по правилу „заборављају” чим напустимо групу. Дакле, ако желимо да ConTeXt привремено измени свој уобичајени начин функционисања, најбољи начин да то постигнемо је да креирамо групу, па да унутар ње изменимо то функционисање. Затим, када напустимо групу, све вредности и формати који су важили пре ње ће се вратити. Већ смо видели неке примере овога када смо поменули команде као што су `\it`, `\bf`, `\sc`, итд. Али ово се не дешава само са командама форматирања: група на неки начин *изолује* свој садржај, тако да било каква измена било које од многих интерних

¹ Појам *куџија* је такође основни ConTeXt појам, али се његово објашњење не даје у овом уводу.

променљивих којима ConTeXt константно управља важи само док се налазимо унутар групе у којој је промена направљена. Слично, команда дефинисана унутар групе не постоји ван ње.

Тако да ако обрадимо следећи пример

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

видећемо да када се први пут изврши команда `\A`, резултат одговара њеној почетној дефиницији (`'B'`). Затим смо креирали групу и у њој редефинисали команду `\A`. Ако је сада извршимо унутар групе, команда ће нам вратити нову дефиницију (`'C'` у нашем примеру), али када напустимо групу у којој је команда `\A` била редефинисана и извршимо је још једном, она ће поново да испише `'B'`. Дефиниција направљена унутар групе се „заборавља” чим напустимо ту групу.

Још једна могућа употреба група се тиче команди или инструкција дефинисаних да се примене само на карактер написан иза њих. У овом случају, ако желимо да се команда примени на више карактера, морамо да их поставимо унутар групе. Тако на пример, резервисани карактер `„^”` који, као што већ знамо, конвертује наредни карактер у експонент када се користи унутар математичког окружења; па ако на пример напишемо `„4^2x”` добићемо `„42x”`. Али ако напишемо `„4^{2x}”` добићемо `„42x”`.

Конечно, трећа употреба груписања је да се систему ConTeXt каже да оно што се налази унутар групе третира као целину. То је разлог што је раније у (одељку 3.5) речено да је у неким ситуацијама боље да се садржај неке опције команде стави унутар витичастих заграда.

3.8.2 Димензије

Мада би систем ConTeXt могли савршено добро да користимо без бриге о димензијама, не бисмо могли да искористимо све могућности подешавања ако се не упознамо са њима. Јер типографска перфекција система T_EX и оних изведених из њега у великој мери зависи од тога што систем интерно води рачуна о димензијама. Карактери имају димензије; размак између речи, или између линија, или између пасуса има димензије; линије имају димензије; маргине, заглавља и подножја. Постојаће димензија за скоро сваки елемент стране који може да нам падне на памет.

Димензије се у систему ConTeXt наводе децималним бројем иза кога следи јединица мере. У [табели 3.2](#) су наведене јединице које могу да се употребе.

Прве три јединице у [табели 3.2](#) су стандардне јединице за дужину; прва се користи у неким деловима света енглеског говорног подручја, а остале ван њега или у неким његовим деловима. Остале јединице долазе из света типографије. Последње две, за које нисам навео еквивалент, су релативне јединице мере чија је основа текући фонт. 1 „em” представља

Име	Име у ConTeXt	Еквивалент
Инч	in	1 in = 2.54 cm
Центиметар	cm	2.54 cm = 1 инч
Милиметар	mm	100 mm = 1 cm
Тачка	pt	72.27 pt = 1 инч
Велика тачка	bp	72 bp = 1 инч
Скалирана тачка	sp	65536 sp = тачка
Пика	pc	1 pc = 12 тачака
Дидо	dd	1157 dd = 1238 тачака
Цицero	cc	1 cc = 12 дидоа
	em	
	ex	

Табела 3.2 Јединице мере у систему ConTeXt

ширину карактера „М”, а „ex” представља ширину карактера „х”. Употреба мера везаних за величину фонта омогућава креирање макроа који изгледају једнако добро без обзира на то који извор се користи у датом тренутку. Зато се у општем случају и препоручује њихова употреба.

Уз неколико изузетака, можемо да користимо било коју јединицу мере која нам одговара, јер ће ConTeXt интерно да их конвертује. Али увек када се наведе димензија, обавезно је да се наведе и јединица мере, па чак и ако желимо да задамо меру „0”, морамо да напишемо '0pt' или '0cm'. Између броја и имена јединице можемо, али не морамо да уметнемо размак. Ако јединица има децимални део, можемо да употребимо децимални граничник, или (.) или запету (,).

Мере се обично користе као опција неке команде. Али можемо и директно да доделимо вредност некој интерној мери система ConTeXt само је потребно да знамо њено име. На пример, увлачење прве линије пасуса се у систему ConTeXt интерно контролише променљивом која се назива `\parindent`. Ако јој експлицитно наведемо вредност, променићемо од тада па надаље меру коју користи ConTeXt. Па тако, ако желимо да елиминишемо увлачење прве линије, потребно је само да у изворном фајлу напишемо:

```
\parindent=0pt
```

Такође смо могли да напишемо и `\parindent 0pt` (без знака једнакости) или `\parindent0pt` без размака између имена јединице и вредности.

Међутим, сматра се да директна додела интерној мери „није елегантна”. У општем случају, препоручује се да се употребе команде које контролишу ту променљиву, и да се то уради у преамбули изворног фајла. Ако се тако не уради, добија се изворни фајл у којем се грешке врло тешко отклањају јер се све конфигурационе команде не налазе на истом месту, па је заиста тешко да се достигне одређена конзистентност типографских карактеристика.

Неке од димензија које користи ConTeXt су „еластичне”, то јест, у зависности од контекста, оне имају једну или другу вредност. Ове мере се наводе следећом синтаксом:

```
\ИмеМере plus МаксУвећање minus МаксУмањење
```

На пример

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

Ова инструкција систему ConTeXt говори да димензији `\parskip` (која задаје вертикални размак између пасуса) *нормалну* меру од 3 тачке, али тако да ако композиција странице то захтева, мера може да буде до 5 тачака (3 плус 2) или само 2 тачке (3 минус 1). У овим случајевима ће ConTeXt изабрати размак за сваку страницу између минимално 2 тачке и максимално 5 тачака.

3.9 Метода за самостално учење система ConTeXt

Испоставља се да је огромна количина ConTeXt команди и опција заиста поражавајућа и може да нам остави утисак да никада нећемо успети да радимо у њему на одговарајући начин. Овај утисак може да завара, јер је једна од предности система ConTeXt уједначен начин на који ради са свим својим структурама: ако добро научимо неколико структура, и ако, мање више знамо чему служе остале, биће нам релативно лако да научимо како се користе када нам буде била потребна нека додатна могућност. Стога овај увод посматрам као неку врсту *обуке* која ће да нас приреди за своја сопствена истраживања.

Да бисте креирали документ системом ConTeXt, вероватно је неопходно да познајете само следећих пет основних ствари (могли бисмо их назвати ConTeXt *Тоџ* 5):

1. Како да креирате изворни фајл или пројекат; ово се објашњава у [поглављу 4](#) овог увода.
2. Поставите главни фонт документа и знате основне команде за промену фонта и боја ([поглавље 6](#)).
3. Основне команде за креирање структуре садржаја документа, као што су поглавља, одељци, пододељци, итд. Све ово је објашњено у [поглављу 7](#).
4. Можда и како да управљате окружењем *набрајања*, што је детаљно описано у [одељку 12.3](#).
5. ... још понешто.

Што се тиче осталог, све што је потребно да знате је да је то уопште могуће. Сигурно је да нико неће користити могућност ако не зна да она уопште и постоји. Многе од њих су описане у овом уводу; али првенствено, увек можемо да посматрамо како се систем ConTeXt понаша када наиђе на одређену врсту конструкције:

- Прво ће постојати команда која имплементира могућност.
- Друго, скоро увек и команда која нам омогућава да конфигуришемо и унапред одредимо како ће се задатак извршити; команда чије име почиње са `\setup` и која се обично подудара са основном командом.
- Коначно, обично је могуће да се креира нова команда која обавља сличне задатке, али са другачијом конфигурацијом.

Ако желите сазнати да ли ове команде постоје или не, претражите званичну листу команди (погледајте [одељак 3.6](#)), која ће вас такође упознати са конфигурационим опцијама које могу да се користе са командама. И мада на први поглед имена ових команди могу изгледати *неразумљиво*, убрзо ћемо видети да постоје опције које се понављају у многим командама и које у свим тим командама раде на исти или врло сличан начин. Ако нисмо сигурни у то шта нека опција ради, или како ради, биће довољно да креирамо документ и тестирамо је. Такође можемо да погледамо у обимну ConTeXt документацију. Као што је уобичајено у свету слободног софтвера, „ConTeXt Standalone” садржи изворне фајлове скоро комплетне документације у дистрибуцији. Када желимо да знамо да ли се опција која нас интересује употребљава у било ком од ових изворних фајлова, алат као што је „gтер” (за GNU Linux системе) нам може помоћи да их све претражимо и видимо на конкретном примеру како се опција користи и шта ради.

Ово је начин на који је замишљено учење система ConTeXt: увод детаљно објашњава пет (тачније четири) аспекта које сам истакао, и још више: док читамо, у глави ћемо формирати јасну слику редоследа: *команда која извршава задатак – команда која конфигурише претходну – команда која омогућава креирање сличне команде*. Такође ћемо научити и неке од основних структура система ConTeXt и знаћемо за шта се оне користе.

Глава 4

Изворни фајлови и пројекти

Садржај: 4.1 Кодирање изворних фајлова; 4.2 Карактери у изворном фајлу које ConTeXt третира на посебан начин; 4.2.1 Размаци и табови; 4.2.2 Преломи линија; 4.2.3 Линије/црте; 4.3 Прости пројекти и пројекти са више фајлова; 4.4 Структура изворног фајла у простим пројектима; 4.5 Рад са више фајлова у TeX стилу; 4.5.1 Команда `\input`; 4.5.2 `\readfile` и `\readfile`; 4.6 ConTeXt пројекти у ужем смислу; 4.6.1 Фајлови *окружења*; 4.6.2 Компоненте и производи; 4.6.3 Пројекти у ужем смислу; 4.6.4 Заједнички аспекти окружења, компоненти, производа и пројеката;

Као што већ знамо, када радимо у систему ConTeXt увек почињемо текст фајлом у којем се, уз садржај финалног документа, налази и већи број инструкција које наводе трансформације помоћу којих ConTeXt из изворног фајла генерише коректно форматирани излазни документ у PDF формату.

Имајући у виду читаоце који су до сада познавали само рад у текст процесорима, мислим да вреди уложити нешто времена у сам изворни фајл. Или пре, изворне фајлове, јер понекада постоји само један изворни фајл, а некада користимо већи број изворних фајлова да дођемо до финалног документа. У овом последњем случају говоримо о „пројектима са више фајлова”.

4.1 Кодирање изворних фајлова

Изворни фајл(ови) мора(ју) бити текст фајл(ови). У компјутерској терминологији, овако се назива фајл који садржи само читљиви текст и који не садржи никакав бинарни код. Ови фајлови се такође називају и *чисти* *текст* или *чисти* *текст* фајлови.

Пошто компјутерски системи интерно обрађују само бинарне бројеве, текст фајл се уствари састоји од *бројева* који представљају *карактере*. За повезивање броја са карактером се користи *табела*. За текст фајлове постоји више могућих табела. Израз *кодирање текста фајла* се односи на одређену табелу мапирања карактера коју користи дати текст фајл.

Постојање различитих табела кодирања текст фајлова је последица саме историје компјутерске науке. У раним фазама развоја, када је капацитет меморије и складишта компјутерских уређаја био скроман, одлучено је да се користи табела под именом ASCII (што је акроним од „*American Standard Code for Information Interchange*”) која је дозвољавала само 128 карактера и успоставио ју је Амерички комитет за стандарде 1963. године. Очигледно је да 128 карактера није довољно да се представе сви карактери и симболи који се користе у свим светским језицима; али је било више него довољно да се представи енглески језик који, од свих западних језика има најмање карактера, јер не употребљава дијакритике (акценте и остале знаке изнад, испод, или кроз остала слова). Предност употребе ASCII табеле је што текст фајлови заузимају врло мало простора, јер 127 (највећи број у табели) може да се представи бинарним бројем од 7 цифара, а први компјутери су користили бајт за мерење меморије, односно бинарни број са 8 цифара. Било који карактер у табели је могао да се смести у један бајт. Пошто бајт има 8 цифара, а ASCII користи само 7, чак је остало простора да се додају у неки други карактери којима могу да се представе остали језици.

Али када се употреба компјутера проширила, постало је очигледно да ASCII није адекватан, па је дошло до развоја *алтернативних* табела у којима су се налазили карактери ван енглеског алфабета, као што је шпанско 'ñ', акцентовани самогласници, каталонско или француско 'ç', итд. С друге стране, није било почетног договора како би те ASCII *алтернативне* табеле требало да изгледају, тако да су различите специјализоване компјутерске компаније постепено саме решавале проблем. Из тог разлога, не само да су развијене одређене табеле

за различите језике или групе језика, већ и различите табеле према компанији која их је креирала (Microsoft, Apple, IBM, итд.).

Идеја да се креира једна табела која би обухватила све језике се појавила тек са повећањем компјутерске меморије и смањењем цене уређаја за складиштење. Али да истакнемо, уствари се није креирала једна табела која садржи све карактере, већ стандардно кодирање (под називом Уникод) уз различите начине за његово представљање (UTF-8, UTF-16, UTF-32, итд.) Од свих ових система, UTF-8 је на крају постао *де факто* стандард којим може да се представи практично било који живи језик, као и многи већ изумрли језици и бројни додатни симболи. Сви они користе бројеве променљиве дужине (између 1 и 4 бајтова), што у суштини помаже да се оптимизује величина текст фајлова. Ова величина се није *значајно* увећала у односу на текст фајлове који користе чисти ASCII.

Све док се није појавио \LaTeX , системи засновани на \TeX – који су такође рођени у САД, па су зато користили енглески као свој природни језик – претпостављали су да је кодирање чисти ASCII; а ако сте желели да користите неко друго кодирање, морали сте на неки начин да га наведете у изворном фајлу.

ConTeXt Mark IV претпоставља да је кодирање UTF-8. Ипак, на мало старијим компјутерским системима се можда још увек као подразумевано кодирање користи неко друго. Нисам потпуно сигуран које подразумевано кодирање користи Windows, када се има у виду стратегија компаније Microsoft да скривањем сложености дође до шире публике (али мада је скривена, то не значи да је елиминисана!). Нема много доступних информација (или можда ја нисам успео да их пронађем) у вези са системом кодирања који подразумевано користи.

У сваком случају, које год да је подразумевано кодирање, сваки текст едитор вам омогућава да фајл сачувате у жељеном кодирању. Изворни фајлови које треба да обради ConTeXt Mark IV морају да се сачувају у UTF-8, осим, наравно, када постоји веома добар разлог да се употреби неко друго (мада не могу да смислим ниједан валидан разлог за то).

Ако желимо да пишемо фајл у неком другом кодирању (можда неки стари фајл) можемо да

- а. Конвертујемо фајл у UTF-8, што се препоручује, и за то постоје разни алати; на Linux систему, на пример, команде `iconv` или `recode`.
- б. Наведемо у изворном фајлу да кодирање није UTF-8. Да бисмо то урадили, морамо да употребимо команду `\enableregime`, чија је синтакса:

`\enableregime[Кодирање]`

где се *Кодирање* односи на име под којим систем ConTeXt зна које је стварно кодирање фајла у питању. У [табели 4.1](#) ћете пронаћи различита кодирања и имена под којим их зна систем ConTeXt.

ConTeXt Mk IV снажно препоручује употребу UTF-8. И ја се слажем са овом препоруком. Од сада па надаље кроз овај увод, можемо претпоставити да је кодирање увек UTF-8.



Уз команду `\enableregime` ConTeXt има и команду `\useregime` која нам омогућава да као њен аргумент наведемо код за једно или остала кодирања. Нисам пронашао никакве информације о овој команди, нити како се она разликује од `\enableregime`, већ само неке примере употребе. Претпостављам да је `\useregime` дизајнирана за сложене пројекте који користе много изворних фајлова, уз очекивање да немају сви исто кодирање. Али то је само нагађање.

Кодирање	Име(на) у ConTeXt	Напомене
Windows CP 1250	cp1250, windows-1250	Западноевропско
Windows CP 1251	cp1251, windows-1251	Ћирилично
Windows CP 1252	cp1252, win, windows-1252	Западноевропско
Windows CP 1253	cp1253, windows-1253	Грчко
Windows CP 1254	cp1254, windows-1254	Турско
Windows CP 1257	cp1257, windows-1257	Балтичко
ISO-8859-1, ISO Latin 1	iso-8859-1, latin1, il1	Западноевропско
ISO-8859-2, ISO Latin 2	iso-8859-2, latin2, il2	Западноевропско
ISO-8859-15, ISO Latin 9	iso-8859-15, latin9, il9	Западноевропско
ISO-8859-7	iso-8859-7, grk	Грчко
Mac Roman	mac	Западноевропско
IBM PC DOS	ibm	Западноевропско
UTF-8	utf	Уникод
VISCII	vis, viscii	Вијетнамско
DOS CP 866	cp866, cp866nav	Ћирилично
KOI8	koi8-r, koi8-u, koi8-ru	Ћирилично
Mac Cyrillic	maccyr, macukr	Ћирилично
Остала	cp855, cp866av, cp866mav, cp866tat, ctt, dbk, iso88595, isoir111, mik, mls, mnk, mos, ncc	Разна

Табела 4.1 Главна кодирања у систему ConTeXt

4.2 Карактери у изворном фајлу које ConTeXt третира на посебан начин

Специјални карактери је име којим ћу називати групу карактера који се разликују од *резервисаних карактера*. Као што смо видели у одељку 3.1, то су они који за систем ConTeXt имају посебно значење, тако да се не могу директно употребљавати као карактери у изворном фајлу. Уз њих, постоји још једна група карактера која, мада их ConTeXt третира као такве када у изворном фајлу наиђе на њих, они се не третирају посебним правилима. У овој групи се налазе размаци, табови, преломи линија и цртице.

4.2.1 Размаци и табови

Табови и размаци се у изворном фајлу за све сврхе третирају на исти начин. ConTeXt ће таб карактер (Tab тастер на тастатури) трансформисати у празан простор. А размаци се апсорбују у било који други празан простор (или таб) који се налази непосредно иза њих. Тако да нема апсолутно никакве разлике ако у изворном фајлу напишете

Станлио и Олио.

или

Станлио и Олио.

ConTeXt сматра да су ове све реченице потпуно исте. Дакле, ако између речи желимо да унесемо додатни размак, морамо да употребимо неке ConTeXt команде које то раде. Обично ће радити са „\ ”, што значи карактер \ иза којег се налази размак. Али постоје и остале процедуре у вези са хоризонталним размаком које ћемо представити у поглављу 10.3.

Апсорпција узастопних размака нам омогућава да изворни фајл пишемо тако да визуелно истакнемо делове које желимо да нагласимо, једноставно повећавајући или умањујући употребљено увлачење, и да не бринемо јер знамо да то неће уопште утицати на финални документ. Дакле, у следећем примеру

```
Музичка група из Мадрида с краја седмдесетих {\em La Romántica
Banda Local} је писала песме еклетичког стила које је било врло тешко
ставити у неку категорију. На пример, у својој песми „El Egipcio” су рекли:
\quotation{\em Esto es una farsa más que una comedia, página muy seria
de la histeria musical; sueños de princesa, vicios de gitano pueden en
su mano acariciar la verdad}}, мешајући речи, фразе, просто зато што
поседују унутрашњи ритам (comedia-histeria-seria, gitano-mano).
```

можемо видети да су неке линије мало увучене удесно. То су линије које су део фрагмената који би требало да се појаве у курзиву. Постојање ових увлачења помаже (аутору) да види где се курзив завршава.

Неко би помислио, каква збрка! Морам ли да се замарам увлачењем линија? Истина је да ово посебно увлачење аутоматски ради мој едитор (GNU Emacs) када уређујем ConTeXt изворни фајл. То је та врста мале помоћи услед које бирате да радите са неким едитором, а не са неким другим.

Правило апсорпције размака се примењује само на узастопне размаке у изворном фајлу. Дакле, ако се у изворни фајл између два размака постави празна група („{}”), мада она у финалном фајлу неће произвести ништа, њено присуство ће обезбедити да два размака нису узастопна. На пример, ако напишемо „Станлио {} и Олио”, добићемо „**Станлио и Олио**”, где, ако пажљиво погледате, између прве две речи постоје два узастопна размака.

Исто се дешава и са резервисаним карактером „~”, мада је његов ефекат да генерише размак чак и ако то није: размак иза кога дође ~ неће апсорбовати наредне, а неће ни размак након ~.

4.2.2 Преломи линија

Када линија достигне максималну ширину, у већини едитора се аутоматски уноси прелом линије. Прелом линије можемо такође и експлицитно да унесемо тако што притиснемо тастер „Ентер” или „Return”.

ConTeXt примењује следећа правила у вези са преломом линије:

- а. За све намене, један прелом линије је исто што и размак. Дакле, ако се непосредно испред или иза прелома линије нађе размак или таб, прелом линије или први размак ће их апсорбовати, а у финални документ ће се уметнути прости размак.
- б. Два или више узастопна прелома линије креирају прелом пасуса. У овом смислу, сматра се да су два прелома линије узастопна ако се између првог и другог не налази ништа друго осим размака или табова (јер их први прелом линије апсорбује); што укратко значи да једна или више узастопних линија које су у изворном фајлу потпуно празне (без икаквих карактера или само са размацима или табовима) постају прелом пасуса.

Запазите да сам рекао „два или више узастопна прелома линије”, па затим „једна или више празних узастопних линија”, што значи да ако желимо да повећамо размак између пасуса,

то не радимо једноставним уметањем још једног прелома линије. За то морамо да употребимо команду која повећава вертикални размак. Ако нам треба само једна додатна линија раздвајања, можемо да употребимо команду `\blank`. Али постоје и остале процедуре за повећање вертикалног размака. Указујем на одељак 11.2.

Пошто прелом линије постаје размак, у неким приликама може да се јави неки нежељени и неочекивани размак. Посебно када пишемо макрое, јер је тада лако да се размак „ушуња” а да то не приметимо. Да бисмо то спречили, можемо употребити резервисани карактер „%” који, као што знамо, тамо где се појави спречава обраду линије, што повлачи да се не обрађује ни прелом на крају реда. Тако на пример, команда

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

која свој први аргумент исписује у курсиву, други у црном слогу, а трећи капиталом би уметнула размак између сваког од ових аргумената, док

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

неће уметнути никакав размак између њих, јер резервисани карактер % спречава да се прелом линије обради, тако да он не постаје размак.

4.2.3 Линије/црте

Црте су добар пример разлике између компјутерске тастатуре и штампаног текста. На нормалној тастатури, обично постоји само један карактер за црту (или линију у типографском изразу) који зовемо цртица или („-”); али штампани текст користи до четири различите дужине линија:

- Кратке линије (цртице), као оне које служе за раздвајање речи на крају реда (-).
- Линије средње дужине (ен црте или ен линије), мало дуже од претходних (—). Имају већи број примена, у неким европским језицима (не толико у енглеском) означавају почетак линије дијалога, или раздвајају мањи од већег броја опсега у датумима или страницама; „стр. 12–33”.
- Дугачке линије (ем црте или ем линије) (—), користе се уместо заграда, за укључивање једне реченице унутар друге.
- Минус знак (—) за представљање одузимања или негативног броја.

Данас је у UTF-8 кодирању доступно све горе наведено и још више. Али пошто још увек не могу све да се генеришу притиском на један тастер на тастатури, није тако једноставно да се уметну у изворни фајл. \TeX је на срећу увидео потребу да се у финални документ умеће више врста линија/црта него што може да се произведе тастатуром, па је дизајнирао просту процедуру за уметање. \ConTeXt је ту процедуру проширио додавањем команди које генеришу разне врсте линија. За генерисање четири врсте линија можемо употребити два приступа: било обичан \ConTeXt начин употребом команде, или директно са тастатуре. Ове процедуре су приказане у [табели 4.2](#):

Врста линије	Изглед	Написана директно	Команда
Цртица	-	-	<code>\hyphen</code>
Ен линија	—	—	<code>\endash</code>
Ем линија	—	—	<code>\emdash</code>
Знак минус	—	\$-\$	<code>\minus</code>

Табела 4.2 Линије/црте у ConTeXt

Имена команди `\hyphen` и `\minus` су она која са обично користе у енглеском језику. Мада их многи у штампарској индустрији називају 'линије', TeX појмови, тачније `\endash` и `\emdash` су такође уобичајени у словослагачкој терминологији. „ен” и „ем” су имена мерних јединица које се користе у типографији. „Ен” представља ширину карактера 'n', док „ем” представља ширину карактера 'm' у фонту који се тренутно користи.

4.3 Прости пројекти и пројекти са више фајлова

У систему ConTeXt можемо да користимо само један изворни фајл у коме се налази комплетан садржај финалног документа као и сви детаљи везани за њега, па у том случају говоримо о „простим пројектима”, или супротно од тога, могли бисмо да употребимо више изворних фајлова који деле садржај финалног документа, па у том случају говоримо о „пројектима са више фајлова”.

Ситуације у којима је уобичајено да се ради са више изворних фајлова су следеће:

- Ако пишемо документ на којем сарађује више аутора, од којих сваки пише по један део на којем ради само он; на пример, ако пишемо фестшрифт са прилозима различитих аутора, или број неког журнала, итд.
- Ако је документ на којем радимо, просто речено велики, тако да се компјутер успори када га уређујемо или када га компајлирамо; у том случају, подела материјала на неколико изворних фајлова у знатној мери убрзава компајлирање сваког дела.
- Исто тако, ако смо написали више макроа које желимо да применимо у неким (или свим) нашим документима, или ако смо генерисали шаблон који контролише или стилизује наше документе и хоћемо да га применимо на сваки документ, итд.

4.4 Структура изворног фајла у простим пројектима

Структура простих пројеката који се развијају само у једном изворном фајлу је веома једноставна и заснива се око „text” окружења које, у суштини, мора да се појави у том истом фајлу. Разликујемо следеће делове фајла:

- **Преамбула документа:** све од прве линије фајла до почетка „text” окружења (`\starttext`).
- **Тело документа:** ово је саджај „text” окружења; или другим речима, све што се налази између `\starttext` и `\stoptext`.

```
% Прва линија документа

% Област преамбуле:
% Садржи глобалне конфигурационе команде
% документа

\starttext % Овде почиње тело документа

...
... % Садржај документа
...

\stoptext % Крај документа
```

Слика 4.1 фајл који садржи прости пројекат

На [слици 4.1](#) видимо веома прост изворни фајл. Потпуно све испред команде `\starttext` (која је на слици у линији 5, ако се броје само линије са текстом), чини преамбулу; све између `\starttext` и `\stoptext` чини тело документа. Било шта након `stoptext` се игнорише.

Преамбула се користи за уметање команди које треба да утичу на документ као целину, односно оне које одређују његову глобалну конфигурацију. Уопште није обавезно да се у преамбули напише било каква команда. Ако ту нема команди, ConTeXt ће усвојити подразумевану конфигурацију која није баш детаљно развијена, али ће послужити за многе документе. У добро планираним документима, преамбула ће садржати све команде које утичу на документ као на целину, као што су макрои и прилагођене команде које ће се користити у изворном фајлу. У типичној преамбули, то би могло да изгледа овако:

- Назначавање главног језика документа (погледајте [одељак 10.5](#)).
- Назначавање величине папира ([одељак 5.1](#)) и распореда стране ([одељак 5.3](#)).
- Особине главног фонта документа ([одељак 6.3](#)).
- Прилагођавања `section` команди које ће се користити ([одељак 7.4](#)) и, ако је потребно, дефиницију нових `section` команди ([одељак 7.5](#)).
- Распоред заглавља и подножја ([одељак 5.6](#)).
- Подешавања за наше нове личне макрое ([одељак 3.7](#)).
- Итд.

Преамбула је намењена за глобалну конфигурацију документа; тако да ту не би требало да буде ништа што се тиче *садржаја* документа, или текста који се обрађује. У теорији, сав текст за обраду се у преамбули игнорише, мада понекад, ако је ту, изазваће грешку компајлирања.

Тело документа, окружено командама `\starttext` и `\stoptext` садржи стварни садржај, што значи текст који се обрађује, заједно са ConTeXt командама које не би требало да утичу на цео документ.

4.5 Рад са више фајлова у \TeX стилу

Да би омогућио рад са више изворних фајлова, \TeX је понудио примитиву која се зове `\input`. Та примитива такође функционише и у систему \ConTeXt , мада он прихвата и две специфичне команде које донекле усавршавају начин на који функционише `\input`.

4.5.1 Команда `\input`

Команда `\input` умеће садржај фајла који јој се проследи као аргумент. Њен формат је:

`\input ИмеФајла`

где је *ИмеФајла* име фајла који треба да се уметне. Обратите пажњу да име не мора да се постави унутар витичастих заграда, мада се неће јавити грешка чак и ако се то уради. Међутим, име никада не би требало поставити у велике заграде. Ако је екстензија фајл „.tex”, може да се изостави.

Када \ConTeXt компајлира документ и наиђе на `\input` команду, он тражи наведени фајл и наставља компајлирање као да је тај фајл био део фајла који га читава. Када заврши компајлирање, он се враћа у оригинални фајл и наставља са места одакле је ушао у читавани фајл; тако да је резултат, практично, као да је садржај фајла наведеног у команди `\input` уметнут на место на којем се она налази. Фајл који се позива командом `\input` мора имати важеће име у нашем оперативном систему и не сме да садржи размаке. \ConTeXt ће га потражити у радном директоријуму, па ако га ту не пронађе, потражиће га у директоријумима наведеним у \TeX ROOT променљивој окружења. Ако се фајл на крају не пронађе, вратиће се грешка компајлирања.

Најчешћа употреба `\input` команде изгледа овако: запише се фајл, назовимо га „glavni.tex”, који ће се користити као контејнер за позивање `\input` командом разних фајлова који чине наш пројекат. Ово је приказано у следећем примеру:

```
% Опште конфигурационе команде:

\input MojaKonfiguracija

\starttext

\input NaslovnaStrana
\input Predgovor
\input Glava1
\input Glava2
\input Glava3

...

\stoptext
```

Запазите како смо за општу конфигурацију документа позвали фајл „MojaKonfiguracija.tex” за који се претпоставља да садржи глобалне команде које желимо да применимо. Затим, између команди `\starttext` и `\stoptext` позивамо неколико фајлова у којима се налази садржај различитих делова нашег документа. Ако у неком тренутку пожелимо да убрзамо процес компајлирања, потребно је да изоставимо компајлирање неких фајлова, па је

потребно само да на почетак линије која позива одређени фајл ставимо знак коментара. На пример, ако пишемо треће поглавље и желимо да га компајлирамо једноставно само да проверимо има ли у њему грешака, не морамо да компајлирамо остатак, па можемо да напишемо:

```
% Опште конфигурационе команде:

\input MojaKonfiguracija

\starttext

% \input NaslovnaStrana
% \input Predgovor
% \input Glava1
% \input Glava2

\input Glava3

...

\stoptext
```

па ће се компајлирати само Глава 3. С друге стране, запазите да измена редоследа поглавља значи једноставно измену редоследа линија које их позивају.

Када у пројекту који се састоји из више фајлова изузмемо неки фајл из процеса компајлирања, добијамо на брзини обраде, али резултат тога је да све референце из дела који се не компајлира а указују на остале делове који још увек нису комајлирани неће више радити. Погледајте [одељак 9.2](#).

Важно је да буде јасно да када радимо са командом `\input`, само главни фајл, онај из којег се позивају сви остали, сме да садржи команде `\starttext` и `\stoptext`, јер ако се налазе и у осталим фајловима, јавиће се грешка компајлирања. С друге стране, ово значи да не можемо директно да компајлирамо појединачне фајлове који чине пројекат, већ је неопходно да се они компајлирају из главног фајла, то јест оног који наводи основну структуру документа.

4.5.2 `\ReadFile` и `\readfile`

Као што смо управо видели, ако ConTeXt не пронађе фајл који се позива командом `\input`, јавиће се грешка. У ситуацији када желимо да уметнемо фајл само ако постоји, а дозвољава се и могућност да фајл можда не постоји, ConTeXt пружа варијацију команде `\input`. То је

`\ReadFile{ИмеФајла}`

Ова команда је слична команди `\input` у сваком погледу, осим што у случају да се фајл не пронађе, компајлирање се наставља и не јавља се никаква грешка. Такође се разликује од команде `\input` и по синтакси, јер знамо да за `\input` није обавезно да се име фајла постави у витичасте заграде. Али са `\ReadFile` је то обавезно. Ако не употребимо витичасте заграде, ConTeXt ће мислити да је име фајла који треба да нађе исто као и први карактер који следи иза команде `\ReadFile`, уз екстензију `.tex`. Тако да ако, на пример, напишемо

`\ReadFile MojFajl`

ConTeXt ће разумети да фајл који треба да прочита има име „`M.tex`”, јер је карактер непосредно након команде `\ReadFile` (изузимајући размаке који се, као што знамо, игноришу

на крају имена команде) 'М'. Пошто ConTeXt у општем случају неће пронаћи фајл под именом „M.tex”, а \ReadFile не генерише грешку ако не пронађе фајл, ConTeXt ће наставити компајлирање након 'М' у „MojFajl”, па ће да уметне текст „ojFajl”.

\readfile је софистициранија верзија команде \ReadFile чији је формат

\readfile{ИмеФајла}{ТекстАкоПостоји}{ТекстАкоНеПостоји}

Први аргумент је сличан аргументу за \ReadFile: то је име фајла унутар витичастих заграда. Други аргумент је текст који се исписује у случају да фајл постоји, пре него што се садржај фајла уметне. Трећи аргумент је текст који се исписује ако се наведени фајл не пронађе. То значи да у зависности од тога да ли се пронађе фајл наведен као први аргумент, извршавају се други (ако фајл постоји), или трећи аргумент (ако фајл не постоји).

4.6 ConTeXt пројекти у ужем смислу

Трећи механизам који систем ConTeXt нуди за рад на пројектима са више фајлова је сложенији и комплетнији: он почиње тако што се прави разлика између фајлова пројекта, фајлова производа, фајлова компоненти и фајлова окружења. Да бисмо разумели међусобне везе и функцију сваког од ових типова, сматрам да је најбоље објаснити сваки од њих појединачно:

4.6.1 Фајлови окружења

Фајл окружења је фајл који чува макрое и конфигурације специфичних стилова намењених да се примене на неколико докумената, било да су они потпуно независни документи, било да су делови неког сложеног документа. Дакле, фајл окружења може да садржи све што бисмо иначе писали испред \starttext; то јест: општу конфигурацију документа.

За ове врсте фајлова сам задржао израз „фајлови окружења” како се не бих удаљио од званичне ConTeXt терминологије, мада верујем да би погоднији израз вероватно био „фајлови формата” или „фајлови глобалне конфигурације”.

Као и сви ConTeXt изворни фајлови, фајлови окружења су текст фајлови који подразумевају да је екстензија „.tex”, мада ако то желимо, можемо да је променимо, највероватније на „.env”. Међутим, у систему ConTeXt се ово обично не ради. Најчешће се фајл окружења препознаје тако што почиње или се завршава са 'env'. На пример: „MojMakroi.env.tex” или „env_MojMakroi.tex”. Унутрашњост једног таквог фајла окружења би могла да изгледа овако:

```
\startenvironment MojeOkruzenje

\mainlanguage[sr]

\setupbodyfont
[dejavu]

\setupwhitespace
[big]

...

\stopenvironment
```

Или другим речима, дефиниције и конфигурационе команде се постављају између `\startenvironment` и `\stopenvironment`. Непосредно иза `\startenvironment` пишемо име којим називамо то окружење, па онда наводимо све команде које треба да чине окружење.

Што се тиче имена окружења, према резултатима мојих тестова, име које стављамо непосредно иза `\startenvironment` је чисто индикативно, тако да и ако га не наведемо, ништа (лоше) се не догађа.

Предвиђено је да фајлови окружења функционишу заједно са компонентама и производима (који су објашњени у наредном одељку). То је разлог због којег једно или више окружења може да се позове из компоненте или производа командом `\environment`. Али ова команда такође ради и ако се употреби у конфигурационој области (преамбули) било ког ConTeXt изворног фајла, чак и ако то није изворни фајл предвиђен да се компајлира у деловима.

Команда `\environment` може да се позове употребом било којег од следећа два формата:

`\environment` Фајл

`\environment[Фајл]`

У сваком случају, резултат команде ће бити читавање садржаја фајла који се наведе као њен аргумент. Ако се тај фајл не пронађе, компајлирање ће се наставити на уобичајен начин, без враћања било какве грешке. Ако је екстензија фајла `„.tex”`, може да се изостави.

4.6.2 Компоненте и производи

Ако замислимо књигу у којој је свако поглавље у различитом изворном фајлу, онда можемо да кажемо да су поглавља *компоненте*, а да је књига *производ*. Ово значи да је *компоненте* самостални део *производа*, који може да поседује сопствени стил и може да се компајлира независно. Свака компонента ће имати различит фајл, а уз то ће постојати и фајл производа који обједињује све компоненте у целину.

Типични фајл компоненте изгледа овако

```
\environment MojeOkruzenje
\environment MojiMakroi

\startcomponent Glava1

  \startchapter[title={Глава 1}]

  ...

\stopcomponent
```

А фајл производа би могао да изгледа овако:

```
\environment MojeOkruzenje
\environment MojiMakroi

\startproduct MojaKnjiga

  \component Glava1
  \component Glava2
  \component Glava3

  ...

\stopproduct
```

Запазите да ће стварни садржај нашег документа бити распоређен по разним фајловима 'компоненти', а да је фајл производа ограничен на успостављање редоследа компоненти. С друге стране, (појединачне) компоненте и производи могу директно да се компајлирају. Компајлирање производа ће генерисати PDF фајл који садржи све компоненте тог производа. А ако се компајлира појединачна компонента, то ће генерисати PDF фајл који садржи само компоненту која се компајлира.

Унутар фајла компоненте, пре команде `\startcomponent`, са `\environment ИмеОкружења` можемо да позовемо један или више фајлова окружења. Исто можемо да урадимо и у фајлу производа, пре `\startproduct`. Истовремено може да се учита неколико фајлова окружења. На пример, можемо да имамо своју омиљену колекцију макроа и разних стилова које примењујемо на документе који се налазе у различитим фајловима. Међутим, имајте на уму да када користимо два или више окружења, она се читавају у редоследу у којем су позивана, тако да ако се иста конфигурациона команда налазу у више од једног окружења, а има различите вредности, примениће се вредности оног последње учитаног. С друге стране, окружења се читавају само једном, тако да ако у претходним примерима у којима се окружење позива из фајла производа и из одређених фајлова компоненти, ако компајлирамо производ, то је тренутак читавања окружења у редоследу у којем су ту наведена; када се окружење позове из било које од компоненти, ConTeXt ће проверити да ли је окружење већ једном било учитано, па у том случају неће урадити ништа.

Име компоненте која се позива из производа мора бити име фајла који садржи ту компоненту, мада ако је екстензија тог фајла „.tex”, она може да се изостави.

4.6.3 Пројекти у ужем смислу

У већини случајева је разлика између производа и компоненти довољна. Исто тако, ConTeXt поседује чак и виши ниво у којем можемо да групишемо већи број производа: то је *пројекат*.

Типични фајл пројекта би отприлике изгледао овако

```
\startproject MojaKolekcija

\environment MojeOkruzenje
\environment MojiMakroi

\product Knjiga1
\product Knjiga2
\product Knjiga3

...

\stopproject
```

Сценарио у којем би нам био потребан пројекат је, на пример, онај у којем је потребно да уредимо колекцију књига, све са истим спецификацијама формата; или ако уређујемо неки журнал: таква колекција књига или журнала би била пројекат; свака књига или сваки број журнала би био производ; а свако поглавље књиге или сваки чланак у броју журнала би био компонента.

С друге стране, пројекти нису предвиђени да се директно компајлирају. Имајте на уму да би по дефиницији сваки производ који припада пројекту (свака књига у колекцији, или сваки број журнала) требало да се компајлира одвојено и да се генерише његов PDF. Стога се у њега поставља команда `\product` којом се назначава који производи припадају неком пројекту, а која у суштини не ради ништа: она је просто подсетник за аутора.

Јасно је да се неко може запитати зашто имамо пројекте ако не могу да се компајлирају: одговор је зато што фајл пројекта везује одређена окружења за пројекат. То је разлог што ће ConTeXt, ако у фајл компоненте или производа уметнемо команду `\project ИмеПројекта`, да прочита фајл пројекта и аутоматски да учита окружења везана за њега. Зато у пројектима команда `\environment` мора да дође након `\startproject`; међутим, у производима и компонентама, `\environment` мора да дође *пре* `\startproduct` или `\startcomponent`.

Исто као и са командама `\environment` и `\component`, команда `\project` нам омогућава да наведемо име пројекта било унутар великих заграда, било да уопште не користимо велике заграде. Ово значи да су `\project ИмеФајла` и `\Project[ИмеФајла]` еквивалентне команде.

Резиме различитих начина за учитавање окружења

Из претходног следи да окружење може да се учита било којом од следећих процедура:

- а. Уметањем команде `\environment ИмеОкружења` испред `\starttext` или `\startcomponent`. То ће да учита окружење само за компајлирање тог фајла.
- б. Уметањем команде `\environment ИмеОкружења` у фајл производа испред `\startproduct`. То ће да учита окружење када се компајлира производ, али неће у случају да се његове компоненте компајлирају посебно.
- в. Уметањем команде `\project` у производ или окружење: то ће да учита сва окружења везана за пројекат (у фајлу пројекта).

4.6.4 Заједнички аспекти окружења, компоненти, производа и пројеката

Имена окружења, компоненти, производа и пројеката: Већ смо видели да се, за све ове елементе, након `\start` команде која започне одређено окружење, компоненту или производ, његово име мора ручно да се наведе. Као правило, ово име мора да се подудара са именом фајла који садржи окружење, компоненту ли производ јер, на пример, када ConTeXt компајлира производ, па сагласно са фајлом производа мора да учита окружење или компоненту, не постоји начин да се зна који фајл представља то окружење или компоненту осим ако фајл има исто име као и елемент који треба да се учита.

Иначе је, према резултатима мојих тестова, име које се пише након `\startproduct` или `\startenvironment` у фајловима производа и окружења чисто индикативно. Ако се изостави, или ако се не подудара са именом фајла, не дешава се ништа лоше. Међутим, у случају компоненти је важно да се име компоненте подудара са именом фајла који је садржи.

Структура директоријума пројекта: Знамо да ConTeXt подразумевано тражи фајлове у радном директоријуму и на путањи коју наводи променљива TEXROOT. Међутим, када користимо `\project`, `\product`, `\component` или `\environment` команде, претпоставља се да пројекат има структуру директоријума у којој се заједнички елементи налазе у директоријуму родитељу, а они специфични у неким дете директоријумима. Дакле, ако се наведени фајл не пронађе у радном директоријуму, потражиће се у његовим родитељ директоријумима, па ако се ни тамо не пронађе, у родитељ директоријуму тог директоријума и тако даље.

II

Глобални аспекти документа

Глава 5

Странице и пагинација документа

Садржај: 5.1 Величина странице; 5.1.1 Подешавање величине странице; 5.1.2 Употреба нестандартних величина странице; 5.1.3 Промена величине папира на било ком месту у документу; 5.1.4 Подешавање величине странице према садржају; 5.2 Елементи на страници; 5.3 Распоред странице (`\setuplayout`); 5.3.1 Додела величине различитим компонентама странице; 5.3.2 Подешавање распореда странице; 5.3.3 Употреба више различитих распореда страница; 5.3.4 Остале ствари у вези са распоредом странице; А Разликовање непарних и парних страница; Б Странице са више од једне колоне; 5.4 Нумерација страница; 5.5 Форсирани или предложени преломи странице; 5.5.1 Команда `\page`; 5.5.2 Спајање одређених линија пасуса тако да се спречи уметање прелома странице између њих; 5.6 Заглавља и подножја; 5.6.1 Команде за постављање саржаја заглавља и подножја; 5.6.2 Форматирање заглавља и подножја; 5.6.3 Дефинисање специфичних заглавља и подножја и везивање за `section` команде; 5.7 Уметање текст елемената у ивице и маргине странице;

ConTeXt трансформише изворни документ у коректно форматиране *странице*. Изглед страница, распоред текста и празнина, као и елементи на страницама су од пресудног значаја за словослагање. Ово поглавље је посвећено свим овим питањима, а и још неким стварима које се тичу пагинације.

5.1 Величина странице

5.1.1 Подешавање величине странице

ConTeXt подразумевано претпоставља да ће документ бити А4 величине, односно следи европски стандард. Командом `\setuppapersize` можемо да подесимо неку другу величину. То је команда која се типично поставља у преамбулу документа. Уобичајена синтакса ове команде је:

```
\setuppapersize[ЛогичкаСтраница][ФизичкаСтраница]
```

где су оба аргумента симболичка имена.¹ Први аргумент који сам назвао *ЛогичкаСтраница*, представља величину странице која треба да се узме у обзир за словослагање; а други аргумент *ФизичкаСтраница*, представља величину папира на којој ће се документ штампати. Обично су обе величине исте, па онда други аргумент може да се изостави; међутим, у ситуацијама када су две величине различите, на пример, када се књига штампа у листовима од по 8 или 16 страница (уобичајена техника штампе, посебно за академске књиге до око 1960их година). У тим случајевима ConTeXt нам омогућава да направимо разлику између обе величине; а ако је идеја да се неколико страница штампа на једном листу папира, можемо такође да наведемо и шему савијања која треба да се поштује користећи команду `\setuparranging` (за коју се у овом уводу не даје објашњење).

¹ Присетимо се да сам у одељку 3.5 навео да у општем случају ConTeXt команде прихватају две врсте опција: симболичка имена, чије значење ConTeXt већ познаје, и променљиве чије вредности морамо експлицитно да наведемо.

За величину словослагања можемо да наведемо било коју од предефинисаних величина које се користе у индустрији папира (или коју ми користимо). То су:

- Било која из А, В и С серија дефинисаних ISO-216 стандардом (од A0 до A10, од B0 до B10 и од C0 до C10), обично се користе у Европи.
- Било која од америчких стандардних величина: letter, ledger, tabloid, legal, folio, executive.
- Било која од RA и RSA величина дефинисаних ISO-217 стандарду.
- G5 и E5 величина дефинисаних швајцарским SIS-014711 стандардом (за докторске тезе).
- За коверте: било која од величина дефинисаних северноамеричким стандардом (envelope 9 до envelope 14) или ISO-269 стандардом (C6/C5, DL, E4).
- CD, за CD омоте.
- S3 – S6, S8, SM, SW за величине екрана у документима који нису намењени штампању, већ приказу на екран.

Заједно са величином папира, командом `\setuppapersize` можемо да назначимо и оријентацију странице: „portrait” (вертикална) или „landscape” (хоризонтална).



Према ConTeXt викију, остале опције које прихвата `\setuppapersize` су „rotated”, „90”, „180”, „270”, „mirrored” и „negative”. У мојим личним текстовима сам приметио неке видљиве промене са „rotated” које инвертују страницу, мада није прецизна инверзија. Бројчане вредности би требало да произведу одговарајући степен ротације, самостално или у комбинацији са „rotated”, али нисам успео то да постигнем. Нити сам могао успешно да откријем чему служе „mirrored” и „negative”.

Други аргумент команде `\setuppapersize`, за који сам већ напоменуо да се може изоставити када је величина штампе иста као величина словослагања, може да узме потпуно исте вредности као и први, и означава величину папира и оријентацију. Његова вредност такође може бити и „oversized” – што према ConTeXt викију – додаје центиметар и по сваком углу папира.

Према викију, постоје и остале могуће вредности за други аргумент: „undersized”, „doublesized” и „doubleoversized”. Али у мојим личним тестовима нисам видео никакву промену када се употреби било која од ових вредности; а ни званична дефиниција команде (погледајте [одељак 3.6](#)) не помиње ове опције.

5.1.2 Употреба нестандартних величина странице

Ако желимо да употребимо нестандартну величину странице, морамо да урадимо две ствари:

1. Употребимо нестандартну синтаксу команде `\setuppapersize` која нам омогућава да наведемо висину и ширину папира као димензије.
2. Дефинишемо нашу сопствену величину странице, да јој доделимо име и да је користимо као да је у питању стандардна величина папира.

Алтернативна синтакса команде `\setuppapersize`

Уз синтаксу коју смо већ видели, `\setuppapersize` нам омогућава да употребимо и следећу:

`\setuppapersize[Име][Опције]`

где је *Име* необавезни аргумент који представља име које се командом `\definepapersize` (коју ћемо представити следећу) додељује величини папира, а *Опције* су врста аргумента којим експлицитно додељујемо вредност. Међу дозвољеним вредностима можемо да истакнемо следеће:

- `width`, `height` које представљају редом ширину и висину странице.
- `page`, `paper`. Прва се односи на величину странице на коју се слаже текст, а друга на величину странице на коју ће се физички штампати. То значи да је „page” исто што и први аргумент команде `\setuppapersize` у њеној уобичајеној синтакси (која је објашњена изнад) и „paper” као други аргумент. Ове опције могу имати исте вредности као што је раније речено (A4, S3, итд.).
- `scale`, наводи фактор скалирања за страницу.
- `topspace`, `backspace`, `offset`, додатни размаци.

Дефинисање прилагођене величине странице

За дефинисање прилагођене величине странице, користимо команду `\definepapersize` чија је синтакса

`\definepapersize[Име][Опције]`

где се *Име* односи на име које се даје новој величини, а *Опције* могу бити:

- Било која од дозвољених вредности за `\setuppapersize` у њеној уобичајеној синтакси (A4, A3, B5, CD, итд.).
- Мере за ширину (папира), висину (папира) и померај, или вредност скалирања („scale”).

Није могуће да се мешају вредности дозвољене за `\setuppapersize` са мерама или факторима скалирања. Разлог за то је што су у првом случају опције симболичке речи, а у другом променљиве којима се дају конкретне вредности; а у систему ConTeXt, као што сам већ рекао, није дозвољено мешање ове две врсте опција.

Када користимо `\definepapersize` да назначимо величину папира која се подудара са неким од стандардних величина, уместо да дефинишемо нову величину папира, ми ћемо уствари да дефинишемо ново име за већ постојећу величину папира. Ово може да буде корисно ако желимо да комбинујемо величину папира са оријентацијом. Тако на пример, можемо да напишемо

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

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

5.1.3 Промена величине папира на било ком месту у документу

У већини случајева документи користе само једну величину папира и то је разлог што је команда као што је `\setuppapersize` типична команда која се поставља у преамбулу и која се користи само једном у сваком документу. Међутим, у неким ситуацијама може бити неопходно да се на неком месту у документу промени величина; на пример, ако од се неког места надаље умеће анекс чији су листови положени.

У таквим случајевима можемо употребити `\setuppapersize` тачно на месту где та промена треба да се догоди. Али пошто би се промена тренутно применила, обично се пре команде `\setuppapersize` умеће форсирани прелом странице, па се тако спречавају нежељени резултати.

Ако је потребно да се величина промени само је појединачну страницу, уместо да команду `\setuppapersize` употребимо двапут, једном да подесимо нову величину, а други пут да је вратимо на оригиналну, можемо да употребимо команду `\adaptpapersize` која мења величину странице, а касније је аутоматски ресетује на вредност која је важила пре позива команде. Исто као и са `\setuppapersize`, пре употребе `\adaptpapersize` би требало да уметнемо форсирани прелом странице.

5.1.4 Подешавање величине странице према садржају

Постоје три окружења у систему ConTeXt која генеришу странице довољне величине тако да на њих стане предвиђени садржај. То су `\startMPpage`, `\startpagefigure` и `\startTEXpage`. Прво генерише страницу која садржи графику дефинисану у MetaPost, језиком за графички дизајн који се интегрише са ConTeXt, али којим се нећу бавити у овом уводу. Друго ради исто са сликом и можда са нешто текста испод ње. Оно узима сва аргумента: први идентификује фајл који садржи слику. О овоме ћу говорити у поглављу посвећеном спољним сликама. Треће (`\startTEXpage`) садржи текст који је њен садржај на страници. Њена синтакса је:

```
\startTEXpage[Опције] ... \stopTEXpage
```

где Опције могу бити било шта од следећег:



- **strut**. Нисам сигуран колико је ова опција корисна. У ConTeXt терминологији, *strut* је карактер који нема ширину, али има максималну висину и дубину, мада ми није јасно какве то везе има општом употребном вредности ове команде. Према викију, ова опција дозвољава вредности „yes”, „no”, „global” и „local”, при чему је подразумевана вредност „no”.
- **align**. Назначава поравнање текста. Ово може бити „normal”, „flushleft”, „flushright”, „middle”, „high”, „low” или „lohi”.
- **offset** за назначаване количине празног простора око текста. Може бити „none”, „overlay” у случају да желите ефекат прекривања, или тачну димензију.

- `width`, `height` за које можемо навести ширину и висину странице, или вредност „fit” тако да ширина и висина постану такве да одговарају тексту који се налази у окружењу.
- `frame` која је подразумевано „off”, али може да узме вредност „on” ако желимо оквир око текста на страници.

5.2 Елементи на страници

ConTeXt препознаје различите елементе на страницама и њихове димензије се конфигуришу командом `\setuplayout`. Убрзо ћемо се позабавити овим, али најбоље би било да пре тога опишемо сваки од елемената странице, и да наведемо име под којим их команда `\setuplayout` познаје:

- **Ивице:** празан простор око површине текста. Мада их већина текст процесора назива „маргине”, пожељно је да се користи ConTeXt терминологија, јер нам омогућава да направимо разлику између ивица у ужем смислу, тамо где нема текста (мада се у електронским документима ту могу налазити дугмад за навигацију и слично), и маргине у којима се понекада могу налазити текст елементи, као што су на пример, белешке на маргинама.
 - Висина горње ивице се контролише помоћу две мере: самом горњом ивицом („top”) и растојањем између горње ивице и заглавља („topdistance”). Збир ове две мере се назива „topspace”.
 - Величина леве и десне ивице зависи од мера „leftedge” и „rightedge” редом. Ако желимо да и једна и друга имају исту дужину, можемо истовремено да их подесимо опцијом „edge”.

У документима намењеним за двострану штампу, не говоримо о левој и десној ивици, већ о унутрашњој и спољашњој; прва је ивица најближа месту на којем ће се листови хефтати или прошивати, тј. лева ивица на непарно нумерисаним страницама, а десна ивица парним страницама. Спољашња ивица је насупрот унутрашње.

 - Висина ниже ивице се назива „bottom”.
- **Маргине** у ужем смислу. ConTeXt маргинама назива само бочне маргине (леву и десну). Маргине се налазе између ивице и простора главног текста и предвиђено је да се у њима налазе одређени текст елементи као што су, на пример, белешке на маргинама, наслови одељака или њихови бројеви.

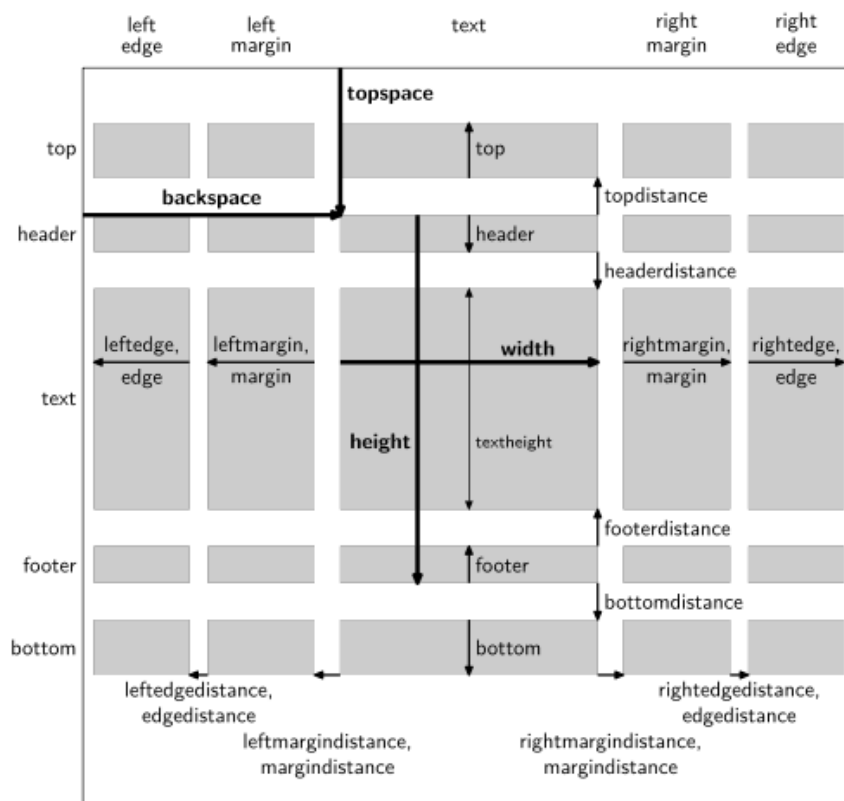
Димензије које контролишу величину маргина су:

- `margin`: користи се када истовремено желимо да поставимо маргине на исту величину.
- `leftmargin`, `rightmargin`: постављају величину леве и десне маргине, редом.
- `edgedistance`: растојање између ивице и маргине.

- `leftedgedistance`, `rightedgedistance`: растојање између ивице и леве и десне маргине, редом.
 - `margindistance`: растојање између маргине и области главног текста.
 - `leftmargindistance`, `rightmargindistance`: растојање између области главног текста и леве и десне маргине, редом.
 - `backspace`: ова мера представља размак између левог угла папира и почетка области главног текста. Дакле, она представља збир „`leftedge`” + „`leftedgedistance`” + „`leftmargin`” + „`leftmargindistance`”.
- **Заглавље и подножје:** заглавље и подножје странице су две области које се редом налазе изнад и испод површине странице по којој се пише. Обично садрже информације које помажу да се одреди контекст текста, као што су бројеви страница, име аутора, наслов дела, наслов поглавља или одељка, итд. На ове делове странице се у систему ConTeXt утиче следећим димензијама:
 - `header`: висина заглавља.
 - `footer`: висина подножја
 - `headerdistance`: растојање од заглавља до области главног текста на страници.
 - `footerdistance`: растојање између подножја и области главног текста на страници.
 - `topdistance`, `bottomdistance`: већ су раније поменуте. Представљају растојање између горње ивице и заглавља или доње ивице и подножја, редом.
 - **Главна област текста:** ово је најшира област странице у којој се налази текст документа. Његова величина зависи од променљивих „`width`” и „`textheight`”. Променљива „`height`”, представља збир „`header`”, „`headerdistance`”, „`textheight`”, „`footerdistance`” и „`footer`”.

Све ове области можемо видети на [слици 5.1](#) заједно са именима одговарајућих мера и стрелицама које означавају докле се простиру. Дебљина стрелице заједно са величином имена стрелица одговара важности сваке од ових раздаљина за распоред странице. Ако пажљиво погледамо, видећемо да ова слика приказује да се страница може представити као табела са 9 редова и 9 колона, или ако не узмемо у обзир вредности размака између различитих области, онда је то пет редова и пет колона од којих текст може да буде само у три реда и три колоне. Пресек средњег реда са средњом колоном чини област главног текста и обично ће чинити већи део странице.

У фази креирања распореда нашег документа, све мере странице можемо видети командом `\showsetups`. Ако желите да на страници видите визуелну представу обриса дистрибуције текста, можете да употребите команду `\showframe`; а командом `\showlayout` можете видети комбинацију претходне две команде.



Слика 5.1 Области и мере на страници

5.3 Распоред странице (`\setuplayout`)

5.3.1 Додела величине различитим компонентама странице

Дизајн странице подразумева доделу одређених величина одговарајућим областима странице. Ово се ради командом `\setuplayout`. Она нам омогућава да изменимо било коју од димензија поменутих у претходном одељку. Њена синтакса је следећа:

`\setuplayout[Име][Опције]`

где је *Име* необавезни аргумент који се користи само у случају када дизајнирамо више распореда (погледајте одељак 5.3.3), а опције су, поред осталих које ћемо видети касније, било које од претходно поменутих. Међутим, имајте на уму да су ове мере у међусобној вези, јер укупан збир компоненти које утичу на ширину, као и оних које утичу на висину мора да се подудари са ширином и висином странице. Ово у суштини значи да када мењамо неку хоризонталну дужину, морамо да променимо и остатак хоризонталних дужина; а исто је и када подешавамо вертикалну дужину.

ConTeXt подразумевано извршава аутоматска подешавања димензија само у неким случајевима, који нису на потпун или систематичан начин наведени у његовој документацији.

Извођењем неколико тестова сам потврдио, на пример, да ручно увећање или смањење висине заглавља или подножја повлачи и измену „textheight”; међутим, ручна измена неке од маргина не подешава аутоматски и ширину текста, „width” (барем је то резултат мојих тестова). Због тога је најефикаснији начин за спречавање неконзистентности између величине странице (постављене са `\setuppapersize`) и величине њених одговарајућих компоненти следећи:

- Што се тиче хоризонталних мера:
 - Подешавањем „backspace” (које укључује „leftedge” и „leftmargin”).
 - Подешавањем „width” (ширине текста) не са димензијама, већ са вредностима „fit” или „middle”:
 - ★ fit израчунава ширину текста на основу ширине осталих хоризонталних компоненти странице.
 - ★ middle ради исто, али најпре поставља десну и леву маргину на исту вредност.
- Што се тиче вертикалних мера:
 - Подешавањем „topspace”.
 - Подешавањем „fit” или „middle” вредности на „height”. Оне раде на исти начин као код ширине. Прва израчунава висину према осталим компонентама, а друга прво поставља горњу и доњу маргину на исту вредност, па онда израчунава висину текста.
 - Када се подеси „height”, по потреби се подешавају висина заглавља или подножја, знајући да ће се „textheight” у тим случајевима аутоматски поново подесити.
- Још једна могућност за индиректно одређивање висине области главног текста је навођењем броја линија које треба да стану у њу (имајући у виду текући размак између линија и величину фонта). Ово је разлог што `\setuplayout` поседује „lines” опцију.

Постављање логичке странице на физичку страницу

У случајевима када величина логичке странице није иста као величина физичке странице (погледајте одељак 5.1.1) `\setuplayout` нам омогућава да конфигуришемо неке додатне опције које утичу на постављање логичке странице на физичку:

- **location**: ова опција одређује место на физичкој страници на којем ће се поставити логичка. Могуће вредности су left, middle, right, top, bottom, singlesided, doublesided или duplex.
- **scale**: наводи фактор скалирања странице пре постављања на физичку страницу.
- **marking**: на страници ће се штампати визуелни маркери који показују где се сече папир.
- **horoffset, veroffset, clipoffset, cropoffset, trimoffset, bleedoffset, artoffset**: низ мера које наводе различите помераје на физичкој страници. Већина њих је објашњена у референтном упутству из 2013. године.

Ове `\setuplayout` опције морају да се комбинују са знацима из `\setuparranging` које говоре како се на физичкој страници ређају логичке странице. Ове команде нећу да описујем у уводу јер их уопште нисам тестирао.

Добијање ширине и висине области текста

Команде `\textwidth` и `\textheight` редом враћају ширину и висину области текста. Вредности које се добију помоћу њих не могу директно да се покажу у финалном документу, али могу да се употребе за постављање мера висине и ширине осталих команди. На пример, ако желимо да наведемо слику чија ће ширина бити 60% ширине линије, као вредност опције „width” за слику морамо да поставимо: „width=0.6\textwidth”.

5.3.2 Подешавање распореда странице

Може се догодити да наш распоред странице на одређеној страници не даје жељени резултат; на пример, последња страница поглавља са само једом или две линије, што није пожељно ни типографски, ни естетски. За решавање оваквих ситуација, ConTeXt обезбеђује команду `\adaptlayout` која нам омогућава да променимо величину области текста на једној или на више страница. Предвиђено је да се ова команда користи само онда када смо завршили са писањем документа и радимо само ситна финална подешавања. Стога, њена природна позиција је у преамбули документа. Синтакса команде је:

`\adaptlayout[Странице] [Опције]`

где се *Странице* односи на број странице или страница чији распоред желимо да променимо. То није обавезни аргумент и он би требало да се користи само када се команда `\adaptlayout` поставља у преамбулу. Можемо да назначимо само једну страницу, или неколико њих, тако што бројеве раздвајамо запетама. Ако изоставимо овај први аргумент, `\adaptlayout` ће утицати само на страницу која садржи команду.

Опције могу да буду:

- **height**: дозвољава нам да наведемо, као меру, висину странице у питању. Можемо да задамо апсолутну висину (нпр. „19cm”) или релативну висину (нпр. „+ 1cm”, „-0.7cm”).
- **lines**: можемо да наведемо број линија који треба да се дода или одузме. Линије се додају ако се испред броја стави +, а одузимају ако се постави знак – (не просто цртица).

Имајте на уму да када мењамо број линија на страници, такође може да се промени и пагинација остатка документа. Из тог разлога је пожељно да се команда `\adaptlayout` користи само на крају, када више нема накнадних измена документа, и да се то уради у преамбули. Затим одлазимо на прву страницу коју желимо да изменимо, извршимо измену, па проверимо како то утиче на наредне странице; ако је измена таква да још нека страница мора да се промени, додајемо и њен број па компајлирамо још једном, и тако даље.

5.3.3 Употреба више различитих распореда страница

Ако је потребно да за различите делове документа употребимо различите распореде страница, најбољи начин да се почне је да се дефинише *омишћи* распоред, па затим разни алтернативни који мењају само димензије које се разликују од оних у општем. Ови алтернативни

распоређи наслеђују све особине општег распоређа које се не мењају као део своје дефиниције. За навођење алтернативног распоређа, његово називање именом којим га касније можемо позвати, користимо команду `\definelayouth` чија је општа синтакса:

```
\definelayouth[Име/Број] [Конфигурација]
```

где је *Име/Број* име које се придружује новом дизајну, или број странице на којем се нови распоређ аутоматски активира, а *Конфигурација* садржи аспекте распоређа које желимо да променимо у односу на општи распоређ.

Када се новом распореду придружи име, на одређеном месту у документу га позивамо помоћу:

```
\setuplayouth[ИмеРаспоређа]
```

а враћамо се на општи распоређ са:

```
\setuplayouth[reset]
```

С друге стране, ако је нови распоређ придружен одређеној страници, он ће се аутоматски активирати када се достигне та страница. Међутим, једном када се активира, враћање на општи дизајн се ради експлицитно, мада ово можемо да *полупрограмирамо*. На пример, ако желимо да применимо распоређ само на странице 1 и 2, у преамбули документа можемо да напишемо следеће:

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

Резултат ових команди је да се распоређ дефинисан у првој линији активира на страници 1, а на страници 3 се активира други распоређ чија је функција просто враћање на општи распоређ.

Са `\definelayouth[even]` креирамо распоређ који се активира на свим парним страницама; а са `\definelayouth[odd]` распоређ који се примењује на све непарне странице.

5.3.4 Остале ствари у вези са распоређом странице

А. Разликовање непарних и парних страница

У документима припремљеним за двострану штампу је чест случај да се заглавље, нумерација страница и бочне маргине разликују на непарним и парним страницама. Парно нумерисане странице се такође називају и леве (версо) странице, а непарне странице десне (ректо) странице. У овим случајевима је такође обичај да се мења терминологија која се тиче маргина, па говоримо о унутрашњим и спољашњим маргинама. Ове раније се налазе на тачки најближој месту на којем ће се странице хефати или прошивати, а касније на супротној страни. На непарно нумерисаним страницама, унутрашња маргина одговара левој маргини, а на парно нумерисаним спољашња маргина одговара десној маргини.

`\setuplayouth` нема ниједну опцију која нам омогућава да директно кажемо да желимо разлику између распоређа парних и непарних страница. То је зато што се систему ConTeXt

разлика између две врсте страница поставља другачијом опцијом: `\setuppagenumbering` коју ћемо видети у одељку 5.4. Када се ово подеси, ConTeXt претпоставља да је страница описана са `\setuplayout` непарна страница и парну страницу изграђује тако што на њу примени инвертоване вредности за непарну страницу: спецификације које се примењују за непарно нумерисане странице се примењују на леву, а на парно нумерисаним страницама на десну; и обрнуто: оне које се примењују на непарно нумерисану страницу на десну, у парно нумерисаној страници се примењују на леву.

Б. Странице са више од једне колоне

Помоћу `\setuplayout` текст нашег документа такође можемо да видимо распоређен у две или више колона, као у неким новинама и магазинима, на пример. Ово се контролише опцијом „columns” чија вредност мора да буде цео број. Када постоји више од једне колоне, размак између њих се задаје опцијом „columndistance”.

Ова опција је намењена за документе у којима је сав текст (или већи део текста) распоређен у више колона. Ако у документу који је углавном у једној колони пожељимо да одређени део буде у две или три колоне, не морамо да мењамо распоред странице, већ просто употребимо „columns” окружење (погледајте одељак 12.2).

5.4 Нумерација страница

ConTeXt подразумевано користи арапске бројеве за нумерацију страница и број се појављује у заглављу, хоризонтално центриран. Ако ове особине желите да промените, ConTeXt вам нуди различите процедуре за то, које према мом мишљењу, ствар чине непотребно компикованом.

Најпре, најосновније карактеристике нумерације се контролишу помоћу две различите команде: `\setuppagenumbering` и `\setupuserpagenumber`.

`\setuppagenumbering` нуди следеће опције:

- **alternative:** ова опција контролише да ли је документ дизајниран тако да су заглавље и подножје идентични на свим страницама („singlesided”), или се прави разлика између парних и непарних страница („doublesided”). Када ова опција има последњу вредност, аутоматски се утиче на вредности уведене са „setuplayout”, тако да се претпоставља да се подешавање „setuplayout” односи само на непарно нумерисане странице, па дакле, оно што се постави за леву маргину уствари је за унутрашњу (што је на парно нумерисаним страницама на десној страни) и да се оно што се постави за десну страну уствари односи на спољашњу маргину, која је на парно нумерисаним страницама на левој страни.
- **state:** означава да ли ће се број странице приказивати, или не. Могуће су две вредности: start (број странице се приказује) и stop (бројеви страница се не приказују). Имена ових вредности (старт и стоп) би могла да нас наведу на помисао да када имамо „state=stop” странице се не нумеришу, а када је „state=start” нумерација поново почиње. Али није тако: ове вредности утичу само на то да ли се број приказује или не.

- **location:** означава где ће се приказивати број. Обичне је потребно да наведемо две вредности у овој опцији, раздвојене запетом. Најпре морамо да наведемо да ли желимо број странице у заглављу („header”), или у подножју („footer”), па затим, где: то може да буде „left”, „middle”, „right”, „inleft”, „inright”, „margin”, „inmargin”, „atmargin” или „marginedge”. На пример: ако желимо да се прикаже десно поравната нумерација у подножју, требало би да наведемо „location={footer,right}”. Иначе, погледајте како смо ову опцију поставили унутар витичастих заграда, тако да ConTeXt исправно може да интерпретира раздвајајућу запету.
- **style:** назначава величину фонта и стил који треба да се користи за бројеве страница.
- **color:** назначава боју која треба да се примени на број странице.
- **left:** прихвата команду или текст који треба да се изврши лево од броја странице.
- **right:** прихвата команду или текст који треба да се изврши десно од броја странице.
- **command:** прихвата команду којој ће се број странице проследити као параметар.
- **width:** назначава ширину коју заузима број странице.
- **strut:** нисам сигуран у вези са овим. У мојим тестовима, када је „strut=no”, број се штампа тачно на горњој ивици заглавља или на дну подножја, док када је „strut=yes” (подразумевана вредност) између броја и странице се примењује размак.

`\setupuserpagenumber` прихвата и следеће додатне опције:

- **numberconversion:** контролише врсту набрајања која може бити арапска („n”, „numbers”), мала слова („a”, „characters”), велика слова („A”, „Characters”), капитал („KA”), римско малим („i”, „r”, „romannumerals”), римско великим („I”, „R”, „Romannumerals”) или римско капиталом („KR”).
- **number:** наводи број који се додељује првој страници, на основу којег се израчунавају остали.
- **numberorder:** ако овоме као вредност доделимо „reverse”, нумерација страница ће се вршити у опадајућем редоследу; то значи да ће последња страница бити 1, претпоследња 2, итд.
- **way:** омогућава нам да одредимо како ће се наставити са нумерацијом. Може бити: `byblock`, `bychapter`, `bysection`, `bysubsection`, итд.
- **prefix:** дозвољава нам да наведемо префикс бројевима страница.
- **numberconversionset:** објашњено је испод.

Уз ове две команде, такође је неопходно да се узме обзир контрола бројева која се тиче макроструктуре документа (погледајте [одељак 7.6](#)). Са ове тачке гледишта, `\defineconversionset` нам омогућава да наведемо другачију врсту нумерације са сваки макроструктурни блок. На пример:

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

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

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

ће уредити да се први блок нашег документа (frontmatter) броји римским бројевима исписаним малим словима, средишњи блок (bodymatter) арапским бројевима, а додаци великим словима.

Следећим командама можемо добити број странице:

- `\userpagenumber`: враћа број странице онакав како је конфигурисан командама `\setuppagenumbering` и `\setupuserpagenumber`.
- `\pagenumber`: враћа исти број као претходна команда, али још увек арапским бројевима.
- `\realpagenumber`: враћа реални број странице у арапским бројевима без потребе било које од ових спецификација.

Ако желимо да добијемо број последње странце у документу, постоји три команде које одговарају претходним. То су: `\lastuserpagenumber`, `\lastpagenumber` и `\lastrealpagenumber`.

5.5 Форсирани или предложени преломи странице

5.5.1 Команда `\page`

Алгоритам за распоред текста у систему ConTeXt је прилично сложен и заснива се на многим израчунавањима и интерним променљивама које програму говоре где се из перспективе типографске коректности налази најбоља тачка за уметање стварног прелома странице. Команда `\page` нам омогућава да утичемо на овај алгоритам:

- а. Сугерисањем одређених тачака као најбољих или најгорих места за уметање прелома странице.
 - `по`: наводи да место на којем се налази команда није добар кандидат за уметање прелома странице, тако да би прелом требало да се уметне на неком другом месту у документу, што је даље могуће.
 - `bigpreference`: наводи да је тачка на којој се наиђе на команду *врло добро место* да се покуша прелом странице, али не толико да се прелом форсира.

Запазите да ове две опције нити форсирају, нити спречавају преломе страница, већ само говоре систему ConTeXt да би требало да узме у обзир оно што је наведено у команду

када тражи најбоље место за прелом. Међутим, у крајњој линији, ConTeXt ће и даље одлучивати о месту на којем ће се уметнуто прелом странице.

б. Форсирањем прелома странице на одређеном месту; у овом случају можемо такође да назначимо колико прелома страница би требало да се направи, као и одређене особине страница које се умећу.

- **yes**: форсира прелом странице на овом месту.
- **makeup**: слично као и „yes”, али форсирани прелом се ради одмах, не поставља се пре тога ниједан плутајући објекат који чека на постављање (погледајте одељак 13.1).
- **empty**: умеће потпуно празну страницу у документ.
- **even**: умеће потребан број страница тако да наредна страница постане парна страница.
- **odd**: умеће потребан број страница тако да наредна страница постане непарна страница.
- **left, right**: слично као и претходне две опције, али се примењује само на документе предвиђене за двострану штампу, са различитим заглављима, подножјима или маргинама у зависности од тога да ли је страница парна или непарна.
- **quadruple**: умеће потребан број страница потребан да наредна страница буде умножак броја 4.

Уз ове опције које су намењене директној контроли пагинације, `\page` нуди и остале опције које утичу на начин функционисања ове команде. Истичу се опција „disable” која чини да ConTeXt игнорише `\page` команде на које наиђе од овог места па надаље, као и опција „reset” која има супротан ефекат, која дакле враћа ефекат наредних `\page` команди на које се наиђе.

5.5.2 Спајање одређених линија пасуса тако да се спречи уметање прелома странице између њих

Понекада када желимо да спречимо прелом странице између неколико пасуса, употреба команде `\page` може бити заморна, јер би требало да се напише на свакој тачки где је могуће да се уметне прелом странице. Једноставнија процедура за то је да се материјал које желимо да буде на истој страници постави у оно што \TeX назива *вертикална кутија*.

На почетку овог документа (на [страници 17](#)) сам напоменуо да је интерно за \TeX све *кутија*. Појам кутије је у \TeX основа за било коју врсту *напредне* операције; али управљање њима је сувише сложено да би се укључило у овај увод. То је разлог што кутије само повремено помињем.

Једном када се креирају, \TeX кутије су недељиве, што значи да не можемо уметнути прелом странице који би кутију поделио на две. Зато ако материјал који желимо да држимо заједно поставимо у невидљиву кутију, спречићемо уметање прелома странице који би поделио тај материјал. То се ради командом `\vbox`, чија је синтакса

`\vbox{Материјал}`

где је *Материјал* текст који желимо да буде заједно.

Нека ConTeXt окружења постављају свој садржај у кутију. На пример, „framedtext”, тако да ако уоквиримо материјал који желимо да држимо заједно у ово окружење и поставимо да је оквир невидљив (што може да се уради опцијом `frame=off`), постићи ћемо исту ствар.

5.6 Заглавља и подножја

5.6.1 Команде за постављање саржаја заглавља и подножја

Ако смо у распореду странице заглављу и подножју доделили одређену величину, у њих можемо уметнути текст командама `\setupheadertexts` и `\setupfootertexts`. Оне су сличне, једина разлика је што прва активира садржај заглавља, а друга садржај подножја. Обе имају од једног до пет аргумената.

1. Ако се употреби са једним аргументом, он ће садржати текст који ће се у заглављу или подножју поставити на средину ширине странице. На пример: `\setupfootertexts[pagenu-
mber]` ће на средини подножја писати број странице.
2. Ако се употреби са два аргумента, садржај првог аргумента ће се поставити на леву страну заглавља или подножја, а онај другог аргумента на десну страну. На пример, `\setupheadertexts[Предговор][pagenu-
mber]` ће сложити заглавље странице у којем се реч „предговор” исписује на левој страни, а број странице на десној страни.
3. Ако се наведу три аргумента, први ће назначити *обласћ* у којој ће се штампати преостала два. *Обласћ* се односи на *обласћ* странице поменуте у одељку 5.2, другим речима: `edge`, `margin`, `header`... Остала два аргумента садрже текст који треба да се постави у леву ивицу, маргину, десну ивицу, маргину.

Када се користи са четири или пет аргумената, то је исто као и када се користи са два или три аргумента, само у случајевима када се прави разлика између парних и непарних страница, а који се јављају, као што знамо, када се са `\setuppagenumbering` постави „`alternative=doublesided`”. У том случају, два могућа аргумента се додају тако да означавају садржај леве и десне стране парних страница.

Важна карактеристика ове две команде је да када се користе са два аргумента, претходно централно заглавље или подножје (ако је постојало) се не преписује, што нам омогућава да у свакој области напишемо другачији текст све док прво напишемо централни текст (позивом команде са једним аргументом), па затим напишемо текстове за обе стране (тако што је позовемо поново, али сада са два аргумента). Дакле, ако напишемо следеће команде

```
\setupheadertexts[и]  
\setupheadertexts[Станлио][Олио]
```

прва команда ће написати „и” на средини заглавља, а друга ће написати „Станлио” на левој и „Олио” на десној страни, остављајући средишњу област нетакнутом, јер јој није наложено да је поново испише. Коначно заглавље ће се сада приказивати овако



Објашњење рада ових команди које сам управо изложио је мој закључак након многих тестова. ConTeXt *excursion* даје објашњење ових команди засновано на верзији са пет аргумената; а оно у референтном упутству из 2013. године је засновано на верзији са три аргумента. Чини ми се да је моје јасније. С друге стране, нисам наишао на објашњење зашто други позив команде не преписује претходни позив, али то је начин на који она ради ако прво напишемо централну ставку у заглављу или подножју, па затим оне са обе стране. Али ако прво упишемо бочне ставке заглавља/подножја, наредни позив команде који уписује централну ставку ће обрисати претходна заглавља или подножја. Зашто? Немам појма. Чини ми се да ови мали детаљи уносе непотребну компликацију и требало би да се јасно објасне у званичној документацији.

Штавише, као садржај заглавља или подножја можемо да наведемо било коју комбинацију текста и команди. А такође и следеће вредности:

- `date`, `currentdate`: ће написати (било које од њих) текући датум.
- `pagenumber`: ће написати број странице.
- `part`, `chapter`, `section...`: ће написати наслов који одговара делу, поглављу, одељку... или било које структурне поделе која постоји.
- `partnumber`, `chapternumber`, `sectionnumber...`: ће написати број дела, поглавља, одељка... или било које структурне поделе која постоји.

Пажња: ова симболичка имена (`date`, `currentdate`, `pagenumber`, `chapter`, `chapternumber`, итд.) се интерпретирају као таква само ако је једини садржај аргумента то симболичко име; али ако додамо још неки текст или команду форматирања, ове речи ће се интерпретирати дословно, па тако, ако напишемо, на пример, `\setupheadertexts[chapternumber]` добићемо број текућег поглавља; али ако напишемо `\setupheadertexts[Поглавље chapternumber]` завршићемо са: „Поглавље chapternumber”. У овим случајевима, када садржај команде није само симболичка реч, морамо да:

- За `date`, `currentdate` и `pagenumber` употребимо не симболичку реч, већ команду са истим именом (`\date`, `\currentdate` или `\pagenumber`).
- За `part`, `partnumber`, `chapter`, `chapternumber`, итд. употребимо `\getmarking[Mark]` команду која враћа садржај *Mark* који се тражи. Тако ће, на пример, `\getmarking[chapter]` да врати наслов текућег поглавља, док ће `\getmarking[chapternumber]` вратити број текућег поглавља.

Ако на одређеној страни желимо да искључимо заглавља и подножја, користимо команду `\noheaderandfooterlines` која ради само на страници на којој се нађе. Ако на некој страници само желимо да обришемо број странице, морамо да употребимо команду `\page[blank]`.

5.6.2 Форматирање заглавља и подножја

У аргументима за `\setupheadertexts` или `\setupfootertexts` може да се наведе одређени формат у којем се приказује текст заглавља или подножја користећи одговарајуће команде форматирања. Међутим, ово такође можемо и глобално да конфигуришемо са `\setupheader` и `\setupfooter` које нуде следеће опције:

- **state:** прихвата следеће вредности: `start`, `stop`, `empty`, `high`, `none`, `normal` или `nomarking`.
- **style**, **leftstyle**, **rightstyle:** конфигурација стила за текст заглавља или подножја. `style` утиче на све странице, `leftstyle` на парне странице, а `rightstyle` на непарне странице.
- **color**, **leftcolor**, **rightcolor:** боја заглавља или подножја. Може да утиче на све странице (`color` опција) или само на парне странице (`leftcolor`) или непарне странице (`rightcolor`)
- **width**, **leftwidth**, **rightwidth:** ширина свих заглавља и подножја (`width`) или заглавља/подножја на парним страницама (`leftwidth`) или непарним (`rightwidth`).
- **before:** команда која треба да се изврши пре писања заглавља или подножја.
- **after:** команда која ће се извршити након писања заглавља или подножја.
- **strut:** ако има вредност „yes”, појављује се вертикални раздвајајући простор између заглавља и ивице. Када има вредност „no”, заглавље или подножје додирује области горње и доње ивице.

5.6.3 Дефинисање специфичних заглавља и подножја и везивање за `section` команде

ConTeXt систем заглавља и подножја нам омогућава да се текст у заглављу или подножју аутоматски промени када мењамо поглавља или одељке; или када мењамо странице, у случају да имамо различити сет заглавља или подножја за парне и непарне странице. Али он нам не омогућава да правимо разлику између прве странице (документа, или поглавља или одељка) и осталих страница. Да бисмо то постигли, морамо:

1. Да дефинишемо специфично заглавље или подножје
2. Да га повежемо са одељком на који се односи.

Дефинисање специфичних заглавља или подножја се врши командом `\definertext`, чија је синтакса:

```
\definertext
  [Име] [Тип]
  [Садржај1] [Садржај2] [Садржај3]
  [Садржај4] [Садржај5]
```

где је *Име* име које се додељује заглављу или подножју са којим радимо; *Тип* може бити `header` или `footer`, у зависности од тога шта дефинишемо, а осталих пет аргумената држе садржај новог заглавља или подножја, на сличан начин као што смо видели код `\setupheadertexts` и `\setupfootertexts`. Кад то урадимо, потребно је да ново заглавље или подножје повежемо са неким одређеним одељком помоћу `\setuphead` користећи опције `header` и `footer` (које су описане у поглављу 7).

Дакле, наредни пример ће да сакрије заглавље на првој страници сваког поглавља, а у подножју ће се појавити центриран број странице:

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

5.7 Уметање текст елемената у ивице и маргине странице

Горња и доња ивица, као и десна и лева маргина обично не садрже никакав текст. Међутим, ConTeXt омогућава да ту поставимо неке текст елементе. Тачније, за то су доступне следеће команде:

- `\setuptoptexts`: омогућава нам да поставимо текст у горњу ивицу странице (изнад области заглавља).
- `\setupbottomtexts`: омогућава нам да поставимо текст у доњу ивицу странице (испод области подножја).
- `\margintext`, `\atleftmargin`, `\atrightmargin`, `\ininner`, `\ininneredge`, `\ininnermargin`, `\inleft`, `\inleftedge`, `\inleftmargin`, `\inmargin`, `\inother`, `\inouter`, `\inouteredge`, `\inoutermargin`, `\inright`, `\inrightedge`, `\inrightmargin`: дозвољавају постављање текста у бочне ивице и маргине документа.

Прве две команде раде исто као `\setupheadertexts` и `\setupfootertexts` и формат ових текстова може чак унапред да се конфигурише помоћу `\setuptop` и `\setupbottom`, слично као што нам `\setupheader` дозвољава да конфигуришемо текстове за `\setupheadertexts`. Што се тиче свега наведеног, указујем на оно што сам већ рекао у одељку 5.6. Само је потребно је да се дода мали детаљ да текст који се подеси са `\setuptoptexts` или `\setupbottomtexts` неће бити видљив ако се у распореду странице не резервише простор за горњу (top) или доњу (bottom) ивицу. У вези тога, погледајте одељак 5.3.1.

Што се тиче команди намењених постављају текста у маргине документа, све оне имају сличну синтаксу:

```
\ИмеКоманде[Референца] [Конфигурација] {Текст}
```

где *Референца* и *Конфигурација* нису обавезни аргументи; први се користи са могуће унакрсно референцирање, а други нам омогућава да подесимо маргинални текст. Последњи аргумент постављен у витичасте заграде садржи текст који се поставља у маргину.

Општија команда од ових овде је `\margintext` јер омогућава постављање текста у било коју маргину или бочну ивицу странице. Остале команде, као што њихово име наговештава, постављају текст на саму маргину (десну или леву, унутрашњу или спољашњу). Оне су уско везане за распоред странице, јер ако на пример, употребимо `\inrightedge` а нисмо резервисали довољно простора у распореду странице за десну ивицу, неће се видети ништа.

Конфигурационе опције за `\margintext` су следеће:

- **location:** наводи у коју маргину ће се сместити текст. Може бити `left`, `right` или, у документима за двострану штампу, `outer` или `inner`. Подразумевано је `left` за документе предвиђене за једнострану штампу, а `outer` у онима за двострану.
- **width:** ширина доступна за штампање текста. Подразумевано је то пуна ширина маргине.
- **margin:** наводи да ли се текст поставља у саму маргину (`margin`) или у ивицу (`edge`).
- **align:** поравнање текста. Овде се користе исте вредности као за `\setupalign 11.6.1`.
- **line:** омогућава да наведемо број линија за који ће се померити текст у маргини. Тако да `line=1` значи да ће се текст померити једну линију наниже, а `line=-1` за једну линију навише.
- **style:** команда или команде које наводе стил текста који се поставља у маргине.
- **color:** боја маргиналног текста.
- **command:** име команде којој ће се текст намењен маргини проследити као аргумент. Ова команда ће се извршити пре исписивања текста. На пример, ако желимо да исцртамо оквир око текста, могли бисмо да употребимо „`[command=\framed]{Текст}`”.

Остале команде прихватају исте опције, осим за `location` и `margin`. Тачније, команде `\atrightmargin` и `\atleftmargin` постављају текст потпуно наслоњен на тело странице. Простор раздвајања можемо да подесимо опцијом `distance`, коју нисам поменуо када сам говорио о `\margintext`, јер приликом својих тестова нисам приметио никакав ефекат на ту команду.



Уз горе поменуте опције, ове команде подржавају и остале опције (`strut`, `anchor`, `method`, `category`, `scope`, `option`, `hoffset`, `voffset`, `dy`, `bottomspace`, `threshold` и `stack`) које нисам поменуо јер нису документоване, а искрено, нисам ни сигуран чему служе. За оне са именима као што је `distance` можемо да погодимо, али шта је са осталима? Вики помиње само опцију `stack`, за коју каже да се користи за емулацију команде `\marginpars` у \LaTeX , али то ми баш и није јасно.

Команда `\setupmargindata` нам омогућава да глобално конфигуришемо текстове у свакој маргини. Тако, на пример

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

обезбеђује да се сви текстови у десној маргини исписују косим слогом.

Такође можемо да креирамо и прилагођену команду са

```
\definemargindata[Име][Конфигурација]
```

Глава 6

Фонтови и боје у систему ConT_EXt

Садржај: 6.1 Типографски фонтови укључени у „ConT_EXt Standalone”; 6.2 Особине фонта; 6.2.1 Фонтови, *стил*ови и стилске варијанте; 6.2.2 Величина фонта; 6.3 Постављање главног фонта документа; 6.3.1 Како изгледа фонт; 6.4 Промена неких особина фонта; 6.4.1 Команде `\setupbodyfont` и `\switchtobodyfont`; 6.4.2 Брза измена стила, алтернативе и величине; 6.4.3 Дефинисање команди и кључних речи за величине фонта, стилове и алтернативе; 6.5 Остале ствари везане за употребу неких алтернатива; 6.5.1 Курзив, коса слова и наглашавање; 6.5.2 Капитал и лажни капитал; 6.6 Употреба и конфигурација боја; 6.6.1 Процедуре за слагање фрагмената текста у боји; 6.6.2 Промена боје позадине и предњег плана документа; 6.6.3 Комадне за бојење одређених фрагмената текста; 6.6.4 Предефинисане боје; 6.6.5 Да видимо доступне боје; 6.6.6 Дефинисање сопствених боја;

6.1 Типографски фонтови укључени у „ConT_EXt Standalone”

ConT_EXt систем за фонтове пружа многе могућности, али је такође и прилично сложен. У овом упутству нећу анализирати све напредне могућности, већ ћу се ограничити на претпоставку да радимо са неким од 21 фонтова који се испоручују у ConT_EXt Standalone инсталацији, онима приказаним у [табели 6.1](#).

Званично име	Име(на) у ConT _E Xt	Пример
Latin Modern	modern, modern-base	Emily Brontë's book
Antykwa Poltawskiego	antypolpoltawskiego	Emily Brontë's book
Antykwa Toruńska	antypol	Emily Brontë's book
Cambria	cambria	Књига Емили Бронте Emily Brontë's book
DejaVu	dejavu	Књига Емили Бронте Emily Brontë's book
DejaVu Condensed	dejavu-condensed	Књига Емили Бронте Emily Brontë's book
Gentium	gentium	Књига Емили Бронте Emily Brontë's book
Iwona	iwona	Emily Brontë's book
Latin Modern Variable	modernvariable, modern-variable	Emily Brontë's book
PostScript	postscript	Emily Brontë's book
TeX Gyre Adventor	adventor, avantgarde	Emily Brontë's book
TeX Gyre Bonum	bonum, bookman	Emily Brontë's book
TeX Gyre Cursor	cursor, courier	Emily Brontë's book
TeX Gyre Heros	heros, helvetica	Emily Brontë's book
TeX Gyre Schola	schola, schoolbook	Emily Brontë's book
TeX Gyre Chorus	chorus, chancery	Emily Brontë's book
TeX Gyre Pagella	pagella, palatino	Emily Brontë's book
TeX Gyre Termes	termes, times	Emily Brontë's book
Euler	eulernova	Emily Brontë's book
Stix2	stixtwo	Књига Емили Бронте Emily Brontë's book
Xits	xits	Књига Емили Бронте Emily Brontë's book

Табела 6.1 Фонтови који су део ConT_EXt дистрибуције

Средишња колона [табеле 6.1](#) наводи име или имена под којим ConT_EXt познаје одређени фонт. Када има два имена, она су синоними. Последња колона приказује како изгледа фонт.

Они који подржавају ћирилицу имају исписан пример и ћирилицом. Што се тиче редоследа у којем су приказани, први је фонт који ConTeXt подразумевано користи, а остали су поређани по абеди, док су последња три фонтови дизајнирани специјално за математику. Обратите пажњу да Euler фонт не може директно да прикаже акцентована слова, тако да видимо Bront's уместо Brontë's.

За читаоце који долазе из Windows света и познају његове подразумеване фонтове, нагласићу да је *heros* исто што и Arial у Windows, док је *termes* исто што и Times New Roman. Они нису потпуно идентични, већ довољни слични, толико да морате бити заиста оштрог ока да бисте приметили разлику.

Фонтови које користи Windows нису *слободан софтвер* (уствари, скоро све у Windows систему није *слободан софтвер*), тако да не смеју бити део ConTeXt дистрибуције. Међутим, ако је ConTeXt инсталиран на Windows систему, онда су ови фонтови већ инсталирани и могу да се користе као и било који други фонт инсталиран на систему који извршава ConTeXt. Ипак, у овом уводу се нећу бавити начином употребе фонтова који су већ инсталирани на систему. Помоћ у вези са овим стварима можете пронаћи на [ConTeXt викију](#).

6.2 Особине фонта

6.2.1 Фонтови, *сџилови* и стилске варијанте

Терминологија у вези са фонтовима је донекле збуњујућа, јер је оно што се понекада назива фонт уствари *фамилија фонтова*. Она се састоји од различитих стилова и варијанти које деле основни дизајн. Нећу се упуштати у расправу о томе која терминологија је тачнија; интересује ме само да разјасним терминологију коју користи ConTeXt. Ту се прави разлика између фонтова, стилова и варијанти (или алтернатива) сваког стила. *Фонтови* укључени у ConTeXt дистрибуцију (то су уствари *фамилије фонтова*) су они које смо видели у претходном одељку. Сада ћемо погледати у *сџилове* и *алтернативе*.

Стилови фонтова

Доналд Е. Кнут је за TeX дизајнирао *Computer Modern* фонт и дао му три различита *сџила* која се називају *roman*, *sans serif* и *teletype*. *Roman* стил је дизајн у коме карактери имају декоративне украсе који су у типолошкој терминологији познати под именом *сџиловице* (*serif*), па је из тог разлога овај стил фонта познат и под називом *serif*. Овај стил се узима као *нормални* или подразумевани стил. *Sans serif* стил, као што му име наговештава, нема ове украсе, па је стога једноставнији, сведенији фонт, понекада познат под другим именима, нпр. линеарни, на шпанском, *paloseco*; овај фонт може бити главни фонт у документу, али је такође погодан за употребу у одређеним фрагментима текста чији је главни фонт *roman* стила, као на пример, у насловима или заглављима страница. Коначно, *teletype* стил је укључен у *Computer Roman* јер је он био дизајниран за писање књига о компјутерском програмирању, које имају дугачке одељке компјутерског *кода*, а он се на штампаним материјалима традиционално представља стилем једнаке ширине који имитира компјутерске термине и старе писаће машине.

Уз ова три *стила* фонтова би могао да се дода и четврти стил намењен за математичке фрагменте. Али пошто TeX аутоматски користи овај стил када уђе у математички режим, и не поседује команде којима се експлицитно укључује или искључује, нити поседује *варијанте* или алтернативе осталих стилова, није уобичајено да се сматра за *стил* у ужем смислу.

ConTeXt нуди команде за два могућа стила: рукописни и калиграфски. Нисам потпуно сигуран шта је разлика између њих јер, с једне стране, ниједан од фонтова укључених у ConTeXt дистрибуцију нема ове стилове у свом дизајну, а са друге стране, према мом мишљењу, калиграфско писање је такође рукописно. Ове команде које ConTeXt нуди за укључивање оваквих стилова, ако се користе са фонтом који их не имплементира, неће изазвати никакву грешку приликом компајлирања: једноставно се ништа неће догодити.

Алтернативне форме фонта

Сваки *стил* поседује више алтернативних форми, а ConTeXt их тако и назива, (*алтернативе*):

- Регуларни или нормални („tf”, од *typeface*).
- Црни слог („bf”, од *boldface*).
- Курзив („it” од *italic*)
- Црни курзив („bi” од *bold italic*)
- Коси („sl” од *slanted*)
- Црни коси („bs” од *bold slanted*)
- Капитал („sc” од *small caps*)
- Средњовековни („os” од *old style*)

Ове *алтернативе* су, као што им име наговештава, узајамно искључиве: када се једна укључи, остале се искључују. То је разлог што ConTeXt нуди команде за укључивање алтернатива, а команде за искључивање не постоје; јер кад укључимо једну алтернативу, искључујемо ону која је била у употреби до тад; тако на пример, ако пишемо курзивом и укључимо црни слог, онде се курзив искључује. Ако желимо да користимо црни слог и курзив истовремено, не морамо да их укључујемо појединачно, већ само треба да укључимо алтернативу која има оба („bi”).

С друге стране, треба имати на уму да мада ConTeXt претпоставља да сваки фонт поседује ове алтернативе, па зато и обезбеђује команде које их укључују, да би оне функционисале и давале видљиви резултат у финалном документу, потребно је и да дизајн фонта поседује одређене форме за сваки стил и алтернативу.

Уствари, многи фонтови не праве разлику у дизајну између косих слова и курзива, нити обезбеђују специјалне облике за капитал.

Разлика између курзива и косих слова

Сличност типографске функције коју имају курзив и коса слова је узрок што многи људи мешају ове две алтернативе. Коса слова се добијају благом ротацијом основног облика. Али курзив је – барем у неким фонтовима – другачији дизајн у којем слова *излегају* искошена јер су тако исцртана; али у стварности нема стварног искошења. Ово се може видети у наредном примеру, у којем смо исписали исту реч три пута у истој величини, довољној да се лако виде разлике. У првој верзији је употребљена регуларна форма, у другој искошена, а у трећој курзив:

italics – *italics* – italics

Обратите пажњу како је у прве две речи дизајн карактера потпуно исти, али у трећој постоје суптилне, али врло очигледне разлике у потезима код неких слова, посебно у начину на који је исписано слово 'а', мада разлике постоје у скоро свим карактерима.

Уобичајена употреба курзива и косих слова је слична и свако сам одлучује да ли да користи једно или друго. Постоји слобода, али би требало да напоменемо да ће документ бити боље сложен и изгледаће лепше ако је употреба курзива и косих слова *конзистентна*. Иначе је у многим фонтовима разлика између курзива и косих слова врло мала, тако да није важно да ли користимо једно или друго.

С друге стране, и курзив и коса слова су алтернативе фонта, што у суштини значи две ствари:

1. Можемо да их користимо само када су дефинисани у фонту.
2. Када укључујемо једну од њих, искључујемо алтернативу која се до тада користила.

Уз команде за курзив и коса слова, ConTeXt обезбеђује и додатне команде за *најлашавање* одређеног текста. Њихова употреба уноси суптилне разлике у поређењу са курзивом или косим словима. Погледајте [одељак 6.5.1](#).

6.2.2 Величина фонта

Сви фонтови са којим ради ConTeXt су засновани на векторској графици, тако да теоретски они могу да се прикажу у било којој величини, мада, као што ћемо видети, то зависи од конкретних инструкција које употребимо за одређивање величине фонта. Ако се другачије не назначи, претпостављамо да је величина фонта 12 тачака.

Сви фонтови које користи ConTeXt су засновани на векторској графици, па су то дакле OpenType или Type 1 фонтови, што значи да фонтови чији су корени из времена пре ових технологија морају поново да се имплементирају. На пример, подразумевани T_EX фонт, *Computer Modern*, који је дизајнирао Кнут, постоји само у одређеним величинама, тако да је поново имплементиран у дизајн под називом *Latin Modern* и који користи ConTeXt, мада у многим документима наставља да се назива *Computer Modern* услед јаког симболизма који фонт још увек има за T_EX системе, јер их је започео и развио Кнут заједно са још једним програмом под називом MetaFont, чија је намена дизајнирање фонтова који би могли да раде са системом T_EX.

6.3 Постављање главног фонта документа

Ако се наведе неки други фонт, ConTeXt ће подразумевано као главни фонт да користи *Latin Modern Roman* у величини од 12 тачака. Овај фонт је оригинално дизајнирао КНУТ да се имплементира у T_EX. То је елегантни фонт римског стила са одличним пропорционалним и декоративним „украсима” – који се називају *сѿойице* (*серифи*) – на појединим потезима, што је врло погодно и за штампани текст и за приказ на екран; мада – и ово је само лично мишљење – није тако погодан за мале екране као што је екран *телефона*, јер се *серифи* или украси нагомилавају, чиме се отежава читање.

Ако желимо да користимо неки други фонт, употребљавамо команду `\setupbodyfont` која нам омогућава не само да променимо тренутни фонт, већ и његову величину и стил. Када желимо да се то примени на цео документ, потребно је да се она постави у преамбулу изворног фајла. Али ако једноставно желимо да променимо фонт на неком месту у тексту, онда морамо да наведемо ово што следи.

Формат команде `\setupbodyfont` је следећи:

`\setupbodyfont`[Опције]

а различите опције команде нам дозвољавају да наведемо:

- **Име фонта**, то може бити било које од симболичких имена фонта која се налазе у [табели 6.1](#).
- **Величину**, која може да се наведе или димензијама (користећи тачку као јединицу мере) или одређеним симболичким именима. Али имајте на уму да мада сам раније рекао како фонтови које користи ConTeXt могу да се скалирају на практично било коју величину, `\setupbodyfont` подржава само оне које су цели бројеви између 4 и 12, као и вредности 14.4 и 17.3. Подразумевано се узима величина од 12 тачака.

`\setupbodyfont`, успоставља оно што бисмо могли да назовемо *основне величине* документа; другим речима, *нормалну* величину карактера на основу које се израчунавају остале величине, на пример за наслове и фусноте. Када командом `\setupbodyfont` променимо главну величину фонта, мењају се и све остале величине које се израчунавају према величини главног фонта.

Осим директно наведене величине карактера (10pt, 11pt, 12pt, итд.) такође можемо да употребимо и нека симболичка имена која на основу текуће величине израчунавају величину карактера која треба да се примени. Та симболичка имена су, поређана од највеће до најмање: `big`, `small`, `script`, `x`, `scriptscript` и `xx`. Тако на пример, ако желимо да поставимо тело текста са `\setupbodyfont` тако да буде веће од 12 тачака, то можемо да урадимо са „`big`”.

- **Стил фонта**, који, као што смо управо навели, може бити `roman` (са серифима), или без серифа (`sans serif`), или стил писаће машине, а за неке фонтове и рукописни (`handwritten`) и калиграфски (`calligraphic`) стил. `\setupbodyfont` нуди различита симболичка имена којима се назначавашу разни стилови. Она се налазе у [табели 6.2](#):

Стил	Дозвољена симболичка имена
Roman	rm, roman, serif, regular
Sans Serif	ss, sans, support, sansserif
Monospaced	tt, mono, type, teletype
Handwritten	hw, handwritten
Calligraphic	cg, calligraphic

Табела 6.2 Силови у `\setupbodyfont`

Колико ја знам, ова различита имена која се подржавају за сваки од стилова су синоними.

6.3.1 Како изгледа фонт

Пре него што одлучимо да у документу употребимо одређени фонт, обично хоћемо да видимо како он изгледа. То скоро увек може да се уради из оперативног система, пошто уз њега обично долази неки алат за испитивање изгледа фонтова инсталираних на систему; али да би олакшао ствар, сам ConTeXt обезбеђује алат који нам омогућава да видимо изглед било ког фонта који може да се користи у систему ConTeXt. То је команда `\showbodyfont`, која генерише табелу са примерима фонта који наведемо.

Формат команде `\showbodyfont` је следећи:

`\showbodyfont[Опције]`

где као опције можемо да наведемо потпуно иста симболичка имена као у `\setupbodyfont`. Тако на пример, `\showbodyfont[schola, 8pt]` ће нам приказати табелу испод, у којој се налазе различити примери фонта schola када је основна величина 8 тачака:

[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>\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

Обратите пажњу да се у првом реду и првој колони табеле налазе одређене команде. Касније, када будемо објаснили значење ових команди, још једном ћемо погледати табеле које генерише команда `\showbodyfont`.

Ако желимо да видимо комплетан опсег карактера у одређеном фонту, можемо да употребимо команду `\showfont[ИмеФонта]`. Ова команда ће нам приказати основни дизајн сваког од карактера без примене команди стила и алтернатива.

6.4 Промена неких особина фонта

6.4.1 Команде `\setupbodyfont` и `\switchtobodyfont`

Када желимо да променимо фонт, стил или величину, можемо да употребимо исту команду којом смо успоставили фонт на почетку документа, онда када нисмо хтели да користимо подразумевани ConTeXt фонт: `\setupbodyfont`. Све што је потребно је да ову команду поставимо на место у документу на којем желимо да променимо фонт. Она врши *сјалну* промену фонта, што значи да ће директно да утиче на главни фонт, а посредно и на све фонтове који зависе од њега.

Команда `\switchtobodyfont` је врло слична команди `\setupbodyfont`. Обе команде нам омогућавају да променимо исте аспекте фонта (сам фонт, стил и величину), али интерно, оне функционишу на другачији начин и предвиђене су за различите потребе. Команда `\setupbodyfont` је предвиђена за *успостављање* главног фонта (и обично јединог) у документу; није ни уобичајено, ни типографски коректно да документ користи више од једног главног фонта (из тог разлога се и назива *главни* фонт). За разлику од тога, `\switchtobodyfont` је намењена за писање делова текста различитим фонтом, или за доделу одређеног фонта посебној врсти пасуса који желимо да дефинишемо у нашем документу.

Ово претходно се уствари тиче интерног функционисања ових команди – али и из угла корисника постоје неке разлике у употреби једне или друге команде. Тачније:

1. Као што већ знамо, `\setupbodyfont` је ограничена на одређени опсег величина, док нам `\switchtobodyfont` омогућава да наведемо практично било коју величину, тако да ако фонт не постоји у тој величини, скалираће се на њу.
2. `\switchtobodyfont` не утиче на текст елементе ни на који начин осим тамо где се користи, за разлику од `\setupbodyfont` која, као што је поменуто изнад, успоставља главни фонт и пошто га мења, такође мења и фонт свих текстуалних елемената чији се фонт израчунава на основу главног фонта.

С друге стране, обе команде не мењају само фонт, стил и величину, већ и остале аспекте придружене фонту, као што је на пример, проред.

`\setupbodyfont` генерише грешку компајлирања ако се затражи недозвољена величина фонта; али не генерише грешку ако се затражи непостојећи фонт, јер се у том случају активира подразумевани (*Latin Modern Roman*) фонт. `\switchtobodyfont` се понаша на исти начин када је у питању фонт, а када је у питању величина, као што сам већ напоменуо, покушава да је постигне скалирајући фонт. Међутим, постоје фонтови који не могу да се скалирају на одређене величине, па ће се и у том случају активирати подразумевани фонт.

6.4.2 Брза измена стила, алтернативе и величине

Измена стила и алтернативе

Уз команду `\switchtobodyfont`, ConTeXt обезбеђује скуп команди које нам омогућавају да брзо променимо стил, алтернативу или величину. Што се тиче ових команди, ConTeXt вики нас упозорава да понекад, када се нађу на почетку пасуса, могу да изазову неке нежељене споредне ефекте, тако да се у тим случајевима препоручује постављање команде `\dontleavehmode` испред.

Стил	Команде које га укључују
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>
Monospaced	<code>\tt</code> , <code>\mono</code> , <code>\teletype</code> ,
Handwritten	<code>\hw</code> , <code>\handwritten</code> ,
Calligraphic	<code>\cf</code> , <code>\calligraphic</code>

Табела 6.3 Команде за прелаз на различите стилове

Табела 6.3 садржи команде које нам омогућавају да променимо стил без измене било ког другог аспекта; а табела 6.4 садржи команде које нам омогућавају да променимо само алтернативу.

Алтернатива	Команде које је укључују
Normal	<code>\tf</code> , <code>\normal</code>
Italic	<code>\it</code> , <code>\italic</code>
Bold	<code>\bf</code> , <code>\bold</code>
Bold-italic	<code>\bi</code> , <code>\bolditalic</code> , <code>\italicbold</code>
Slanted	<code>\sl</code> , <code>\slanted</code>
Bold-slanted	<code>\bs</code> , <code>\boldslanted</code> , <code>\slantedbold</code>
Small caps	<code>\sc</code> , <code>\smallcaps</code>
Medieval	<code>\os</code> , <code>\mediaeval</code>

Табела 6.4 Команде за укључивање одређене алтернативе

Све ове команде задржавају своје важење док се експлицитно не укључи други стил или алтернатива, или док се не заврши *група* унутар које је декларисана команда. Дакле, када желимо да команда утиче само на одређени део текста, потребно је да тај део поставимо у групу, као у наредном примеру, где се креирањем групе реч *thought* исписује курзивом, сваки пут када се појављује као именица, а не као глагол.

```
I thought a {\it thought} but
the {\it thought} I thought wasn't
the {\it thought} I thought I thought.
If the {\it thought} I thought I thought
had been the {\it thought} I thought
I wouldn't have thought so much!
```

```
I thought a thought, but the thought I thought, wasn't the tho-
ught I thought I thought. If the thought I thought I thought
had been the thought I thought I wouldn't have thought so
much!
```

Суфикси за истовремену измену алтернативе и величине

Команде које мењају стил или алтернативу у својим двословним верзијама (`\tf`, `\it`, `\bf`, итд.) подржавају разне *суфиксе* који утичу на величину фонта. Суфикси `a`, `b`, `c` и `d` повећавају величину фонта, тако што је редом множе са 1.2 , $1.2^2 (= 1.44)$, $1.2^3 (= 1.728)$ или $1.2^4 (= 2.42)$. Погледајмо пример:

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

тест, тест, тест, тест, ТЕСТ

суфикси `x` и `xx` смањују величину фонта тако што је редом множе са $0,8$ и $0,6$:

```
\tf тест, \tfx тест, \tfxx тест
```

ТЕСТ, тест, тест

Суфикси `'x'` и `'xx'` примењени на `\tf` нам омогућавају да скратимо команду, тако да `\tfx` може да се напише као `\tx`, а `\tfxx` као `\txx`.

Доступност ових разних суфикса зависи од конкретне имплементације фонта. Према ConTeXt референтном упутству из 2013. године (који се углавном односи на Mark II) једини суфикс за који се гарантује да ради је `'x'`, а остали могу а не морају да буду имплементирани; или могу бити имплементирани само за неке алтернативе.

Да би се спречиле недоумице, увек можемо да употребимо команду `\showbodyfont` о којој сам раније говорио (у одељку 6.3.1). Ова команда приказује табелу која не само што даје изглед фонта, већ и то како он изгледа у сваком од својих стилова и алтернатива, као и све доступне суфиксе за промену величине.

Хајде да још једном погледамо табелу коју приказује `\showbodyfont`:

[modern-designsize] [modern-designsize,10pt]													
	<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

Ако пажљиво погледамо табелу, видећемо да прва колона садржи стилове фонта (`\rm`, `\ss` и `\tt`). Први ред, са леве стране, садржи алтернативе (`\tf`, `\sc`, `\sl`, `\it`, `\bf`, `\bs` и `\bi`), док десна страна првог реда садржи остале доступне суфиксе, мада само са регуларним алтернативама.

Важно је приметити да ће промена величине фонта било којим од ових суфикса то урадити само у буквалном смислу, док ће остале вредности обично придружене величини фонта, као што је проред, остати нетакнуте.

Прилагођавање фактора скалирања суфикса

Ако желимо да прилагодимо фактор скалирања можемо да употребимо команду `\definebodyfontenvironment` чији је формат:

```
\definebodyfontenvironment[одређена величина][скалирања]
\definebodyfontenvironment[default][скалирања]
```

У првој верзији бисмо најпре редефинисали скалирање за одређену величину главног фонта која се поставља помоћу `\setupbodyfont` или са `\switchtobodyfont`. На пример:

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

би обезбедило да када је главни фонт величине 10 тачака, суфикс 'a' мења на 12 тачака, суфикс 'b' на 14, суфикс 'c' би помножио оригиналну величину фонта са 2, а суфикс 'd' са 3. Запазите да је за а и b наведена фиксна димензија, али је за c и d наведен фактор множења оригиналне величине.

Али када је први аргумент команде `\definebodyfontenvironment` има вредност „default”, онда ћемо редефинисати вредности скалирања за све могуће величине фонта, а као вредност скалирања можемо само да наведемо фактор множења. Дакле, када напишемо:

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

наводимо да која год да је величина главног фонта, суфикс a би требало да је множи са 1,3, b са 1,6, c са 2 и d са 4.

Као и суфиксе `xx`, `x`, `a`, `b`, `c` и `d`, командом `\definebodyfontenvironment` можемо да доделимо вредности скалирања и за кључне речи „`big`”, „`small`”, „`script`” и „`scriptscript`”. Ове вредности се додељују свим величинама придруженим овим речима у командама `\setupbodyfont` и `\switchtobodyfont`. Такође се примењују и на следеће команде, чија корисност (барем тако мислим) може да се изведе из њиховог имена:

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

Ако желимо да видимо подразумеване величине одређеног фонта, можемо да употребимо команду `\showbodyfontenvironment[Фонт]`. Када је, на пример, извршимо за `modern` фонт, добијамо следећи резултат:

[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 Дефинисање команди и кључних речи за величине фонта, стилове и алтернативе

Предефинисане команде за измену величине фонта, стилова и варијанти су довољне. Ипак, ConTeXt нам омогућава да:

1. Додамо своју команду за измену стила фонта, величине или варијанте.
2. Додамо синониме за стилове или варијанте имена која препознаје команда `\switchtobodyfont`.

За то он обезбеђује следеће команде:

- `\definebodyfontswitch`: нам омогућава да дефинишемо команду за промену величине фонта. На пример, ако желимо да дефинишемо команду `\osam` (или команду `\viii`¹) којом величину фонта постављамо на 8 pt, потребно је да напишемо:

```
\definebodyfontswitch[osam][8pt] или \definebodyfontswitch[viii][8pt]
```

- `\definefontstyle`: нам омогућава да дефинишемо једну или више речи које могу да се употребе у командама `\setupbodyfont` или `\switchtobodyfont` за постављање одређеног стила фонта; тако на пример, ако *sans serif* желимо да зовемо неким другим именом (нпр. „linear”) можемо да напишемо

```
\definefontstyle[linear][ss]
```

Особеност команде `\definefontstyle` је што дозвољава да се неколико речи истовремено користи за исти стил, тако да можемо наставити пример:

```
\definefontstyle[linear, tehn, bezstopica][ss]
```

- `\definealternativestyle`: нам дозвољава да варијанти фонта придружимо име. Ово име би могло да функционише као команда, или може да је препознаје опција `style` у командама које дозвољавају конфигурацију стила. Дакле, следећи фрагмент

```
\definealternativestyle[strong][\bf][]
```

ће направити команду `\strong` као и кључну реч „strong” коју ће препознавати опција `style` у командама које је подржавају. Могли смо да ставимо и реч „bold”, али је ConTeXt већ користи, тако да сам као алтернативу изабрао израз који се користи у HTML, тачније „strong”.



Не знам шта ради трећи аргумент команде `\definealternativestyle`. Обавезан је, па дакле не може да се изостави; али једина информација коју сам нашао у вези са њим се налази у ConTeXt референтном приручнику где се каже да је овај трећи аргумент релевантан само за наслове поглавља и одељака „где, осим `\cap`, морамо да се држимо фонта који се овде користи” (??)

6.5 Остале ствари везане за употребу неких алтернатива

Међу различитим алтернативама фонта, постоје две које захтевају одређена појашњења:

6.5.1 Курзив, коса слова и наглашавање

И курзив и коса слова се углавном употребљавају за типографско истицање фрагмента текста како би се привукла пажња читаоца на њега. Другим речима, да се фрагмент *нагласи*.

Наравно, текст можемо да нагласимо тако што експлицитно укључимо курзив или коса слова. Али ConTeXt нуди алтернативну команду која много кориснија и интересантнија, а

¹ Упамтите да за разлику од командних симбола, имена команди у систему ConTeXt смеју да се састоје само од слова.

посебно је намењена за наглашавање фрагмента текста. То је команда `\em` чије име долази од речи *emphasis*. За разлику од `\it` и `\sl`, које су чисто типографске команде, `\em` је *концепцијална* команда; она функционише на другачији начин, тако да је свестранија, до те мере да ConTeXt документација препоручује да се радије употребљава `\em` уместо `\it` или `\sl`. Када употребимо ове две последње команде, ми систему ConTeXt говоримо коју алтернативу фонта желимо да користимо; али када употребимо `\em` говоримо му који ефекат желимо да постигнемо, а систему ConTeXt остављамо да одлучи како то да изведе. Обично, ефекат наглашавања или истицања нечега се постиже тако што укључимо курзив или коса слова, али то зависи од контекста. Дакле, ако у тексту који је већ исписан курзивом – или косим словима – употребимо `\em`, команда ће текст да нагласи на супротни начин – у овом случају усправним словима.

Тако да следећи пример:

```
{\em Једна од најлепших  
орхидеја света је  
{\em Thelymitra variegata}  
или Јужна Краљица од Сабе.}
```

*Једна од најлепших орхидеја света је Thelymitra
variegata или Јужна Краљица од Сабе.*

Приметите да прво `\em` укључује курзив (уствари коса слова, погледајте испод), а да га друго `\em` искључује и исписује речи „Thelymitra variegata” нормалним усправним стилем.

Још једна предност команде `\em` је што то није алтернатива, тако да не искључује алтернативу коју смо до тада имали активну, па на пример, у тексту исписаним црним слогом, командом `\em` ћемо добити црна искошена слова без потребе да експлицитно позовемо команду `\bs`. Слично, ако се у већ истакнутом тексту појави команда `\bf`, истицање неће да престане.

Команда `\em` подразумевано укључује коса слова а не курзив, али то можемо да променимо командом `\setupbodyfontenvironment[default][em=italic]`.

6.5.2 Капитал и лажни капитал

Капитал је типографски ресурс који је често много бољи од верзала (великих слова). Капитал нам пружа облик великих слова, али висину на линији одржава истом као за курент (мала слова). То је разлог што је капитал стилска варијанта курента. У одређеним контекстима, капитал замењује верзал и посебно је користан за писање римских бројева, или наслова поглавља. У академским текстовима је такође обичај да се капиталом исписују имена цитираних аутора.

Проблем је што капитал не имплементирају сви фонтови, а и они који то раде, не раде за све своје *стилове фонџа*. Уз то, пошто је капитал алтернатива курзиву, црном слогу или косим словима, сагласно општим правилима која смо поставили у овом поглављу, све ове типографске могућности не смеју истовремено да се користе.

Ови проблеми се решавају употребом *лажној калџиџала*. ConTeXt нам омогућава да га креирамо командама `\cap` или `\Cap`; о њима се говори у одељку 10.2.1.

6.6 Употреба и конфигурација боја

ConTeXt нуди команде за промену боја комплетног документа, неких његових елемената, или одређених делова текста. Такође нам обезбеђује команде за учитавање стотина предефинисаних боја у меморију и могућност да видимо које су њихове компоненте.

6.6.1 Процедуре за слагање фрагмената текста у боји

Већина подесивих команди система ConTeXt подржава опцију под именом „color” која нам омогућава да наведемо боју којом ће се исписати текст на који утиче та команда. Дакле, ако желимо да наведемо да наслови поглавља треба да се исписују плавом бојом, потребно је само да напишемо:

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

Употребом ове процедуре можемо да обојимо наслове, заглавља, фусноте, маргиналне напомене, линије и штрафте, табеле, наслове табела или слика, итд. Предност је што ће коначни резултат бити конзистентан (сви текстови који врше исту функцију ће бити исписивани истом бојом) и моћи ће лакше глобално да се промени.

Фрагмент текста можемо и директно да обојимо, мада да би се спречила претерана употреба различитих боја, што није пријатно из типографске перспективе, или неконзистентна употреба, у општем случају се не препоручује директно бојење, већ употреба онога што бисмо могли да назовемо *семантичко бојење*, то јест, уместо да напишемо

```
\color[red]{Врло важан текст}
```

дефинишемо команду за врло важан текст којој се додели боја. На пример

```
\definehighlight[vazno][color=red]
\vazno{Врло важан текст}
```

6.6.2 Промена боје позадине и предњег плана документа

Ако желимо да променимо боју комплетног документа, зависно од тога да ли треба да утичемо на боју позадине или боју предњег плана (текста), користићемо `\setupbackgrounds` или `\setupcolors`. Тако, на пример

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

Ова команда ће поставити позадину страница на плаво. Као вредност за „backgroundcolor” можемо да употребимо име било које од предефинисаних боја.

Ако глобално желимо да променимо боју предњег плана кроз цео документ (од тачке на којој се уметне команда) користимо `\setupcolors`, где опција „textcolor” контролише боју текста. На пример:


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

ће поставити боју текста на црвено.

6.6.3 Комадне за бојење одређених фрагмената текста

Општа команда за бојење малих делова текста је

```
\color[ИмеБоје]{Текст који се боји}
```

За веће делове текста пожељно је да употреби

```
\startcolor[ИмеБоје] ... \stopcolor
```

Обе користе име неке предефинисане боје. Ако желимо да дефинишемо боју док радимо, можемо да употребимо команду `\colored`. На пример:

```
Три \colored[r=0.1, g=0.8, b=0.8]
{обојене} мачке.
```

Три обојене мачке.

6.6.4 Предефинисане боје

ConTeXt читава најчешће коришћене предефинисане боје наведене у [табели 6.5](#).¹

Име	Светли тон	Средњи тон	Тамни тон
black			
white			
gray	lightgray	middlegray	darkgray
red	lightred	middlered	darkred
green	lightgreen	middlegreen	darkgreen
blue	lightblue	middleblue	darkblue
cyan		middlecyan	darkcyan
magenta		middlemagenta	darkmagenta
yellow		middleyellow	darkyellow

Табела 6.5 Предефинисане ConTeXt боје

Постоје и остале колекције боја које се подразумевано не читавају, али које могу да се прочитају командом

```
\usecolors[ИмеКолекције]
```

где ИмеКолекције може бити

- „crayola”, 235 боја које имитирају нијансе маркера.
- „dem”, 91 боја.
- „ema”, 540 дефиниција боја базираних на бојама које користи Emacs.
- „rainbow”, 91 боја за употребу у математичким формулама.
- „ral”, 213 дефиниција боја из *Deutsches Institut für Gütesicherung und Kennzeichnung* (Немачки институт за осигурање квалитета и означавање).

¹ Ова листа може да се пронађе у референтном упутству и на ConTeXt викију, али сам прилично сигуран да није комплетна јер у овом документу, на пример, без потребе за читавањем било какве додатне боје, користимо „наранџасту” – која није у [табели 6.5](#) – за наслове одељака.

- „rgb”, 223 боје.
- „solarized”, 16 боја базираних на solarized шеми.
- „svg”, 147 боја.
- „x11”, 450 стандардних боја за X11.
- „xwi”, 124 боје.

Фајлови дефиниција боја се налазе у директоријуму „context/base/mkiv” дистрибуције и њихово име одговара шеми „colo-imp-IME.mkiv”. Информације о разним колекцијама предефинисаних боја које сам управо навео су засноване на дистрибуцији коју ја поседујем. Одређене колекције, или број боја дефинисаних у њима би могао да се промени у будућности.

Да бисмо видели боје које свака од ових колекција садржи, можемо да употребимо команду `\showcolor[ИмеКолекције]` која је описана испод. Да бисмо користили неке од ових боја, најпре морамо да их учитамо у меморију командом (`\usecolors[ИмеКолекције]`), па затим морамо задамо `\color` или `\startcolor` командама име боје. На пример, следећи низ:

```
\usecolors[xwi]
\color[darkgoldenrod]{Станлио и Олио}
```

ће написати

Станлио и Олио

6.6.5 Да видимо доступне боје

Команда `\showcolor` приказује листу боја у којој можете видети изглед боје, њен изглед када се користи у сивим тоновима, црвену, зелену и плаву компоненту боје, као и име под којим је познаје ConTeXt. Ако се употреби без аргумената, `\showcolor` ће приказати боје које се користе у текућем документу. Али, као аргумент можемо да наведемо било коју од предефинисаних колекција боја о којима смо говорили у одељку 6.6.4, тако да ће `\showcolor[solarized]` на пример, да нам прикаже 16 solarized боја у тој колекцији:

		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

Ако желимо да видимо `rgb` компоненте одређене боје, можемо да употребимо команду `\showcolorcomponents[ИмеБоје]`. Ово је корисно ако покушавамо да дефинишемо одређену боју, тако да видимо састав неке боје која јој је слична. На пример, `\showcolorcomponents[darkgoldenrod]` ће да нам прикаже:

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

6.6.6 Дефинисање сопствених боја

Команда `\definecolor` нам омогућава да или клонирамо постојећу боју, или дефинишемо нову боју. Клонирање постојеће боје је просто као креирање алтернативног имена за њу. Да бисте то урадили, потребно је да напишете:

```
\definecolor[Нова боја][Стара боја]
```

Ово ће обезбедити да је „Нова боја” потпуно иста као и „Стара боја”.

Али основна употреба команде `\definecolor` је за креирање нових боја. Да би се то урадило, команда мора да се употреби на следећи начин:

```
\definecolor[ИмеБоје][Дефиниција]
```

где *Дефиниција* може да се уради применом до шест различитих шема генерисања боје:

- **RGB боје:** дефиниција RGB боја је једна од најраспрострањенијих; заснива се на идеји да је могуће представити боју мешањем, сабирањем, три примарне боје: црвене ('r' за *red*), зелене ('g' за *green*) и плаве ('b' за *blue*). Свака од ових компоненти се наводи као децимални број између 0 и 1.

```
\definecolor[limeta 1][r=0.75, g=1, b=0]: Текст у „limeta 1”.
```

- **Hex боје:** овај начин представљања боја је такође заснован на RGB шеми, али се црвена, зелена и плава компонента представљају као тробајтни хексадецимални број у којем први бајт представља вредност црвене, други вредност зелене, а трећи вредност плаве. На пример

```
\definecolor[limeta 2][x=BFFF00]: Текст у „limeta 2”.
```

- **CMYK боје:** овај модел генерисања боја је оно што се назива „субтрактивни модел” и заснива се на мешавини пигмената следећих боја: цијан ('c'), магента ('m'), жута ('y', од *yellow*) и црна ('k', од *key*). Свака од ових компоненти се наводи као децимални број између 0 и 1.

```
\definecolor[limeta 3][c=0.25, m=0, y=1, k=0]: Текст у „limeta 3”.
```

- **HSL/HSV:** овај модел боја се заснива на мерењу нијансе ('h', од *hue*), засићења ('s' од *saturation*) и луминансе ('l' или понекад 'v', од *value*). Нијанса одговара броју између 0 и 360; засићење и луминанса морају бити децимални бројеви између 0 и 1. На пример

```
\definecolor[limeta 4][h=75, s=1, v=1]: Текст у „limeta 4”
```

- **HWB боје:** HWB модел је предложени стандард за CSS4 који мери нијансу ('h', од *hue*), и ниво беле ('w', од *whiteness*) и црне ('b', од *blackness*). Нијанса одговара броју између 0 и 360, док су белина и црнина представљени децималним бројем између 0 и 1.

```
\definecolor[Azure][h=75, w=0.2, b=0.7] Текст у „Azure”.
```

- **Скала сиве:** заснована на компоненти под називом ('s', од *scale*) која мери количину сиве. Мора бити број између 0 и 1. На пример:

```
\definecolor[svetlo siva][s=0.65] Текст у „svetlo siva”.
```

Такође је могуће да се дефинише нова боја из неке друге боје. На пример, боја којом су исписани наслови у овом уводу је дефинисана као

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

Глава 7

Структура документа

Садржај: 7.1 Структурна подела у документима; 7.2 Типови одељака и њихова хијерархија; 7.3 Заједничка синтакса команди поделе; 7.4 Формат и конфигурација одељака и њихових наслова; 7.4.1 Команде `\setuphead` и `\setupheads`; 7.4.2 Делови наслова одељака; 7.4.3 Контрола нумерације (у нумерисаним одељцима); 7.4.4 Боја и стил наслова; 7.4.5 Место броја и текста наслова; 7.4.6 Команде или акције које се извршавају пре или након исписивања наслова; 7.4.7 Остале особине које могу да се подесе; 7.4.8 Остале `\setuphead` опције; 7.5 Дефинисање нових команди поделе; 7.6 Макроструктура документа;

7.1 Структурна подела у документима

Осим за врло кратке текстове (на пример, као што је писмо), документ се обично организује у блокове или групе текста које у општем случају следе хијерархијски редослед. Нема стандардног начина за давање имена овим блоковима: у романима, на пример, структурна подела се обично назива „поглавља” мада неки – они дужи – имају и веће блокове који се обично називају „делови” и који групишу већи број поглавља. Драмска дела праве разлику између „чинова” и „сцена”. Академски приручници су (понекад) подељени на „делове” и „лекције”, „теме” или „поглавља” која такође и сама често имају сопствене поделе; иста врста сложене хијерархијске поделе често постоји и у осталим академским или техничким документима (као што је овај који читате, посвећен објашњавању компјутерског програма или система). Чак су и закони организовани у „књиге” (најдуже и најсложеније, као што су кодекси) „насловe”, „поглавља”, „одељке”, „пододељке”. Научни и технички документи такође могу достићи до шест, седам, па чак и осам нивоа угњеждавања ових врста подела.

Ово поглавље се бави анализом механизма које ConTeXt нуди као подршку за ове структурне поделе. На све њих ће се односити један свеобухватни појам „одељци”.

Не постоји јасан појам који нам омогућава да уопштено говоримо о свим овим врстама структурне поделе. Појам „одељак” који сам ја изабрао, концентрише се на структурну поделу више него било који други, мада је лоша страна то што се једна од предефинисаних структурних подела у систему ConTeXt назива „section” (односно одељак). Надам се да то неће унети забуну и верујем да ће бити прилично једноставно да се из контекста одреди да ли говоримо о одељку као општој одредници за структурне поделе, или о специфичној подели коју ConTeXt назива одељак.

Сваки „одељак” (у општем смислу) подразумева:

- Разумно велику *структурну поделу докумената* која може да укључи и остале поделе нижег нивоа. Из ове перспективе, „одељци” подразумевају блокове текста који међу собом имају хијерархијски однос. Из угла својих одељака, документ као целина може да се посматра као стабло. Документ *per se* је стабло, свако од његових поглавља је грана, а свака од њих може да има гранчице које се такође деле, и тако даље.

Да би се документ читао и разумео, врло је важно да има што јаснију структуру. Слово-слагач то не ради, ово је ауторов задатак. И мада није до система ConTeXt да нас учини бољим ауторима него што то јесмо, велики скуп команди за поделу документа које

обезбеђује, при чему је хијерархија међу њима прилично јасна, може да нам помогне у писању докумената који имају бољу структуру.

- *Име сџрукџуре* које бисмо називали њен „наслов” или „лабела”. То име се штампа:
 - Увек (или скоро увек) на месту у документу где почиње структурна подела.
 - Повремено и у садржају, у заглављу или подножју страница које садрже одређени одељак.

ConTeXt нам омогућава да ове задатке аутоматизујемо на такав начин да је особине форматирања којима се наслов структурне јединице штампа потребно навести само једном, као и то да ли би он требало или не да буде део садржаја, или заглавља или подножја. Да би то извео, ConTeXt једино мора да зна где је почетак и крај сваке структурне јединице, како се зове и на којем је хијерархијском нивоу.

7.2 Типови одељака и њихова хијерархија

ConTeXt разликује *нумерисане* и *нenumерисане* одељке. Ови први се, као што им име наговештава, аутоматски нумеришу и шаљу у садржај, а понекада и у заглавља и/или подножја страница.

ConTeXt има хијерархијски поређане предефинисане команде за дељење приказане у [табели 7.1](#).

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

Табела 7.1 ConTeXt команде за поделу

Потребно је пружити и следећа објашњења у вези са предефинисаним одељцима:

- У [табели 7.1](#) су команде поделе приказане у свом традиционалном облику. Али управо ћемо видети да оне могу да се користе и као *окружења* (`\startchapter ... \stopchapter`, на пример) и да се тај приступ уствари препоручује.
- У садржај се поставља само првих 6 нивоа поделе. Међутим, моји тестови су показали да постоји до 12 нивоа: након „`\subsubsubsection`” долази „`\subsubsubsubsection`”, и тако даље, све до „`\subsubsubsubsubsubsubsubsubsubsection`”, или „`\subsubsubsubsubsubsubsubsubsubject`”.

Али требало би да имамо у виду ће ови (превише дубоки) нижи нивои представљени изнад врло тешко да унапреде разумевање текста! Врло је вероватно да имамо велике одељке који се неизбежно баве са

неколико тема, па ће ово читаоцу да отежа схватање њиховог садржаја. Ако се иде предубоко у нивое, читалац губи осећај комплетног текста, и то је последица превелике фрагментације материјала. Ја лично сматрам да је у општем случају четири нивоа сасвим довољно; врло ретко би било потребно да се иде до шест или седам нивоа, али било која већа дубина обично није добра идеја.

Из перспективе писања изворног фајла, чињеница да креирање додатних поднивоа значи додавање још једног „sub” на претходни ниво, може учинити изворни фајл скоро нечитљивим: није једноставно разумети ниво команде која се зове „subsubsubsubsubsection” јер морам да бројим све те „sub”-ове! Мој савет је, дакле, да ако је заиста потребно толико нивоа дубине, од петог нивоа па надаље (subsubsection) дефинишемо сопствене команде за поделу (погледајте одељак 7.5) и да им дамо имена која су јаснија од предефинисаних.

- Највиши ниво (`\part`) постоји само за нумерисане наслове и има особину да се наслов дела не штампа. Међутим, чак и ако се наслов не штампа, умеће се празна страница (па можемо претпоставити да ће се наслов штампати онда када корисник реконфигурише команду), а нумерација *дела* се узима у обзир за израчунавање нумерације поглавља и осталих одељака.

Према ConTeXt викију, разлог што подразумевана верзија команде `\part` не штампа ништа је што скоро увек наслов на овом нивоу тражи посебан распоред; и мада је то тачно, мени се не чини као довољно добар разлог, јер у пракси се често редефинишу и поглавља и одељци, а чињеница да делови не штампају ништа приморава почетнике да *уроне* у документацију како би открили шта није у реду.

- Мада је први ниво поделе „део”, то је само теоретски и апстрактно. У одређеном документу први ниво поделе ће бити онај који одговара првој команди поделе у документу. Значи, у документу који нема делове већ поглавља, поглавље ће бити први ниво. Али ако документ нема ни поглавља, већ само одељке, онда хијерархија тог документа почиње од одељака.

7.3 Заједничка синтакса команди поделе

Све команде поделе, укључујући и све нивое које креира корисник (погледајте одељак 7.5), подржавају следеће алтернативне облике синтаксе (ако, на пример, користимо ниво „section”):

```
\section [Лабела] {Наслов}
\section [Опције]
\startsection [Опције] [Променљиве] ... \stopsection
```

У сва три начина, аргументи унутар великих заграда нису обавезни и могу да се изоставе. Њима ћемо се позабавити одвојено, али прва синтакса од ове три помаже да се објасни да се у Mark IV препоручује употреба трећег приступа.

- Први облик синтаксе, који би могли да назовемо „*класична*”, команде прихватају два аргумента, један необавезан унутар великих заграда, а други обавезан унутар витичастих заграда. Необавезни аргумент служи да се команди придружи лабела која ће се користити за интерне референце (погледајте одељак 9.2). Обавезан аргумент унутар витичастих заграда је наслов одељка.
- Остала два облика су више у ConTeXt стилу: све што је команди потребно се преноси вредностима и опцијама унутар великих заграда.

Присетимо се да сам у одељцима 3.3.1 и 3.4 рекао да се у систему ConTeXt опсег важења сваке команде наводи витичастим заградама, а њене опције у великим заградама. Али ако размислимо о томе, наслов одређене команде поделе није опсег важења њене примене, па да би се остало у сагласности са општом синтаксом, не би требало да се наводи унутар витичастих заграда, већ као опција. ConTeXt дозвољава овај изузетак јер је то класични начин рада у систему TeX, али нуди и алтернативне облике синтаксе који су конзистентнији са општим дизајном система.

Тип опција је додела вредности (ИмеОпције = Вредност) и оне су следеће:

- **reference**: лабела за унакрсне референце.
- **title**: наслов одељка који ће се штампати у телу документа.
- **list**: наслов одељка који ће се штампати у садржају.
- **marking**: наслов одељка који ће се штампати у заглављу или подножју страница.
- **bookmark**: наслов одељка који ће се конвертовати у *bookmark* у PDF фајлу.
- **ownnumber**: ова опција се користи у случају када се одељак не нумерише аутоматски; тада ће ова опција доделити одељку наведени број.

Наравно, опције „list”, „marking” и „bookmark” би требало да се користе само онда када нам је потребан другачији наслов који ће заменити главни постављен опцијом „title”. Ово је веома корисно, на пример, када је наслов сувише дугачак за заглавље; мада то можемо да постигнемо и помоћу команди `\nomarking` и `\nolist` (нешто врло слично). С друге стране, треба имати на уму да ако текст наслова (опција „title”) садржи запете, онда мора да се постави унутар витичастих заграда, и комплетан текст и сама запета, како би се обезбедило да ConTeXt зна да је запета део наслова. Исто важи и за остале опције: „list”, „marking” и „bookmark”. Дакле, да не бисмо морали да водимо рачуна о томе има ли у наслову запета, мислим да је добра идеја развити навику да се вредност било које од ових опција поставља унутар витичастих заграда.

На пример, следеће линије ће креирати поглавље под називом „Тест поглавље” којем се придружује лабела за унакрсне референце „test”, док ће наслов у тексту бити „Поглавље тест” уместо „Тест поглавље”.

```
\chapter
[
  title={Тест поглавље},
  reference={test},
  marking={Поглавље тест}
]
```

Синтакса `\startТипСекције` претвара одељак у *окружење*. Она је конзистентнија са чињеницом да је, као што сам рекао на почетку, позадина сваког одељка издвојени блок текста, мада *окружења* која креирају команде поделе ConTeXt подразумевано не сматра за *групе*. Ова процедура је оно што Mark IV препоручује; врло могуће јер овај начин успостављања одељака од нас захтева да експлицитно наведемо где сваки одељак почиње и где се завршава. Тако се лакше обезбеђује конзистентност структуре, а вероватно и боља подршка за XML и EPUB излаз. Уствари, то је за XML излаз од суштинске важности.

Када користимо `\startИмеСекције` унутар великих заграда може да се наведе једна или више променљивих. Њихове вредности могу касније да се употребе на осталим местима у документу командом `\structureuservariable`.

Постојање корисничких променљивих у систему ConTeXt омогућава врло напредну употребу јер помоћу њих могу да се донесу одлуке да ли да се фрагмент компајлира или не, или на који начин да се то уради, или којим шаблоном, све у зависности од вредности одређене променљиве. Међутим, ови ConTeXt алати излазе ван оквира материјала којим желим да се бавим у овом уводу.

7.4 Формат и конфигурација одељака и њихових наслова

7.4.1 Команде `\setuphead` и `\setupheads`

ConTeXt подразумевано додељује одређене карактеристике сваком нивоу поделе које углавном (али не само то) утичу на формат у којем се наслов приказује у главном телу документа, али не и начин на који се наслов приказује у садржају или заглављима и подножјима. Ове особине можемо да променимо командом `\setuphead` чија синтакса је следећа:

`\setuphead[Одељци][Опције]`

где

- **Одељци** представља име једног или више одељака (раздвојених запетама) на које утиче команда. Може бити:
 - Било који од предефинисаних одељака (part, chapter, title, итд.), па у том случају можемо на њих да указујемо било по имену, било по њиховом нивоу. За указивање по нивоу користимо реч „section-*БрНивоа*”, где је *БрНивоа* број нивоа жељеног одељка. Тако да је „section-1” исто што и „part”, „section-2” исто што и „chapter”, итд.
 - Било која врста одељка који смо сами дефинисали. Што се тога тиче, погледајте одељак 7.5.
- **Опције** представља конфигурационе опције. То су опције типа експлицитне доделе вредности (ИмеОпције = вредност). Број доступних опција је огроман (преко шездесет) па ћу их зато објаснити груписањем у категорије сагласно њиховој функцији. Међутим, морам истаћи да нисам успео одредити шта неке од ових опција раде нити како се користе. О њима нећу да говорим.

Раније сам рекао да `\setuphead` утиче на одељке који се експлицитно наведу. Али то не значи да измена одређеног одељка не би требало да утиче на остале осим ако експлицитно нису били поменути у команди. Уствари је тачно супротно: измена одељка утиче на остале одељке *који су везани* за њега, чак и када то није експлицитно наведено у команди. Постоје две врсте везе између различитих одељака:

- Ненумерисане команде су везане са одговарајућом нумерисаном командом истог нивоа, тако да промена изгледа нумерисане команде утиче и на изглед ненумерисане команде

истог нивоа; али не и обрнуто: промена нумерисане команде не утиче на нумерисану. Ово значи да ако, на пример, променимо неки аспект „chapter” (ниво 2) такође мењамо и тај аспект за „title”; али ако изменимо „title”, то неће утицати на „chapter”.

- Команде су повезане хијерархијски, тако да ако на неком нивоу изменимо *одређене особине*, та измена ће утицати и на све нивое који долазе након тог нивоа. Ово важи само за неке особине. На пример, боју: ако подесимо да се пододељци исписују црвеном бојом, такође мењамо и боју за подпододељке, подподододељке на црвену, итд. Али то се не дешава за остале особине као што је стил фонта, на пример.

Уз команду `\setuphead`, ConTeXt обезбеђује и команду `\setupheads` која глобално утиче на све команде поделе. У вези ове команде, ConTeXt вики наводи да су неки људи пријавили да не функционише. Резултат мојих тестова је да команда ради за неке опције, али за неке не ради. Тачније, не ради са опцијом „style”, што је изненађујуће, јер је стил наслова највероватнија ствар коју би корисник пожелео глобално да промени за све наслове. Али према мојим тестовима, команда ради са осталим опцијама, као што су на пример, „number” или „color”. Тако ће на пример, `\setupheads[color=blue]` обезбедити да се сви наслови у нашем документу исписују плавом бојом.

Пошто сам помало лењ да тестирам сваку опцију и видим да ли ради или не са командом `\setupheads` (упамтите да постоји више од шездесет опција), у даљем тексту ћу помињати само `\setuphead`.

Коначно: пре испитивања одређених опција, требало би да обратимо пажњу на нешто што је речено на ConTeXt викију, мада се вероватно не налази на одговарајућем месту: неке опције функционишу само ако користимо синтаксу `\startИмеОдељка`.

Ова информација је дата у вези са командом `\setupheads`, а не у вези са `\setuphead`. Пошто је то место на којем је објашњен највећи део опција, чини се да је то најразумније место да се ова информација изложи, ако већ треба да се каже само на једном месту. С друге стране, информација помиње само опцију „insidesection” и није јасно да ли то исто важи и за остале опције.

7.4.2 Делови наслова одељка

Пре него што се упустимо у одређене опције којима можемо подесити изглед наслова, корисно је да истакнемо да наслов може имати до три различита дела. ConTeXt нам омогућава да их форматирамо појединачно, или заједно. Елементи наслова су следећи:

- **Сам наслов**, то јест текст који га сачињава. У принципу, увек се приказује овај наслов, осим за одељке типа „part” за које се подразумевано наслов не исписује. Опција која контролише да ли се наслов приказује или не је „placehead” и њене вредности могу бити „yes”, „no”, „hidden”, „empty” или „section”. Значење прве две је јасно. Али нисам сигуран у вези резултата осталих вредности ове опције.

Дакле, ако желимо да се наслов приказује у првом нивоу одељака, подешавање би требало да буде:

```
\setuphead
[part]
[placehead=yes]
```



Као што већ знамо, наслов одређених одељака може аутоматски да се пошаље у заглавља и садржај. Опцијама команди поделе `list` и `marking` можемо да наведемо алтернативни наслов који ће се послати тамо. Такође је могуће да се у току писања наслова употребе команде `\nolist` или `\nomarking` тако да се одређени делови наслова замене елипсисима у садржају или заглављу. На пример:

```
\chapter{Утицаји импресионизма \nomarking{19. века у 21. веку}}
```

ће у заглављу да напише „Утицаји импресионизма ...”.

- **Нумерација.** Ово је случај само за нумерисане одељке (`part`, `chapter`, `section`, `subsection`...), али не важи за ненумерисане (`title`, `subject`, `subsubject`). Уствари, то да ли ће се одређени одељак нумерисати или не зависи од опција „`number`” и „`incrementnumber`” чије су могуће вредности „`yes`” и „`no`”. У нумерисаним одељцима се обе ове опције постављају на `yes` а у ненумерисаним на `no`.

Зашто постоје две опције које контролишу исту ствар? Јер у суштини, оне контролишу различите ствари; једна контролише да ли се одељак уопште нумерише (`incrementnumber`), а друга да ли се број приказује (`number`). Ако се за одељак постави `incrementnumber=yes` и `number=no`, добићемо одељак који је споља (визуелно) ненумерисан, али се интерно још увек броји. То може бити корисно да се такав одељак постави у садржај, јер се он обично састоји само од нумерисаних одељака. У вези овога, погледајте [пододељак А у одељку 8.1.7](#).

- **Лабела** за наслов. У принципу, овај елемент је у насловима празан. Али можемо да му доделимо вредност, па ће се у том случају испред броја и самог наслова исписивати лабела коју смо доделили за тај ниво. На пример, у насловима поглавља желимо да видимо исписану реч „Поглавље”, или реч „Део” за делове. За ово не користимо команду `\setuphead` већ `\setuplabeltext`. Ова команда нам омогућава да доделимо текстуалну вредност лабелама различитих нивоа поделе. Тако на пример, ако желимо да се у нашем документу испред наслова поглавља исписује „Поглавље”, треба да поставимо:

```
\setuplabeltext
[chapter=Поглавље~]
```

У примеру сам иза додељеног имена поставио резервисани карактер „`~`” који иза речи умеће размак на којем не може да се направи прелом реда. Ако нам не смета да се ред преломи између лабеле и броја, једноставно можете да напишете размак. Али тај размак (било које врсте) је важан; без њега ће број бити спојен са лабелом и видели бисмо, на пример „Поглавље1” уместо „Поглавље 1”.

7.4.3 Контрола нумерације (у нумерисаним одељцима)

Већ знамо да предефинисани нумерисани одељци (`part`, `chapter`, `section`...) и то да ли се одређени одељак нумерише или не, зависи од опција „`number`” и „`incrementnumber`” постављених командом `\setuphead`.

Нумерација разних нивоа је подразумевано аутоматска, осим ако опцији „`ownnumber`” не доделимо вредност „`yes`”. Када је „`ownnumber=yes`” мора да се назначи број који се додељује свакој команди. Ово се ради:

- Ако је команда позвана употребом класичне синтаксе, додавањем аргумента са бројем пре текста наслова. На пример:
`\chapter{13}{Наслов поглавља}` ће да генерише поглавље којем је ручно додељен број 13.
- Ако је команда позвана синтаксом која је специфична за ConTeXt (`\ТипОдељка [Опције]` или `\startТипОдељка [Опције]`), опцијом „ownnumber”. На пример:
`\chapter[title={Наслов поглавља}, ownnumber=13]`, ће да генерише поглавље којем је ручно додељен број 13.

Када ConTeXt аутоматски врши нумерацију, он употребљава интерне бројаче који чувају бројеве различитих нивоа; тако постоји бројач за делове, други за поглавља, још један за одељке, итд. Сваки пут када ConTeXt пронађе команду поделе, он извршава следеће акције:

- Увећава за '1' бројач придружен нивоу који одговара команди.
- Ресетује на 0 бројаче придружене свим нивоима испод оног којем одговара команда.

Ово значи да ће сваки пут када се пронађе ново поглавље, бројач поглавља да се увећа за 1 и сви одељци, пододељци, подпододељци итд. се враћају на 0; али се не дирају бројачи придружени деловима.

Ако желите да утичете на број од којег се почиње нумерација, употребите команду `\setupheadnumber` као што следи:

`\setupheadnumber[ТипОдељка][Број од којег почиње бројање]`

где је *Број од којег почиње бројање* број од којег почиње бројање било ког типа одељака. Дакле ако је *Број од којег почиње бројање* нула, први одељак ће бити 1; ако је 10, први одељак ће бити 11.

Ова команда омогућава и да се утиче на шаблон аутоматског увећавања; тако на пример, можемо подесити да се поглавља или одељци броје у паровима, или по три. Дакле, `\setupheadnumber[section][+5]` ће поставити да се одељци броје као 5 од пет; а `\setupheadnumber[chapter][14, +5]` ће се постарати да прво поглавље почне од 15 (14 + 1), друго ће бити 20 (15 + 5), треће 25, итд.

Нумерација поделе се подразумевано исписује арапским бројевима, и укључује се нумерација свих претходних нивоа. То значи: у документу који се састоји од делова, поглавља, одељака и пододељака, одређени пододељак ће навести којем делу, поглављу и одељку припада. Тако је четврти пододељак другог одељка у трећем поглављу првог реда бити „1.3.2.4”.

Начин приказа бројева контролишу две основне опције:

- **conversion:** она контролише врсту набрајања која ће се користити. Нуди већи број вредности, зависно од типа нумерације који желимо:
 - **Нумерација арапским бројевима:** класична нумерација: 1, 2, 3, ... се добија вредностима n , N или `numbers`.
 - **Нумерација римским бројевима.** Постоји три начина за ово:

- ★ Римски бројеви од великих слова: I, R, Romannumerals.
 - ★ Римски бројеви од малих слова: i, r, romannumerals.
 - ★ Римски бројеви у капиталу: KR, RK.
- **Нумерација словима.** Постоји три начина за ово:
- ★ Велика слова: A, Character
 - ★ Мала слова: a, character
 - ★ Капитал: AK, KA
- **Нумерација речима.** Значи да пишемо реч која означава број. Тако на пример, '3' постаје 'Three'. Постоји два начина да се ово уради:
- ★ Речи почињу великим словом: Words.
 - ★ Речи исписане само малим словима: words.
- **Нумерација симболима:** Нумерација базирана на симболима користи различите сетове симбола у којима је сваком симболу додељена нумеричка вредност. Пошто сетови симбола које користи ConTeXt имају врло ограничен број симбола, онда је погодно да се користи ова врста нумерације када очекивани максимални број није сувише велики. ConTeXt редом обезбеђује четири различита сета симбола: set 0, set 1, set 2 and set 3. Испод је наведен сваки од ових сетова који се користе за нумерацију. Запазите да је за сетове set 0 и set 1 максимални број који може да се достигне 9, а за сетове set 2 и set 3 12:

```

Set 0: • - * ▷ ° ○ □ ✓
Set 1: * ** *** ‡ ‡‡ ‡‡‡ * ** ***
Set 2: * † ‡ ** †† ‡‡ *** ††† ‡‡‡ **** †††† ‡‡‡‡
Set 3: * ** *** ‡ ‡‡ ‡‡‡ ¶ ¶¶ ¶¶¶ § §§ §§§

```

- **sectionsegments:** ова опција нам омогућава да контролишемо да ли се приказује нумерација претходних нивоа. Можемо да назначимо претходне нивое који ће се приказивати. То се ради тако што се идентификује почетни и крајњи ниво који се приказују. Идентификација нивоа може да се изврши његовим бројем (part=1, chapter=2, section=3, etc.), или именом (part, chapter, section, etc.). Тако на пример, „sectionsegments=2:3” наводи да би требало приказивати нумерацију поглавља и одељака. Потпуно је исто као да кажемо „sectionsegments=chapter:section”. Ако желимо назначити да се приказују сви бројеви изнад одређеног нивоа, можемо као вредност опције „optionsegments” да употребимо *ПочетниНиво:all*, или *ПочетниНиво:**. На пример, „sectionsegments=3:*” наводи да се приказује нумерација почевши од нивоа 3 (section).

Замислимо, на пример, да хоћемо да се делови нумеришу римским бројевима исписаним великим словима; поглавља арапским бројевима, али без приказа броја дела којем припадају; одељци и пододељци арапским бројевима уз приказ бројева поглавља и одељка, а подододељци великим словима. Трбало би да напишемо следеће:

```

\setuphead[part][conversion=I]
\setuphead[chapter] [conversion=n, sectionsegments=2]
\setuphead[section] [conversion=n, sectionsegments=2:3]
\setuphead[subsection] [conversion=n, sectionsegments=2:4]
\setuphead[subsubsection][conversion=A, sectionsegments=5]

```

7.4.4 Боја и стил наслова

Следеће опције контролишу стил и боју:

- **Стил** се контролише опцијама „style”, „numberstyle” и „textstyle” у зависности од тога да ли желимо да утичемо за комплетан наслов, само на нумерацију или само на текст. Овим опцијама можемо да укључимо команде које утичу на фонт; одређени фонт, стил (roman, sans serif или typewriter), алтернативу (italic, bold, slanted...) и величину. Ако само желимо да наведемо једну особину стила, то можемо да урадимо употребом имена стила (на пример, „bold” за црни слог), или наводећи одговарајућу скраћеницу („bf”), или команду која је генерише (`\bf`, у случају црног слога). Ако желимо да наведемо неколико особина истовремено, то морамо да урадимо помоћу команди које их генеришу, тако што их пишемо једну за другом. С друге стране, имајте на уму да ако наведемо само једну особину, остале особине стила ће се успоставити аутоматски на подразумеване вредности за документ, а то је разлог што се ретко саветује успостављање само једне особине стила.
- **Боја** се поставља опцијама „color”, „numbercolor” и „textcolor” у зависности од тога да ли боју желимо да поставимо за комплетан наслов, или само за набрајање или само за текст. Боја која се овде наводи може да буде једна од предефинисаних ConTeXt боја, или нека друга боја коју смо раније сами дефинисали и доделили јој име. Међутим, овде не можемо директно да употребимо команду за дефиницију боје.

Уз ових шест опција, постоји још пет које служе за постављање софистициранијих особина којима можемо урадити скоро све што замислимо. То су: „command”, „numbercommand”, „textcommand”, „deepernumbercommand” и „deepercommand”. Хајде да најпре објаснимо прве три:

- `command` наводи команду која ће прихватити два аргумента, број и наслов одељка. То може бити обична ConTeXt команда, или команда коју смо сами дефинисали.
- `numbercommand` личи на „command”, али ова команда прихвата само аргумент са бројем одељка.
- `textcommand` такође личи на „command”, али она прихвата само аргумент са текстом наслова.

Ове три опције нам омогућавају да урадимо практично било шта што пожелимо. На пример, ако желим да одељци буду десно поравнати, уоквирени и са преломом линије између броја и текста, једноставно могу да креирам команду која то ради, па да је онда наведем као вредност опције „command”. То би се постигло следећим линијама:

```

\define[2]\AlignSection
  {\framed[frame=on, width=broad,align=flushright]{#1\#2}}

\setuphead
  [section]
  [command=\AlignSection]

```

Када истовремено поставимо опције „command” и „style”, команда се примењује на наслов са задатим стилем. Ово значи, на пример, да ако смо поставили „textstyle=\em”, и „textcommand=\WORD”, команда `\WORD` (која текст који узима као аргумент претвара у велика слова) ће се применити на наслов са постављеним стилем, тј.: `\WORD{\em Текст наслова}`. Ако желимо да буде обрнуто, тј. да се стил примени на садржај наслова након примене команде, требало да уместо опција „textcommand” и „numbercommand” употребимо опције „deepercommand” и „deepernumbercommand”. То би у горњем примеру генерисало „`{\em\WORD{Текст наслова}}`”.

У већини случајева не би било разлике ако се уради на један или други начин. Али у неким случајевима може бити разлике.

7.4.5 Место броја и текста наслова

Опција „alternative” истовремено контролише две ствари: место набрајања у односу на текст наслова и место самог наслова (укључујући и број и текст) у односу на страницу на којој се приказује и садржај одељка. То су две различите ствари, али њима управља иста опција, контролишу се истовремено.

Место наслова у односу на страницу и први пасус садржаја одељка се контролише следећим могућим вредностима опције „alternative”:

- **text**: наслов одељка се интегрише у први пасус садржаја одељка. Резултат је сличан ономе што производи L^AT_EX са `\paragraph` и `\subparagraph`.
- **paragraph**: наслов одељка ће бити независан пасус.
- **normal**: наслов одељка ће се поставити на подразумевано место које ConT_EXt обезбеђује за одређени тип одељка у питању. Обично је то „paragraph”.
- **middle**: наслов се пише као самостални, центрирани пасус. Ако је команда набрајања, број и текст се постављају у различите линије, обе центриране.

Сличан резултат ономе што се добија са „alternative=middle”, добија се опцијом „align” која контролише поравнање наслова. Може да има вредности „left”, „middle” или „flushright”. Али ако овом опцијом центрирамо наслов, број и текст ће се појавити на истој линији.

- **margin**: ова опција исписује комплетан наслов (набрајање и текст) у простор резервисан за маргину.

Место броја у односу на текст наслова се назначава следећим вредностима опције „alternative”:

- **margin/inmargin:** наслов је посебан пасус. Набрајање се исписује у простору резервисаном за маргину. Нисам открио разлику између „margin” и „inmargin”.
- **reverse:** наслов чини посебна пасус, али се уобичајени редослед обрће, па се прво исписује текст, а онда број.
- **top/bottom:** у насловима чији текст заузима више од једне линије, ове две опције контролишу да ли ће набрајање да се поравнава са првом или са последњом линијом наслова.

7.4.6 Команде или акције које се извршавају пре или након исписивања наслова

Постоји могућност да се наведе једна или више команди које се извршавају пре исписивања наслова (опције „before”) или након (опције „after”). Ове опције се често употребљавају за визуелно истицање наслова. На пример: ако између наслова и текста који му претходи желимо да уметнемо вертикални размак, „before=\blank” ће да дода празну линију. Ако желимо још већи размак, могли бисмо да напишемо „before={\blank[3*big]}”. У овом случају смо вредност опције поставили унутар витичастих заграда да спречимо грешку. Такође смо визуелно могли да нагласимо размак између претходног текста и наредног са „before=\hairline, after=\hairline”, што би исцртало хоризонталну линију пре и после наслова.



Врло сличне опцијама „before” и „after” су опције „commandbefore” и „commandafter”. Према резултатима мојих тестова, изводим закључак да је разлика у томе што ове претходне две извршавају акције пре и након почетка слагања наслова као таквог, док последње две показују на команде које ће се извршити пре и након слагања *текста наслова*.

Ако испред наслова желимо да уметнемо прелом странице, потребно је да употребимо опцију „page” која између осталих, нуди и вредност „yes” за уметање прелома странице, „left” да се уметне потребан број прелома странце тако да се обезбеди да наслов почне на парној страници, „right” да се обезбеди да наслов почне на непарној страници, или „no” ако желимо да онемогућимо форсирани прелом странице. С друге стране, ова опција ће за нивое испод „chapter” функционисати само ако се употреби „continue=no”, у супротном неће радити ако је одељак, пододељак или команда на првој страници поглавља.

У систему ConTeXt, поглавља подразумевано почињу на новој страници. Ако се подеси да и одељци такође почињу на новој страници, јавља се проблем ако се први одељак поглавља јави на самом почетку поглавља: ако и тај одељак уметне прелом странице, имаћемо страницу која отвара поглавље и садржи само наслов. То није баш естетски задовољавајуће. То је разлог што можемо поставити опцију „continue”, чије ми име, морам да признам, баш и није јасно: ако је „continue=yes”, прелом странице се неће примењивати за одељке који су на првој страници поглавља. Ако је „continue=no” прелом странице ће се употребљавати.

Ако уместо команди поделе употребљавамо окружења одељака (`\start ... \stop`), имамо и опцију „insidesection” којом можемо назначити једну или више команди које ће се извршити онда када се испише наслов и већ се налазимо унутар одељка. Ова опција нам омогућава да, на пример, обезбедимо да се непосредно након почетка поглавља аутоматски сложи садржај („insidesection=\placecontent”)

7.4.7 Остале особине које могу да се подесе

Уз оне које смо видели до сада, са `\setuphead` можемо да подесимо и следеће додатне особине:

- **Проред.** Контролише се са „interlinespace” која као своју вредност узима име претходно креиране команде прореда са `\defineinterlinespace` и подешене са `\setupinterlinespace`.
- **Поравнање.** Опција „align” утиче на поравнање пасуса у којем се налази наслов. Неке од могућих вредности које може да има су: „flushleft” (лево), „flushright” (десно), „middle” (центрирано), „inner” (унутрашња маргина) и „outer” (спољашња маргина).
- **Маргина.** Опцијом „margin” можемо ручно да поставимо маргину наслова.
- **Увлачење првог пасуса.** Вредност опције „indentnext” (која може бити „yes”, „no” или „auto”) контролише да ли се прва линија првог пасуса у одељку увлачи. Обично је питање увлачења те линије (у документу у којем се све прве линије пасуса увлаче) ствар укуса.
- **Ширина.** Наслови подразумевано заузимају потребну ширину, осим ако је она већа од ширине линије, па ће у том случају заузимати више од једне линије. Опцијом „width” можемо наслову доделити неку одређену ширину. Опције „numberwidth” и „textwidth” редом додељују ширину набрајања или ширину текста наслова.
- **Раздвајање броја и текста.** Опције „distance” и „textdistance” нам омогућавају да контролишемо размак који постоји између броја и текста наслова.
- **Стил заглавља и подножја одељка.** За ово користимо опције „header” и „footer”.

7.4.8 Остале `\setuphead` опције

Опције са којима смо се до сада упознали нам нуде скоро неограничене могућности за подешавање наслова одељака. Међутим, `\setuphead` нуди још око тридесет опција које нисам поменуо. Већину углавном зато што нисам открио чему служе или како се користе, а неколико зато што би њихово објашњење захтевало да зађем у аспекте о којима не желим да се бавим у овом уводу.



7.5 Дефинисање нових команди поделе

Своје сопствене команде за поделу можемо дефинисати командом `\definehead` чија је синтакса:

```
\definehead[ИмеКоманде][Модел][Конфигурација]
```

где

- **ИмеКоманде** представља име нове команде поделе.
- **Модел** је има постојеће команде поделе која ће се користити као модел на основу којег ће нова команда иницијално да наследи све њене карактеристике.

У суштини, нова команда наслеђује много више од почетних карактеристика модела: она постаје нека врста прилагођене инстанце модела, али са њим дели, на пример, интерни бројач који контролише нумерацију.

- **Конфигурација** је прилагођена конфигурација наше нове команде. Овде можемо да употребимо потпуно исте опције као у `\setuphead`.

Није неопходно да се нова команда конфигурише у време када се креира. То може и касније да се уради командом `\setuphead`, а уствари, у примерима датим у ConTeXt упутствима и његовом викију, изгледа да је то и уобичајени начин.

7.6 Макроструктура документа

Поглавља, одељци, пододељци, наслови..., структура документа; они га организују. Али уз структуру која је резултат ове врсте команди, у неким штампаним књигама, посебно у онима које се пишу у академским круговима, постоји врста *макро-редоследа* материјала књиге, који не узима у обзир само садржај књиге, већ и функцију коју сваки од ових великих делова има у књизи. Тако разликујемо:

- Почетни део документа који садржи насловну страницу, страницу са признањима, страницу са посветом, садржај, можда предговор, страницу презентације, итд.
- Главно тело документа, које садржи основни текст документа, подељен у делове, поглавља, одељке, пододељке, итд. Овај део је обично и најопширнији и најважнији.
- Додатни материјал који се састоји од додатака и анекса који развијају или дају пример неких ствари којима се бавило у главном телу, или обезбеђују додатну документацију коју није написао аутор главног тела, итд.
- Завршни део документа у коме можемо пронаћи библиографију, индексе, речнике, итд.

Сваки од ових делова у изворном фајлу можемо да омеђимо окружењима наведеним у [табели 7.2](#).

Део документа	Команда
Почетни део	<code>\startfrontmatter [Опције] ... \stopfrontmatter</code>
Главно тело	<code>\startbodymatter [Опције] ... \stopbodymatter</code>
Додаци	<code>\startappendices [Опције] ... \stopappendices</code>
Завршни део	<code>\startbackmatter [Опције] ... \stopbackmatter</code>

Табела 7.2 Окружења која одражавају макроструктуру документа

Четири окружења прихватају исте четири опције: „page”, „before”, „after” и „number”, а њихове вредности и резултат употребе су исти као код `\setuphead` (погледајте [одељак 7.4](#)), мада би требало запазити да ће опција „number=no” овде да елиминише нумерацију свих команди поделе унутар окружења.

Уметање било које од ових великих подела у документ има смисла само ако оно служи да успостави неку врсту разликовања између њих. Можда заглавља или нумерација страница у почетном делу. Конфигурација сваког од ових блокова се врши командом `\setupsectionblock` чија је синтакса:

```
\setupsectionblock[Име блока] [Опције]
```

где *Име блока* може да буде `frontpart`, `bodypart`, `appendix` или `backpart`, а опције су оне које смо управо навели: „page”, „number”, „before” и „after”. На пример, ако желимо подесити да се *frontmatter* (почетном делу) странице нумеришу римским бројевима, у преамбули нашег документа треба да напишемо:

```
\setupsectionblock
[frontpart]
[
  before={\setuppagenumbering[conversion=Romannumerals]}
]
```

Из подразумеване ConTeXt конфигурације ова четири блока следи да:

- Четири блока почињу нову страницу.
- Нумерација одељака се мења у сваком од ових блокова:
 - У `frontmatter` и `backmatter` се сви нумерисани одељци подразумевано не нумеришу.
 - У `bodymatter` поглавља користе арапске бројеве.
 - У `appendices` се поглавља нумеришу великим словима.

Такође је могуће и креирање нових блокова одељака командом `\definesectionblock`.

Глава 8

Садржаји, индекси, листе

Садржај: 8.1 **Садржај**; 8.1.1 Општи преглед садржаја; 8.1.2 Потпуно аутоматски садржај са насловом; 8.1.3 Аутоматски садржај без наслова; 8.1.4 Елементи који се постављају у садржај: опција `criterion`; 8.1.5 Распоред садржаја: опција `alternative`; 8.1.6 Формат ставки садржаја; 8.1.7 Ручна подешавања садржаја; А Укључивање нумерисаних одељака у садржај; Б Ручно уметање ставки у садржај; В Искључивање из садржаја одређеног одељка који припада врсти која чини део садржаја; Г Наслов одељка у садржају који се разликује од наслова у телу документа; 8.2 **Листе, комбиноване листе и садржаји базирани на листи**; 8.2.1 Листе у систему ConTeXt; 8.2.2 Листе или индекси слика, табела и осталих ставки; 8.2.3 Комбиноване листе; 8.3 **Индекс**; 8.3.1 Генерисање индекса; А Претходна дефиниција ставки индекса и обележавање места у изворном фајлу која се односе на њих; Б Генерисање коначног индекса; 8.3.2 Форматирање индекса тема; 8.3.3 Креирање осталих индекса;

Садржај и индекс представљају глобални аспект документа. Сви документи ће поседовати садржај, док ће само неки имати индекс. На многим језицима (али не и у српском) и садржај и индекс потпадају под општи појам 'индекс'. За енглеске читаоце, садржај обично долази на почетку (документа, или у неким случајевима, и на почецима поглавља), а индекс долази на крају.

И једно и друго претпостављају одређену употребу механизма интерних референци чије објашњење је дато у одељку 9.2.

8.1 Садржај

8.1.1 Општи преглед садржаја

У претходном поглављу смо испитивали команде које омогућавају да се током писања успоставља структура документа. Овај одељак се бави садржајем и индексом, који на неки начин *ограђавају* структуру документа. Садржај је веома користан за схватање документа као целине (јер помаже да се информација стави у контекст) и за проналажење тачног места на којем би могао да се налази одређени пасаж. Чини се да књиге веома сложене структуре, са много одељака и подељака разних нивоа дубине, захтевају различиту врсту садржаја, јер је онај са мало детаља (можда само са прва два или три нивоа поделе) помаже да се стекне општа слика садржаја документа, али није баш користан за проналажење одређеног пасаж; док се у веома детаљном садржају, у којем је лако да се од дрвета не види шума, губи општи поглед на документ. То је разлог што књиге са заиста сложеном структуром понекада имају више од једног садржаја: на почетку један не толико детаљан, који приказује главне делове, и садржај са више детаља на почетку сваког поглавља, а можда и индекс на крају књиге.

У систему ConTeXt релативно лако можемо аутоматски да их генеришемо. Можемо да:

- Генеришемо комплетан или делимични садржај на било ком месту у документу.
- Одлучимо шта ће сваки да садржи.

- Конфигуришемо њихов изглед, све до најситнијих детаља.
- У садржај уметнемо хиперлинкове који нам омогућавају да скочимо директно на жељени одељак.

У суштини је ова последња могућност подразумевано активирана у свим садржајима, ако је за документ укључена функција интерактивности. Што се тога тиче, погледајте [одељак 9.3](#).

Објашњење овога у ConTeXt референтном упутству је, по мом мишљењу, донекле нејасно услед чињенице да се одједном уводи превише информација. ConTeXt механизам за изградњу садржаја чине многи делови; па је тешко да текст који све њих одједном покушава да објасни буде јасан. Посебно читаоцу који се тек упознаје са системом. За разлику од тога, објашњење у [викију](#), је практично ограничено на примере: веома је корисно за учење *џирикова* али није адекватно – барем ја тако мислим – за разумевање механизма и начина на који он функционише. Из тог разлога сам у овом уводу одлучио да употребим стратегију објашњавања по којој се почиње претпостављајући нешто што није у потпуности тачно (или није пуна истина): да у систему ConTeXt постоји нешто што се назива *садржај*. Почевши овако, објашњавају се *обичне* команде за генерисање садржаја, па када се ове команде и њихова конфигурација добро упознају, сматрам да је тренутак за увођење – мада више на теоретском него на практичном нивоу – информације о оним деловима механизма који су до тада били изостављени. Познавање тих додатних *делова* нам омогућава да креирамо прилагођеније садржаје него што су то они које називамо *обични*, креирани командама које су објашњене до тог места; међутим, у већини случајева неће бити потребно да то радимо.

8.1.2 Потпуно аутоматски садржај са насловом

Основне команде за аутоматско генерисање садржаја од нумерисаних одељака документа (делова, поглавља, одељака, итд.) су `\completecontent` и `\placecontent`. Основна разлика између ове две команде је што прва у садржај умеће *наслов*; да би се то урадило, она непосредно испред садржаја умеће *нунумерисано ђоїлавље* чији је подразумевани наслов Садржај.

Дакле, `\completecontent`:

- Умеће, на место на којем се нађе, нунумерисано поглавље под називом „Садржај”.

Присетимо се да је у систему ConTeXt команда за генерисање нунумерисане поделе која је на истом нивоу као поглавља `\title` (погледајте [одељак 7.2](#)). Стога у стварности `\completecontent` не умеће *поглавље* (`\chapter`) већ *наслов* (`\title`). Нисам то овако рекао јер мислим да читаоца овде може да збуни употреба имена команди нунумерисаних подела, пошто појам *наслов* такође има и шири смисао, па је лако да га читалац не поистовети са конкретним нивоом поделе на који се овде односи.

- Ово *ђоїлавље* (уствари, `\title`) се форматира потпуно исто као и остатак нунумерисаних поглавља у документу; а која подразумевано укључују и прелом странице.
- Садржај се штампа непосредно након наслова.

У почетку је генерисани садржај *комїлїїшан*, што можемо закључити из имена команде која га генерише (`\completecontent`). Али с једне стране, ми можемо да ограничимо дубину садржаја, као што је објашњено у [одељку 8.1.3](#), а с друге, пошто је ова команда *осеїїїва* на место у изворном фајлу на којем се пронађе (погледајте шта сам рекао у вези са

`\placecontent`), ако се `\completecontent` не пронађе на почетку документа, могуће је да генерисани садржај неће бити комплетан; а на неким местима у изворном фајлу је чак могуће и да се команда привидно игнорише. Ако се то догоди, решење је да се команда позове са опцијом „`criterium=all`”. Што се тиче ове опције, погледајте такође одељак 8.1.3.

Да бисмо променили подразумевани наслов који се додељује садржају, користимо команду `\setupheadtext` чија је синтакса:

```
\setupheadtext[Језик][Елемент=Име]
```

где *Језик* није обавезно и односи се на идентификатор језика који користи ConTeXt (погледајте одељак 10.5), а *Елемент* се односи на елемент чије име желимо да променимо („`content`” у случају садржаја) и *Име* представља име или наслов који желимо да дамо нашем садржају. На пример

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

ће обезбедити да се садржај генерисан командом `\completecontent` зове „`Contents`” уместо „`Table of Contents`”.

Чак штавише, `\completecontent` омогућава све конфигурационе опције као и `\placecontent`, за чије објашњење треба да погледате (следећи одељак).

8.1.3 Аутоматски садржај без наслова

Општа команда за уметање садржаја без наслова, који се аутоматски генерише из команди поделе документа је `\placecontent` и њена синтакса је:

```
\placecontent[Опције]
```

У суштини, садржаји ће да обухвате апсолутно све нумерисане одељке, мада командом `\setupcombinedlist` можемо да ограничимо дубину нивоа који ће обухватити (о овој команди ћемо говорити касније). Тако ће, на пример:

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

да ограничи састав садржаја на поглавља и одељке.

Особеност ове команде је да је осетљива на место у изворном фајлу на којем се налази. Ово је веома једноставно да се објасни на неколико примера, али је много компликованије ако желимо да наведемо како тачно команда ради и који наслови се укључују у садржај у сваком случају. Па хајде да почнемо са примерима:

- `\placecontent` постављено на почетку документа, пре прве команде поделе (`part`, `chapter` или `section`, у зависности од ситуације) ће да генерише комплетан садржај.

Нисам потпуно сигуран да је садржај који се подразумевано генерише *комплетан*, мада верујем да се у њега укључује довољно нивоа поделе тако да је у већини случајева комплетан; али сумњам да неће ићи иза осмог нивоа поделе. У сваком случају, као што је поменуто изнад, ниво поделе који улази у садржај можемо да подесимо помоћу

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

- За разлику од тога, иста ова команда постављена унутар дела, поглавља или одељка ће генерисати садржај у којем се налази само оно што је део тог елемента, или другим речима, поглавља, одељке и остале ниже нивое поделе тог одређеног дела, или одељке (и остале нивое) одређеног поглавља, или пододељке одређеног одељка.

Што се тиче техничког и детаљног објашњења, да би се исправно разумело подразумевано функционисање команде `\placecontent`, од суштинске важности је присетити се да су за ConTeXt Mark IV различите поделе у суштини *окружења* која почињу са `\startТипПоделе`, а које се завршавају са `\stopТипПоделе` и да у њима могу да се налазе остале команде поделе нижег нивоа. Па када се то узме у обзир, можемо рећи да `\placecontent` подразумевано генерише садржај који ће само да обухвати:

- Елементе који припадају *окружењу* (нивоу поделе) у који се постави команда. Ово значи да када се команда постави у поглавље, садржај неће обухватити одељке или пододељке осталих поглавља.
- Елементе који имају ниво поделе нижи од нивоа који одговара тачки на којој се налази команда. Ово значи да ако се команда нађе у поглављу, део садржаја постају само одељци, пододељци и остали нижи нивои; али ако је команда у одељку, он ће се поделити тако да се направи садржај нивоа пододељка.

Уз то, да би се генерисао садржај, потребно је да се `\placecontent` пронађе *пре* првог одељка поглавља у којем се налази, или пре првог пододељка одељка у којем се налази, итд.

Нисам сигуран да сам пружио јасно објашњење изнад. Можда уз донекле детаљнији пример од претходног можемо боље разумети на шта мислим: хајде да замислимо следећу структуру документа:

- Поглавље 1
 - Одељак 1.1
 - Одељак 1.2
 - ★ Пододељак 1.2.1
 - ★ Пододељак 1.2.2
 - ★ Пододељак 1.2.3
 - Одељак 1.3
 - Одељак 1.4
- Поглавље 2

Дакле: када се `\placecontent` постави испред Поглавља 1, генерисаће комплетан садржај, сличан ономе који се генерише са `\completecontent`, само без наслова. Али ако се команда постави унутар Поглавља 1, а испред одељка 1.1, садржај ће приказивати само из чега се састоји поглавље; а ако се постави на почетак одељка 1.2, садржај ће приказивати само оно што се налази у том одељку. Али ако се команда постави, на пример, између одељака 1.1 и 1.2, игнорисаће се. Такође ће се игнорисати и ако се постави на крај одељка, или на крај документа.

Све ово се, наравно, односи само на случај када се команди не поставе никакве опције. Тачније, опција `criterium` ће променити то подразумевано понашање.

Од свих опција које подржава команда `\placecontent` објаснићу само њих две, најважније за постављање садржаја и још важније, само оне које су (делимично) документоване у ConTeXt референтном упутству. Опција `criterium` која утиче на оно што се поставља у садржај у зависности од места у изворном фајлу у којем се налази команда; и опција `alternative`, која утиче на општи распоред генерисаног садржаја.

8.1.4 Елементи који се постављају у садржај: опција `criterium`

Подразумевано понашање команде `\placecontent` у вези са местом команде у изворном фајлу је објашњено изнад. Опција `criterium` мења ово понашање. Уз остале, она може да има и следеће вредности:

- `all`: садржај ће бити комплетан, без обзира на место у изворном фајлу на којем се нађе команда.
- `previous`: садржај ће обухватити само команде поделе (нивоа на ком се налазимо) *ис-прег* команде `\placecontent`. Ова опција је предвиђена за садржаје који се исписују на крају документа или одређеног одељка.
- `part, chapter, section, subsection...`: говоре да би садржај требало да се ограничи само на наведени ниво поделе.
- `component`: у пројектима из више фајлова (погледајте [одељак 4.6](#)), генерисаће се само садржај који одговара *компоненти* у којој је пронађена команда `\placecontent` или `\completecontent`.

8.1.5 Распоред садржаја: опција `alternative`

Опција `alternative` контролише општи распоред садржаја. Њене главне вредности можете видети у [табели 8.1](#).

alternative	Из чега се састоји садржај	Напомене
a	Број – Наслов – Страница	Једна ставка по линији
b	Број – Наслов – Размац – Страница	Једна ставка по линији
c	Број – Наслов – Водеће тачке – Страница	Једна ставка по линији
d	Број – Наслов – Страница	Континуални садржај
e	Наслов	Уоквирен
f	Наслов	Лево поравнат, десно поравнат или центриран
g	Наслов	Центриран

Табела 8.1 Начини форматирања садржаја

Прве четири вредности за `alternative` обезбеђују све информације о сваком одељку (његов број, његов наслов и страницу на којој почиње), па су зато погодне и за папирне и електронске документе. Последње три алтернативе нас информишу само о наслову, тако да су погодни само за електронске документе у којима није неопходно да се зна број странице на којој почиње одељак, ако садржај има хиперлинк на њу, а они се у систему ConTeXt подразумевано постављају.

Иначе, ако читалац жели да на најбољи начин схвати разлике између различитих алтернатива, сматрам да треба да генерише тест документ у којем може детаљно да их анализира.

8.1.6 Формат ставки садржаја

Већ смо видели да нам `alternative` опција команде `\placecontent` или `\completecontent` омогућава да контролишемо општи *распоред* садржаја, тј. које информације ће се приказати за сваки наслов, и хоће ли или неће бити прелома реда којим се раздвајају различити наслови. Завршна подешавања сваке ставке садржаја се врше командом `\setuplist` чија је синтакса:

`\setuplist[Елемент][Конфигурација]`

где се *Елемент* односи на одређену врсту одељка. То може да буде `part`, `chapter`, `section`, итд. Такође можемо истовремено да конфигуришемо више од једног елемента, тако што их раздвојимо запетама. *Конфигурација* има до 54 могућности, од којих многе, као и обично, нису експлицитно документоване; али то не спречава да оне које су документоване, или оне које нису довољно јасне омогуће потпуно подешавање садржаја.

Сада ћу да објасним најважније опције, тако што ћу их груписати према њиховој корисности, али пре него што се упустимо у њих, присетимо се да ставка садржаја, у зависности од вредности опције `alternative`, може имати до три различите компоненте: број одељка, наслов одељка и број странице. Конфигурационе опције нам омогућавају да конфигуришемо различите компоненте глобално, или појединачно:

- *Укључивање (или не) различитих компоненти*: ако смо изабрали алтернативу која уз наслов укључује и број одељка и страницу (алтернативе 'a' 'b' 'c' или 'd'), опције `headnumber=no` или `pagenumber=no` значе да се за одређени ниво који конфигуришемо, број одељка (`headnumber`) или број странице (`pagenumber`) не приказује.
- *Боја и стил*: већ знамо да ставка која генерише одређени одељак у садржају може имати (у зависности од алтернативе) до три различите компоненте: број одељка, наслов и број странице. Опцијама `style` и `color` можемо свима одједном да одредимо стил и боју, или да опцијама `numberstyle`, `textstyle` или `pagestyle` (за стил) и `numbercolor`, `textcolor` или `pagecolor` (за боју) то посебно урадимо појединачно за сваку.

Да би се контролисао изглед сваке ставке, уз сам стил можемо да на комплетну ставку или на неку од њених различитих делова применимо неке команде. За то служе опције `command`, `numbercommand`, `pagecommand` и `textcommand`. Команда која се овде наводи може да буде стандардна `ConTeXt` команда, или команда коју сами направимо. Број одељка, текст наслова и број странице се прослеђују као аргументи опцији `command`, док се опцији `textcommand` прослеђује наслов одељка, а опцији `pagecommand` број странице. Тако ће, на пример, следећа реченица обезбедити да се наслови одељака исписују (лажним) капиталом:

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

- *Раздвајање од осталих елемената садржаја*: опције `before` и `after` нам омогућавају да наведемо команде које ће се извршити пре (`before`) и након (`after`) слагања ставке садржаја. Обично се ове команде користе било за постављање размака или неког раздвајајућег елемента између претходне и наредне ставке.

- *Увлачење елемената*: поставља се опцијом `margin` која нам омогућава да поставимо величину увлачења које ће имати ставке нивоа који конфигуришемо.
- *Упућени хиперлинкови у садржај*: ставке садржаја подразумевано поседују хиперлинк на страницу документа на којој почиње одељак о којем је реч. Употребом опције `interaction` можемо искључити ову функцију (`interaction=no`), или можемо да је ограничимо на део ставке у којој ће се налазити хиперлинк, што може бити број одељка (`interaction=number` или `interaction=sectionnumber`), наслов одељка (`interaction=text` или `interaction=title`) или број странице (`interaction=page` или `interaction=page-number`).
- *Остали аспекти*:
 - `width`: наводи величину размака између броја и наслова одељка. Може да буде димензија, или кључна реч `fit` која поставља тачну ширину броја одељка.
 - `symbol`: омогућава да се број одељка замени симболом. Подржане су три могуће вредности: `one`, `two` и `three`. Вредност `none` за ову опцију уклања из садржаја број одељка.
 - `numberalign`: наводи поравнања елемената нумерације; може да буде `left`, `right`, `middle`, `flushright`, `flushleft`.

Међу многим опцијама за конфигуравање садржаја, не постоји ниједна која нам омогућава директну контролу размака између линија. Он ће подразумевано да буде исти као онај који се примењује на комплетан документ. Међутим, често је пожељно да су линије садржаја мало збијеније од остатка документа. Да бисмо то постигли, требало би да команду која генерише садржај (`\placecontent` или `\completecontent`) поставимо унутар групе у којој се подешава различити размак између линија. На пример:

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

8.1.7 Ручна подешавања садржаја

Већ смо објаснили две основне команде за генерисање садржаја (`\placecontent` и `\completecontent`), као и њихове опције. Са ове две команде се садржаји аутоматски генеришу, и конструишу од постојећих нумерисаних одељака у документу, или у блоку или сегменту документа на који се садржај односи. Сада ћу објаснити одређена подешавања која можемо поставити тако да садржај није толико аутоматизован. Ово подразумева:

- Могућност да се у садржај укључе и наслови неких ненумерисаних одељака.
- Могућност ручног слања одређене ставке у садржај која не одговара присуству нумерисаног одељка.
- Могућност да се одређени нумерисани одељак искључи из садржаја.
- Могућност да се наслов за одређену ставку у садржају не подударе у потпуности са насловом који се налази у телу документа.

А. Укључивање нумерисаних одељака у садржај

Механизам којим ConTeXt изграђује садржај подразумева да се сви нумерисани одељци аутоматски укључују, што, као што сам већ рекао (погледајте одељак 7.4.2) зависи од две опције (`number` и `incrementnumber`) које можемо да променимо командом `\setuphead` за сваку врсту одељка. Тамо је такође је објашњено да ће тип одељка за који је `incrementnumber=yes` и `number=no` бити интерно, а не видљиво нумерисани одељак.

Стога, ако желимо да одређени нумерисани тип одељка – на пример, `title` – буде део садржаја, морамо да променимо вредност опције `incrementnumber` за тај тип одељка, тако што је поставимо на `yes`, па да онда укључимо у садржај тај тип одељка, што се ради, као што је објашњено изнад, командом `\setupcombinedlist`:

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

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

Затим, ако то желимо, командом `\setuplist` можемо ту ставку да форматирамо на потпуно исти начин као и било коју другу; на пример:

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

Напомена: управо објашњеном процедуром се у садржај из нашег документа укључују све инстанце типа нумерисаног одељка о којем је реч (у овом примеру, одељци типа `title`). Ако желимо да се у садржај укључи само одређено појављивање тог типа одељка, пожељно је да се то уради процедуром која је објашњена испод.

Б. Ручно уметање ставки у садржај

Са било ког места у изворном фајлу у садржај можемо да пошаљемо или ставку (симулирањем постојања одељка који у стварности не постоји), или команду.

Да бисмо послали ставку која симулира постојање одељка који уствари не постоје, користимо команду `\writetolist` чија је синтакса:

```
\writetolist[ТипОдељка][Опције]{Број}{Текст}
```

где

- Први аргумент наводи ниво који ова ставка мора да има у садржају: `chapter`, `section`, `subsection`, etc.
- Други аргумент, који није обавезан, омогућава да се ова ставка конфигурише на одређени начин. Ако се ручно послати унос изостави, форматираће се као и све остале ставке нивоа који је наведен првим аргументом; мада морам истаћи да у мојим тестовима ово није успело да проради.



И у званичној листи ConTeXt команди (погледајте одељак 3.6) и у викију нам се каже да овај аргумент прихвата исте вредности као и `\setuplist`, што је команда која нам омогућава да форматирамо различите ставке садржаја. Али, инсистирам, у мојим тестовима ни на који начин нисам успео да променим изглед ставке која је ручно послата у садржај.



- Трећи аргумент би требало да пресликава нумерацију коју има елемент послат у садржај, али ни то нисам успео да постигнем у својим тестовима.
- Последњи аргумент садржи текст који треба да се пошаље у садржај.

Ово је корисно, на пример, ако желимо да пошаљемо одређени нумерисани одељак у садржај, али само њега. У одељку А се објашњава како да се у садржај пошаље цела категорија нумерисаних одељака; али ако у садржај желимо да пошаљемо само једно одређено појављивање неког типа одељка, много је zgodније да се употреби команда `\writetolist`. Па тако, на пример, ако желимо да одељак нашег документа који садржи библиографију не буде нумерисани одељак, али да ипак буде део садржаја, могли би да напишемо:

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

Приметите како за одељак користимо нумерисану верзију команде `section`, што је `subject`, али је у садржај шаљемо ручно, као да је у питању нумерисани одељак (`section`).

Још једна команда која је предвиђена за ручно утицање на садржај је `\writebetweenlist` и она се користи да се са одређеног места у документу у садржај не пошаље сама ставка, већ команда. На пример, ако желимо да између две ставке садржаја уметнемо линију, могли бисмо да напишемо следеће на било ком месту у документу које се налази између два одељка у питању:

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

В. Искључивање из садржаја одређеног одељка који припада врсти која чини део садржаја

Садржај се конструише од *типова одељака* који су, као што већ знамо, одређени опцијом `list` команде `\setupcombinedlist`, па ако одређени *тип одељка* мора да се појави у садржају, нема начина да се неки одређени одељак тог типа из било ког разлога изузме из садржаја.

Обично, ако не желимо да се тамо појави одељак, употребљавамо његов *нумерисани еквивалент* што значи, на пример, `title` уместо `chapter`, `subject` уместо `section`, итд. Ови одељци се не шаљу у садржај, нити се нумеришу.

Међутим, ако из неког разлога желимо да се одређени одељак нумерише, али да се не појави у садржају, чак и ако се остали одељци тог типа појављују у садржају, можемо употребити *трик* који се састоји у креирању новог типа одељка који је *клон* одељка у питању. На пример:

```
\definehead[MySubsection][subsection]
\section{Први одељак}
\subsection{Први пододељак}
```

```
\MySubsection{Други пододељак}
\subsection{Трећи пододељак}
```

Ово ће обезбедити да када се уметне тип одељка `MySubsection`, увећава се бројач пододељака, али се не мења садржај, јер он подразумевано не укључује одељке типа `MySubsection`.

Г. Наслов одељка у садржају који се разликује од наслова у телу документа

Ако не желимо да наслов одређеног одељка који је део садржаја буде исти као онај који се приказује у телу документа, можемо употребити једну од две процедуре које су нам на располагању:

- Креирање одељка без употребе традиционалне синтаксе (`ТипОдељка{Наслов}`) већ помоћу `\ТипОдељка [Опције]`, или са `\startТипОдељка [Опције]`, и додељивање текста који желимо да се упише у садржај опцији `list` (погледајте одељак 7.3).
- Када се одељак о коме је реч пише у телу документа, употреби се команда `\nolist`: она чини да се текст који је њен аргумент у садржају замењује елипсисом. На пример:

```
\chapter
[title={\nolist{Приближан и донекле понављајући}
увод у реалност очигледног}]
```

би сложило наслов поглавља у телу документа, „Приближан и донекле понављајући увод у реалност очигледног”, али би у садржај послало следећи текст „... увод у реалност очигледног”.

Пажња: ово што сам управо истакао у вези са командом `\nolist` се наводи и у `ConTeXt` референтном упутству и у [викију](#). Међутим, код мене производи грешку при компајлирању која ме обавештава да команда `\nolist` није дефинисана.

8.2 Листе, комбиноване листе и садржаји базирани на листи

Интерно за `ConTeXt`, садржај није ништа више од *комбиноване листе*, која се, као што јој име наговештава, састоји од комбинације простих листи. Стога је основна представа на основу које `ConTeXt` изграђује садржај, листа. Неколико листи се комбинује тако да формирају садржај. `ConTeXt` подразумевано садржи предефинисану комбиновану листу која се назива „content” и то је оно са чиме раде команде које смо до сада истражили: `\placecontent` и `\completecontent`.

8.2.1 Листе у систему `ConTeXt`

Листа је за `ConTeXt` опсег нумерисаних елемената о којима треба да се памте три ствари:

1. Број.
2. Име или наслов.
3. Страница на којој се налази.

Ово се дешава са нумерисаним одељцима; али исто тако и са осталим елементима документа као што су слике, табеле, итд. У општем случају, оним елементима за које постоји команда чије име почиње са `\place` и која их поставља у документ, као што су `\placetable`, `\placefigure`, итд.

У свим овим случајевима, ConTeXt аутоматски генерише листу различитог броја појављивања у документу елемента чији је тип у питању, његов број, наслов и страницу. Тако, на пример, постоји листа поглавља, под називом `chapter`, друга за одељке, под називом `section`; али и једна за табеле (која се назива `table`) или слике (под називом `figure`). Листе које ConTeXt аутоматски генерише се увек називају исто као и ставка коју чувају.

Листа ће са такође аутоматски креирати ако, на пример, направимо нови тип нумерисаног одељка: када је креирамо, имплицитно се креира и листа која их чува. А ако за нумерисани одељак поставимо опцију `incrementnumber=yes`, чиме га претварамо у нумерисани одељак, такође ћемо имплицитно да креирамо и листу под тим именом.

Уз имплицитне листе (које ConTeXt аутоматски дефинише) командом `\definelist` можемо да креирамо и своје листе. Њена синтакса је

```
\definelist[ИмеЛисте][Конфигурација]
```

Ставке листе се додају:

- У листе предефинисане у систему ConTeXt, или у оне које се аутоматски креирају као резултат прављења новог плутајућег објекта (погледајте одељак 13.5), сваки пут када се ставка из листе уметне у документ, било командом поделе, било командом `\placeшта-год` за остале типове листи. На пример: `\placefigure` ће да уметне било какву слику у документ, али ће такође да уметне и одговарајућу ставку на листу.
- У било коју врсту листе ручно командом `\writetolist[ИмеЛисте]`, која је већ објашњена [подељку Б одељка 8.1.7](#). Такође је доступна и команда `\writebetweenlist`. И она је објашњена у том одељку.

Када се листа креира и у њу поставе све њене ставке, постоје три основне команде везане за листу `\setuplist`, `\placelist` и `\completelist`. Прва нам омогућава да конфигуришемо како изгледа листа; последње две умећу листу о којој се ради на место у документу на којем се нађу команде. Разлика између `\placelist` и `\completelist` је слична разлици између `\placecontent` и `\completecontent` (погледајте одељке 8.1.2 и 8.1.3).

На пример,

```
\placelist[section]
```

ће да уметне листу одељака, постављајући и хиперлинкове на њих у случају да је укључена интерактивност документа, тј. ако у `\setuplist` није постављено `set interaction=no`.

Листа одељака није у потпуности иста као садржај базиран на одељцима: појам садржаја обично укључује и ниже нивое (пододељке, подпододељке, итд.). Али листа одељака ће да укључи само одељке и ништа више.

Синтакса ових команди је:

```
\placelist[ИмеЛисте][Опције]
```

```
\setuplist[ИмеЛисте][Конфигурација]
```

`\setuplist` опције су већ објашњене у одељку 8.1.6, а опције за `\placelist` су исте као опције за `\placecontent` (погледајте одељак 8.1.3).

8.2.2 Листе или индекси слика, табела и осталих ставки

Из онога што је до сада речено, може се видети да пошто ConTeXt аутоматски креира листу слика које су постављене у документ командом `\placefigure`, генерисање листе или индекса слика на одређеном месту у документу је просто колико и употреба команде `\placelist[figure]`. А ако желимо да генеришемо листу са насловом (слично ономе што добијамо са `\completecontent`) потребно је да употребимо команду `\completelist[figure]`. Слично можемо да урадимо и са остале четири предефинисане врсте плутајућих објеката у систему ConTeXt: табелама („table”), графицима („graphic”), *интермецима* („intermezzo”) и хемијским формулама („chemical”), мада за одређене случајеве ових, ConTeXt већ нуди команду која их генерише без наслова: (`\placelistoffigures`, `\placelistoftables`, `\placelistofgraphics`, `\placelistofintermezzi` и `\placelistofchemicals`), а још једну која их генерише са насловом: (`\completelistoffigures`, `\completelistoftables`, `\completelistofgraphics`, `\completelistofintermezzi` и `\completelistofchemicals`), на сличан начин као са `\completecontent`.

На исти начин, за плутајуће објекте које смо сами креирали (погледајте одељак 13.5) аутоматски ће се креирати `\placelistof<ИмеПлутајућег>` и `\completelistof<ИмеПлутајућегFloatName>`.

За листе које смо сами креирали командом `\definelist`, индекс можемо да направимо помоћу `\placelist[ИмеЛисте]` или помоћу `\completelist[ИмеЛисте]`.

8.2.3 Комбиноване листе

Комбинована листа је, као што јој име наговештава, листа која комбинује ставке из различитих претходно дефинисаних листи. ConTeXt подразумевано дефинише комбиновану листу за садржај чије име је „content”, али сами можемо да креирамо остале комбиноване листе командом `\definecombinedlist` чија је синтакса:

```
\definecombinedlist[Име][Листе][Опције]
```

где је

- *Име*: име које се додељује новој комбинованој листи.
- *Листе*: односи се на имена листи које се комбинују, раздвојена запетама.
- *Опције*: конфигурационе опције листе. Могу да се наведу у време дефинисања листе, или, што је вероватно боље, када се позива листа. Главне опције (које су већ раније објашњене) су `criterium` (поделењак 8.1.4 одељка 8.1.3) и `alternative` (у поделењу 8.1.5 истог одељка).

Упутни ефекат креирања комбиноване листе командом `\definecombinedlist` је да се такође креира и команда која се зове `\placeИмеЛисте` и која служи да се листа позове, то јест: да се уметне у излазни фајл. Тако ће, на пример,

```
definecombinedlist[TOC]
```

да креира команду `\placeTOC`; а

```
definecombinedlist[content]
```

да креира команду `\placecontent`

Али само мало, `\placecontent`! Зар то није команда која се користи за креирање *обично* садржаја? Заиста: ово значи да ConTeXt стандардни садржај уствари креира помоћу следеће команде:

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

Када се дефинишемо нашу комбиновану листу, можемо да је конфигуришемо (или реконфигуришемо) командом `\setupcombinedlist` која прихвата већ објашњене опције `criterium` (погледајте поделењак 8.1.4 у одељку 8.1.3) и `alternative` (погледајте поделењак 8.1.5 у истом одељку), као и опцију `list` за *измену* листи које чине комбиновану листу.

Званична листа ConTeXt команди (погледајте одељак 3.6) међу опцијама које прихвата команда „`\setupcombinedlist`” не помиње опцију `list`, али она се користи у неколико примера употребе ове команде у викију (који је, штавише, не помиње ни на страници посвећеној овој команди). Лично сам проверио да ова опција функционише.

8.3 Индекс

8.3.1 Генерисање индекса

Индекс појмова се састоји од битних појмова и обично се налази на крају документа. Он наводи странице на којима се може пронаћи та тема.

Када се словослагање књига радило ручно, генерисање индекса појмова је био сложен задатак, и веома заморан. Било каква промена у пагинацији би могла да утиче на све ставке индекса. Из тог разлога, индекси и нису били уобичајени. Данас, компјутерски механизми

словослагања значе да мада је задатак још увек заморан, више није толико сложен ако се има у виду да компјутерском систему није проблем да листу података који се тичу ставке индекса одржава ажурном.

За генерисање индекса појмова нам је потребно да:

1. Одредимо које речи, појмови или концепти треба да буду део индекса. Ово је задатак који може да изврши само аутор.
2. Проверимо на којим местима у документу се појављује свака ставка будућег индекса. Мада, да будемо прецизни, уместо да *проверимо* места у изворном фајлу на којима се дискутује о концепту или проблему, када радимо у систему ConTeXt, потребно је да та места *означимо* уметањем команде која ће затим служити за аутоматско генерисање индекса. Ово је заморни део.
3. Коначно, генеришемо формат и индекс постављајући га на жељено место у документу. Ово последње је прилично једноставно у систему ConTeXt и захтева само једну команду: `\placeindex`.

А. Претходна дефиниција ставки индекса и обележавање места у изворном фајлу која се односе на њих

Најважнији посао је други корак. Тачно је да га компјутерски систем олакшава, у смислу да можемо извршити глобалну претрагу текста и тако пронађемо места у изворном фајлу на којима се говори о одређеној теми. Али такође не би требало ни да се слепо поуздамо у такве претраге текста: добар индекс појмова мора детектовати свако место на коме се дискутује о одређеној теми, чак и ако притом не користи *сшандардни* појам када се о њој говори.

Да бисмо *обележили* конкретно место у изворном фајлу и тако га придружили уз реч, појам, или идеју која ће се појавити у индексу, користимо команду `\index` чија је синтакса:

```
\index[Абецедно]{Ставка индекса}
```

Абецедно није обавезан аргумент и користи се за навођење алтернативног текста саме ставке индекса како би могла да се сложи по абецедном редоследу, а *Сшавка индекса* је текст који ће да се појави у индексу, придружен овој ознаци. Такође можемо да применимо особине форматирања која желимо да употребимо, а ако се у тексту јављају резервисани карактери, они морају да се напишу на уобичајени ConTeXt начин.

Могућност за абецедно сортирање ставке индекса на различити начин од онога како се заиста пише је врло корисна. Узмимо, на пример, овај документ и да сам хтео да генеришем ставку индекса за сва помињања команде `\TeX`. Низ `\index{\backslash TeX}` ће, на пример, команду поставити не према слову 't' из 'TeX', већ међу симболе, јер појам који се шаље у индекс почиње обрнутом косом цртом. Ово се решава тако што се напише `\index[tex]{\backslash TeX}`.

Сшавке индекса ће бити оне које желимо. Да би индекс тема био заиста користан, морамо да радимо мало напорније и да се питамо које концепте ће читалац највероватније да потражи; тако да на пример, може бити боље да се ставка дефинише као „болест, Хоџкинова” него да се дефинише као „Хоџкинова болест”, јер је општији појам „болест”.

Према конвенцији, ставке у индексу тема се увек пишу курентом, осим ако су лична имена.

Ако индекс има неколико нивоа дубине (дозвољено је до три), одређена ставка индекса се придружује неком нивоу употребом карактера '+'. То се ради овако:

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

У првом случају смо дефинисали други ниво под именом *Ставка 2* који ће бити подставка *Ставке 1*. У другом случају смо дефинисали трећи ниво под називом *Ставка 3* који ће бити подниво *Ставке 2*, а која је подставка *Ставке 1*. На пример

```
Му \index{пас}пас, је \index{пас+хрт}хрт који се зове Ракета.
Он не воли \index{мачка+луталица}мачке луталице.
```

Вреди запазити неке детаље у вези са овим што је претходно речено:

- Команда `\index` се обично поставља *испрег* речи којој се придружује и обично се не раздваја од ње размаком. Разлог за то је да се обезбеди да се команда нађе на истој страници на којој се нађе и реч за коју је везана:
 - Ако би постојао размак који их раздваја, постојала би могућност да ConTeXt изабере баш тај размак за прелом линије, што би онда могло да заврши и као прелом странице, па би у том случају команда била на једној страници а реч за коју је везана на наредној страници.
 - Ако би команда дошла *након* речи, било би могуће да се та реч разбије по слоговима и да се између два слога уметне прелом линије који би могао да буде и прелом странице. У том случају би команда показивала на наредну страницу на којој је прва реч она на коју показује.
- Погледајте како се појмови другог нивоа уводе у другом и трећем облику команде.
- Приметите како се у трећој употреби `\index` команде, мада је реч која се појављује у тексту „мачке”, термин који се шаље у индекс је „мачка”.
- Коначно: приметите како су три ставке за индекс тема написане у само две линије. Раније сам рекао да је обележавање прецизних места у изворном фајлу заморно. Сада ћу додати и да је обележавање превише места контрапродуктивно. Преопширан индекс уопште није бољи од концизног индекса у којем су све информације битне. Због тога сам раније рекао да би одлука о томе које речи треба да генеришу индекс требало да буде резултат свесне ауторове одлуке.

Ако желимо да наш индекс буде заиста користан, појмови који се користе као синоними морају да се у индексу групишу под један главни појам. Али пошто читалац може да претражује индекс на неку информацију по било ком од других главних појмова, уобичајено је да индекс садржи ставке које упућују на друге ставке. На пример, индекс тема неког правног приручника би врло лако могао да буде нешто слично овоме

уговорно неважење
погледајте *ништавност*.

Ово не постижемо командом `\index`, већ са `\seeindex` чији је формат:

```
\seeindex[Абецедно] {Ставка1} {Ставка2}
```

где је *Ставка1* ставка индекса које ће указивати на неку другу; а *Ставка2* је циљ указивања. У нашем претходном примеру, требало би да напишемо:

```
\seeindex{уговорно неважење}{ништавност}
```

И у `\seeindex` такође можемо да употребимо знак `' + '` да назначимо поднивое за било који од њена два аргумента у великим заградама.

Б. Генерисање коначног индекса

Када у изворном фајлу обележимо све ставке за наш индекс, он се заиста и генерише командама `\placeindex` или `\compleindex`. Ове две команде скенирају изворни фајл и траже `\index` команде, па генеришу листу свих ставки које би индекс требало да садржи, придружујући појам са бројем странице која одговара месту на којем је пронађена команда `\index`. Оне затим по абецедном редоследу уређују листу појмова који се појављују у индексу и спајају случајеве у којима се појам појављује више од један пут, па коначно, умећу коректно форматирани резултат у финални документ.

Разлика између `\placeindex` и `\compleindex` је слична разлици између `\content` и `\completecontent` (погледајте одељак 8.1.2): `\placeindex` је ограничена на генерисање и уметање индекса, док `\compleindex` најпре у финални документ умеће ново поглавље под подразумеваним именом „Индекс”, па у њему слаже индекс.

8.3.2 Форматирање индекса тема

Индекси тема су одређена примена општије структуре коју ConTeXt назива „*рејистар*”; тако да се индекс форматира командом:

```
\setupregister[index][Конфигурација]
```

Овом командом можемо:

- Одредити како ће индекс и његови различити елементи да изгледају. Наиме:
 - Наслови индекса који обично представљају слова абецеде. Они су подразумевано исписана курентом. Са `alternative=A` можемо подесити да буду у верзалу.
 - Саме ставке и њихов број странице. Изглед зависи од опција `textstyle`, `textcolor`, `textcommand` и `deertextcommand` за текућу ставку, а `pagestyle`, `pagecolor`

и `pagescommand`, за број странице. Са `pagenumber=no` такође можемо да генеришемо и индекс тема без бројева страница (мада не знам да ли ово може бити корисно).

- Опција `distance` мери ширину раздвајања између имена ставке и бројева страница; али такође мери и количину увлачења подставки.

Мислим да су имена опција `style`, `textstyle`, `pagestyle`, `color`, `textcolor`, и `pagecolor` довољно јасна да нам наговесте шта ради свака од њих. Што се тиче `command`, `pagescommand`, `textcommand` и `deeptextcommand`, указујем на објашњење слично названих опција у одељку 7.4.4, који се тиче конфигурације команди поделе.

- Поставити општи изглед индекса, што између осталог укључује команде које се извршавају пре (`before`) или након (`after`) индекса, број колона који он треба да има (`n`), да ли би колоне требало да буду једнаке ширине или не (`balance`), поравнање ставки (`align`), итд.

8.3.3 Креирање осталих индекса

Објаснио сам индекс тема као да је само та врста индекса могућа у документу; али истина је да документи могу да имају онолико индекса колико је потребно. На пример, могао би да постоји индекс личних имена, који скупља имена особа које се помињу у документу, са ознаком места на којем се цитирају. И то је врста индекса. Правни текстови би такође могли да имају специјални индекс за помињања грађанског законика; или у документу као што је овај који читате, индекс макроа који су у њему објашњени, итд.

За креирање додатних индекса у документу користимо команду `\defineregister` чија је синтакса:

```
\defineregister[ИмеИндекса] [Конфигурација]
```

где је *ИмеИндекса* име који ће се доделити новом индексу, а *Конфигурација* контролише начин на који он функционише. Такође је могуће да се индекс касније конфигурише помоћу

```
\setupregister[ИмеИндекса] [Конфигурација]
```

Када се креира нови индекс под именом *ИмеИндекса* на располагању ћемо имати команду `\ИмеИндекса` којом обележавамо ставке које ће тај индекс да садржи, на сличан начин као што се ставке обележавају са `\index`. Команда `\seeИмеИндекса` нам такође омогућава да креирамо ставке које указују на друге ставке.

На пример: могли бисмо да креирамо индекс ConTeXt команди у овом документу командом:

```
\defineregister[macro]
```

која би креирала команду `\macro`. Она ми омогућава да обележим све референце на ConTeXt команде као ставку индекса, па да затим командама `\placemacro` или `\completemacro` генеришем индекс.

Креирање новог индекса укључује команду `\ИмеИндекса` којом се обележавају његове ставке, и команде `\placeИмеИндекса` и `\completeИмеИндекса` којима се индекс генерише. Али ове две последње команде су уствари скраћенице две општије команде које се примењују на дати индекс. Тако је `\placeИмеИндекса` еквивалентна са `\placereregister[ИмеИндекса]`, а `\completeИмеИндекса` је еквивалентна са `\completeregister[ИмеИндекса]`.

Глава 9

Референце и хиперлинкови

Садржај: 9.1 Типови референци; 9.2 Интерне референце; 9.2.1 Лабела у циљу референце; 9.2.2 Команде на месту извора референце које преузимају податке са места циља; А Основне команде за преузимање информација од лабеле; Б Преузимање информације придружене лабели командом `\ref`; В Откривање места на које води линк; 9.2.3 Аутоматско генерисање префикса којим се спречавају лабеле дупликати; 9.3 Интерактивни електронски документи; 9.3.1 Укључивање интерактивности у документима; 9.3.2 Основна конфигурација интерактивности; 9.4 Хиперлинкови на спољашње документе; 9.4.1 Команде које помажу у словослагању хиперлинкова, али их не креирају; 9.4.2 Команде које успостављају линк; 9.5 Креирање маркера у финалном PDF фајлу;

9.1 Типови референци

Научни и технички документи су пуни референци:

- Оне понекада указују на остале документе који су основа за оно што је речено или што је у контрадикцији са оним што се објашњава, или што развија или нијансира идеју којом се бави, итд. У овим случајевима се каже да је референца *спољашња* и, ако документ треба да буде академски ригорозан, референца узима облик *цитирања* из литературе.
- Али је такође уобичајено да се у једном од одељака неког документа указује на неки други одељак у њему, и у том случају се каже да је референца *интерна*. Такође постоји и интерна референца када се на неком месту у документу коментарише неки аспект одређене слике, табеле, напомене или елемента сличне природе, тако што се на њега указује према његовом броју, или према броју странице на којој се налази.

Из разлога прецизности, интерне референце би требало да указују на тачно место, као и на место које се лако проналази. Из тог разлога су ове врсте референци увек референца или на нумерисани елемент (као када, на пример, кажемо „погледајте табелу 3.2”, или „поглавље 7”), или бројеви странице. Нејасне референце типа „као што смо већ рекли” или „као што ћемо касније видети” нису праве референце, и нема посебног захтева за начин њиховог словослагања, нити постоји било какав специјални алат за то. И ја лично одвраћам своје докторанте или магистранте од навике да користе ову праксу.

Интерне референце се такође уобичајено називају и „унакрсне референце” мада ћу у овом документу једноставно да користим израз „референце” у општем смислу, а „интерне референце” када желим да будем одређенији.

Да би разјаснио терминологију коју користим за референце, место у документу на којем се уводи референца ћу називати *извор*, а место на које она указује, *циљ*. Када се посматра на овај начин, могли бисмо рећи да је референца интерна када су извор и циљ у истом документу, а спољашња када су извор и циљ у различитим документима.

Из угла словослагања документа:

- Спољашње референце не представљају неки посебан проблем, тако да у принципу не захтевају било какав алат којим се уведе: сви потребни подаци из циљног документа су

ми доступни и могу да их користим у референци. Међутим, ако је документ извора електронски документ, а циљни документ је такође доступан на мрежи, онда у референцу може да се уметне хиперлинк који омогућава директан скок на циљ. У овим случајевима се каже да је документ извора *интерактиван*.

- За разлику од тога, интерне референце представљају изазов за словослагање документа, јер свако ко има искуства у припреми умерено дугачких научних или техничких документа зна да је скоро неизбежно да се током припреме документа нумерација страница, одељци, слике, теореме, или сличне ствари које се наводе у референци, мењати. Тако одржавање референци ажурним постаје врло тешко.

У време пре компјутера, аутори су избегавали интерне референце; а оне које су биле неизбежне, као што је садржај (који је, ако уз њега иде и број странице сваког одељка, пример интерне референце), писали су на крају.

Чак и најограниченији системи за словослагање, као што су текст процесори, омогућавају уметање неке врсте интерних унакрсних референци као што су садржаји. Али то није ништа у поређењу са свеобухватним механизмом за управљање референцама који је саставни део система ConTeXt. Његова одлика је могућност комбинације механизма за управљање интерним референцама, намењеним за одржавање референци ажурним, са употребом хиперлинкова, што очигледно није искључиво само за спољашње референце.

9.2 Интерне референце

За успостављање интерне референце, потребне су две ствари:

1. Лабела или идентификатор на циљној тачки. Док компајлира документ, ConTeXt ће овој лабели да придружи одређене податке. Врсте лабеле ће одредити тачно које податке; то може бити број одељка, број слике, број који је придружен одређеној ставки нумерисане листе, наслов одељка, итд.
2. Команда на месту извора која чита податке придружене лабели повезаној са местом циља и која их уметне на место извора. Команда варира у зависности од тога које податке лабеле желимо да уметнемо на место извора.

Када говоримо о референци, то радимо у релацији „извор → циљ”, тако да би можда требало најпре да објаснимо ствари које се тичу извора, па тек онда ствари које се тичу циља. Међутим, сматрам да је разумевање логике референци лакше ако се редослед објашњавања обрне.

9.2.1 Лабела у циљу референце

У овом поглављу под појмом *лабела* подразумевам текст стринг који ће се придружити месту циља референце и који ће се интерно користити за додавање одређених информација у вези са местом циља референце, као што су, на пример, број странице, број одељка, итд. У ствари, информације придружене свакој лабели зависе од процедуре којом се лабела креира. ConTeXt ове лабеле назива *референце*, мада ја сматрам да је овај други појам мање јасан јер поседује много шире значење.

Лабела придружена циљу референце:

- Захтева да је сваки потенцијални циљ у документу јединствен, тако да се може одредити једнозначно. Ако исту лабелу користимо за различите циљеве, ConTeXt неће пријавити грешку, већ ће учинити да две референце указују на прву лабелу коју пронађе (у изворном фајлу), па ће споредни ефекат тога бити да су можда неке од наших референци погрешне, а што је још горе, нећемо то приметити. Из тог разлога је врло важно обезбедити да када се креира лабела, новој лабели којој вршимо доделу није већ раније извршена додела.
- Може да се састоји из слова, цифри, знакова интерпункције, размака, итд. На местима на које се наиђе на размаке, и даље се примењују општа ConTeXt правила која се тичу ове врсте карактера (погледајте одељак 4.2.1), тако да се, на пример, „Moja fina labela” и „Moja fina labela” третирају као иста лабела, мада је у обе примењен различит број размака.

Пошто нема ограничења у карактерима који могу да се користе у имену лабеле, а ни у броју карактера, мој савет је да се употребе имена лабела која су јасна, па ће нам то помоћи да разумемо изворни фајл када га будемо читали много касније након што је написан. То је разлог што пример који сам малопре навео („Moja fina labela”) није баш добар пример, јер нам не говори ништа о циљу на који указује лабела. На пример, за наслов овог одељка би лабела 'sec:Labela cilja' била боља.

За придруживање одређеног циља лабели, у суштини постоје две процедуре:

1. Путем аргумента или опције команде која се користи за креирање елемента на који ће лабела да указује. Са ове тачке гледишта, све команде које креирају неку врсту структуре или текст елемент који може бити циљ референце прихватају и опцију која се назива „reference” и која се користи за уметање лабеле. Понекада је лабела уместо опције садржај комплетног аргумента.

Добар пример за ово што сам навео се налази у командама поделе, које као што знамо из (одељка 7.3), омогућавају неколико различитих синтакси. Команда се у класичној синтакси пише као:

```
\section[Лабела]{Наслов}
```

а у синтакси специфичној за ConTeXt Mark IV, команда се пише као

```
\startsection
[title=Наслов, reference=Лабела, ... ]
```

У оба случаја команда форсира уметање лабеле која ће се придружити одељку (или поглављу, пододељку, итд.) о коме је реч.

Рекао сам да се ова могућност налази у свим командама које нам омогућавају да креирамо текст елемент који може да постане циљ референце. Све су то текст елементи који се нумеришу, од којих су неки као што су то одељци, плутајући објекти свих врста (табеле, слике и слично), фусноте или напомене на крају, цитати, нумерисане листе, описи, дефиниције, итд.

Када се лабела уноси директно аргументом, а не као опција којој се додељује вредност, у систему ConTeXt је могуће да се неколико лабела придружи истом циљу. На пример:

```
\chapter[labela1, labela2, labela3] {Моје поглавље}
```



Није ми потпуно јасно која би била предност имати више различитих лабела за један циљ, али претпостављам да је то могуће не због неке предности, већ услед неког *интерног* захтева система ConTeXt који важи за одређене врсте аргумената.

2. Помоћу команди `\pagereference`, `\reference`, или `\textreference` чија је синтакса:

```
\pagereference[Лабела]
\reference[Лабела]{Текст}
\textreference[Лабела]{Текст}
```

- Лабела која се креира са `\pagereference` нам омогућава да добијемо број странице.
- Лабеле које се креирају са `\reference` и `\textreference` нам омогућавају да добијемо и број странице као и текст који се придружује уз њих и наведен је као аргумент.

И у `\reference` и у `\textreference`, текст који се везује за лабелу не постоји у финалном документу на месту на којем се налази команда (циљ референце), али може да се врати и појави на месту извора референце.

Раније сам рекао да се свакој лабелу придружују одређене информације у вези места циља. Од типа лабеле зависи које су то информације:

- Све лабеле *ћамће* (у смислу да омогућавају враћање) број странице команде која их је креирала. За лабеле придружене одељцима који се можда простиру на више страница, тај број ће бити број странице где одељак о којем је реч почиње.
- Лабеле уметнуте командом која креира нумерисани текст елемент (одељак, напомена, табела, слика, итд.) *ћамће* број придружен том елементу (број одељка, број напомене, итд.)
- Ако овај елементи има *наслов*, као што је то случај са, на пример, одељцима, али такође и табелама ако су биле уметнуте командом `\placetable`, она ће памтити тај наслов.
- Лабеле креиране командом `\pagereference` *ћамће* само број странице.
- Оне креиране командама `\reference` или `\textreference` такође памте текст који им је придружен и које ове команде узимају као аргумент.



Уствари нисам сигуран шта је стварна разлика између команди `\reference` и `\textreference`. Мислим да је могуће да дизајн три команде које омогућавају креирање лабеле покушава да се одговара дизајну три команде које омогућавају добијање информација од лабела (које ћемо представити ускоро); али истина је да су, према мојим тестовима, `\reference` и `\textreference` редундантне команде.

9.2.2 Команде на месту извора референце које преузимају податке са места циља

Команде које ћу управо да представим преузимају информације из лабела, а ако је наш документ интерактиван, генеришу и везу са циљем референце. Али битна ствар у вези са

овим командама је да се информације преузимају од лабеле. Ако само желимо да креирамо везу, без преузимања било каквих информација од лабеле, морамо да употребимо команду `\goto` која је објашњена у одељку 9.4.2.

А. Основне команде за преузимање информација од лабеле

Имајући у виду да свака лабела која се придружи месту циља може да чува различите информације, логично је да ConTeXt нуди три различите команде за преузимање тих информација; у зависности од тога коју информацију са места циља референце желимо да преузмемо, користимо једну од следећих команди:

- Команда `\at` нам омогућава да преузмемо број странице на којој је лабела.
- За лабеле које памте број елемента (број одељка, број напоменем број ставке, број табеле, итд.) уз број странице, команда `\in` нам омогућава да преузмемо и овај број.
- Коначно, за лабеле које памте текст који се придружује лабели (наслов одељка, наслов слике уметнуте командом `\placefigure`, итд.) команда `\about` нам омогућава да преузмемо тај текст.

Ове три команде, `\at` `\in` `\about`, имају исту синтаксу:

```
\at{Текст}[Лабела]
\in{Текст}[Лабела]
\about{Текст}[Лабела]
```

- *Лабела* је лабела од које желимо да преузмемо информацију.
- *Текст* је текст написан непосредно испред информације коју желимо да преузмемо командом. Између текста и податка из лабеле ће се уметнути размак који не може да се преломи, а ако је функција интерактивности укључена на такав начин да команда осим преузимања информације генерише и линк који нам омогућава да скочимо на место циља, текст који је наведен као аргумент ће бити део линка (на њега ће моћи да се кликне).

У наредном примеру можемо видети како `\in` преузима број одељка, а `\at` број странице.

У одељку `\in{одељку}[sec:target labels]`, који почиње на `\at{страници}[sec:target labels]`, објашњавају се карактеристике лабела које се користе за интерне референце.

У одељку 9.2.1, који почиње на страници 142, објашњавају се карактеристике лабела које се користе за интерне референце.

Запазите да је ConTeXt аутоматски креирао хиперлинкове (погледајте одељак 9.3), и да је текст наведен као аргумент за `\in` и `\at` део линка. Али да смо ово написали на другачији начин, резултат би био следећи:

У одељку `\in{}[sec:target labels]`, који почиње на страници `\at{}[sec:target labels]`, објашњавају се карактеристике лабела које се користе за интерне референце.

У одељку 9.2.1, који почиње на страници 142, објашњавају се карактеристике лабела које се користе за интерне референце.

Остатак текста остаје исти, али речи *одељку* и *сјраници* које се налазе испред референце нису део линка, јер више нису део команде.

Ако ConTeXt није могао да пронађе лабелу на коју указују команде `\at`, `\in` или `\about`, неће се јавити грешка у компајлирању, али ће се у финалном документу, на месту на којем би информација коју ове команде преузимају требало да се прикаже, исписати „??”.

Постоје два разлога услед којих ConTeXt не може да пронађе лабелу:

1. Направили смо грешку док смо је писали.
2. Компајлирамо само део документа, а лабела указује на део који још увек није компајлиран (погледајте одељке 4.5.1 и 4.6).

У првом случају ће бити потребно да се грешка исправи. Стога је добра идеја да када завршимо компајлирање комплетног документа (па више није могуће да дође до другог случаја), у PDF фајлу потражимо сва појављивања „??” и тако потврдимо да у документу нема прекинутих референци.

Б. Преузимање информације придружене лабели командом `\ref`

Свако од `\at`, `\in` и `\about` преузима неке елементе лабеле. Постоји још једна команда која нам омогућава да преуземо неке елементе лабеле који јој се наведу. То је команда `\ref`, а њена синтакса је:

`\ref[Елемент који се преузима][Лабела]`

где први аргумент може да буде:

- `text`: преузима текст придружен лабели.
- `title`: преузима наслов придружен лабели.
- `number`: преузима број повезан са лабелом. На пример, у одељцима је то број одељка.
- `page`: преузима број странице.
- `realpage`: преузима стварни број странице.
- `default`: преузима оно што ConTeXt сматра за *природни* елемент лабеле. У општем случају се ово подудара са оним што преузима `number`.

Уствари, команда `\ref` је много прецизнија него `\at`, `\in` или `\about`, па тако на пример, прави разлику између броја странице и стварног броја странице. Број странице не мора да се подудари са стварним бројем странице ако, на пример, нумерисање документа почне од 1500 (јер је документ наставак неког претходног) или ако су странице у преамбули биле нумерисане римским бројевима, па се у главном делу поново почиње. Слично овоме, `\ref` прави разлику и између *шексиа* и *наслова* придруженог референци, нешто што `\about`, на пример, не ради.

Ако се `\ref` употреби за преузимање информације од лабеле која такву информацију нема (нпр. наслов лабеле придружене фусноти), команда ће да врати празан string.

В. Откривање места на које води линк

ConTeXt такође обезбеђује и две команде осетљиве на *адресу линка*. „Адресом линка” желим да одредим да ли се циљ линка у изворном фајлу проналази испред или иза извора. На пример: пишемо наш документ и желимо да укажемо на одељак који би у коначном садржају могао да дође испред или иза онога који управо пишемо. Просто јер још увек нисмо донели одлуку. У оваквој ситуацији би било корисно имати команду која пише једно или друго у зависности од тога да ли циљ на крају дође испред или иза извора у финалном документу. Као одговор на потребе као што је ова, ConTeXt обезбеђује команду `\somewhere` чија је синтакса:

`\somewhere{Текст ако је испред}{Текст ако је иза}[Лабела].`

На пример, у следећем тексту:

Команда `\type{\somewhere}` такође може и да детектује адресу хиперлинка. На овај начин можемо такође да пронађемо поглавља или остале текст елементе `\somewhere{испред}{иза}[sec:references]` и да о њиховим описима дискутујемо на неком другом месту `\somewhere{испред}{иза}[sec:interactivity]`.

Команда `\somewhere` такође може и да детектује адресу хиперлинка. На овај начин можемо такође да пронађемо поглавља или остале текст елементе *испред* и да о њиховим описима дискутујемо на неком другом месту *иза*.

За овај пример сам употребио две стварне лабеле у изворном фајлу овог поглавља.

Још једна команда која може да детектује да ли се лабела на коју покажемо налази испред или иза је команда `\atpage` чија синтакса је:

`\atpage[лабела]`

Ова команда је врло слична претходној, али уместо да нам омогући да сами наведемо текст, у зависности од тога да ли се лабела налази испред или иа, `\atpage` умеће подразумевани текст за свака од два случаја, а ако је документ интерактиван, умеће и хиперлинк.

Текст који умеће `\atpage` је онај који се придружи уз „precedingpage” ознаке у случају када се лабела коју прихвата као аргумент налази *испред* команде, а „hereafter” у супротном случају.

Када сам стигао до овде, схватио сам да ме је претходна одлука преварила: у овом одељку сам одлучио да оно што ConTeXt назива „референца”, зовем „лабела”. Изгледало ми је јасно. Али одређени фрагменти текста које генеришу ConTeXt команде, као што је команда `\atpage`, такође се називају „лабеле”¹ (овога пута у другом смислу). (Погледајте [одељак 10.5.3](#)). Надам се да се читалац неће збунити. Сматрам да се из контекста може разумети на које различито значење речи *лабела* мислим у сваком случају.

Дакле, текст који умеће команда `\atpage` можемо да изменимо на исти начин као што мењамо текст било које друге ознаке:

`\setuplabeltext[sr][precedingpage=Нови текст]`
`\setuplabeltext[sr][hereafter=Нови текст]`

¹ У српском преводу је за овај појам изабран израз *ознаке*, тако да не долази до овог проблема – *ѝрим. ѝрев*.

Верујем да на овом месту постоји ситна грешка у „ConTeXt Standalone” (дистрибуцији коју користим). Испитујући имена предефинисаних ознака у систему ConTeXt које могу да се промене командом `\setuplabeltext`, проналазе се два пара ознака које су добри кандидати да их користи команда `\atpage`:

- „precedingpage” и „followingpage”.
- „hencefore” и „hereafter”.

Можемо претпоставити да би `\atpage` користила или први или други пар. Али уствари, за ставке које долазе испред, она користи „precedingpage”, а за оне које долазе иза користи „hereafter”, па ми се чини да то није конзистентно.

9.2.3 Аутоматско генерисање префикса којим се спречавају лабеле дупликати

У великом документу није увек једноставно да се спречи дуплирање лабела. Стога се препоручује да се уведе неки ред у начин који изаберемо за именовање лабела. Једна пракса која помаже је да се користе префикси за лабеле који се мењају у зависности од типа лабеле. На пример, „sec:” за одељке, „fig:” за слике, „tbl:” за табеле, итд.

У вези овога, ConTeXt обезбеђује колекцију алата који омогућавају да:

- Сам ConTeXt аутоматски генерише лабеле за све дозвољене елементе.
- Свака ручно генерисана лабеле може да има префикс, било неки који смо ми унапред одредили, било онај који аутоматски генерише ConTeXt.

Детаљно објашњење овог механизма је дугачко, и мада су ово несумњиво корисни алати, не сматрам да су од суштинске важности. Дакле, пошто не могу да се објасне у неколико речи, радије нећу да их објашњавам, већ ћу да упутим на оно што се о њима говори у ConTeXt референтном упутству, или на [викију](#).

9.3 Интерактивни електронски документи

Само електронски документи могу да буду интерактивни; али нису сви. *Електронски* документ је онај који се чува у компјутерском фајлу и који може да се отвори и чита директно на екрану. С друге стране, електронски документ опремљен алатима који кориснику омогућавају да врши *интеракцију* са њим, назива се интерактивни; то јест: можемо са њим још нешто осим пуког читања. Интерактивност постоји када, на пример, документ поседује дугмад која врше неку акцију, или линкове којима можемо да скочимо на неко друго место у документу, или на спољашњи документ; или када постоје површине документа у којима корисник може да пише, или постоје аудио или видео клипови који могу да се пусте, итд.

Сви документи које генерише ConTeXt су електронски (пошто ConTeXt генерише PDF који је по својој дефиницији електронски документ), али они нису увек интерактивни. Укључивање интерактивности мора експлицитно да се уради на начин који је показан у следећем одељку.

Ипак, имајте на уму да иако ConTeXt генерише интерактивни PDF, потребан вам је PDF читач који омогућава да употребите ту интерактивност, пошто не може баш сваки постојећи

PDF читач да користи хиперлинкове, дугмад и сличне ствари које се тичу интерактивних докумената.

9.3.1 Укључивање интерактивности у документима

ConTeXt подразумевано не користи интерактивне функције, осим ако му експлицитно не кажете да их користи. То се обично ради у преамбули документа. Команда која укључује ову могућност је:

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

Када желимо да генеришемо интерактивни документ, ова команда се обично употреби само једном у преамбули. Али у суштини можемо да је користимо колико год желимо често када је потребно да изменимо стање интерактивности документа. Опција „state=start” укључује интерактивност, док је опција „state=stop” искључује, тако да интерактивност можемо да искључимо у неким поглављима или *деловима* нашег документа где тако желимо.

Не пада ми на памет ниједан разлог због којег би пожелели да имамо неинтерактивне делове документа који је интерактиван. Али оно што је важно у вези са филозофијом система ConTeXt је да ако је нешто технички могуће, чак и ако није вероватно да ћемо га користити, систем нуди процедуру да се то уради. Оваква филозофија је разлог што систем ConTeXt поседује огроман број могућности и спречава да прост увод, као што је овај, буде *кратак*.

Једном када се успостави интерактивност:

- Одређене ConTeXt команде ће већ поседовати хиперлинкове. Дакле:
 - Команде за креирање садржаја ће бити, у принципу и само ако се експлицитно не наведе другачије, интерактивне, тј. клик на ставку садржаја врши скок на страницу на којој почиње тај одељак.
 - Команде за интерне референце које смо видели у првом делу овог поглавља, где се кликом на њих аутоматски скаче на циљ референце.
 - Фусноте и белешке на крају, где када се кликом на сидро у главном телу текста скаче на страницу на којој је написана сама напомена, а кликом на ознаку напомене у тексту напомене се скаче на место у главном тексту на којем је извршен позив на напомену.
 - Итд.
- Укључује се могућност употребе осталих команди посебно дизајнираних за интерактивне документе, као што су презентације. Оне употребљавају многе алате везане за интерактивност, као што су дугмад, менији, слике за пресвлачење, уграђени звук или видео, итд. Објашњење свега овога би било предугачко, а и презентације су прилично специјална врста документа. У следећим редовима ћу зато описати једну особину везану за интерактивност: хиперлинкове.

9.3.2 Основна конфигурација интерактивности

Уз укључивање и искључивање интерактивности, команда `\setupinteraction` нам омогућава и да конфигуришемо неке ствари везане за интерактивност; углавном, мада не само то, боју и стил линкова. То се ради следећим опцијама команде:

- `color`: контролише *нормалну* боју линкова.
- `contrastcolor`: одређује боју линкова којима је циљ на истој страници на којој је и извор. Препоручујем да се ова опција увек постави на исти садржај као и претходна.
- `style`: контролише стил линка.
- `title`, `subtitle`, `author`, `date`, `keyword`: вредности додељене овим опцијама се конвертују у метаподатке PDF фајла који генерише ConTeXt.
- `click`: ова опција контролише да ли линк треба да се истакне када се на њега кликне.

9.4 Хиперлинкови на спољашње документе

Направићу разлику између команди које не креирају линк већ помажу код словослагања URL адреса линка, и команди које креирају хиперлинк. Хајде да их погледамо одвојено:

9.4.1 Команде које помажу у словослагању хиперлинкова, али их не креирају

URL адресе су углавном врло дугачке и садрже све врсте карактера, чак и карактере који су резервисани у систему ConTeXt и не могу директно да се користе. Уз то, када у документу треба да се прикаже URL, веома је тешко сложити пасус, јер URL може да прекорачи дужину линије и уопште не садржи размаке који би могли да се искористе за уметање прелома линије. Чак штавише, у URL адресама није разумно растављати речи да би се уметнули преломи редова, јер је читаоцу врло тешко да зна да ли је цртица последица хифенације, или је уствари део саме URL адресе.

ConTeXt зато обезбеђује две могућности за *словослагање* URL адреса. Прва је првенствено намењена URL адресама које ће се користити интерно, али се уствари неће приказивати у документу. Друга служи за URL адресе које треба да се напишу у документ. Хајде да их погледамо једну по једну:

`\useURL`

Ова команда нам омогућава да у преамбули документа напишемо URL и да му придружимо неко име, па када желимо да га употребимо у документу, можемо да га позовемо тим именом. Посебно је корисно са URL адресама које ће се у документу користити више пута.

Команда подржава две начина употребе:

1. `\useURL[Придružено име][URL]`
 2. `\useURL[Придružено име][URL][][Текст линка]`
- У првој верзији, URL адреси се просто придружује име којим ће се позивати у документу. Али онда, да би употребили URL, када га позивамо мораћемо на неки начин да наведемо текст који ће се приказати у документу и на који ће моћи да се кликне.
 - У другој верзији последњи аргумент садржи текст на који ће моћи да се кликне. Трећи аргумент постоји у случају када URL желимо да поделимо на два дела, тако да први део садржи приступну адресу, а други део име одређеног документа или страницу коју желимо да отворимо. На пример: адреса документа који објашњава шта је ConTeXt: <http://www.pragma-ade.com/general/manuals/what-is-context.pdf>. Ова адреса може да се у потпуности напише као други аргумент, чиме трећи остаје празан:

```
\useURL [WhatIsCTX]
[http://www.pragma-ade.com/general/manuals/what-is-context.pdf]
[]
[Шта је \ConTeXt?]
```

али такође можемо и да је поделимо на два аргумента:

```
\useURL [WhatIsCTX]
[http://www.pragma-ade.com/general/manuals/]
[what-is-context.pdf]
[Шта је \ConTeXt?]
```

У оба случаја ћемо ту адресу придружити уз реч „WhatIsCTX”, тако да укључи линк на ту адресу, а ту команду ћемо користити да креирамо линк; уместо саме URL адресе, просто можемо да напишемо „WhatIsCTX”.

Ако на било које место у тексту желимо да уметнемо ту URL адресу коју смо придружили имену командом `\useURL`, можемо да употребимо `\url[Придružено име]` што у документ уметне URL адресу придружену том имену. Али мада уписује URL адресу, ова команда не креира било какав линк.

Формат у којем се командом `\url` приказују URL адресе није онај који се поставља на општи начин помоћу команде `\setupinteraction`, већ помоћу посебне команде која постоји за ову команду. То је команда `\setupurl` која нам омогућава да поставимо стил (опцијом `style`) и боју (опцијом `colour`).

`\hyphenatedurl`

Ова команда је намењена за URL адресе које ће се писати у тексту нашег документа, и ако је потребно, ConTeXt мора да унутар URL адресе уметне преломе линије како би исправно сложио пасус. Њен формат је:

```
\hyphenatedurl{URLадреса}
```

Упркос имену команде `\hyphenatedurl`, она не врши хифенацију имена URL адресе. Она сматра да су одређени чести карактери у URL адресама добре тачке за уметање

прелома линије пре или иза њих. Карактере које желимо можемо да додамо на листу карактера код којих се дозвољава прелом линије. За то имамо три команде на располагању:

```
\sethyphenatedurlnormal{Карактери}
\sethyphenatedurlbefore{Карактери}
\sethyphenatedurlafter{Карактери}
```

Ове команде редом додају карактере који им се проследе као аргументи на листу карактера који подржавају прелом линије испред и иза, листу карактера који подржавају само прелом линије и оних који дозвољавају прелом линије уназад.

`\hyphenatedurl` може да се користи кадгод мора да се упише URL адреса која ће се појавити у финалном документу онаква каква је. Чак може да се користи и као последњи аргумент команде `\useURL` у верзији те команде где последњи аргумент доставља текст на који може да се кликне и који се приказује у финалном документу. На пример:

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

У аргументу команде `\hyphenatedurl` могу да се користе сви резервисани карактери осим три који морају да се замене командама:

- % мора да се замени са `\letterpercent`
- # мора да се замени са `\letterhash`
- \ мора да се замени са `\letterescape` или `\letterbackslash`.

Сваки пут када `\hyphenatedurl` уметне прелом линије, она извршава команду `\hyphenatedurlseparator` која подразумевано не ради ништа. Али ако је редedefинишемо, у URL се умеће репрезентативни карактер на сличан начин на који се то дешава са обичним речима у којима се умеће цртица која означава да се реч наставља у наредној линији. На пример:

```
\def\hyphenatedurlseparator{\curvearrowright}
```

ће дакле приказати следећу прилично дугачку веб адресу на следећи начин:

[https://support.microsoft.com/?scid=http://support.microsoft.com:80~
/support/kb/articles/Q208/4/27.ASP&NoWebContent=1.](https://support.microsoft.com/?scid=http://support.microsoft.com:80/~support/kb/articles/Q208/4/27.ASP&NoWebContent=1)

9.4.2 Команде које успостављају линк

Да би помоћу `\useURL` успоставили линкове на предефинисане URL адресе можемо да употребимо команду `\from`, која је ограничена на успостављање линка, али не уписује никакав текст на који може да се кликне. Подразумевани текст у `\useURL` ће се употребити као текст линка. Њена синтакса је:

```
\from[Име]
```


где је *Име* име које раније придружено URL адреси командом `\useURL`.

Да бисмо креирали линкове и придружили их тексту на који може да се клике, а који није раније дефинисан, на располагању имамо команду `\goto` која се користи за генерисање и интерних и спољашњих линкова. Њена синтакса је:

`\goto{Текст на који може да се кликне}[Циљ]`

где је *Текст* на који може да се кликне текст који ће се приказати у финалном документу и на који када се кликне мишем, изврши се скок, а *Циљ* може да буде:

- Лабела из нашег документа. У овом случају `\goto` ће да генерише скок на сличан начин као, на пример, команде `\in` или `\at` које смо већ истражили. Али за разлику од тих команди, не преузима се никаква информација придружена лабели.
- Сама URL адреса. У овом случају се мора експлицитно навести да је то URL адреса, тако што се команда пише на следећи начин:

`\goto{Текст на који може да се кликне}[url(URL)]`

где URL може бити редом, име претходно придружено URL адреси командом `\useURL`, или сама URL адреса, у ком случају када се пише URL, морамо обезбедити да се ConTeXt резервисани карактери пишу онако како то захтева ConTeXt. Писање URL адресе сагласно са ConTeXt правилима неће утицати на функционалност линка.

9.5 Креирање маркера у финалном PDF фајлу

PDF фајлови могу да имају интерну листу маркера свог садржаја која омогућава да читалац види садржај документа у посебном прозору програма за преглед PDF фајлова, и да се креће кроз њега простим кликтањем на сваки од одељака и пододељака.

ConTeXt подразумевано у излазном PDF фајлу не креира у маркер листу садржаја, мада то може да се добије једноставним уметањем команде `\placebookmarks`, чија је синтакса:

`\placebookmarks[Листа одељака]`

где је *Листа одељака* листа нивоа одељака раздвојена запетама који би требало да се појаве у листи садржаја.

Имајте на уму следећа запажања везана за ову команду:

- Према мојим тестовима `\placebookmarks` не функционише ако се налази у преамбули документа. Али унутар тела документа (између `\starttext` и `\stoptext`, или између `\startproduct` и `\stopproduct`), није битно на које место је поставите: листа маркера ће укључити и одељке или пододељке који се налазе испред команде. Међутим, верујем да је за исправно разумевање изворног фајла, најпаметније да се команда постави на почетак.
- Типови одељака које дефинише корисник (командом `\definehead`) се не налазе увек на исправном месту у листи маркера. Пожељно је да се не укључе у листу.

- Ако наслов било ког одељка у себи садржи белешку на крају или фусноту, текст фусноте ће се сматрати као део маркера.
- Уместо листе одељака, као аргумент можемо једноставно да наведемо симбол „all” који, као што му име наговештава, укључује све одељке; међутим, према мојим тестовима, ова реч уз оно што су без сумње одељци, укључује и текстове који се ту постављају неким командама које не служе за поделу, тако да је добијена листа донекле непредвидљива.

Не омогућавају нам сви програми за преглед PDF фајлова да видимо маркере; а и многи који то омогућавају, немају ту могућност подразумевано активирану. Зато, да бисмо проверили резултат ових функција, морамо бити сигурни да наш PDF читач ту функцију подржава и да је укључена. Чини ми се да Acrobat, на пример, подразумевано не приказује маркере, мада постоји тастер на алатној линији којим можемо да их прикажемо.

III

Специфична питања

Глава 10

Карактери, речи, текст и хоризонтални размак

Садржај: 10.1 Добијање карактера којима нормално не може да се приступи са тастатуре; 10.1.1 Дијакритици и специјална слова; 10.1.2 Традиционалне лигатуре; 10.1.3 Грчка слова; 10.1.4 Разни симболи; 10.1.5 Дефинисање карактера ; 10.1.6 Употреба предефинисаних скупова симбола; 10.2 Специјално форматирање карактера; 10.2.1 Верзал, курент и лажни капитал; 10.2.2 Текст у експоненту или индексу; 10.2.3 Дословни текст; 10.3 Размак између карактера и речи; 10.3.1 Аутоматско постављање хоризонталног размака; 10.3.2 Промена размака између карактера у речи; 10.3.3 Команде за додавање хоризонталног размака између речи; 10.4 Сложенице; 10.5 Језик текста; 10.5.1 Постављање и измена језика; 10.5.2 Конфигурисање језика; 10.5.3 Ознаке које се придружују одређеним језицима; 10.5.4 Неке команде које се тичу језика; А Команде у вези са датумом; Б Команда `\translate`; В Команде `\quote` и `\quotation`;

Основни централни елемент свих текст докумената је карактер: карактери се групишу у речи, које формирају линије, које формирају пасусе, а који формирају странице.

Ово поглавље које почиње од „*каракѿера*” објашњава неке од алата система ConTeXt који се тичу карактера, речи и текста.

10.1 Добијање карактера којима нормално не може да се приступи са тастатуре

У текст фајлу кодираном као UTF-8 (погледајте [одељак 4.1](#)) можемо да користимо било који карактер или симбол, и из живих и из многих изумрлих језика. Али, пошто су могућности тастатуре ограничене, већини карактера и симбола који су дозвољени у UTF-8 обично не може директно да се приступи са тастатуре. Ово је посебно случај са многим дијакритицима, тј. знацима који се постављају изнад (или испод) одређених слова, и који им дају посебну вредност; али такође и многи остали карактери као што су математички симболи, традиционалне лигатуре, итд. Многе од ових карактера у систему ConTeXt можемо добити користећи команде.

10.1.1 Дијакритици и специјална слова

Скоро сви западни језици имају дијакритике (уз важан изузетак највећег дела енглеског језика), а у општем случају, тастатуре могу да генеришу дијакритике који одговарају регионалним језицима. Тако шпанска тастатура може да генерише све дијакритике потребне у шпанском језику (углавном акценте и дијерезис) као и неке дијакритичке знаке који се користе у осталим језицима, као што је каталонски (гравис акценти и дијерезис) или француски (седиј, гравис и циркумфлекс акценти); мада не и неке који се користе, на пример, у португалском језику као што је тилда над неким самогласницима у речима као што је „*navegação*”.

TeX је дизајниран у Сједињеним Америчким Државама где тастатуре у општем случају не омогућавају добијање дијакритика; зато му је ДОНАЛД КНУТ направио скуп команди којима можемо да добијемо скоро све познате дијакритичке знаке (барем оне у језицима који користе латиницу). Ако користимо шпанску тастатуру, нема много смисла употребљавати ове команде за оне дијакритике који могу да се добију директно са тастатуре. Ипак је важно знати да ове команде постоје, и како се називају, јер шпанским (или италијанским, или француским...) тастатурама не можемо да генеришемо све могуће дијакритике.

Име	Карактер	Скраћеница	Команда
Акут акцент	acute	<code>\'u</code>	<code>\uacute</code>
Гравис акцент	grave	<code>\`u</code>	<code>\ugrave</code>
Циркумфлекс акцент	circumflex	<code>\^u</code>	<code>\ucircumflex</code>
Дијерезис или умлаут (трема)	diaeresis	<code>\"u</code>	<code>\udiaeresis, \uumlaut</code>
Тилда	tilde	<code>\~u</code>	<code>\utilde</code>
Макрон	macron	<code>\=u</code>	<code>\umacron</code>
Брев	breve	<code>\u u</code>	<code>\ubreve</code>

Табела 10.1 Акценти и остали дијакритици

У табели 10.1 се налазе команде и скраћенице којима можемо да добијемо ове дијакритике. У свим случајевима није битно да ли користимо команду или скраћеницу. У табели сам користио слово 'u' као пример, али ове команде функционишу за сваки самогласник (односно за већину њих¹), а и за неке сугласнике и полусамогласнике.

- Пошто су већина скраћених команди *контролни симболи* (погледајте одељак 3.2), слово на којем треба да се нађе дијакритик може да се напише непосредно након команде, или одвојено од ње. Тако на пример: ако желимо да добијемо португалско 'ã', можемо да напишемо `\=a` или `_a` карактере.² Али у случају брев (`\u`), ради се о *контролној речи*, па је размак обавезан.
- У случају дугачке верзије команде, слово на које се поставља дијакритик ће бити прво слово имена команде. Тако на пример, `\emacron` ће да постави макрон изнад малог слова 'е' (ë), `\Emacron` ће урадити исто за велико слово 'Е' (Ë), док ће `\Amacron` урадити исто за велико слово 'А' (Ä).

Мада команде у табели 10.1 раде са самогласницима и неким сугласницима, постоје остале команде за генерисање неких дијакритика и специјалних слова које функционишу само за једно или неколико слова. Оне су приказане у табели 10.2.

Желео бих да истакнем да неке од команди у горњој табели генеришу карактере од осталих карактера, док друге команде раде само ако фонт који користимо поседује дати карактер. На пример, што се тиче немачког есцет (ß), табела приказује две команде, али само један карактер, јер фонт који овде користим поседује само верзал верзију немачког есцет (што је уобичајено у фонтовима).

¹ Од свих команди које се налазе у табели 10.1, тилда не функционише са словом 'е' и није ми јасно зашто.

² Упамтите да када је у овом документу важно да се виде, размаке представљамо са ' '.

Име	Карактер	Скраћеница	Команда
Скандинавско О	ø, Ø	\o, \O	
Скандинавско А	å, Å	\aa, \AA, {\r a}, {\r A}	\aring, \Aring
Пољско L	ł, Ł	\l, \L	
Немачко есцет	ß	\ss, \SS	
'i' and 'j' без тачке	i, j	\i, \j	
Мађарски умлаут	ű, Ű	\H u, \H U	
Седиј	ç, Ç	\c c, \c C	\ccedilla, \Ccedilla

Табела 10.2 Још дијакритика и специјалних слова

Вероватно је то разлог што не могу да добијем верзал скандинавског А мада команде „{\r A}” и `\Aring` раде како треба.

Мађарски умлаут такође ради са словом 'о', а седиј са словима 'k', 'l', 'n', 'r', 's' и 't', у верзалу и у куренту. Потребно је редом употребити команде `\kcedilla`, `\lcedilla`, `\ncedilla` ...

10.1.2 Традиционалне лигатуре

Лигатура се формира спајањем две или више графема које се обично пишу одвојено. Ово „стапање” два карактера је често почело као врста скраћивања у рукописним текстовима, све док коначно нису достигли одређену типографску самосталност. Неке од њих су чак биле део карактера који се обично дефинишу у типографском фонту, као што су амперсанд, '&', који је настао као сажимање латинске копуле (конјункције) „et”, или немачко есцет односно оштро С (ß), које је, као што му име наговештава, почело као комбинација 's' и 'z'. У неким дизајнима фонтова, чак и данас можемо пратити порекло ових карактера; или можда их ја видим јер знам да су ту. Тачније, '&' у фонту Pagella и 'ß' у фонту Bookman.

Као вежбу, предлажем (након читања поглавља 6, где се објашњава како се то ради) да покушате приказати ове карактере тим фонтовима у довољној величини (на пример, 30 pt) како би могли да уочите њихове компоненте.

Остале традиционалне лигатуре које нису постале толико популарне, али се данас још увек повремено користе, су латински завршеци „oe” и „ae” који су повремено писане као 'œ' или 'æ' како би се назначило да у латинском формирају дифтонг. Ове лигатуре се у систему ConTeXt добијају командама из табеле 10.3

Лигатура	Скраћеница	Команда
æ, Æ	\ae, \AE	\aeligature, \AEligature
œ, Æ	\oe, \OE	\oeligature, \OEligature

Табела 10.3 Традиционалне лигатуре

Лигатура која је у шпанском (кастиљском) била традиционална и која се данас обично не налази у фонтовима је 'Ð': сажимање 'D' и 'E'. Колико знам, у ConTeXt не постоји команда која нам омогућава да то урадимо,¹ али можемо да је сами креирамо, као што је објашњено у одељку 10.1.5.

¹ Док у L^AT_EX можемо да употребимо команду `\DH` коју имплементира пакет „fontenc”.

Уз претходне лигатуре, које сам назвао *традиционалне* јер имају порекло у рукописима, након проналаска штампарске пресе развијене су одређене штампане лигатуре које ћу да зовем „типograфске лигатуре”, које ConTeXt посматра као алате фонта и којима аутоматски управља програм, мада можемо утицати на начин на који се ради са тим алатима (укључујући и лигатуре) командом `\definefontfeature` (која у овом уводу није објашњена).

10.1.3 Грчка слова

У математичким и физичким формулама је уобичајена употреба грчких слова. ConTeXt је из тог разлога омогућио генерисање свих слова грчког алфабета, и малих и великих. Овде се команда гради од енглеског имена датог слова. Ако се први карактер напише малим словом, добиће се мало грчко слово, а ако се напише великим словом, добиће се велико грчко слово. На пример, команда `\mu` ће да генерише курент верзију овог слова (μ), док ће `\Mu` да генерише верзал верзију (M). Табела 10.4 приказује које команде генеришу свако слово грчког алфабета, мала и велика слова.

Српско име	Карактер (к/в)	Команде (к/в)
Алфа	α , A	<code>\alpha</code> , <code>\Alpha</code>
Бета	β , B	<code>\beta</code> , <code>\Beta</code>
Гама	γ , Γ	<code>\gamma</code> , <code>\Gamma</code>
Делта	δ , Δ	<code>\delta</code> , <code>\Delta</code>
Епсилон	ϵ , ε , E	<code>\epsilon</code> , <code>\varepsilon</code> , <code>\Epsilon</code>
Зета	ζ , Z	<code>\zeta</code> , <code>\Zeta</code>
Ета	η , H	<code>\eta</code> , <code>\Eta</code>
Тета	θ , ϑ , Θ	<code>\theta</code> , <code>\vartheta</code> , <code>\Theta</code>
Јота	ι , I	<code>\iota</code> , <code>\Iota</code>
Капа	κ , \varkappa , K	<code>\kappa</code> , <code>\varkappa</code> , <code>\Kappa</code>
Ламбда	λ , Λ	<code>\lambda</code> , <code>\Lambda</code>
Ми	μ , M	<code>\mu</code> , <code>\Mu</code>
Ни	ν , N	<code>\nu</code> , <code>\Nu</code>
Кси	ξ , Ξ	<code>\xi</code> , <code>\Xi</code>
Омикрон	\omicron , O	<code>\omicron</code> , <code>\Omicron</code>
Пи	π , ϖ , Π	<code>\pi</code> , <code>\varpi</code> , <code>\Pi</code>
Ро	ρ , ϱ , P	<code>\rho</code> , <code>\varrho</code> , <code>\Rho</code>
Сигма	σ , ς , Σ	<code>\sigma</code> , <code>\varsigma</code> , <code>\Sigma</code>
Тау	τ , T	<code>\tau</code> , <code>\Tau</code>
Ипсилон	υ , Υ	<code>\upsilon</code> , <code>\Upsilon</code>
Фи	ϕ , φ , Φ	<code>\phi</code> , <code>\varphi</code> , <code>\Phi</code>
Хи	χ , X	<code>\chi</code> , <code>\Chi</code>
Пси	ψ , Ψ	<code>\psi</code> , <code>\Psi</code>
Омега	ω , Ω	<code>\omega</code> , <code>\Omega</code>

Табела 10.4 Грчки алфабет

Запазите како курент верзије неких карактера (епсилон, капа, тета, пи, ро, сигма и фи) имају две могуће варијанте.

10.1.4 Разни симболи

Уз карактере које смо управо видели, TeX (па дакле и ConTeXt) обезбеђује команде за генерисање било ког броја симбола. Постоји много таквих команди. У [додатку Б](#) представљам проширену листу која ипак није комплетна.

10.1.5 Дефинисање карактера

Ако је потребно да користимо било који карактер којем не може да се приступи са наше тастатуре, увек можемо да пронађемо веб страницу са тим карактерима и да их копирамо у наш изворни фајл. Ако користимо UTF-8 кодирање (као што се и препоручује) то ће скоро увек да функционише. А на ConTeXt викију такође постоји страница пуна симбола који просто могу да се копирају и налепе у наш документ. Да бисте дошли до ње, кликните [на следећи линк](#).

Међутим, ако неки од ових карактера треба да користимо више пута, онда копирање/налепљивање баш и није ефикасан начин. Било би добро да се дефинише карактер који се придружи команди, а она ће га генерисати сваки пут када нам је потребан. То се ради командом `\definecharacter` чија је синтакса:

```
\definecharacter Име Карактер
```

где

- **Име** представља име које се придружује новом карактеру. То не би требало да буде име неке постојеће команде, јер би је тако преписали.
- **Карактер** представља карактер који се генерише сваки пут када извршимо `\Име`. Постоји три начина да наведемо овај карактер:
 - Просто га напишемо или га налепимо у изворни фајл (ако смо га копирали из неког другог електронског документа или веб странице).
 - Тако што наведемо број придружен том карактеру у фонту који тренутно користимо. Да бисте видели карактере који постоје у фонту, као и бројеве који су им придружени, употребите команду `\showfont[Име фонта]`.
 - Тако што изградимо нови карактер једном од команди за изградњу композитних карактера које ћемо ускоро упознати.

Као пример првог начина употребе, вратимо се на тренутак на одељке који се баве лигатурама (10.1.2). Тамо сам говорио о традиционалним лигатурама у шпанском језику које у данашњим фонтовима углавном не можемо да пронађемо: 'Ð'. Овај карактер би могли да придружимо, на пример, команди `\decontract` тако да се карактер генерише кадгод напишемо `\decontract`. То можемо да урадимо са:

```
\definecharacter decontract Ð
```

Ако пожелимо да изградимо карактер који се не налази у нашем фонту и не може да се добије нашом тастатуром, као што је случај у примеру који сам управо навео, најпре морамо да пронађемо неки текст у којем постоји тај карактер, копирамо га и налепимо у нашу дефиницију. У датом примеру, 'Ð' сам копирао са Википедије.

ConTeXt такође обезбеђује и неке команде које нам омогућавају да креирамо композитне карактере и које могу да се користе у комбинацији са `\definecharacter`. Под појмом композитни карактери подразумевам карактере који имају дијакритике. Ево тих команди:

```
\buildmathaccent Акцент Карактер
```



```

\buildtextaccent Акцент Карактер
\buildtextbottomcomma Карактер
\buildtextbottomdot Карактер
\buildtextcedilla Карактер
\buildtextgrave Карактер
\buildtextmacron Карактер
\buildtextogonek Карактер

```

На пример: као што већ знамо, ConTeXt подразумевано нуди команде за писање само одређених слова са седиј дијакритиком (с, k, l, n, r, s и t), који се обично имплементирају у фонтовима. Ако желимо да користимо 'b' могли би да употребимо команду `\buildtextcedilla` на следећи начин:

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

Ова команда ће да креира нову `\bcedilla` команду која ће да генерише слово 'b' са седиј: 'ḅ'. Ове команде буквално „изграђују” нови карактер који ће се генерисати чак и ако га наш фонт нема. Оно што ове команде раде је да преклопе један карактер преко другог, па затим том резултату дају име.

У мојим тестовима команде `\buildmathaccent` или `\buildtextogonek` нису прорадиле. Тако да их надаље нећу помињати.

`\buildtextaccent` као аргумент узима два карактера и преклапа један преко другог, мало издижући један од њих. Мада се назива „`buildtextaccent`”, није обавезно да било који од карактера који се проследе као аргументи буде акценат; али резултат преклапања ће бити бољи ако је тако, јер у том случају је мање вероватно да ће преклапање акцента преко карактера преписати део карактера. С друге стране, на преклапање два карактера који под уобичајеним околностима имају исту основну линију утиче чињеница да команда мало издиже један од карактера изнад другог. То је разлог што, на пример, ову команду не можемо да употребимо за добијање сажимања 'Ð' које је поменуто изнад, јер ако у нашем изворном фајлу напишемо

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

благо померање изнад основне линије слова 'D' које прави ова команда значи да ефекат који се добија командом („Ð”) није баш како треба. Али ако висина карактера то дозвољава, могли би да креирамо комбинацију. На пример

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

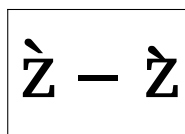
би дефинисало "̣" карактер који би био придружен команди `\unusual`.

Остале `build` команде прихватају један аргумент – карактер којем ће се додати дијакритик генерисан сваком командом. Испод ћу приказати пример сваке од њих, изграђен на слову 'z':

- `\buildtextbottomcomma` додаје запету испод карактера који узима као аргумент ('z').
- `\buildtextbottomdot` додаје тачку испод карактера који узима као аргумент ('z').
- `\buildtextcedilla` додаје седиј испод карактера који узима као аргумент ('z').

- `\buildtextgrave` додаје гравис акценат изнад карактера који узима као аргумент (`'ž`).
- `\buildtextmacron` додаје малу црту испод карактера који узима као аргумент (`'z̄`).

На први поглед, изгледа да је `\buildtextgrave` сувишно јер имамо `\buildtextaccent`; међутим, ако проверите гравис акценат који се генерише првом од ове две команде, он изгледа мало боље. Следећи пример приказује резултат обе команде, на величини фонта која је довољна да се лако види разлика:



10.1.6 Употреба предефинисаних скупова симбола

„ConTeXt Standalone” уз сам ConTeXt доноси и већи број предефинисаних скупова симбола које можемо да користимо у својим документима. Ови скупови се називају „cc”, „cow”, „fontawesome”, „jmn”, „mvs” и „nav”. Сваки од њих има и неке подскупове:

- `cc` поседује „cc”.
- `cow` поседује „cownormal” и „cowcontour”.
- `fontawesome` поседује „fontawesome”.
- `jmn` поседује „navigation 1”, „navigation 2”, „navigation 3” и „navigation 4”.
- `mvs` поседује „astronomic”, „zodiac”, „europe”, „martinvogel 1”, „martinvogel 2” и „martinvogel 3”.
- `nav` поседује „navigation 1”, „navigation 2” и „navigation 3”.

Вики такође помиње и скуп под називом `was` који поседује „wasy general”, „wasy music”, „wasy astronomy”, „wasy astrology”, „wasy geometry”, „wasy physics” и „wasy apl”. Али ја у својој дистрибуцији нисам успео да их пронађем, и сви моји покушаји да их добијем нису успели.

Да би се видело који симболи се налазе у сваком од ових скупова, користи се следећа синтакса:

```
\usesymbols[Скуп]
\showsymbolset[Подскуп]
```

На пример: ако желимо да видимо симболе који постоје у „mvs/zodiac”, онда у изворном фајлу треба да напишемо:

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

и добићемо следећи резултат:

Aquarius	♈	♈
Aries	♈	♈
Cancer	♋	♋

Capricorn	♐	
Gemini	♊	
Leo	♌	
Libra	♎	
Pisces	♓	
Sagittarius	♐	
Scorpio	♏	
Taurus	♉	
Virgo	♍	

Запазите да се уз симбол наводи и његово име. Команда `\symbol` нам омогућава да употребимо било који од симбола. Њена синтакса је следећа:

```
\symbol[Подскуп][ИмеСимбола]
```

где је *Подскуи* један од подскупова придружен било ком од скупова које смо претходно учили командом `\usesymbols`. На пример, ако желимо да употребимо астролошки симбол који је придружен Водолији (налази се у `mvs/zodiac`) требало би да напишемо

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

што ће нам дати „♈“, и то ће се за све намене и сврхе третирати као „карактер“, па на њега утиче активна величина фонта када се штампа. Можемо такође да употребимо и `\definecharacter` да команди придружимо жељени симбол. На пример

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

ће да креира нову команду под именом `\Aries` која ће да генерише карактер „♈“.

Ове симболе би такође могли да користимо, на пример, у окружењу набрајања. На пример:

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

ће да произведе

```
☞ ставка
☞ ставка
☑ ставка
☑ ставка
☞ ставка
```

10.2 Специјално форматирање карактера

У ужем смислу, команде *форматирања* утичу на фонт који се користи, његову величину, стил или варијанту. Ове команде су објашњене у [поглављу 6](#). Међутим, ако се посматра *уопштено*, за команде форматирања можемо да сматрамо и оне које на неки начин мењају карактере који им се проследе као аргумент (дакле оне које им мењају изглед). У овом одељку ћемо се упознати са неким од тих команди. Остале, као што су подвучени или текст са линијом изнад или испод текста (нпр. где желимо да обезбедимо простор за одговор на питање) ћемо видети у [одељку 12.5](#).

10.2.1 Верзал, курент и лажни капитал

Сама слова могу бити у верзалу или у куренту. За ConTeXt, верзал и курент слова су различити карактери, тако да ће у принципу словослагати слова као што су написана. Међутим, постоји група команди које нам омогућавају да обезбедимо да се текст који им се проследи као аргумент увек пише у верзалу или у куренту:

- `\word{текст}`: конвертује текст који узима као аргумент у курент.
- `\Word{текст}`: конвертује прво слово текста који узима као аргумент у верзал.
- `\Words{текст}`: конвертује прво слово сваке речи које узима као аргумент у верзал; остатак слова је у куренту.
- `\WORD{текст}` или `\WORDS{текст}`: пише текст који узима као аргумент у верзалу.

`\cap` и `\Cap` су врло сличне овим командама: оне текст који узимају као аргумент такође претварају у верзал, али на њега примењују и фактор скалирања, исти као онај који се примењује 'x' суфиксом у командама промене фонта (погледајте [одељак 6.4.2](#)) тако да ће, у већини фонтова, верзал бити исте висине као и курент, чиме добијамо једну врсту ефекта *лажног калшијала*. У поређењу са правим капиталом (погледајте [одељак 6.5.2](#)) ово има следеће предности:

1. `\cap` и `\Cap` ће радити са било којим фонтом, за разлику од правог капитала који функционише само са фонтовима и стиловима који га заиста имају.
2. С друге стране, прави капитал је варијанта фонта, која као таква није компатибилна са било којом другом варијантом као што је црни слог, курзив или коси слог. Међутим, `\cap` и `\Cap` су потпуно компатибилне са било којом варијантом фонта.

Разлика између `\cap` и `\Cap` је у томе што ова прва примењује фактор скалирања на сва слова речи које чине њен аргумент, док `\Cap` уопште не примењује скалирање на прво слово сваке речи, чиме се добија ефекат сличан оном који се добија када се у тексту писаном капиталом употреби верзал. Ако се текст који се прослеђује команди 'caps' састоји од неколико речи, одржаваће се величина верзал слова у првом слову сваке речи.

Па тако, у следећем примеру

УН, чији \Cap{председник} има своју канцеларију у седишту \cap{уН} ...

УН, чији ПРЕДСЕДНИК има своју канцеларију у седишту уН ...

најпре треба да запазимо разлику у величини између првог пута када смо написали „УН” (у верзалу) и другог пута (у капиталу, „УН”). У примеру сам други пут написао \cap{уН} тако да можемо видети да нема разлике ако је аргумент који узима команда \cap у верзалу или куренту: команда конвертује сва слова у верзал, па онда примењује фактор скалирања; за разлику од команде \Cap која не скалира прво слово.

Ове команде могу и да се у^инезде, па се у том случају фактор скалирања примењује још једном, чиме се величина додатно смањује, као што се види у наредном примеру где се реч „капитал” у првој линији још једном скалира:

\cap{Људи који су нагомилили свој
\cap{капитал} на штету других су у
револуциона времена чешће
{\bf обезглављивани} него што нису}.

ЉУДИ КОЈИ СУ НАГОМИЛАЛИ СВОЈ КАПИТАЛ НА ШТЕТУ ДРУГИХ СУ У РЕВОЛУЦИОНАРНА ВРЕМЕНА ЧЕШЋЕ ОБЕЗГЛАВЉИВАНИ НЕГО ШТО НИСУ.

Команда \noscap примењена на текст над којим је примењена команда \cap поништава \cap ефекат у тексту који је њен аргумент. На пример:

\cap{When I was One I had just begun,
when I was Two I was \noscap{nearly}
new (A.A. Milne)}.

WHEN I WAS ONE I HAD JUST BEGUN, WHEN I WAS TWO I WAS nearly NEW (A.A. MILNE).

Начин на који команда \cap функционише можемо да конфигуришемо помоћу команде \setupcapitals, а можемо и да дефинишемо различите верзије команде, тако да свака има своје сопствено име и одређену конфигурацију. Ово можемо да урадимо помоћу \definecapitals.

Обе команде раде на сличан начин:

\definecapitals[Име][Конфигурација]
\setupcapitals[Име][Конфигурација]

Параметар „Име” у команди \setupcapitals није обавезан. Ако се не употреби, конфигурација утиче на саму команду \cap. Ако се употреби, морамо да наведемо име које је претходно командом \definecapitals додељено некој одређеној конфигурацији.

У ове команде, конфигурација омогућава подешавање три опције: „title”, „sc” и „style”, при чему прва и друга прихватају за вредност „yes” и „no”. Са „title” наводимо да ли ће капитализација да утиче и на наслове (што је подразумевано), а са „sc” наводимо да ли команда треба да буде прави капитал („yes”), или лажни капитал („no”). Подразумевано користи лажни капитал, који има предност да команда ради чак и када употребљавате фонт који нема имплементирану капитал варијанту. Трећа вредност „style” нам омогућава да наведемо команду стила која ће се применити на текст на који утиче команда \cap.

10.2.2 Текст у експоненту или индексу

Већ знамо (погледајте одељак 3.1) да ће у математичком режиму резервисани карактери „_” и „^” конвертовати карактер или групу која непосредно следи у експонент или индекс. Да би се тај ефекат постигао и ван математичког режима, ConTeXt обезбеђује следеће команде:

- `\high{Текст}`: текст који узима као аргумент исписује у експоненту.
- `\low{Текст}`: текст који узима као аргумент исписује у индексу.
- `\lohi{Индекс}{Експонент}`: исписује оба аргумента, један изнад другог: први аргумент на дну, а изнад њега други аргумент, чиме се постиже занимљив ефекат:

```
\lohi{испод}{изнад} | изнад
                     | испод
```

10.2.3 Дословни текст

Латински израз *verbatim* (од *verbum* = *реч* + суфикс *atim*), који може да се преведе као „дословно” или „реч за реч”, се користи у програмима за обраду текста као што је ConTeXt да се означи да фрагменти текста не би уопште требало да се обраде, већ да се у финални фајл препишу онако како су написани. ConTeXt за ово употребљава команду `\type`. Она је намењена за кратке текстове који не заузимају више од једне линије, а окружење `typing` је намењено за текстове који су дужи од једне линије. Ове команде се често користе у компјутерским књигама за приказивање фрагмената кода, и ConTeXt ове текстове форматира словима фиксне ширине, као што би изгледао откуцан писаћом машином или на екрану компјутерског терминала. У оба случаја се текст шаље у финални документ без *обраде*, што значи да у њима смеју да се нађу резервисани карактери или специјални карактери који ће се у финални фајл пренети *какви јесу*. Исто тако, ако аргумент команде `\type`, или садржај окружења `\starttyping` садржи команду, она ће се *исписати* у финални документ, неће се извршити.

Комада `\type` има и следећу особеност: њен аргумент *може* да се налази унутар витичастих заграда (као што је уобичајено у систему ConTeXt), али за огорађивање (окруживање) аргумента може да се користи и било који други карактер.

Када ConTeXt чита `\type` команду, он претпоставља да карактер који није размак непосредно иза имена команде служи као граничник њеног аргумента; тако да сматра да садржај аргумента почиње са наредним карактером, а завршава се са карактером испред наредног појављивања *граничника*.

Неки примери ће нам помоћи да ово лакше разумемо:

```
\type 1Станлио о Олио1
\type |Станлио и Олио|
\type zСтанлио и Олиоz
\type (Станлио и Олио(
```

Запазите да је у првом примеру први карактер након имена команде '1', у другом '|', а у трећем 'z'; дакле: у сваком од ових случајева, ConTeXt ће сматрати да је аргумент команде `\type` све до наредног појављивања тог истог карактера. Исто важи и за последњи пример, који је врло поучан, јер у принципу бисмо могли да претпоставимо да ако је отварајући граничник аргумента '(', онда затварајући треба да буде ')', али то није тако, јер су '(' и ')' различити карактери, а команда `\type`, као што сам рекао, тражи карактер затварајућег граничника који је исти као и претходни карактер који се употребио да започне аргумент.

Постоје само два случајева у којима команда `\type` дозвољава да отварајући и затварајући граничник буду различити карактери:

- Ако је отварајући граничник карактер '{', он сматра да ће затварајући граничник бити '}'.
- Ако је отварајући граничник '<<', он сматра да ће затварајући граничник да буде '>>'. Овај случај је јединствен и у томе што се два узастопна карактера користе као граничници.

Међутим: чињеница да команда `\type` омогућава било који граничник не значи да би требало да користимо „чудне” граничнике. Ако се посматра из угла *читљивости* и *разумљивости* изворног фајла, најбоље је да се аргумент команде `\type` огради витичастим заградама где је то могуће, као што је и уобичајено у систему ConTeXt; а када то није могуће јер се витичасте заграде налазе у аргументу команде `\type`, употребите симбол: по могућству онај који није један од ConTeXt резервисаних карактера. На пример: `\type *Ово је затварајућа витичаста заграда: '{'*`.

И `\type` и `\starttyping` могу да се конфигуришу командама `\setuptype` и `\setuptyping`. Такође можемо да креирамо прилагођене верзије ових команди са `\definetype` и `\definetyping`. Што се тиче актуелних опција конфигурације за ове команде, упућујем на „setup-en.pdf” (у директоријуму `tex/texmf-context/doc/context/documents/general/qrcs`).

Две врло сличне команде команди `\type` су:

- `\typ`: функционише слични као и `\type`, али не искључује поделу речи на крају редова.
- `\tex`: команда која је намењена писању текстова о TeX или ConTeXt: она додаје обрнуту косу црту испред текста који узима као аргумент. Иначе, ова команда се разликује од команде `\type` у томе да обрађује неке од резервисаних карактера на које наиђе у тексту који јој се проследи као аргумент. Тачније, витичасте заграде унутар `\tex` ће се третирали на исти начин на који се и иначе третирају у систему ConTeXt.

10.3 Размак између карактера и речи

10.3.1 Аутоматско постављање хоризонталног размака

ConTeXt размак између различитих карактера и речи (који се у TeX назива *хоризонтални размак*) обично поставља аутоматски:

- Размак између карактера који чине реч дефинише сам фонт, што, осим у случају фонтова фиксне ширине, обично значи већу или мању количину празног простора у зависности од тога који карактери се раздвајају, па је тако на пример, размак између 'Л' и 'А' ('ЛА') обично мањи од размака између 'П' и 'О' ('ПО'). Међутим, изузев ових могућих варијација у зависности од комбинације слова у питању и које су предефинисане у фонту, размак између карактера који чине реч је, у општем случају, фиксна и непроменљива мера.
- За разлику од тога, размак између речи на истој линији може бити еластичнији.
 - У случају речи у линији чија ширина мора бити иста као ширина осталих линија у пасусу, један од механизма које ConTeXt користи да добије линије исте ширине је варијација размака између речи, што је детаљније објашњено у одељку 11.3. У

овим случајевима, ConTeXt ће да успостави потпуно исти хоризонтални размак између свих речи у линији (осим за правила дата испод), и обезбедиће да је размак између речи у различитим линијама пасуса што је могуће сличнији.

- Међутим, уз потребу да се простор између речи развуче или скупи тако да се линије поравнају, у зависности од активног језика ConTeXt узима у обзир и одређена типографска правила, па се тако на пример у неким деловима енглеске типографске традиције након тачке на крају реченице уметне додатни размак.

Овај додатни размак функционише за енглески и можда још неке језике (мада је исто тако тачно да данас у многим случајевима издавачи на енглеском језику не уметну додатни размак након тачке), али не и за шпански или српски за које је типографска традиција другачија. Тако да ову функцију можемо привремено да укључимо помоћу `\setupspacing[broad]` и да је искључимо са `\setupspacing[packed]`. Такође бисмо могли и да променимо подразумевану конфигурацију за шпански (а и за било који други језик, укључујући и српски), као што је објашњено у одељку 10.5.2.

10.3.2 Промена размака између карактера у речи

Измена подразумеваног размака између карактера који чине реч се из угла типографије сматра за врло лошу праксу, осим у насловима. ConTeXt ипак обезбеђује команду за промену овог размака између карактера у речи:¹ `\stretched`, чија је синтакса:

`\stretched[Конфигурација]{Текст}`

где Конфигурација омогућава било коју од следећих опција:

- **factor:** цео или децимални број који представља размак који треба да се постигне. Не би требало да буде сувише велики број. Фактор од 0.05 је већ видљив голим оком.
- **width:** наводи укупну ширину коју мора да има текст достављен команди, тако да ће сама команда да израчуна неопходно размицање тако да распореди карактере у тај простор.

Према мојим тестовима, када је ширина постављена опцијом `width` мања од оне која је неопходна да се текст представи уз *factor* од 0.25, опција `width` и овај фактор се игноришу. Претпостављам да је то зато што нам команда `\stretched` омогућава само да *увећамо* размак између карактера у речи, а не и да га смањимо. Али не разумем зашто је ширина потребна да се представи текст уз фактор од 0.25 узета као минимална мера за опцију `width`, а не *природна ширина* текста (са фактором 0).

- **style:** команда или команде стила које се примењују на текст који се узима као аргумент.
- **color:** боја којом ће се исписати текст узет као аргумент.

Тако у наредном примеру можемо графички видети како би команда радила када би се применила на исту реченицу, али са различитим ширинама:

¹ За филозофију система ConTeXt је врло типично да се обезбеди команда која ради нешто што сама ConTeXt документација саветује да се не ради. Мада се иде за типографском перфекцијом, циљ је и да се аутору омогући апсолутна контрола над изгледом свог документа: на аутору је одговорност да ли је то боље или лошије.


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

t	e	s	t		t	e	x	t									
t		e		s		t			t		e	x	t				
t			e		s			t			t		e	x	t		
t				e			s			t				t	e	x	t

У овом примеру се може видети да расподела хоризонталног размака између различитих карактера није униформна. 'x' и 't' у „text”, а 'e' и 's' у „test”, се увек појављују ближе један уз други него остали карактери. Нисам успео да пронађем разлог зашто се то дешава.

Ако се примени без аргумената, команда ће употребити пуну ширину линије. С друге стране, унутар текста који представља аргумент ове команде, команда `\` се рedefинише, па уместо прелома линије умеће хоризонтални размак. На пример:

```
\stretched{test\text}           t e s t                               t e x t
```

Подразумевану конфигурацију команде можемо да променимо командом `\setupstretched`.



Не постоји команда `\definestretched` која би нам омогућила да поставимо прилагођене конфигурације придружене имену команде, међутим, у званичној листи команди (погледајте одељак 3.6) пише да `\setupstretched` долази од `\setupcharacterkerning` и постоји команда `\definecharacterkerning`. Ипак, у мојим тестовима овом последњом командом нисам успео да поставим било какву прилагођену конфигурацију за команду `\stretched`, мада морам признати да баш и нисам уложио много времена у то.

10.3.3 Команде за додавање хоризонталног размака између речи

Већ знамо да ако желимо да повећамо размак између речи, то нећемо постићи додавањем два или више узастопна размака, јер ConTeXt апсорбује све узастопне размаке, као што је објашњено у одељку 4.2.1. Ако желимо већи размак између речи, морамо да употребимо неку од команди које нам то омогућавају:

- `\,` умеће у документ врло мали размак (који се назива танки размак). Користи се, на пример, за раздвајање хиљада у бројевима (нпр. 1.000.000), или за раздвајање обрнутог апострофа од обрнутих навода. На пример: „`1\,473\,451`” производи „1 473 451”.
- `\space` или „`\`” (обрнута коса црта иза које следи размак, који сам представио са „`\`”, пошто је невидљив) уноси додатни размак.
- `\enskip`, `\quad` и `\qqquad` умећу у документ редом размак ширине пола ема, 1 ем или 2 ема. Присетите се да је ем мера која зависи од величине фонта и износи ширину слова 'm', која се обично подудара са величином фонта у тачкама. Дакле, ако се користи фонт величине 12 тачака, `\enskip` уноси размак од 6 тачака, `\quad` размак од 12 тачака, а `\qqquad` размак од 24 тачака.

Уз ове команде које уносе размаке прецизне димензије, команде `\hskip` и `\hfill` уносе хоризонтални размак променљиве ширине:

`\hskip` нам омогућава да наведемо тачно колико празног простора желимо да додамо. Овако:

```
Oво је \hskip 1cm 1 центиметар\\
Oво је \hskip 2cm 2 центиметра\\
Oво је \hskip 2.5cm 2,5 центиметра\\
```

```
Oво је      1 центиметар
Oво је      2 центиметра
Oво је      2,5 центиметра
```

Наведени размак може да буде и негативан, па се у том случају текст пише преко другог. Овако:

```
Oво је пре фарса него
\hskip -1cm комедија
```

```
Oво је пре фарсе комедија
```

`\hfill`, са друге стране, уноси онолико празног простора колико је потребно да се заузме комплетна линија, па на тај начин можемо да постигнемо ефекат десно поравнатог текста, центрираног текста или текста на обе стране линије, као што је приказано у наредном примеру:

```
\hfill На десној страни\\
На обе\hfill стране
```

```
На десној страни
На обе
странице
```

10.4 Сложенице

Под „сложеницама” у овом одељку подразумевам речи које се формално схватају као једна реч, а не просто спојене две речи. Није увек једноставно схватити ту разлику: „радиоактиван” је јасно састављено од две речи („радио + активан”) али нико ко говори српски неће помислити о сложеници на било који други начин осим као на једну реч. С друге стране, имамо речи које се понекад спајају уз помоћ цртице (полусложенице). Те две речи имају одвојена значења и употребе, али су спојене (а у неким случајевима могу да постану и једна реч, мада не још увек!). Тако на пример, можемо да пронађемо речи као што су „француско–канадски” или „радио–техника”.

Сложенице пред `ConTeXt` постављају неке проблеме који се углавном тичу њихове потенцијалне поделе на крају реда. Ако је цртица елемент који их спаја, онда из типографске перспективе нема проблема поделе на крају реда, али би требало да избегнемо поделу у другом делу речи јер на тај начин две узастопне цртице отежавају разумљивост.

Постоји команда „`|`” која систему `ConTeXt` говори да две речи чине полусложеницу. Ова команда не почиње са обрнутом косом цртом и дозвољава два различита начина коришћења:

- Можемо да употребимо две узастопне вертикалне црте и да напишемо, на пример, „Горе| | доле”.
- Између две вертикалне црте може да се налази симбол спајања/раздвајања, као на пример, у „спајање|/| раздвајање”.

У оба случаја `ConTeXt` ће знати да има посла са сложеницом, па ће применити одговарајућа правила растављања на крају реда за тај тип речи. Разлика између употребе две узастопне вертикалне линије и окруживања граничника речи њима је што у првом случају `ConTeXt`

користи граничник који је предефинисан као `\setuphyphenmark`, или другим речима цртицу, што је подразумевано „-”). Дакле, ако напишемо „слика | | рам”, `ConTeXt` ће генерисати „слика-рам”.

Командом `\setuphyphenmark` можемо да променимо подразумевани граничник (у случајевима када су нам потребне две вертикалне црте). Вредности које су дозвољене за ову команду су „-”, „—”, „-”, „-”, „(”, „)”, „=”, „/”. Међутим, имајте на уму да вредност „=” постаје ем црта (исто што и „-”).

Уобичајена употреба „| |” је са цртицама, јер се она обично користи у полусложеницама. Али понекада граничник може да буде заграда, ако желимо на пример, „(међу)простор”, или може бити коса црта, као у „улаз/излаз”. Ако у овим случајевима желимо да се примењују уобичајена правила поделе речи на крају реда за сложенице, могли би да напишемо „(међу|) | простор” или „улаз | / | излаз”. Као што сам раније рекао, сматра се да је „|=|” скраћеница за „|-|” и као граничник умеће ем црту (—).

10.5 Језик текста

Карактери који чине речи обично припадају неком језику. За `ConTeXt` је важно да зна на ком језику пишемо текст, јер од тога зависи много битних детаља. Неке од њих су:

- Растављање речи на крају реда.
- Излазни формат неких речи.
- Одређена словослагачка питања везана за типографску традицију језика о којем је реч.

10.5.1 Постављање и измена језика

`ConTeXt` претпоставља да је језик енглески. Можемо да га променимо једном од следеће две процедуре:

- Користећи команду `\mainlanguage` у преамбули да поставимо главни језик документа.
- Користећи команду `\language` намењену за промену активног језика на неком месту у документу.

Обе команде очекују аргумент који се састоји од неког идентификатора језика (или кода). За идентификацију језика може да се користи или међународни двословни код језика наведен у ISO 639-1, а то је исто што се употребљава, на пример, на вебу, или енглеско име језика о ком је реч.

Табела 10.5 приказује комплетну листу језика које подржава `ConTeXt`, заједно са ISO кодом за сваки језик, а тамо где постоје, и кодове одређених варијанти које могу експлицитно да се наведу.¹

¹ Табела 10.5 приказује кратак преглед листе која се добија следећим командама:

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

Ако овај документ читате доста касније од времена када је написан (2020), могуће је да `ConTeXt` садржи и додатне језике, тако да би била добра идеја да покренете ове команде и видите ажурирану листу језика.
Током 2020. је додата подршка за српски језик — *прим. прев.*

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

Табела 10.5 Језичка подршка у систему ConTeXt

Тако на пример, ако желимо да као главни језик документа поставимо шпански (кастиљански), могли би да употребимо било коју од следеће три команде:

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

Ако одређени језик желимо да укључимо *унушар* документа, можемо да употребимо или команду `\language[Код језика]`, или специфичну команду која активира тај језик. Тако на пример, `\en` активира енглески језик, `\fr` активира француски, `\es` шпански, или `\ca` каталонски. Једном када се неки језик активира, он остаје активан све док експлицитно не променимо језик, или док се не затвори група у којој је дошло до промене језика. Тако да језици функционишу као и команде за промену фонтова. Међутим, запазите да језик који се постави командом `\language` или неком од њених скраћеница (`\en`, `\fr`, `\de`, итд.) не утиче на језик којим се штампају ознаке (погледајте [одељак 10.5.3](#)).

Мада може бити заморно да се обележи језик свих речи и израза које користимо у свом документу, а које нису на главном језику документа, веома је важно да се то уради ако желимо да на крају имамо документ који је правилно сложен, посебно ако је у питању било какав професионални рад. Не би требало да обележимо комплетан текст, већ само онај који није на главном језику. Обележавање језика понекада може да се аутоматизује

употребом макроа. На пример, оригинални језик овог документа у којем се често цитирају ConTeXt команде је енглески, па сам направио макро који, осим што команде исписује у одређеном формату и боји, обележава их као енглеску реч. У свом професионалном раду морам често да цитирам много француске и италијанске библиографије, тако да сам у своју библиографску базу података унео поље које чува језик рада, тако да у цитирањима и листама библиографских референци могу да аутоматизујем навођење језика.

Ако у истом документу користимо два језика који користе различите алфабете (на пример енглески и српски, или енглески и грчки), постоји трик којим се избегава потреба да се обележава језик израза који се исписују различитим алфабетом: изменом подешавања главног језика (погледајте наредни одељак) тако да се читају и подразумевани шаблони за поделу речи на крају реда за језик који користи другачији алфабет. На пример, ако желимо да користимо енглески и старогрчки, следећом командом неће бити потребе да ручно означавамо језик текста на грчком:

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

Ово функционише само зато јер енглески и грчки не користе исти алфабет, тако да не може доћи до конфликта у шаблонима за растављање речи на крају реда та два језика, па одједном можемо да учитамо оба. Али за два језика који користе исти алфабет, истовремено учитавање шаблона за поделу речи на крају реда ће несумњиво да доведе до неправилног раздвајања речи на крају реда.

10.5.2 Конфигурисање језика

ConTeXt функционисање одређених алата придружује одређеном језику који је активан у датом тренутку. Подразумевано придруживање може да се промени командом `\setuplanguage` чија је синтакса:

```
\setuplanguage[Језик][Конфигурација]
```

где је *Језик* код језика који желимо да конфигуришемо, а *Конфигурација* садржи одређену конфигурацију коју желимо да поставимо (или да изменимо) за тај језик. Тачније, дозвољено је до 32 различите опције конфигурације, али ја ћу се бавити само са онима које изгледају пригодно за уводни текст као то је овај:

- **date:** омогућава нам да конфигуришемо подразумевани формат датума. Погледајте даље на [страници 175](#).
- **lefthyphenmin, righthyphenmin:** минимални број карактера који морају бити са леве или десне стране да би се извршила подела речи на крају реда. На пример `\setuplanguage[en][lefthyphenmin=4]` неће делити било коју реч која лево од потенцијалне цртице поделе има мање од 4 карактера.
- **spacing:** могуће вредности ове опције су „broad” или „packed”. У првом случају (broad), примениће се правила раздвајања речи у енглеском језику, што значи да ће се након тачке на крају реченице иза које следи неки други карактер додати одређена количина додатног размака. С друге стране, „spacing=packed” ће спречити примену ових правила. За енглески језик, вредност broad је подразумевана.
- **leftquote, rightquote:** наводе карактере (или команде), редом, које ће команда `\quote` користити лево и десно од текста који је њен аргумент (у вези са овом командом, погледајте [страницу 176](#)).
- **leftquotation, rightquotation:** наводе карактере (или команде), редом, које ће команда `\quotation` користити лево и десно од текста који је њен аргумент (у вези са овом командом, погледајте [страницу 176](#)).

10.5.3 Ознаке које се придружују одређеним језицима

Многе од ConTeXt команди аутоматски генеришу одређене текстове (или *ознаке*), као на пример, команда `\placetable` која исписује ознаку „Табела xx” испод табеле коју умеће, или `\placefigure` која умеће ознаку „Слика xx”.

Ове *ознаке* зависе од језика подешеног са `\mainlanguage` (али не ако је језик постављен са `\language`) и можемо да их променимо са

```
\setuplabeltext[Језик][Кључ=Ознака]
```

где је *Кључ* израз под којим ConTeXt познаје ознаку, а *Ознака* је текст који желимо да ConTeXt генерише. Тако на пример,

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

би подесило да када је главни језик шпански, слике које се умећу командом `\placefigure` не називају „Figure x”, већ „Imagen x”. Запазите да након текста саме ознаке мора да се остави размак, како би се обезбедило да се ознака не прикачи за наредни карактер. У примеру сам употребио резервисани карактер „~”; такође сам могао да напишем и „[figure=Imagen{ }]” постављајући размак унутар витичастих заграда тако да га ConTeXt не уклони.

Које ознаке можемо да редифинишемо командом `\setuplabeltext`? ConTeXt документација није комплетна онако како би неко очекивао у овом тренутку. Референтно упутство из 2013. године (које највише објашњава ову команду) помиње „chapter”, „table”, „figure”, „appendix”... и додаје „остале сличне текст елементе”. Можемо претпоставити да ће имена бити енглеска имена елемената о којима се ради.



Једна од предности *слободног либре софтвера* је да су кориснику доступни изворни фајлови; тако да их можемо погледати. Ја сам то учинио, и *њушкајући* кроз изворне фајлове система ConTeXt, открио сам фајл „lang-txt.lua”, који се налази у `tex/texmf-context/tex/context/base/mkiv` и за који мислим да је онај који садржи предефинисане ознаке и њихове различите преводе; тако да ако ConTeXt било када генерише редефинисани текст који желимо да променимо, треба да отворимо овај фајл и да пронађемо име те ознаке. На овај начин можемо видети име које је придружено ознаци.

Ако негде у документ желимо да уметнемо текст придружен одређеној ознаци, то можемо урадити командом `\labeltext`. Тако на пример, ако желим да укажем на табелу, и да обезбедим да је назовем на исти начин на који је назива ConTeXt у команди `\placetable`, могу да напишем: „Као што приказује `\labeltext{table}` на наредној страници.” Овај текст ће у документу чији је `\mainlanguage` српски да произведе: „Као што приказује Табела на наредној страници.”

Неке од ознака које могу да се редифинишу командом `\setuplabeltext` су подразумевано празне; као на пример, „chapter” или „section”. То је зато што ConTeXt подразумевано не додаје ознаке командама поделе. Ако желимо да променимо ово подразумевано понашање, потребно је да у преамбули документа редифинишемо само ове ознаке, па тако на пример, `\setuplabeltext[chapter=Поглавље~]` подешава да испред поглавља стоји реч „Поглавље”.

Конечно, важно је истаћи да, мада у општем случају ConTeXt команде које као аргумент прихватају неколико опција раздвојених запетама, последња опција може да се заврши запетом

и не дешава се ништа лоше. У команди `\setuplabeltext` би то проузроковало грешку приликом компајлирања.

10.5.4 Неке команде које се тичу језика

А. Команде у вези са датумом

ConTeXt поседује три команде везане за датум које свој излаз производе на језику који је активан у време када се изврше. То су:

- `\currentdate`: када се изврши без аргумената у документу чији је главни језик српски, враћа системски датум у формату „Дан. Месец Година”. На пример: „11. септембар 2020”. Али можемо да јој наложимо да користи другачији формат (што би се догодило у САД и неким другом деловима света енглеског говорног подручја који следе свој систем стављања месеца пре дана, па отуда неславни датум 9/11), или да додамо и име дана у недељи (weekday), или само неке елементе датума (day, month, year)

За назначавање другачијег формата датума, „dd” или „day” представља дане, „mm” месеце (као бројеве), „month” месеце написане малим словима, а „MONTH” великим. Што се тиче године, „yy” ће исписати само последње цифре, док ће „year” или „y” да испише све четири. Ако између компоненти желимо и неки граничник, морамо да га експлицитно напишемо. На пример

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

ће када се изврши 9. септембра 2020 да испише „среда 9 септембар”.

- `\date`: ова команда, када се изврши без аргумената, производи потпуно исти резултат као и `\currentdate`, што значи тренутни датум у стандардном формату. Међутим, као аргумент може да јој се наведе неки одређени датум. За ово су потребна два аргумента: првим аргументом наводимо дан („d”), месец („m”) и годину („y”) који одговарају датуму који желимо да наведемо, док другим аргументом (који није обавезан) наводимо формат приказаног датума. На пример, ако желимо да знамо дан у недељи када су се упознали Џон Ленон и Пол Мекартни, што је према Википедији догађај који се догодио 6. јула 1957. године, требало би да напишемо

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

па бисмо сазнали да се тај историјски догађај десио у суботу.

- `\month` као аргумент узима број и враћа име месеца који одговара том броју.

Б. Команда `\translate`

Команда `translate` подржава низ фрази придружених одређеном језику, тако да ће у зависности од језика који је активан у неко време, у финални документ да се уметне једна или друга. У следећем примеру се користи команда `translate` да придружи четири фразе на шпанском и српском, које се чувају у меморијском баферу (што се тиче `buffer` окружења, погледајте [одељак 12.6](#)):

```

\startbuffer
\starttabulate[|*{4}{lw(.25\textwidth)}|]
\NC \translate[es=Su carta de fecha, sr=Ваше писмо датирано]
\NC \translate[es=Su referencia, sr=Ваша референца]
\NC \translate[es=Nuestra referencia, sr=Наша референца]
\NC \translate[es=Fecha, sr=Датум] \NC\NR
\stoptabulate
\stopbuffer

```

па ако *дафер* уметнемо на место у документу на којем је активан шпански језик, исписаће се шпанске фразе, али ако је то место у документу на којем је активан српски језик, уметнуће се српске фразе. Дакле:

```

\language[es]
\getbuffer

```

ће да генерише

Su carta de fecha	Su referencia	Nuestra referencia	Fecha
-------------------	---------------	--------------------	-------

док ће

```

\language[sr]
\getbuffer

```

да генерише

Ваше писмо датирано	Ваша референца	Наша референца	Датум
---------------------	----------------	----------------	-------

В. Команде `\quote` и `\quotation`

Једна од најчешћих типографских грешака у текст документима се дешава када се знаци цитирања (полунаводници или наводници) отворе, али се експлицитно не затворе. Да би се то спречило, `ConTeXt` обезбеђује команде `\quote` и `\quotation` које цитирају текст који им се зада као аргумент; `\quote` ће употребити полунаводнике, док ће `\quotation` да употреби знаке навода.

Ове команде зависе од активног језика па користе подразумевани карактер или команду постављену за дати језик којим се отвара или затвара цитат (погледајте [одељак 10.5.2](#)); па тако на пример, ако желимо да користимо шпански као подразумевани стил за знаке навода – угласте заграде) типичне за шпански, италијански, француски, написали бисмо:

```

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

```

Међутим, ове команде не обрађују угњеждене наводе; мада можемо креирати алат који то ради, а који користи чињеницу да су `\quote` и `\quotation` уствари примене онога што `ConTeXt` назива *delimitedtext*, и да командом `\definedelimitedtext` може да се креира још примена. Тако ће наредни пример:

```

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

```



```
[CommasLevelB]
[left=", right="]

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

да креира три команде које ће омогућити до три различита нивоа цитирања. Први ниво са угластим наводницима, други са обичним наводницима, а трећи са полунаводницима.

Наравно, ако енглески користимо као главни језик, онда ће се аутоматски користити полунаводи и наводи (од запета, не равни као у овом документу!).

Глава 11

Пасуси, линије и вертикални размак

Садржај: 11.1 Пасуси и њихове карактеристике; 11.1.1 Аутоматско увлачење прве линије пасуса; 11.1.2 Специјално увлачење пасуса; 11.2 Вертикални размак између пасуса; 11.2.1 `\setupwhite-space`; 11.2.2 Пасуси без додатног међусобног размака; 11.2.3 Уметање додатног вертикалног размака на одређено место у документу; 11.2.4 `\setupblank` и `\defineblank`; 11.2.5 Остале процедуре за добијање више вертикалног размака; 11.3 Како ConTeXt изграђује линије које формирају пасусе; 11.3.1 Употреба резервисаног карактера '~'; 11.3.2 Растављање речи на крају реда; 11.3.3 Ниво толеранције за преломе линија; 11.3.4 Форсирање прелома линије на одређеном месту; 11.4 Проред; 11.5 Остале ствари у вези са линијама; 11.5.1 Конвертовање прелома линија у изворном фајлу у преломе линија у финалном документу; 11.5.2 Нумерисање линија; 11.6 Хоризонтално и вертикално поравнање; 11.6.1 Хоризонтално поравнање; 11.6.2 Вертикално поравнање;

Као што смо видели у поглављу 5, општи изглед документа углавном одређују величина страница и распоред елемената на њима, фонт који смо изабрали, о чему се говори у поглављу 6, као и остале ствари као што су проред, поравнање пасуса и размак између њих, итд. Ово поглавље се бави тим осталим стварима.

11.1 Пасуси и њихове карактеристике

Пасус је основна јединица текста за ConTeXt. Постоје два начина за отварање пасуса:

1. Уметање једне или више узастопних празних линија у изворни фајл.
2. Команде `\par` или `\endgraf`.

Обично се користи први начин јер је једноставнији и јер се тако добијају фајлови који се лакше читају и разумевају. Експлицитно уметање прелома пасуса командом се обично ради у макроима (погледајте одељак 3.7.1) или у ћелијама табела (погледајте одељак 13.3).

У документу који је са типографске тачке гледишта добро сложен, важно је да се пасуси визуелно истичу. То се обично постиже на један од два начина: малим увлачењем прве линије сваког пасуса, или уметањем размака између пасуса, а понекад и комбинацијом оба ова начина, мада се то на неким местима не препоручује јер се сматра удвајањем типографских елемената.

Не слажем се у потпуности. Просто увлачење прве линије прве линије не пружа увек довољно визуелно истцање пасуса; али повећање размака које не прати и увлачење прави проблеме у случају када пасус почиње на врху странице, па нисмо сигурни да ли је у питању почетак новог пасуса, или наставак пасуса са претходне странице. Комбинација оба поступка отклања сваку сумњу.

Хајде да најпре видимо како се у систему ConTeXt постиже увлачење линија и пасуса.

11.1.1 Аутоматско увлачење прве линије пасуса

Аутоматско уметање малог увлачења пре линије пасуса је подразумевано искључено. Командом `\setupindenting` можемо да га укључимо, поново искључимо ако је укључено, као и да назначимо величину увлачења. Ова команда прихвата следеће вредности:

- **always**: сви пасуси се увлаче, увек.
- **yes**: укључује *нормално* увлачење пасуса. Одређени пасуси испред којих се налази додатни вертикални размак, као што су први пасуси одељака, или пасуси који долазе иза одређених окружења се неће увлачити.
- **no, not, never, none**: искључује аутоматско увлачење прве линије пасуса.

У случајевима када смо укључили аутоматско увлачење, истом командом можемо да наведемо и колико то увлачење треба да буде. То можемо да наведемо експлицитном мером (као што је 1.5 cm) или симболичким речима „small”, „medium” и „big” које означавају да желимо мало, средње или велико увлачење.

У неким типографским традицијама (међу њима и шпанској), подразумевано увлачење је било два квада. У типографији, квад (оригинално *квадрат*) је био метални раздвајач који се користио у словослагању на штампарским пресама. Израз је касније усвојен као опште име за две уобичајене величине размака у типографији, без обзира на технику која се користи за штампу. Ем квад је размак ширине једног ема; широк као и висина датог фонта (Википедија). Тако да би за слово величине 12 тачака, квад био ширине 12 тачака и 12 тачака висине. ConTeXt поседује две квад команде: `\quad` која генерише размак наведен малочас, и `\qqquad` која генерише дупло већи размак, али према величини фонта који се тренутно користи. Увлачење од два квада за слово величине 11 тачака ће бити 22 тачке, а са словом величине 12 тачака, 24 тачке.

Када се укључи увлачење, а не желимо да се одређени пасус увуче, треба да употребимо команду `\noindentation`.

У општем случају, ја у својим документима укључујем аутоматско увлачење са `\setupindenting[yes, big]`. Међутим, у овом документу то нисам тако урадио, јер да јесам, велики број кратких реченица и примера би изгледао неуредно на страницама.

11.1.2 Специјално увлачење пасуса

Један од графичких поступака за истицање пасуса је да се увуче било десна, било лева (или обе) страна пасуса. Ово се користи, на пример, за блок цитат.

ConTeXt поседује окружење која нам омогућава да променимо увлачење пасуса тако да истакнемо текст који се налази у њему. То је „narrower” окружење:

```
\startnarrower[Опције] ... \stopnarrower
```

где *Опције* може да буде:

- **left**: увлачење леве маргине.
- **Br*left**: увлачење леве маргине, уз множење *нормално* увлачења са *Br* (на пример $2*left$).
- **right**: увлачење десне маргине.

- **Br*right**: увлачење десне маргине, уз множење *нормалној* увлачења са *Br* (на пример `2*right`).
- **middle**: увлачење обе маргине. Ово је подразумевано.
- **Br*middle**: увлачење обе маргине, уз множење *нормалној* са *Br*.

У објашњењу опција које сам поменуо, *нормално увлачење* се односи на количину левог и десног увлачења које подразумевано умеће „narrower”. Ова *количина* може да се подеси командом `\setupnarrower` која прихвата следеће конфигурационе опције:

- **left**: количина увлачења која треба да се примени на леву маргину.
- **right**: количина увлачења која треба да се примени на десну маргину.
- **middle**: количина увлачења која треба да се примени на обе маргине.
- **before**: команда која треба да се изврши пре уласка у окружење.
- **after**: команда која треба да се изврши након изласка из окружења.

Ако желимо да користимо другачију конфигурацију narrower окружења, свакој од њих можемо да доделимо различито име са `\definennarrower[Име] [Конфигурација]`

где је *Име* име које се повезује са овом конфигурацијом, а *Конфигурација* прихвата исте вредности као и `\setupnarrower`.

11.2 Вертикални размак између пасуса

11.2.1 `\setupwhitespace`

Као што већ знамо из одељка 4.2.2, за ConTeXt није битно колико у изворном фајлу има узастопних празних линија: једна или више њих ће у финални документ да уметне прелом пасуса. Уметањем додатне празне линије у изворни фајл нећемо моћи да повећамо размак између пасуса. Уместо тога, размак између пасуса се контролише командом `\setupwhitespace` која прихвата следеће вредности:

- **none**: значи да се између пасуса неће правити додатни размак.
- **small, medium, big**: умећу редом мали, средњи или велики вертикални размак. Конкретна величина размака који се умеће овим вредностима зависи од величине фонта.
- **line, halfline, quarterline**: мере додатни размак у односу на висину линија и редом умећу додатну линију, пола линије или четвртину линије.
- **ДИМЕНЗИЈА**: поставља конкретну меру размака између пасуса. На пример, `\setupwhitespace[5pt]`.

У општем случају се не саветује употреба конкретне мере за `\setupwhitespace`. Боље је да се користе симболичке вредности `small`, `medium`, `big`, `line`, `halfline` или `quarterline`. За то постоје два разлога:

- Символичке вредности су еластичне мере (погледајте одељак 3.8.2) што значи да поседују нормалну вредност, али се дозвољава и одређено смањење или увећање те вредности, чиме се систему ConTeXt помаже при словослагању страница омогућавањем да преломи пасуса буду естетски слични. Фиксна вредност раздвајања пасуса отежава постизање добре пагинације документа.
- Символичке вредности `small`, `medium`, `big`, итд. се израчунавају на основу величине фонта, тако да ако се у неким деловима документа она промени, промениће се и величина вертикалног размака између пасуса, па ће крајњи резултат увек бити уравнотежен. За разлику од тога, промене величине фонта неће утицати на фиксну вредност вертикалног размака, што ће за последицу да има документ са лоше распоређеним празним простором (у естетском смислу) и који није у складу са типографским правилима.

Када се за вертикални размак постави вредност, доступне су још две додатне команде: `\nowhitespace`, која елиминише било какав додатни размак између одређених пасуса и `\whitespace` која ради супротно. Међутим, ове команде су ретко потребне, јер је чињеница да ConTeXt и сам веома успешно управља вертикалним размаком између пасуса; посебно ако је као вредност унета нека од предефинисаних димензија, која се израчунава из тренутних вредности величине фонта и прореда.



Значење `\nowhitespace` је очигледно. Али не обавезно и саме команде `\whitespace`, јер шта је сврха наредивања вертикалног размака између одређених пасуса ако је већ успостављен генерално за све пасусе? Па, када се пишу напредни макрои, команда `\whitespace` може бити корисна унутар петље која мора донети одлуку према вредности одређеног услова. То је мање више напредно програмирање, и ја овде нећу детаљније улазити у то.

11.2.2 Пасуси без додатног међусобног размака

Ако желимо да одређени делови нашег документа имају пасусе који нису раздвојени додатним размаком, можемо да изменимо општу конфигурацију команде `\setupwhitespace`, али то је, на неки начин, у супротности са ConTeXt филозофијом која каже да би опште конфигурационе команде требало да се постављају искључиво у преамбулу изворног фајла, тако да се постигне конзистентан изглед докумената који лако може да се промени. Зато постоји „packed” окружење, чија је општа синтакса

```
\startpacked[Размак] ... \stoppacked
```

и где је *Размак* необавезни аргумент који наводи количину жељеног вертикалног размака између пасуса у окружењу. Ако се не наведе, неће се примењивати било какав додатни вертикални размак.

11.2.3 Уметање додатног вертикалног размака на одређено место у документу

Ако на одређеном месту у документу није довољан нормални вертикални размак између пасуса, можемо да употребимо команду `\blank`. Када се користи без аргумената, `\blank` ће да уметне количину вертикалног размака подешену са `\setupwhitespace`. Можемо да наведемо или конкретну вредност унутар великих заграда, или неку од симболичких вредности које се израчунавају из величине фонта: `small`, `medium` или `big`. Те вредности можемо

и да помножимо неким целим бројем, па ће тако на пример, `\blank[3*medium]` да уметне размак који је еквивалентан висини три средња прелома линије. Две величине можемо да поставимо и заједно. На пример, `\blank[2*big, medium]` ће уметнути два велика и један средњи размак.

Пошто је команда `\blank` дизајнирана за повећање вертикалног размака између пасуса, она нема никакав ефекат ако се између два пасуса чији размак желите да повећате уметне прелом странице; а ако уметнемо две узастопне `\blank` команде, примениће се само једна од њих (она која умеће највећи размак). `\blank` команда која се постави иза прелома линије такође нема ефекта. Међутим, у овим случајевима можемо да форсирамо уметање вертикалног простора употребом симболичке речи „force” у опцији команде. Тако на пример, ако желимо да се наслови поглавља у нашем документу појаве ниже на страници, тако да је укупна висина те странице мања од осталих страница (што је релативно честа типографска пракса), онда овако морамо да напишемо конфигурацију `\chapter` команде:

```
\setuphead
[chapter]
[
  page=yes,
  before={\blank[4cm, force]},
  after={\blank[3*medium]}
]
```

Овај низ команди ће обезбедити да поглавља увек почињу на новој страници и да се лабела поглавља помери четири центиметара наниже. Ово не би радило без опције „force”.

11.2.4 `\setupblank` и `\defineblank`

Раније сам поменуо да када се `\blank` употреби без аргумената, еквивалентна је са `\blank[big]`. Међутим, ово можемо променити помоћу `\setupblank`, тако што је подесимо `\setupblank[0.5cm]` на пример, или `\setupblank[medium]`. Када се користи без аргумената, `\setupblank` ће подесити вредност на величину текућег фонта.

Као и са `\setupwhitespace`, празан простор који се умеће са `\blank` је, онда када је њена вредност једна од предефинисаних симболичких вредности, еластична димензија која може да се адаптира у одређеној мери. То можемо да изменимо помоћу „fixed”, уз могућност да касније вратимо подразумевану вредност са „flexible”. Тако на пример, за текст који се слаже у две колоне, препоручује се да се постави `\setupblank[fixed,line]`, а када се враћа назад на једну колону, `\setupblank[flexible,default]`.

Одређену конфигурацију можемо да придружимо имену командом `\defineblank`. Општи формат ове команде је:

```
\defineblank[Име] [Конфигурација]
```

Када дефинишемо своју конфигурацију размака, можемо да је користимо помоћу `\blank[ИмеКонфигурације]`.

11.2.5 Остале процедуре за добијање више вертикалног размака

TeX команда која умеће додатни вертикални простор је `\vskip`. Ова команда, као и скоро све TeX команде, такође функционише у систему ConTeXt али се саветује да се уопште не употребљава јер омета интерно функционисање неких ConTeXt макроа. Уместо ње се саветује употреба команде `\godown` чија је синтакса:

`\godown[Мера]`

где *Мера* мора да буде цео број након којег следи јединица мере. На пример, `\godown[5cm]` ће померити 5 центиметара наниже у страници; мада ако се у оквиру ових 5 центиметара наиђе на прелом странице, команда `\godown` ће померити само до наредне странице. Слично, `\godown` нема ефекта на почетку странице, мада можемо да је *преваримо* тако што напишемо, на пример, „`_ \godown[3cm]`”¹ што ће прво да уметне размак чиме више нисмо на почетку странице, па ће онда отићи три центиметара наниже.

Као што знамо, `\blank` такође прихвата конкретну меру као аргумент. Дакле, из угла корисника, `\blank[3cm]` или `\godown[3cm]` су практично исто. Међутим, између њих постоје суптилне разлике. Тако на пример, две узастопне `\blank` команде се не акумулирају, па када се ово догоди, примењује се само она која задаје већи размак. С друге стране, две или више `\godown` команде се савршено добро надовезују.

Још једна прилично корисна TeX команда чија употреба не прави проблем систему ConTeXt, је `\vfill`. Ова команда умеће флексибилни вертикални размак које се протеже све до краја странице. То је као да команда *џура* наниже све што је написано иза ње. Тако се могу постићи занимљиви ефекти, као на пример, постављање пасуса на дно странице простим навођењем команде `\vfill` испред њега. Мада ефекат команде `\vfill` тешко може да се примети ако се она не комбинује са форсираним преломима странице, јер нема пуно смисла гурати пасус или линију текста наниже кроз пасус пошто када расте, он расте навише.

На пример, ако желимо да обезбедимо да се линија постави на дно странице, треба да напишемо:

```
\vfill
Линија на дну странице
\page[yes]
```

Као и све остале команде које умећу вертикални размак, команда `\vfill` нема ефекта на почетку странице. Али можемо да је *преваримо* тако што испред ње напишемо форсирани размак. На пример:

```
\page[yes]
\ \vfill
Центрирана линија
\vfill
\page[yes]
```

ће фразу „Центрирана линија” поставити вертикално центрирану на страници.

¹ Присетите се да у овом документу карактер ‘`_`’ користимо за представљање размака онда када је важно да га уочимо.

11.3 Како ConTeXt изграђује линије које формирају пасусе

Један од главних задатака словослагачког система је да прихвати дугачки низ речи и да га подели на појединачне линије погодне величине. На пример, сваки пасус у овом тексту је подељен на линије ширине 15 центиметара, али аутор није морао да се бави таквим детаљима, јер ConTeXt сам бира тачке прелома након што сваки пасус сагледа у целини, тако да завршне речи пасуса заиста могу да утичу на поделу прве линије. Резултат тога је да су размаци између речи у целом пасусу уједначени у највећој могућој мери.

Ово је један од аспеката који најбоље приказује разлику у начину на који функционишу текст процесори као и бољи квалитет који се добија системима као што је ConTeXt. Пошто текст процесор, када наиђе на крај линије и прескочи на наредну, подешава размак између речи у линији која је управо попуњена тако да обезбеди исправно поравнање. Он то ради за сваку линију посебно, па на крају свака линија у пасусу има различити размак између речи. Ово може да буде разлог појаве врло лошег ефекта (нпр. 'реке' празног простора које теку кроз текст). С друге стране, ConTeXt пасус обрађује у целисти, и за сваку линију израчунава број прихватљивих тачака прелома реда и величину размака између речи која би настала након прелома на одређеној тачки. Пошто прелом на некој линији утиче на потенцијалне тачке прелома у наредним линијама, укупан број могућности може да буде прилично велики; али то не представља проблем за ConTeXt. Он ће коначну одлуку да донесе на основу комплетног пасуса, обезбеђујући да је размак између речи на свакој линији *што је више могуће сличан*, а резултат тога је да су пасуси много боље сложени; визуелно гушћи.

Да би ово постигао, ConTeXt тестира разне алтернативе и свакој од њих на основу њених параметара придружује *badness* (исквареност) вредност. Начин одређивања ове вредности је одређен након детаљне студије уметности типографије. Коначно, када испита све могућности, ConTeXt бира најмање лошу опцију (ону која има најнижу вредност искварености). У општем случају, ово функционише прилично добро, али ће несумњиво постојати случајеви у којима изабране тачке прелома нису најбоље, или нама не изгледају најбоље. Зато ћемо понекада пожелети да програму кажемо како нека места нису пожељна за прелом реда. А у другим ситуацијама ћемо пожелети да на одређеном месту форсирамо прелом.

11.3.1 Употреба резервисаног карактера '~'

Очигледно су главни кандидати за тачке прелома размаци између речи. Ако желимо да назначимо да одређени размак никада не сме да се замени преломом линије, као што већ знамо, употребићемо резервисани карактер '~', који TeX назива *веза*, а који везује две речи заједно.

Употреба овог непрекидајућег размака се углавном препоручује у следећим случајевима:

- Између делова који чине скраћеницу. На пример U~S.
- Између скраћеница и појма на који се односе. На пример, Др~Ана Петровић или стр.~45.
- Између бројева и појма који иде уз њих. На пример, Елизабета~II, 45~томова.
- Између цифара и симбола који се налазе испред или иза њих, ако нису у експоненту. На пример, 73~км, \$~53; али ипак, 35'.
- У процентима који су изражени речима. На пример, двадесет~процената.

- У групама бројева раздвојених размацима. На пример, 5~357~891. Мада се у оваквим случајевима препоручује употреба онога што се назива *шпанки размак*, који се у систему ConTeXt добија командом `\,`, тако да се напише 5\,357\,891.
- Да се спречи да скраћеница буде сама у линији. На пример:

Постоје сектори као што су забава, комуникациони медији,
трговина,~итд.

Овим случајевима, КНУТ (отац система TeX) додаје и следеће препоруке:

- Након скраћенице која није на крају реченице.
- У референцама на делове документа, као што су поглавља, додаци, слике, итд. На пример, Поглавље~12.
- Између имена и првог слова средњег имена особе, или између првог слова имена и презимена. На пример, Доналд~Е. Кнут, А.~Ајнштајн.
- Између математичких симбола која стоје уз имена. На пример, димензија~\$d\$, шири-на~\$w\$.
- Између симбола у редовима. На пример $\{1,~2,~\dots,~n\}$.
- Када је број стриктно везан са предлогом. На пример, од 0 до~1.
- Када се математички симболи изражавају речима. На пример, једнако~је~\$n\$.
- У листама унутар пасуса. На пример: (1)~зелено, (2)~црвено, (3)~плаво.

Мноштво случајева? Несумњиво, цена типографске перфекције захтева додатни труд. Јасно је да ако не желимо, не морамо да применимо ова правила, али не смета да их познајемо. Осим тога – овде говорим из личног искуства – једном када се навикнемо да их примењујемо (било које од њих), почињемо аутоматски то да радимо. Исто као када на речи постављамо акценте док их пишемо (у шпанском то морамо): за оне од нас који то раде, а постало је аутоматизам, није потребно више времена да се напише реч са акцентом од речи без акцента.

11.3.2 Растављање речи на крају реда

Осим за језике који се углавном састоје од речи из једног слога, веома је тешко да се добије оптимални резултат када су потенцијалне тачке прелома реда само између речи. Због тога ConTeXt такође анализира могућност уметања прелома реда између два слога у речи (у српском је то мало слободније); а да би се то урадило, неопходно је да се зна језик на којем је текст написан, јер се правила растављања речи на крају реда разликују у сваком језику. Због тога је важно да се у преамбули документа наведе команда `\mainlanguage`.

Може се догодити да ConTeXt неку реч не успе да растави. Понекада узрок могу бити сама његова правила за поделу речи (на пример, ConTeXt никада не дели реч на два дела ако делови немају минимални број слова); или јер је реч двосмислена. На крају крајева, шта ConTeXt може да уради са речима као што је „unionised”? Реч би могла да се појави у фрази као што је „the unionised workforce” (синдикализована радна снага), али би могла да се појави и у хемијском тексту као „an unionised particle” (тј. нејонизована честица). А шта ако ConTeXt мора да се носи са речи „manslaughter” (убиство) као последњој речи на страници, пре прелома странице. Могао би да преломи реч као man-slaughter (исправно) али би могао такође да је преломи и као mans-laughter (двозначно).

Штагод да је разлог, ако нисмо задовољни начином на који је реч подељена, или ако је не-исправно подељена, то можемо да променимо тако што контролним симболом `\` - експлицитно назначимо потенцијалне тачке прелома на којима реч може да се преломи. Тако на пример, ако нам реч „unionised” зада било какве проблеме, могли би да је у изворном фајлу напишемо као „union\ -ised”; или ако имамо проблем са „manslaughter”, могли би да је напишемо као „man\ -slaughter”.

Ако се проблематична реч користи неколико пута у документу, онда би требало да у преамбули назначимо како се раставља командом `\hyphenation`: ова команда, која је предвиђена да се наведе у преамбули изворног фајла, као аргумент узима једну или више речи раздвојених запетама, које цртицом наводе тачке на којима се могу поделити. На пример:

```
\hyphenation{union-ised, man-slaughter}
```

Ако реч која се проследи овој команди не садржи цртицу, она се никада неће растављати на крају реда. Исти ефекат се постиже употребом команде `\hbox` која креира недељиву хоризонталну кутију око речи, или `\unhyphenated` која спречава да се реч или речи које јој се проследи као аргумент деле на крају реда. Али док `\hyphenation` функционише глобално, `\hbox` и `\unhyphenated` функционишу локално, што значи да команда `\hyphenation` утиче на сва појављивања речи задате као аргумент у целом документу; док `\hbox` или `\unhyphenated` функционишу само на месту у изворном фајлу на којем се наиђе на њих.

Интерно, начин на који \TeX контролише поделу речи на крају реда је помоћу променљивих `\pretolerance` и `\tolerance`. Прва од ових променљивих контролише прихватљивост поделе која се врши само на размаку. Подразумевано је 100, али ако је променимо, на пример, на 10 000, онда ће \ConTeXt увек сматрати да је прихватљиво да постоји прелом линије који не значи поделу речи на слоге, што *de facto* значи да искључујемо поделу речи према слоговима. А ако вредност `\pretolerance` поставимо на -1, на пример, натерали бисмо \ConTeXt да на крају реда увек дели речи.

Произвољну вредност за `\pretolerance` увек можемо једноставно да поставимо било где у документу. На пример:

```
\pretolerance=10000
```

али ову вредност такође можемо да мењамо и помоћу „lesshyphenation” и „morehyphenation” вредности у `\setupalign`. У вези овога, погледајте одељак 11.6.1.

11.3.3 Ниво толеранције за преломе линија

Када тражи могуће тачке прелома линије, \ConTeXt је обично врло стриктан, што значи да ће дозволити да реч пређе иза десне маргине када не може правилно да је растави, и не жели да умеће прелом линије испред речи ако то прави сувише велики размак између речи на тој линији. Ово подразумевано понашање обично даје оптималне резултате и само изузетно неке линије донекле излазе ван десне маргине. Идеја је да аутор (или словослагач) прегледа ове ретке случајеве када се документ заврши, па да донесе одговарајућу одлуку, која би могла да буде `\break` команда испред речи која излази ван маргине, или преформулисање текста на другачији начин, тако да се позиција те речи помери негде другде.

Међутим, у неким случајевима ниска толеранција система \ConTeXt може представљати проблем. У тим случајевима му можемо рећи да буде толерантнији са размаком у линијама. За то имамо на располагању команду `\setuptolerance` која нам омогућава да променимо ниво толеранције при израчунавању прелома линија коју \ConTeXt назива „хоризонтална

толеранција” (јер утиче на хоризонтални размак) и „вертикална толеранција” када израчунава преломе страница. О овоме ћемо говорити у одељку 11.6.2.

Хоризонтална толеранција (она која утиче на преломе линија) је подразумевано постављена на вредност „verystict”. Ово подешавање можемо да изменимо, као алтернативе, било којом од следећих вредности: „strict”, „tolerant”, „verytolerant” или „stretch”. Тако на пример,

```
\setuptolerance[horizontal, verytolerant]
```

значи да линија скоро никада неће моћи да пређе десну маргину, чак и ако то значи да се поставља врло велики и једва видљив размак између речи у линији.

11.3.4 Форсирање прелома линије на одређеном месту

Ако желимо да форсирамо прелом линије на одређеном месту, користимо `\break`, `\crlf` или `\\` команде. Прва од њих, `\break`, умеће прелом линије на месту на којем се нађе. Линија на којој се нађе команда ће највероватније да буде естетски деформисана, са огромним размаком између речи. Као што можемо видети у наредном примеру у којем команда `\break` у трећој линији (фрагмента извора на левој страни) креира прилично ружну линију (у форматираном тексту са десне стране).

Видео сам га на углу старог кварта како се `\emph{шепури}` онако како`\break` то раде жестоки момци када ходају, увек држећи руке у џеповима својих капута, тако да нико не може знати који од њих носи бодеж.

Видео сам га на углу старог кварта како се *шепури* онако како то раде жестоки момци када ходају, увек држећи руке у џеповима својих капута, тако да нико не може знати који од њих носи бодеж.

Да бисмо спречили овај ефекат, можемо употребити `\\` или `\crlf` команде које такође умећу форсирани прелом линије, али оригиналну линију испуњавају са довољно празног простора, тако да буде лево поравната:

Видео сам га на углу старог кварта како се `\emph{шепури}` онако како`\\` то раде жестоки момци када ходају, увек држећи руке у џеповима својих капута, тако да нико не може знати који од њих носи бодеж.

Видео сам га на углу старог кварта како се *шепури* онако како то раде жестоки момци када ходају, увек држећи руке у џеповима својих капута, тако да нико не може знати који од њих носи бодеж.

Колико ја знам, на *нормалним* линијама нема разлике између `\\` и `\crlf`; али у наслову одељка постоји разлика:

- `\\` генерише прелом линије у телу документа, али не када се наслов одељка преноси у садржај.
- `\crlf` генерише прелом линије који се примењује и на тело документа и када се линија преноси у садржај.

Прелом линије не би требало да се меша са преломом пасуса. Прелом линије просто завршава текућу линију и почиње наредну, али се остаје у текућем пасусу, тако да ће размак

између оригиналне линије и нове одређивати нормалан проред који важи у пасусу. Тако да постоје само три сценарија у којима се може препоручити форсирани прелом линије:

- У изузетним случајевима када ConTeXt није могао да пронађе погодни прелом линије, тако да линија пробија десну маргину. У овим случајевима (који се дешавају врло ретко, углавном када линија има недељиве *куџије*, или *дословни* текст [погледајте [одељак 10.2.3](#)]), помаже команда `\break` непосредно испред речи која залази у десну маргину.
- У пасусима који су уствари сачињени од појединачних линија, од којих су информације у свакој потпуно независне од оних у претходној, на пример, наслов писма у којем прва линија може да садржи име пошиљаоца, друга примаоца, а трећа датум; или у тексту који говори о ауторству дела, у којем се у једној линији налази име аутора, у другој њихова канцеларија или академска титула и можда у трећој датум, итд. У овим случајевима би прелом линије требало да се уради командама `\\` или `\crlf`. Такође је уобичајено да су ове врсте параграфа десно поравнате.
- Када се пишу песме, или слична врста текстова, за раздвајање стихова. Мада је у овом последњем случају боље да се користи `lines` окружење које је објашњено у [одељку 11.5.1](#).

11.4 Проред

Проред је растојање које раздваја линије пасуса. ConTeXt га аутоматски израчунава на основу конкретног фонта који се тренутно користи и првенствено на основу базне величине постављене командом `\setupbodyfont` или `\switchtobodyfont`.

На проред можемо да утичемо командом `\setupinterlinespace` која прихвата три различите синтаксе:

- `\setupinterlinespace [..Проред..]`, где је *Проред* прецизна вредност или симболичка реч која додељује предефинисани проред:
 - Када је у питању прецизна вредност, то може бити димензија (на пример, 15pt), или просто цео или децимални број (на пример, 1.2). У овом другом случају број се интерпретира као „број линија” у односу на ConTeXt подразумевани проред.
 - Када је симболичка реч, може бити „small”, „medium” или „big” које постављају мали, средњи, или велики проред, увек базиран на подразумеваном прореду који би применио ConTeXt.
- `\setupinterlinespace [...]=...]`. У овом режиму, проред се поставља директно изменом мера помоћу којих ConTeXt израчунава одговарајући проред. Већ сам рекао да се проред израчунава на основу конкретног фонта и његове величине; али само да би ствари остале просте: у суштини, фонт и његова величина постављају одређене мере на основу којих се израчунава проред. Онда се `\setupinterlinespace` приступом те мере мењају, па се мења и проред. Конкретне мере и вредности којима може да се манипулише (чије значење нећу да објашњавам, јер је то ван опсега простог

увога) `cy:line, height, depth, minheight, mindepth, distance, top, bottom, stretch` и `shrink`.

- `\setupinterlinespace` [Име]. Овим режимом успостављамо или конфигуришемо одређени прилагођени проред претходно дефинисан са `\defineinterlinespace`.

Са

```
\defineinterlinespace[Име] [Конфигурација]
```

одређеном имену можемо да придружимо неку конфигурацију прореда коју затим можемо просто да укључимо на неком месту у документу командом `\setupinterlinespace`[Име]. Када желимо да се вратимо на нормални проред, потребно је да напишемо `\setupinterlinespace[reset]`.

11.5 Остале ствари у вези са линијама

11.5.1 Конвертовање прелома линија у изворном фајлу у преломе линија у финалном документу

Као што већ знамо (погледајте одељак 4.2.2), ConTeXt подразумевано игнорише преломе линија у изворном фајлу и сматра да су то обични размаци, осим у случају да постоји два или више узастопна прелома линије, када умеће прелом пасуса. Међутим, постоје ситуације у којима желимо да се поштују преломи линија у оригиналном изворном фајлу онако како су уметнути, када на пример, пишемо поезију. ConTeXt за то обезбеђује „lines” окружење чији је формат:

```
\startlines[Опције] ... \stoplines
```

где између осталог, опције могу бити и било шта од следећег:

- **space**: када се ова опција постави на вредност „on”, уз поштовање прелома линија у изворном фајлу, окружење ће такође да поштује и размаке, чиме се привремено игнорише правило апсорпције.
- **before**: текст или команда која треба да се изврши пре уласка у окружење.
- **after**: текст или команда која треба да се изврши након изласка из окружења.
- **inbetween**: текст или команда која треба да се изврши када се уђе у окружење.
- **indenting**: вредност која назначава да ли се у окружењу пасуси увлаче или не (погледајте одељак 11.1.1).
- **align**: поравнање линија у окружењу (погледајте одељак 11.6).
- **style**: команда стила која се примењује унутар окружења.
- **color**: боја која се примењује унутар окружења.

Тако на пример,

<pre>\startlines Ишли смо у Африку, Да садимо паприку. Знате ону жуту, Фину али љуту. \stoplines</pre>	<p>Ишли смо у Африку, Да садимо паприку. Знате ону жуту, Фину али љуту.</p>
--	---

Подразумевани начин на који ради окружење такође можемо да променимо командом `\setuplines` и, као са толико ConTeXt команди, одређеној конфигурацији овог окружења командом `\definelines` може да се додели име. Њена синтакса је:

```
\definelines[Име] [Конфигурација]
```

где као конфигурацију можемо да ставимо исте опције објашњене за ово окружење. Када дефинишемо прилагођено окружење, овако га умећемо у документ:

```
\startlines[Име] ... \stoplines
```

11.5.2 Нумерисање линија

У одређеним врстама текстова је уобичајено да се успостави нека врста нумерисања линија, на пример, у текстовима о компјутерском програмирању где је релативно уобичајено да се као примери наводе фрагменти кода чије су линије нумерисане, или у песмама, критичким издањима, итд. За све ове ситуације ConTeXt обезбеђује `linenumbering` окружење чији је формат

```
\startlinenumbering[Опције] ... \stoplinenumbering
```

Доступне су следеће опције:

- **continue:** у случајевима када има више делова документа у којима је потребна нумерација линија, ова опција уређује да нумерација поново почиње од почетка за сваки од њих („continue=no”, подразумевана вредности). С друге стране, ако је потребно да се нумерација наредног дела настави тамо где се завршила нумерација претходног, треба да поставимо „continue=yes”.
- **start:** назначава број прве линије у случајевима када не желимо да буде '1', или желимо да одговара претходној нумерацији.
- **step:** нумерисаће се све линије које су део окружења, али овом опцијом можемо назначити да се број штампа само у одређеним интервалима. На пример, за песме је уобичајено да се штампа сваки пети број (стихови 5, 10, 15, ...).

У општем случају, командом `\setuplinenumbering` све ове опције могу да се наведу за сва `linenumbering` окружења у нашем документу. Ова команда нам такође омогућава да конфигуришемо и остале аспекте нумерације линија:

- **conversion:** врста нумерације линија. Може бити било која од оних које су објашњене на [страници 116](#) у вези са нумерацијом поглавља и одељака.
- **style:** команда (или команде) које одређују стил који ће се користити за исписивање нумерације линија (фонт, величина, варијанта...).
- **color:** боја којом ће се исписивати бројеви линија.
- **location:** где ће се исписивати бројеви линија. Може бити било шта од следећег: text, begin, end, default, left, right, inner, outer, inleft, inright, margin, inmargin.
- **distance:** растојање између броја линије и саме линије.
- **align:** поравнање броја. Може бити: inner, outer, flushleft, flushright, left, right, middle или auto.
- **command:** команда којој ће се број линије проследити као параметар пре исписивања.
- **width:** ширина простора који се резервише за испис броја линије.
- **left, right, margin:**

Такође можемо да креирамо различите прилагођене нумерације линија командом `\definelinenumbering` којом се конфигурацији придружује име:

```
\definelinenumbering[Име] [Конфигурација]
```

Када се одређена конфигурација дефинише и придружи имену, можемо је употребљавати помоћу

```
\startlinenumbering[Име] ... \stoplinenumbering
```

11.6 Хоризонтално и вертикално поравнање

Команда која у општем случају контролише поравнање је `\setupalign`. Она се користи и за хоризонтално и за вертикално поравнање.

11.6.1 Хоризонтално поравнање

Када *природна* ширина линије текста не заузима сав доступан простор, јавља се питање шта урадити са тим преосталим празним простором.¹ У суштини можемо да урадимо три ствари у вези са овим проблемом:

1. Да га накупимо на једној од две стране линије: ако га накупимо на леву страну, изгледаће као да је линија *мало њојурана* у десно, а ако га накупимо на десну страну, линија остаје на левој. У првом случају се ради о *десном њоравнању*, а у другом о *левом њоравнању*. ConTeXt подразумевано примењује лево поравнање последње линије пасуса.

¹ Под изразом *природна* мислим на ширину линије *пре* него што ConTeXt подеси величину размака између речи како би омогућио поравнање.

Када се неколико узастопних линија лево поравна, десна страна је неправилна; а када је поравнање са десне стране, онда лева страна изгледа неуједначено. Приликом давања имена опцијама које поравнавају једну или другу страну, ConTeXt не мисли на страну на којој су линије поравнате, већ на страну на којој нису уједначене. Тако опција `flushright` даје лево поравнање, а `flushleft` десно поравнање. Као скраћенице за `flushright` и `flushleft`, `\setupalign` као вредности подржава и `right` и `left`. Али **пажња**: овде значење речи вара. Мада `left` значи „лево”, а `right` значи „десно”, `\setupalign[left]` поравнава по десној ивици, а `\setupalign[right]` поравнава по левој. У случају да се читалац пита зашто је наведен овај коментар, вреди цитирати ConTeXt вики: „ConTeXt користи опције `flushleft` и `flushright`. Десно и лево поравнање су супротна од уобичајених смерова у свим командама које прихватају опцију поравнања, у смислу 'искрзана лева' и 'искрзана десна'. Нажалост, када је Ханс писао овај део система ConTeXt, размишљао је о 'искрзана десна' и 'искрзана лева' поравнању, а не о 'равна лева' и 'равна десна'. Па како је већ одавно тако, није могуће да се промени јер би промена покварила компатибилност уназад са свим постојећим документима који ово користе.”

У документима припремљеним за двострану штампу, уз леву и десну маргину постоје и унутрашња и спољашња маргина. Вредности `flushinner` (или просто `inner`) и `flushouter` (или просто `outer`) у тим случајевима успостављају одговарајуће поравнање.

2. Да га распоредимо по обе маргине. Резултат ће бити центрирана линија. `\setupalign` опција која то ради је `middle`.
3. Да га распоредимо по свим речима које чине линију, уз неопходно повећање размака између речи, тако да линија потпуно заузме доступан простор. У овим случајевима говоримо о *уједначеним линијама*. Ово је и подразумевана ConTeXt вредност, па зато не постоји посебна `\setupalign` опција за успостављање оваквог поравнања. Међутим, ако смо променили подразумевано уједначено поравнање, можемо га вратити помоћ `\setupalign[reset]`.

Вредност за `\setupalign` коју смо управо видели (`right`, `flushright`, `left`, `flushleft`, `inner`, `flushinner`, `outer`, `flushouter` и `middle`) може да се комбинује са `broad`, па се онда добија донекле грубље поравнање.

Остале могуће вредности `\setupalign` које утичу на хоризонтално поравнање се тичу растављања речи на крају реда, јер и зависности од тога да ли се то ради или не, тачна мера дужине линије је већа или мања; а то онда утиче на преостали празан простор.

`\setupalign` прихвата `morehyphenation` вредност која систему ConTeXt говори да се више потруди приликом проналажења тачака прелома које се добијају растављањем речи на крају реда и `lesshyphenation` која има супротни ефекат. Са `\setupalign[horizontal, morehyphenation]`, преостали празан простор у линијама ће се умањити, па ће онда поравнање бити мање уочљиво. Супротно томе, `\setupalign[horizontal, lesshyphenation]` чини да буде више преосталог празног простора, чиме је поравнање уочљивије.

Предвиђено је да се `\setupalign` уметне у преамбулу и да утиче на комплетан документ, или да се уметне на одређено место, па да утиче на све од тог места до краја. Ако само желимо да променимо поравнање једне или неколико линија, можемо да употребимо:

- „alignment” окружење, намењено да утиче на неколико линија. Његов општи облик је:

```
\startalignment[Опције] ... \stopalignment
```


где су *Ойције* било које од оних које прихвата `\setupalign`.

- Команде `\leftaligned`, `\midaligned` или `\rightaligned` које редом праве лево, центрирано, или десно поравнање; а ако желимо да последња реч у пасусу (али само она, а не и остатак линије) буде десно поравната, можемо да употребимо `\wordright`. Све ове команде траже да им се текст на који утичу достави у витичастим заградама.

С друге стране, запазите да ако речи „right” и „left” у `\setupalign` укључују супротно поравнање од оног на које сугерише име, то се не дешава са командама `\leftaligned` и `\rightaligned` које примењују ону врсту поравнања које одговара њиховом имену: left лево поравнање, а right десно.

11.6.2 Вертикално поравнање

Пошто се хоризонтално поравнање примењује када ширина линије не заузима сав простор који јој је доступан, онда вертикално поравнање утиче на висину целе странице: ако *џриродна* висина текста странице не заузме комплетан доступан простор, шта се ради са остатком празног простора? Можемо да га нагомиламо на врх („height”), што значи да ће се текст странице погурати наниже; можемо да га нагомиламо на дно („bottom”), или да га распоредимо између пасуса („line”). Подразумевана вредност за вертикално поравнање је „bottom”.

Вертикални ниво толеранције

На исти начин на који `\setuptolerance` можемо да променимо ниво толеранције система ConTeXt у погледу количине хоризонталног простора који је дозвољен у линији (хоризонтална толеранција), такође можемо да променимо и његову вертикалну толеранцију, тј. толеранцију на простор између пасуса који је већи од оног који ConTeXt подразумевано сматра разумним за добро сложену страницу. Вредности које су могуће за вертикалну толеранцију су исте као и за хоризонталну: `verystict`, `strict`, `tolerant` и `verytolerant`. Подразумевана вредности је `\setuptolerance [vertical, strict]`.

Контрола удовица и сирочића

Један аспект који индиректно утиче на вертикално поравнање је контрола удовица и сирочића. Оба израза наговештавају да је последица прелома странице то да једна линија пасуса буде изолована од остатка пасуса који се нађе на другој страници. Сматра се да то није типографски погодно. Ако је изолована линија пасуса прва на страници, онда говоримо о *линији удовици*; ако је линија одвојена од свог пасуса последња на страници, онда говоримо о *линији сирочићу*.

ConTeXt подразумевано не имплементира контролу којом би се спречило да дође до оваквих линија. Али то можемо променити изменом неких од интерних променљивих система ConTeXt: `\widowpenalty` контролише линије удовице, а `\clubpenalty` контролише линије сирочиће. Дакле, следеће наредбе у преамбули нашег документа обезбеђују да се ова контрола изврши:

```
\widowpenalty=10000
\clubpenalty=10000
```

Извођење ове контроле значи да ће ConTeXt избегавати уметање прелома странице који прву или последњу линију поставља на суседну страницу оне на којој је остатак пасуса. Ово

спречавање ће бити мање или више ригорозно у зависности од вредности коју доделимо променљивама. Ако поставимо 10 000, као што сам употребио у примеру, контрола ће бити апсолутна; вредност од, на пример, 150 поставља контролу која није тако ригорозна, па се понекад може догодити да се појаве неке линије удовице или сирочићи, углавном онда када је алтернатива у типографском смислу још лошија.

Глава 12

Специјалне конструкције и пасуси

Садржај: 12.1 Фусноте и белешке на крају; 12.1.1 Врсте напомена у систему ConTeXt и команде које су им придружене; 12.1.2 Детаљи о фуснотама и напоменама на крају; 12.1.3 Локалне напомене; 12.1.4 Креирање и употреба прилагођених типова напомена; 12.1.5 Конфигурисање напомена; 12.1.6 Привремено искључивање напомена током компајлирања; 12.2 Пасуси у више колона; 12.2.1 `\startcolumns` окружење; 12.2.2 Паралелни пасуси; 12.3 Уређене листе; 12.3.1 Избор врсте листе и граничник између *сџавки*; А Неуређене листе; Б Уређене листе; 12.3.2 Унос ставки у листу; 12.3.3 Основна конфигурација листе; 12.3.4 Додатна конфигурација листе; 12.3.5 Просте листе са `\items` командом; 12.3.6 Предодређивање понашања листе и креирање сопствене врсте листе; 12.4 Описи и набрајања; 12.4.1 Описи; 12.4.2 Набрајања; 12.5 Линије и оквири; 12.5.1 Просте линије; 12.5.2 Линије везане за текст; 12.5.3 Уоквирене речи или текстови; 12.6 Остала интересантна окружења и конструкције;

12.1 Фусноте и белешке на крају

Напомене су „секундарни текстуални елементи који се користе за различите намене, као што је разјашњавање или проширивање главног текста, приказивање библиографске референце на изворе, укључујући и цитате, указивање на остале документе или исказивање значења текста” [*Libro de Estilo de la Lengua española* (Стилски водич шпанског језика), стр. 195]. Оне су од велике важности у текстовима академске природе. Могу да се налазе на разним местима на страници или у документу. Данас су најраспрострањеније оне које се налазе у подножју странице (па се зато називају фусноте); понекад се налазе и у маргинама (маргиналне напомене), на крају сваког поглавља или одељка, или на крају документа (белешке на крају). У посебно компликованим документима, могу да постоје разне *серџе* напомена: ауторове, напомене преводиоца, ажурирања, итд. Тачније, у критичким издањима апарат напомена може бити изузетно компликован и само неколико словослагачких система има могућност да га подржи. ConTeXt је један од њих. Доступне су бројне команде које успостављају у конфигуришу различите врсте напомена.

Да би се ово објаснило, корисно је почети са истицањем различитих елемената који постоје у напмени:

- *Маркер* или *сигро* напомене: знак који се поставља у тело текста и који означава да постоји напомена везана за њега. Немају све врсте напомена придружено *сигро*, али када постоји, ово *сигро* се појављује на два места: на месту у главном тексту на које се односи напомена и на почетку самог текста напомене. Присуство исте ознаке на оба места омогућава да се напомена придружи главном тексту.
- *ИД* или *идентификаџор* напомене: слово, број или симбол који идентификује напомену и помоћу којег се напомена разликује од осталих напомена. Неке напомене, на пример маргиналне напомене, немају ИД. Када то није случај, ИД се обично подудара са *сигром*.

Ако размишљамо искључиво о фуснотама, видећемо да нема разлике између њих и онога што сам управо назвао *маркер референце* и *ид*. У осталим врстама напомена јасно можемо видети разлику: линијске напомене, на пример, имају ид али немају маркер референце.

- *Текст* или *садржај* напомене, увек се налази на неком другом месту на страници или у документу, различитом од оног на којем је команда која генерише напомену и означава њен садржај.
- *Лабела* придружена напомени: лабела или име придружено напомени које се не приказује у финалном документу, али нам омогућава да се позовемо на њега и да на неком другом месту у документу добијемо њен ИД.

12.1.1 Врсте напомена у систему ConTeXt и команде које су им придружене

ConTeXt познаје различите врсте напомена. За сада ћу само да их наведем, уз опис у општим цртама и командама које служе да се генеришу. Касније ћу детаљније да опишем прве две:

- **Фусноте:** несумњиво најпопуларније, у тој мери да је постало уобичајено да се за све врсте напомена једним изразом каже да су *фусноте*. Фусноте на место у документу на којем се најђе на команду умећу *маркер* са *ид* напомене, а сам текст напомене умећу на дно странице на којој се појави маркер. Креирају се командом `\footnote`.
- **Белешке на крају:** ове напомене се креирају командом `\endnote` и умећу се на место у документу на којем се пронађе маркер са ИД напомене; али садржај напомене се умеће на друго место у документу, и то уметање се постиже другом командом (`\placenames`).
- **Маргиналне напомене:** као што им име наговештава, оне се пишу у маргини текста и у телу документа нема ИД или аутоматски генерисаног маркера или сидра. Две главне команде (мада не и једина) које их креирају су `\inmargin` и `\margintext`.
- **Линијске напомене:** врста напомена која је типична за окружења у којима се линије нумеришу, као што је случај са `\startlinenumbering ... \stoplinenumbering` (погледајте одељак 11.5.2). Сама напомена, која се обично пише на дну, се односи на одређени број линије. Оне се генеришу командом `\linenote` која се конфигурише командом `\setuplinenote`. Ова команда на штампа *маркер* у телу текста, већ у самој напомени исписује број линије на коју се напомена односи (што се користи као *ИД*).

Сада ћу детаљно да прикажем прве две врсте напомена:

- Маргиналне напомене се обрађују на другом месту (одељак 5.7).
- Линијске напомене имају врло специфичну употребу (посебно у критичким издањима) па верујем да је у уводном документу као што је овај, довољно да се читалац упозна са чињеницом да оне постоје.

Међутим, заинтересованом читаоцу препоручујем видео (на шпанском) уз који иде и текст (такође на шпанском) о критичким издањима у систему ConTeXt, чији је аутор Пабло Родригес. Можете га пронаћи на [овој адреси](#). Такође је добар и за разумевање неколико општих подешавања напомена.

12.1.2 Детаљи о фуснотама и напоменама на крају

Синтакса команди за фусноте и напомене на крају као и механизми конфигурације и прилагођавања тих команди су врло слични, јер су у стварности, обе ове врсте напомена конкретне инстанце општијих конструкција (напомена). Командом `\definernote` (погледајте одељак 12.1.4) могу да се дефинишу и друге инстанце напомена.

Синтакса команде која креира сваку од ових врста напомена је:

```
\footnote[Лабела]{Текст}
\endnote[Лабела]{Текст}
```

где је

- *Лабела* аргумент који није обавезан и који напмени додељује лабелу којом можемо негде другде у документу да укажемо на ову напомену.
- *Текст* је садржај напомене. Дужина текста може бити произвољна и може да садржи специјалне пасусе и подешавања, мада би требало напоменути да је што се тиче фуснота, врло тешко постићи коректан распоред странице у документима који имају доста прилично дугачких напомена.

У принципу, било која команда која може да се употреби у главном тексту може да се употреби и у тексту напомене. Међутим, успео сам да потврдим како неке конструкције и карактери који не представљају никакав проблем у главном тексту генеришу грешку приликом компајлирања ако се нађу у тексту напомене. Ове случајеве сам открио током тестирања, али их нисам ни на који начин организовао.

Када се аргумент *Лабела* употреби за постављање лабеле напомене, команда `\note` нам омогућава да добијемо ИД напомене у питању. Ова команда испишује ИД напомене којој је придружена лабела прослеђена као аргумент. Тако на пример:

```
Ја сам црвић Глиша,\footnote[glisa]{Познати
карактер из дечије песмице}
ја сам црвић Глиша\note[glisa]\
Илазим из земље само кад
је киша.\Волим да се играм,
волим да се смејем.\
Лети да се купам, зими да се грејем.
```

```
Ја сам црвић Глиша,1 ја сам црвић Глиша1
Илазим из земље само кад је киша.
Волим да се играм, волим да се смејем.
Лети да се купам, зими да се грејем.
```

¹Познати карактер из дечије песмице

Главна разлика између `\footnote` и `\endnote` је место на којем се појављује напомена:

`\footnote` Као правило, текст напомене штампа на дну странице на којој се налази команда, тако да ће се маркер напомене и њен текст (или почетак текста, ако мора да се распореди на две странице) појавити на истој страници. Да би то постигао, ConTeXt ће урадити неопходна подешавања словослагања странице тако што ће да израчуна простор потребан за место напомене на дну странице.

Али у неким окружењима, `\footnote` текст напомене неће да уметне на дно саме странице, већ испод окружења. Ово је случај, на пример, у табелама, или у `columns` окружењу. Ако у овим случајевима желимо да се напомене унутар окружења сместе на дно странице, треба да уместо `\footnote` команде употребимо `\footnotetext` у комбинацији са командом `\note` која је поменута изнад. Ова прва, која такође као аргумент који није обавезан подржава лабелу, штампа само текст напомене, без маркера. Али пошто `\note` штампа само маркер без текста, комбинација обе нам омогућава да напомену поставимо тамо где желимо.

Тако на пример, могли би да напишемо `\note[MyLabel]` унутар табеле или окружења у више колона, па да онда када изађемо из тог окружења наведемо `\footnotetext[MyLabel]{Текст напомене}`.

Још један пример употребе `\footnotetext` у комбинацији са `\note` би биле напомене унутар других напомена. На пример:

```
Ово%
\footnote{или ово\note[noteB], ако вам више одговара.}%
\footnotetext[noteB]
{или можда чак и ово\note[noteC].}
\footnotetext[noteC]{би могло бити
нешто потпуно другачије.}
је реченица са угнеђеним напоменама.
```

Ово¹ је реченица са угнеђеним напоменама.

¹или ово², ако вам више одговара.

²или можда чак и ово³.

³би могло бити нешто потпуно другачије.

`\endnote` штампа само сидро напомене на месту у изворном фајлу на којем се постави. Конкретан садржај напомене се умеће другом командом на неко друго место у документу, (`\placenotes[endnote]`) која ће, на месту на које се нађе, уметнути садржај свих напомена на крају у документу (или поглављу или одељку у питању).

12.1.3 Локалне напомене

Окружење `\startlocalfootnotes` значи да се фусноте у њему посматрају као *локалне* напомене, што значи да се њихова нумерација ресетује и да се садржај напомена неће аутоматски уметати заједно са осталим напоменама, већ само на оном месту у документу на којем се нађе на команду `\placelocalfootnotes`, што може али и не мора да буде унутар окружења.

12.1.4 Креирање и употреба прилагођених типова напомена

Командом `\definernote` можемо да креирамо специјалне врсте напомена. То може бити корисно у сложеним документима где постоје напомене од различитих аутора, или за различите намене, да се графички разликује свака врста напомене у нашем документу користећи различито форматирање и другачију нумерацију.

Синтакса команде `\definernote` је следећа:

`\definernote[Име][Модел][Конфигурација]`

где је

- *Име* име које додељујемо нашој новој врсти напомене.
- *Модел* модел напомене која ће иницијално да се искористи. Може бити `footnote` или `endnote`; у првом случају ће наш модел напомене радити као фусноте, а у другом као напомене на крају, мада их у документ нећемо постављати `\placenotes[endnote]` већ са `\placenotes[Име]` (име које смо доделили овој врсти напомена).

Теоретски овај аргумент није обавезан, мада у мојим тестовима неке напомене креиране без њега нису биле видљиве, а ја нисам имао стрпљења да откријем зашто се то дешавало.

- *Конфигурација* је други аргумент који није обавезан и који нам омогућава да се наша нова врста напомене разликује од свог модела: било подешавањем другачијег формата, било другачијом нумерацијом, или и једним и другим.

Према званичној листи ConTeXt команди (погледајте одељак 3.6) подешавања која могу да се поставе током креирања нове врсте напомене су базирана на онима која би могла касније да се наведу командом `\setupnote`. Међутим, као што ћемо ускоро видети, уствари постоје две могуће команде за подешавање напомена: `\setupnote` и `\setupnotation`. Тако да мислим да је пожељно изоставити овај аргумент, па онда подесити нашу нови напомену користећи одговарајуће команде. То је барем једноставније да се објасни.

На пример, следећи фрагмент ће креирати нову напомену под именом „BlueNote” која ће личити на фусноте, али ће се њен садржај штампати црним слогом у плавој боји:

```
\definernote [BlueNote] [footnote]
\setupnotation
  [BlueNote]
  [color=blue, style=bf]
```

Када креирамо нову врсту напомена, нпр. *BlueNote*, постаће доступна команда која нам омогућава да је користимо. У нашем примеру ће то бити `\BlueNote` и њена синтакса је слична синтакси команде `\footnote`:

```
\BlueNote[Лабела]{Текст}
```

12.1.5 Конфигурисање напомена

Конфигурација напомена (фуснота или напомена на крају, напомена креираних командом `\definernote` као и линијских напомена подешених са `\linenote`) се врши помоћу две команде: `\setupnote` и `\setupnotation`¹. Синтакса је слична за обе:

```
\setupnote[ТипНапомене][Конфигурација]
\setupnotation[ТипНапомене][Конфигурација]
```

где се *ТипНапомене* односи на врсту напомене коју конфигуришемо (`footnote`, `endnote` или име неке врсте напомена коју смо сами креирали) и *Конфигурација* садржи одређене опције конфигурације за команду.

Проблем је што имена ове две команде не стављају јасно до знања шта је разлика између њих, или које ствари свака од њих конфигурише; а такође не помаже ни чињеница да многе од опција ових команди нису нигде документоване. Након много тестирања, нисам успео да дођем до било каквог закључка који би ми омогућио да разумем зашто се одређене ствари

¹ `\setupnote` има 35 директних опција конфигурације и још 45 додатних наслеђених од `\setupframed`; `\setupnotation` има 45 директних конфигурационих опција и још 23 наслеђених од `\setupcounter`. Пошто ове опције нигде нису документоване, и мада за већину њих из њиховог имена можемо наслутити њихову корисност, морамо проверити да ли је наша интуиција у праву или није; и такође, узимајући у обзир да многе од ових опција прихватају већи број вредности које све треба да се тестирају... Да бих написао ово објашњење, видећете да сам морао извршити поприличан број тестова; и мада је извршавање теста брзо, обављање великог броја тестова је споро и досадно. Тако да се надам да ће ми читалац опростити ако вам кажем да ћу осим две опште конфигурационе команде за напомене које помињем у главном тексту и на које се фокусирам у објашњењу које следи, изоставити остале четири потенцијалне конфигурационе могућности из објашњења:

- `\setupnotes` и `\setupnotations`: другим речима, исто име али у множини. Вики каже да су верзије команде у једнини и множини синоними, и ја верујем да је тако.
- `\setupfootnotes` и `\setupendnotes`: претпостављамо да су ово специфичне примене за фусноте и напомене на крају, редом. Можда би објашњавање конфигурације напомене на основу ових команди било лакше, међутим, пошто прва опција (`numberconversion`) коју сам покушао са `\setupfootnotes` није успела да проради, мада знам да остале опције ових команди функционишу... био сам сувише лењ да многим тестовима које сам морао да урадим како би написао оно што следи додам и тестове потребне да се у објашњење укључе и ове две команде. Али мишљења сам (након неколико случајних тестова које сам извршио) да све што ради у ове две команде, али чије објашњење изостављам, такође ради и у командама за које дајем објашњење.

конфигуришу једном, а остале другом,¹ осим можда што, због избора које сам направио да бих је натерао да ради, `\setupnotation` увек утиче на текст напомене, или на ИД који се штампа уз текст напомене, док `\setupnote` има неке опције које утичу на маркер за напомену који се умеће у главни текст.

Сада ћу покушати да организујем оно што сам пронашао након обављања неких тестова са различитим опцијама обе команде. Већину опција обе команде остављам по страни, јер нису нигде документоване и јер нисам успео да извучем било какве закључке о томе за шта служе или под којим условима би требало да се користе:

- **ИД који се користи за маркер:**

напомене се увек идентификују бројем. Овде можемо да конфигуришемо:

- *Први број*: контролише га опција `start` у `\setupnotation`. Њена вредност мора да буде цео број, и он се користи за почетак набрајања напомена.
- *Систем набрајања*, који зависи од опције `numberconversion` у `\setupnotation`. Њене вредности могу бити:
 - * *Арајски бројеви*: `n`, `N` или `numbers`.
 - * *Римски бројеви*: `I`, `R`, `RomanNumerals`, `i`, `r`, `romanNumerals`. Прве три су римски бројеви исписани великим словима, а последње три малим.
 - * *Набрајање словима*: `A`, `Character`, `Characters`, `a`, `character`, `characters` у зависности од тога које слово желимо да буде велико (прве три опције) или мало (остале).
 - * *Набрајање речима*. Другим речима, пишемо реч која означава број, тако на пример, `'3'` постаје `'three'`. Постоје две метода. Опција `Words` исписује реч великим словима, а `words` малим.
 - * *Набрајање симболима*: можемо да користимо четири различита скупа симбола, зависно од опције која се изабере: `set 0`, `set 1`, `set 2` или `set 3`. На [страници 117](#) постоји пример симбола који се користе у свакој од ових опција.
- *Дојађај који поочиње нумерацију напомена из почетака*: ово зависи од опције `way` у `\setupnotation`. Када је вредност `bytext`, све напомене у документу ће се редно нумерисати, и тај број се никада неће ресетовати. Када је `bychapter`, `bysection`, `bysubsection` итд., бројач напомена ће се ресетовати сваки пут када се промени поглавље, одељак, или пододељак, а када је `byblock` набрајање се ресетује сваки пут када у макроструктури (погледајте [одељак 7.6](#)) променимо блокове. Када је вредност `byrange`, бројач напомена се ресетује при свакој промени странице.
- **Конфигурисање маркера напомене:**
 - Да ли се приказује или не: опција `number` у `\setupnotation`.
 - Позиција маркера у односу на текст напомене: опција `alternative` у `\setupnotation`: може да има било коју од следећих вредности: `left`, `inleft`, `leftmargin`,

¹ Постоји страница у [ConTeXt викију](#) коју сам случајно открио (јер није директно посвећена напоменама), која сугерише да је разлика у томе што `\setupnotation` контролише текст напомене који треба да се уметне, а `\setupnote` окружење напомене у којем ће се она поставити (?) Али то није у складу са чињеницом да се, на пример, ширина текста напомене (која има везе са њеним *умешањем*) контролише опцијом `width` команде `\setupnote`, а не `\setupnotation` опцијом са истим именом. Оно што се овде контролише је ширина простора између маркера и текста напомене.

right, inright, rightmargin, inmargin, margin, innermargin, outermargin, serried, hanging, top, command.

- Формат маркера у самој напмени: опција `numbercommand` у `\setupnotation`.
- Формат маркера у телу текста: опција `textcommand` у `\setupnote`.

Опције `numbercommand` и `textcommand` морају да се састоје од команде која као аргумент узима садржај маркера. То може бити и кориснички дефинисана команда. Међутим, открио сам да раде и једноставне команде форматирања (`\bf`, `\it`, итд.), мада оне нису команде које обавезно прихватају аргумент.

- Размак између маркера и текста (у самој напмени): опције `distance` и `width` у `\setupnotation`. Нисам успео да откријем разлику (ако она уопште и постоји) између употребе једне или друге опције.
- Постојање хиперлинка за скок између маркера у главном тексту и маркера у самој напмени: опција `interaction` у `\setupnote`. `yes` као вредност значи да ће постојати линка, а по значи да неће.

- **Конфигурисање самог текста напомене.**

Можемо да утичемо на следеће аспекте:

- Место: ово зависи од опције `location` у `\setupnote`.

У принципу већ знамо да се фусноте постављају на дно странице (`location=page`), а белешке на крају тамо где се пронађе команда `\placenotes[endnote]` (`location=text`). Међутим ову функцију можемо да прилагодимо на пример, тако да фусноте поставимо као `location=text`, па ће фусноте радили слично као белешке не крају и појављиваће се у документу на месту где се пронађе команда `\placenotes[footnote]`, или специфична команда за фусноте `\placefootnotes`. Овом процедуром би на пример фусноте могли да штампамо испод пасуса у којем се пронађу.

- Раздвајање пасуса између напомена: свака напомена се подразумевано штампа у посебном пасусу, али можемо подесити да се све напомене на једној страници штампају у истом пасусу ако опцију `paragraph` у `\setupnote` поставимо на „yes”.
- Стил у којем се исписује текст саме напомене: опција `style` у `\setupnotation`.
- Величина слова: опција `bodyfont` у `\setupnote`.

Ова опција служи само у случају када ручно желимо да подесимо величину фонта за фусноте. То скоро увек није добра идеја, јер ConTeXt подразумевано подешава величину фонта за фусноте тако да буде мањи од величине главног текста, али користећи величину која је пропорционална величини фонта у главном телу документа.

- Лева маргина текста напомене: опција `margin` у `\setupnotation`.
- Максимална ширина: опција `width` у `\setupnote`.
- Број колона: опција `n` у `\setupnote` одређује да се текст напомене слаже у две или више колона. Вредност „n” мора бити цео број.

- **Размак између напомена или између напомена и текста:** овде имамо следеће опције:

- `rule`, у `\setupnote` одређује да ће постојати линија (`rule`) између простора за напомене и простора за главни текст на страници. Дозвољене вредности су `yes`, `on`, `no` и `off`. Прве две укључују линију, а последња је искључује.
- `before`, у `\setupnotation`: команда или команде које треба да се изврше пре уметања текста напомене. Служи да се уметне додатни размак, линије које раздвајају напомене, итд.
- `after`, у `\setupnotation`: команда или команде које треба да се изврше након уметања текста напомене.

12.1.6 Привремено искључивање напомена током компајлирања

Команде `\notesenabledfalse` и `\notesenabledtrue` говоре систему ConTeXt да редом искључи или укључи компајлирање напомена. Ова функција може бити корисна ако документ има велики број дугачких напомена, па желимо да добијемо верзију без напомена. Из мог личног искуства, када на пример, коригујем докторску тезу, више волим да је прво прочитам у једном пролазу, без напомена, па да онда у другом читању укључим и напомене.

12.2 Пасуси у више колона

Словослагање текста у више од једне колоне се може подесити:

- а. Као општа особина распореда странице.
- б. Као особина одређених конструкција, као што су на пример, уређене листе, или фусноте, или напомене на крају.
- в. Као особина која се примењује на одређени пасус у документу.

У свим овим случајевима, већина команди и окружења ће савршено да ради са више од једне колоне. Ипак има неких ограничења; углавном у вези са плутајућим објектима у општем случају (погледајте [одељак 13.1](#)) и нарочито са табелама ([одељак 13.3](#)) чак и када нису пливајући објекти.

Што се тиче дозвољеног броја колона, ConTeXt нема теоретско ограничење. Међутим, постоје физичка ограничења која треба узети у обзир:

- Ширина папира: неограничени број колона захтева неограничену ширину папира (ако ће се документ штампати) или екрана (ако је предвиђено да се документ приказује на екрану). У пракси, када се узме у обзир *нормална* ширина величина папира које су у продаји и од којих се праве књиге, као и величина екрана компјутера, тешко је се на њих смести текст који је шири од четири или пет колона.
- Величина компјутерске меморије: ConTeXt референтно упутство истиче да *нормалним* системима (који нису ни посебно моћни ни посебно ограничених ресурса), може да се обрађује између 20 и 40 колона.

У овом одељку ћу се фокусирати на механизам више колона у специјалним пасусима или фрагментима, јер је

- Више колона као опција распореда странице већ раније објашњена (у [пододељку Б одељка 5.3.4](#)).
- Могућност коју нуде неке конструкције, као што су уређене листе или фусноте, словослагање текста у више од једне колоне, дискутована у вези са конструкцијом или окружењем о којем се ради.

12.2.1 `\startcolumns` окружење

Уобичајена процедура за уметање фрагмената састављених од неколико колона у документ је да се употреби `columns` окружење чији је формат:

```
\startcolumns[Конфигурација] ... \stopcolumns
```

где нам *Конфигурација* омогућава да контролишемо многе аспекте окружења. Можемо да наведемо жељену конфигурацију сваки пут када позивамо окружење, или да прилагодимо подразумевано понашање окружења за све позиве окружења, што може да се постигне са

```
\setupcolumns[Configuration]
```

Конфигурационе опције су исте у оба случаја. Најважније од њих, поређане по функцији, су следеће:

- **Опције које контролишу број колона и размак између њих:**
 - `n`: контролише број колона. Ако се ово изостави, генерисаће се две колоне.
 - `nleft`, `nright`: ове опције се користе у распореду двостраног документа (погледајте [пододељак А одељка 5.3.4](#)), да се редом успостави број колона на левим (парним) и десним (непарним) страницама.
 - `distance`: размак између колона.
 - `separator`: одређује шта чини граничник између колона. Може бити `space` (подразумевана вредност) или `rule`, у ком случају се генерише линија (`rule`) између колона. У случају да се између колона поставља линија, она може да се конфигурише са следеће две опције:
 - ★ `rulecolor`: боја линије.
 - ★ `rulethickness`: дебљина линије.
 - `maxwidth`: максимална ширина коју колоне могу имати + размак између њих.
- **Опције које контролишу дистрибуцију текста по колонама:**
 - `balance`: `ConTeXt` подразумевано *балансира* колоне, што значи да текст распоређује по њима на такав начин да колоне имају мање више исту количину текста. Међутим,

ову опцију можемо да подесимо на по тако да текст не почне у колони ако претходна није пуна.

- `direction`: одређује у ком смеру се текст распоређује по колонама. Подразумевано се следи природни редослед читања (с лева на десно), али ако овој опцији доделимо вредност `reverse` распоређиваће се с десна у лево.
- **Опције које утичу на слагање текста унутар окружења:**
 - `tolerance`: текст који се пише у више од једне колоне подразумева да је ширина линије колоне мања, па као што је објашњено у опису механизма који ConTeXt користи за конструисање линија (одељак 11.3), то отежава лоцирање оптималних тачака за уметање прелома линије. Ова опција нам омогућава да привремено изменимо хоризонталну толеранцију унутар окружења (погледајте одељак 11.3.3), и тако помогнемо словослагање текста.
 - `align`: контролише хоризонтално поравнање линија унутар окружења. Може имати било коју од следећих вредности: `right`, `flushright`, `left`, `flushleft`, `inner`, `flushinner`, `outer`, `flushouter`, `middle` или `broad`. Што се тиче значења ових вредности, погледајте одељак 11.6.1.
 - `color`: наводи име боје којом ће се штампати текст унутар окружења.

12.2.2 Паралелни пасуси

Паралелни пасуси су посебан начин композиције у више колона. У овој врсти слагања текст се распоређује у две колоне (обично, мада понекад и у више од две), али не може слободно да тече између њих, већ се уместо тога одржава стриктна контрола онога шта се појављује у свакој колони. Ово је веома корисно, на пример, за генерисање докумената који пореде две верзије текста, као што су нова и стара верзија недавно измењеног закона, или у двојезичним издањима; или за писање речника за специфичне дефиниције текста у којима се текст који се дефинише појављује на левој, а дефиниција на десној страни, итд.

Обично бисмо користили механизам табела да обрадимо овакве врсте пасуса; али то је зато што већина текст процесора није тако моћна као ConTeXt. Он поседује команде `\defineparagraphs` и `\setupparagraphs` које изграђују овакве врсте пасуса употребом механизма колоне, а који је, мада има своја ограничења, флексибилнији од механизма табела.

Колико је мени познато, ова врста пасуса нема посебно име. Назвао сам их „паралелни пасуси” јер ми се то чини много сликовитији израз од онога који ConTeXt користи за њих: „*paragraphs*”.

Овде је основна команда `\defineparagraphs` чија је синтакса:

`\defineparagraphs[Име][Конфигурација]`

где је *Име* име које се даје овој конструкцији, а *Конфигурација* представља особине које ће имати, а које и касније могу да се подесе помоћу

`\setupparagraphs[Име][Колона][Конфигурација]`

где је *Име* које је било дато кад се креирала, *Колона* није обавезан аргумент који нам омогућава да конфигуришемо одређену колону, и *Конфигурација* нам омогућава да одредимо како ради у пракси.

На пример:

```
\defineparagraphs
[MursijaCinjenice]
[n=3, before={\blank},after={\blank}]

\setupparagraphs
[MursijaCinjenice][1]
[width=.1\textwidth, style=bold]

\setupparagraphs
[MursijaCinjenice][2]
[width=.4\textwidth]
```

Горњи фрагмент креира окружење у три колоне под називом `MursijaCinjenice`, па затим подешава да прва колона заузима до 10 процената ширине линије. Пошто се трећа колона не конфигурише, заузимаће преосталу ширину, тј. 50%.

Кад се окружење креира, можемо га употребити да напишемо кратку историју града Мурсија:

```
\startMursijaCinjenice
825
\MursijaCinjenice
Основан град Мурсија.
\MursijaCinjenice
Почеци града Мурсија нису сасвим јасни, али постоји доказ да је године
825. Емир од ал-Андалуса Абдерман II наредио оснивање града под именом
Мадина (или Медина) Мурсија, вероватно саграђеног над много старијом
насеобином.
\stopMursijaCinjenice
```

825 Основан град Мурсија.

Почеци града Мурсија нису сасвим јасни, али постоји доказ да је године 825. Емир од ал-Андалуса Абдерман II наредио оснивање града под именом Мадина (или Медина) Мурсија, вероватно саграђеног над много старијом насеобином.

Да смо желели наставити са причом о Мурсији, за нови догађај би била потребна нова инстанца окружења (`\startMursijaCinjenice`), јер овим механизмом не може да се уметне неколико *редова*.

Желео бих да истакнем следеће детаље у вези са овим примером:

- Једном када се окружење креира, рецимо са `\defineparagraphs[BrankoKockica]`, то постаје нормално окружење које почиње са `\startBrankoKockica` и које се завршава са `\stopBrankoKockica`.
- Креира се и команда `\BrankoKockica`, која се користи унутар окружења онда када желимо назначити да треба променити колону.

Што се тиче опција конфигурације за паралелне пасусе (`\setupparagraphs`), јасно ми је да је у овој фази увода, узевши у обзир пример који сам управо представио, читалац већ припремљен да сам одреди сврху сваке од опција, тако да ћу испод навести само имена и врсту опција, а тамо где је то битно, и могуће вредности. Ипак, упамтите да `\setupparagraphs` [Име] [Конфигурација] поставља конфигурацију која утиче на комплетно окружење, док се `\setupparagraphs` [Име] [БрКол] [Конфигурација] наводи конфигурацију која се примењује само на дату колону.

- | | | |
|-------------------------------------|---|--|
| • <code>n</code> : број | • <code>align</code> : изведено из <code>\setupalign</code> | • <code>rulecolor</code> : боја линије |
| • <code>before</code> : команда | • <code>inner</code> : команда | • <code>style</code> : команда стила |
| • <code>after</code> : команда | • <code>rule</code> : on off | • <code>color</code> : боја |
| • <code>width</code> : димензија | • <code>rulethickness</code> : димензија | |
| • <code>distance</code> : димензија | | |

Ова листа опција није комплетна; нисам навео опције које овде не бих објашњавао. Такође сам искористио чињеницу да се налазимо у одељку посвећеном колонама, па сам листу опција навео у три колоне, мада то нисам урадио употребом било које команде објашњене у овом одељку, већ помоћу `columns` опције `itemize` окружења, којем је посвећен следећи одељак.

12.3 Уређене листе

Када се информације приказују на уређени начин, читалац их лакше прихвата. Али ако је уређење и визуелно уочљиво, оно онда читаоцу истиче чињеницу да је пред њим уређени текст. То је разлог што одређене *конструкције* или *механизми* покушавају да истакну визуелну организацију текста, чиме се доприноси самој структури текста. Од свих алата које за ову сврху ConTeXt нуди аутору, најважнији је `itemize` окружење. Оно је и тема овог одељка, и помоћу њега се развија оно што би могли да назовемо *уређене листе*.

Листе се састоје од низа *текстуалних елемената* (које ћу називати *сјавке*), испред којих се налази карактер који помаже да се ставка истакне од осталих и који ћемо називати „граничник”. Граничник може бити број, слово или симбол. Обично су (мада не и увек) *сјавке* пасуси, а листа се форматира тако да се обезбеди *видљивост* граничника сваког елемента; обично примењивањем висећег увлачења које га истиче¹. У случају угнеждених листи, увлачење сваке се постепено повећава. У HTML језику се листе у којима је граничник број или карактер који се редно увећава називају *уређене листе*, што значи да ће свака *сјавка* листе имати различити граничник који нам омогућава да на сваки елемент указујемо бројем или идентификатором; а онима у којима се за сваку ставку користи исти карактер или симбол назива *неуређене листе*.

ConTeXt аутоматски управља азбучним ређањем граничника у нумерисаним листама, као и увлачењем потребним за угнеждане листе; а у случају угнежавања неуређених листи, такође се брине о избору различитог карактера или симбола који омогућава да се ниво *сјавке* у листи на први поглед разликује од симбола који му претходи.

У референтном упутству се каже да је максимални ниво угнежавања у листама 4, али претпостављам да је то био случај када је упутство написано, у 2013. години. Према мојим тестовима, изгледа да нема ограничења

¹ У типографији се увлачење примењено на све линије пасуса осим на прву назива *висеће увлачење*, чиме се лако проналази прва реч или први карактер пасуса.

угнеждавања уређених листи (моји тестови су достигли до 15 нивоа угнеждавања). Мада изгледа да нема ограничења ни за неуређене листе, у смислу да није битно колико нивоа отворимо, грешка се не генерише; али за неуређене листе ConTeXt примењује подразумеване симболе само за првих девет нивоа угнеждавања.

У сваком случају, требало би истаћи да превелика употреба угнеждавања у листама може имати супротан ефекат од оног који желимо, а то је да се читалац осећа изгубљен, да не може пронаћи сваку ставку у општој структури листе. Из овог разлога лично сматрам да иако су листе моћан алат за структурно уређење текста, скоро никада не би требало да се иде даље од трећег нивоа угнеждавања; па чак и трећи ниво треба да се користи само у одређеним случајевима када је то оправдано.

Општи алат за писање листи у систему ConTeXt је `\itemize` окружење чија је синтакса:

```
\startitemize[Опције][Конфигурација] ... \stopitemize
```

при чему ова два аргумента нису обавезна. Први прихвата симболичка имена којима је ConTeXt дефинисао прецизно значење; други аргумент, који се ретко и користи, омогућава да се одређеним променљивама доделе одговарајуће вредности и да се на тај начин утиче на рад окружења.

12.3.1 Избор врсте листе и граничник између *сјавки*

А. Неуређене листе

Листа генерисана помоћу `itemize` је подразумевано неуређена листа у којој ће се граничник аутоматски изабрати у зависности од нивоа угнеждавања:

- | | |
|--------------------------------|-------------------------------|
| • За први ниво угнеждавања. | ○ За шести ниво угнеждавања. |
| – За други ниво угнеждавања. | ○ За седми ниво угнеждавања. |
| ★ За трећи ниво угнеждавања. | □ За осми ниво угнеждавања. |
| ▷ За четврти ниво угнеждавања. | ✓ За девети ниво угнеждавања. |
| ◦ За пети ниво угнеждавања. | |

Међутим, можемо директно да наведемо симбол који желимо да се користи на одређеном нивоу тако што као аргумент наведемо и број нивоа. Тако ће `\startitemize[4]` да генерише неуређену листу у којој ће се као граничник користити карактер `▷` без обзира на ниво угнеждавања листе.

Предефинисани симбол за сваки ниво можемо да променимо командом `\definesymbol`:

```
\definesymbol[Ниво]{Симбол придружен нивоу}
```

На пример

```
\definesymbol[1][\diamond]
```

подешава да први ниво неуређених листи користи симбол `◊`. Истом командом можемо да доделимо симболе и нивоима угнеждавања већим од деветог. Тако на пример

```
\definesymbol[10][\copyright]
```

за ниво угнеждавања 10 поставља међународни симбол *ауторских права*: ©.

Б. Уређене листе

Када хоћемо да генеришемо уређену листу, `itemize` окружењу морамо навести врсту уређења коју желимо. То може бити:

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 , ...				

Разлика између `n` и `m` је у фонту који се користи да се представи број: `n` користи фонт активан у том тренутку, док `m` користи различити, елегантнији, скоро калиграфски фонт.¹

Није ми познато име фонта који се користи за `m`, па зато у горњој листи нисам успео да тачно представим врсту бројева које генерише ова опција. Читаоцима препоручујем да је сами испробају.¹

12.3.2 Унос ставки у листу

Ставке у листи се по правилу креирају помоћу `\startitemize` и уносе командом `\item`, која такође има и верзију у облику окружења, што је више у Mark IV стилу: `\startitem... \stopitem`. Тако наредни пример:

<pre>\startitemize[a, packed] \startitem Први елемент \stopitem \startitem Други елемент \stopitem \startitem Трећи елемент \stopitem \stopitemize</pre>	<pre>a. Први елемент b. Други елемент c. Трећи елемент</pre>
--	--

креирати потпуно исти резултат као и

<pre>\startitemize[a, packed] \item Први елемент \item Други елемент \item Трећи елемент \stopitemize</pre>	<pre>a. Први елемент b. Други елемент c. Трећи елемент</pre>
---	--

`\item` или `\startitem` је *ошћѝа* команда за уметање ставке у листу. Уз њу постоје и следеће додатне команде када желимо постићи специјални резултат:

`\head` Ова команда би требало да се користи уместо команде `\item` када не желимо да се након те ставке уметне прелом линије.

Уобичајена конструкција је да се непосредно након елемента листе уметне угнеждена листа или блок текста, тако да елемент листе у суштини служи као *наслов* подлисте или блока текста. У овим случајевима се не саветује уметање прелома странице након тог елемента и наредних пасуса. Ако за уметање ових елемената уместо `\item` употребимо `\head` ConTeXt ће се *максимално потрудити* да такав елемент не раздвоји од наредног блока.

`\sym` Команда `\sym{Текст}` уметне ставку у којој се као *граничник* уместо броја или симбола користи текст наведен као аргумент команде. Ова листа се дакле конструише од ставки које се умећу са `\sym` уместо са `\item`.

¹ Опција `m` укључује такозване *oldstyle* бројеве. То је стилска алтернатива уграђена у неке фонтове (*општ* OpenType особина) која може да се укључи командом `\os`. — *прим. прев.*

`\sub` Ова команда би требало да се користи само у уређеним листама (где се испред сваке ставке налази број или слово у азбучном редоследу). Она чини да ставка која се њом уноси задржи број претходне ставке, а да би се назначило да понављање броја није грешка, са леве стране се штампа знак '+'. Ово може бити корисно ако указујемо на претходну листу за коју предлажемо измене, али да би све било јасно, задржава се нумерација оригиналне листе.

`\mar` Ова команда одржава нумерацију ставки, али у маргину додаје слово или карактер (који јој се прослеђује као аргумент, унутар витичастих заграда). Нисам сигуран колико је корисна.

Постоје још две команде за унос ставки чија комбинација прави врло *интересантне* ефекте и, ако тако могу да кажем, мислим да је боље објаснити их кроз пример. `\ran` (скраћеница од *range* – опсег) и `\its`, скраћеница од *items* – ставке. Прва прихвата један аргумент (унутар витичастих заграда), а друга понавља симбол који се користи као граничник у листи *x* пута (подразумевано 4, али то можемо променити употребом опције *items*). Следећи пример показује како ове две команде могу заједно да креирају нешто што подсећа на упитник:

Након читања увода, одговорите на следећа питања:

```
\startitemize[5, packed][width=8em, distance=2em, items=5]
\ran{Не \hss Да}
\its Никада нећу користити \ConTeXt, сувише је тежак.
\its Користићу га само за писање великих књига.
\its Користићу га увек.
\its Толико ми се свиђа да ћу свом следећем детету дати име
\quotation{Ханс}, у част Ханса Хагена.
\stopitemize
```

Након читања увода, одговорите на следећа питања:

Не	Да	
○ ○ ○ ○ ○		Никада нећу користити ConTeXt, сувише је тежак.
○ ○ ○ ○ ○		Користићу га само за писање великих књига.
○ ○ ○ ○ ○		Користићу га увек.
○ ○ ○ ○ ○		Толико ми се свиђа да ћу свом следећем детету дати име „Ханс”, у част Ханса Хагена.

12.3.3 Основна конфигурација листе

Присетимо се да „*itemize*” прихвата два аргумента. Већ смо видели како нам први аргумент омогућава да изаберемо жељену врсту листе. Али можемо да га употребимо и за навођење осталих карактеристика листе; то се ради помоћу наредних опција окружења „*itemize*” у његовом првом аргументу:

- `columns`: ова опција одређује да се листа прави у две или више колона. Након опције `columns` мора да се наведе жељени број колона као речи раздвојене запетама: `two`, `three`,

four, five, six, seven, eight или nine. Ако се иза columns не наведе ниједан број, генерисаће се две колоне.

- `intro`: ова опција покушава да листу не раздвоји преломом линије од пасуса испред листе.
- `continue`: у уређеним листама (нумерисаним или азбучним) ова опција чини да листа настави са нумерацијом од последње нумерисане листе. Ако се употреби `continue` опција, није потребно да се наведе жељени тип листе јер се претпоставља да ће бити иста као последња нумерисана листа.
- `racked`: је једна од најчешће коришћених опција. Чини да се вертикални размак између *сјавки* листе смањи колико год је то могуће.
- `nowhite`: има сличан ефекат као и `racked`, али још израженији: не само да смањује вертикални размак између ставки, већ и вертикални размак између околног текста.
- `broad`: повећава хоризонтални размак између граничника ставке и текста ставке. Простор може да се повећа множењем броја са `broad`, на пример `\startitemize[2*broad]`.
- `serried`: уклања хоризонтални размак између граничника ставке и текста.
- `intext`: уклања висеће увлачење.
- `text`: уклања висеће увлачење и смањује вертикални размак између ставки.
- `repeat`: чини да ниво детета у угнежденим листама *јонавља* исти ниво као претходни. Тако бисмо имали на првом нивоу: 1, 2, 3, 4; на другом нивоу: 1.1, 1.2, 1.3, итд. Опција мора да се наведе за унутрашњу листу, не за спољашњу.
- `margin`, `inmargin`: граничник се подразумевано штампа на левој страни, али унутар саме области текста (`atmargin`). Опције `margin` и `inmargin` померају граничник на маргину.

12.3.4 Додатна конфигурација листе

Други аргумент у `\startitemize`, који такође није обавезан, омогућава детаљнију и опсежнију конфигурацију листи.

- `before`, `after`: команде које се редом извршавају пре отварања и након затварања `itemize` окружења.
- `inbetween`: команда која се извршава између две *сјавке*.
- `beforehead`, `afterhead`: команда која се извршава пре или након ставке која се уноси командом `\head`.
- `left`, `right`: карактер који се штампа лево или десно од граничника. На пример, ако желимо да добијемо азбучну листу у којима су слова унутар заграда, треба да напишемо:

```
\startitemize[a][left=(, right=)]
```

- `stopper`: задаје карактер који треба да се испише након граничника. Функционише само у уређеним листама.
- `width`, `maxwidth`: ширина простора резервисаног за гранични, па дакле и за viseће увлачење.
- `factor`: репрезентативни број фактора раздвајања између граничника и текста.
- `distance`: мера растојања између граничника и текста.
- `leftmargin`, `rightmargin`, `margin`: маргина која се додаје лево (`leftmargin`) или десно (`rightmargin`) од ставки.
- `start`: број којим почиње нумерација ставки.
- `symalign`, `itemalign`, `align`: поравнање ставки. Прихвата исте вредности као и `\setupalign`. `symalign` контролише поравнање граничника; `itemalign` текста ставке, а `align` поравнање оба.
- `indenting`: увлачење прве линије пасуса унутар окружења. Погледајте одељак 11.1.1
- `indentnext`: наводи да ли пасус након окружења треба да се увуче. Вредности су *yes*, *no* и *auto*.
- `items`: за ставке унете командом `\its`, наводи колико се пута понавља сепаратор.
- `style`, `color`; `headstyle`, `headcolor`; `marstyle`, `marcolor`; `symstyle`, `symcolor`: ове опције контролишу стил и боју ставки док се уносе у окружење командама `\item`, `\head`, `\mar` или `\sym`.

12.3.5 Просте листе са `\items` командом

Алтернатива `itemize` окружењу за врло просте ненумерисане листе у којима број ставки није сувише велики је команда `\items` чија је синтакса:

`\items[Конфигурација]{Ставка 1, Ставка 2, ..., Ставка n}`

Различите ставке у листи се раздвајају запетама. На пример:

Графички фајлови могу имати,
између осталих, и следеће екстензије:

`\items{png, jpg, tiff, bmp}`

Графички фајлови могу имати, између осталих, и следеће екстензије:

- `png`
- `jpg`
- `tiff`
- `bmp`

Конфигурационе опције које подржава ова команда су подскуп оних за `itemize`, осим две које су специфичне за ову команду:

- `symbol`: ова опција одређује врсту листе која ће се генерисати. Подржава исте вредности за избор врсте листе као и `itemize`.

- `n`: ова опција наводи од које ставке ће почети граничник.

12.3.6 Предодређивање понашања листе и креирање сопствене врсте листе

У претходним одељцима смо видели како се наводи жељена врста листе и какве су њене карактеристике. Али то није ефикасно ако се ради при сваком позиву листе и обично је последица недоследан документ у којем свака листа изгледа другачије, а ниједан од тих изгледа не испуњава било какав критеријум.

Зато би требало:

- Да се предодреди *нормално* понашање окружења `itemize` и команде `\items` у преамбули документа.
- Да креирамо сопствене прилагођене листе. На пример: азбучно обележену листу желимо да назовемо *ListAlpha*, листу која се набраја римским бројевима *ListRoman*), итд.

Прво постижемо командама `\setupitemize` и `\setupitems`. Друго захтева да се употреби или команда `\defineitemgroup`, или команда `\defineitems`. Прва ће креирати окружење листе које личи на `itemize`, а друга команду сличну команди `items`.

12.4 Описи и набрајања

Описи и набрајања су две конструкције које обезбеђују доследно словослагање пасуса или група пасуса које развијају, описују или дефинишу фразу или реч.

12.4.1 Описи

Код описа разликујемо *наслов* и његово објашњење или развој. Опис можемо креирати са:

```
\definedescription[Име] [Конфигурација]
```

где је *Име* име којим називамо ову нову конструкцију, а Конфигурација контролише изглед наше нове структуре. Кад се изврши претходна наредба имаћемо нову команду и окружење са изабраним именом. Дакле:

```
\definedescription[Concept]
```

ће креирати команде:

```
\Concept{Наслов}
\startConcept {Наслов} ... \stopConcept
```

Команду ћемо користити у случају када се текст који објашњава наслов састоји само од једног пасуса, а окружење за наслове чији опис заузима више од једног пасуса. Када се користи команда, пасус који јој непосредно следи је једини који се сматра за опис наслова. Али када се користи окружење, сав садржај у њему ће се форматирати одговарајућим увлачењем како би било јасно да је у вези са насловом.

На пример:

```
\definedescription
[Concept]
[alternative=left, width=3.5cm, headstyle=bold]

\Concept{Контекстуализовање}
```

Постављање нечега у одређени контекст, или слагање текста словослагачким системом под називом `\ConTeXt`. Могућност исправне контекстуализације у било којој ситуацији се сматра знаком интелигенције и доброг опажања.

ће да генерише следећи резултат:

Контекстуализовање Постављање нечега у одређени контекст, или слагање текста словослагачким системом под називом `\ConTeXt`. Могућност исправне контекстуализације у било којој ситуацији се сматра знаком интелигенције и доброг опажања.

Као што је обичај у систему `\ConTeXt`, изглед наше нове конструкције ће се задати аргументом *Конфигурација* у тренутку креирања, или касније са `\setupdescription`:

```
\setupdescription[Име] [Конфигурација]
```

где *Име* представља име нашег новог описа, а *Конфигурација* одређује његов изглед. Међу разним опцијама конфигурацију желим да истакнем следеће:

- `alternative`: оно је опција која суштински контролише изглед конструкције. Она одређује позиционирање наслова у односу на његов опис. Могуће вредности су `left`, `right`, `inmargin`, `inleft`, `inright`, `margin`, `leftmargin`, `rightmargin`, `innermargin`, `outermargin`, `serried`, `hanging` и њихова имена су довољно јасна да се може стећи идеја резултата, мада, у случају сумње, најбоље је урадити тест и видети какав изглед се постиже.
- `width`: контролише ширину кутије у којој ће се исписати наслов. У зависности од вредности опције `alternative`, та величина такође може бити и део увлачења којим ће се исписати текст објашњења.
- `distance`: контролише размак између наслова и објашњења.
- `headstyle`, `headcolor`, `headcommand`: утичу како ће изгледати сам наслов: стил (`headstyle`) и боја (`headcolor`). Помоћу `headcommand` можемо назначити команду којој ће се текст наслова проследити као аргумент. На пример: `headcommand=\WORD` ће обезбедити да се текст наслова испише у верзалу.
- `style`, `color`: контролише изглед текста описа наслова.

12.4.2 Набрајања

Набрајања су нумерисани текст елементи организовани у неколико нивоа. Сваки елемент почиње насловом који се подразумевано састоји од имена структуре и њеног броја, мада

наслов можемо променити опцијом `text`. Она су доста слична описима, мада се разликују у следећем:

- Сви елементи у набрајању деле исти наслов.
- Стога се међусобно разликују по својој нумерацији.

Ова структура може бити веома корисна, на пример, за писање формула, проблема или вежби у уџбеницима, јер обезбеђује да се нумеришу како треба и форматирају на доследан начин.

Набрајање можемо да креирамо помоћу

```
\defineenumeration[Име] [Конфигурација]
```

где *Име* представља име нове конструкције, а *Конфигурација* контролише њен изглед.

Тако смо у наредном примеру:

```
\defineenumeration
[Exercise]
[alternative=top, before=\blank, after=\blank, between=\blank]
```

креирали нову структуру под називом *Exercise* и, као што се дешава са набрајањима, имаћемо на располагању следеће две команде:

```
\Exercise
\startExercise
```

Команда се користи само за *exercise* (вежбу) која се састоји од једног пасуса, а окружење за *вежбе* у више пасуса. Али пошто набрајања могу ићи до четвртог нивоа дубине, креираће се и следеће команде и окружења:

```
\subExercise
\startsubExercise
\stopsubExercise
\subsubExercise
\startsubsubExercise
\stopsubsubExercise
\subsubsubExercise
\startsubsubsubExercise
\stopsubsubsubExercise
```

А за контролу нумерације и следеће додатне команде:

- `\setИмеНабрајања`: поставља текућу вредност нумерације.
- `\resetИмеНабрајања`: поставља бројач набрајања на нулу.
- `\nextИмеНабрајања`: увећава бројач набрајања за један.

Изглед набрајања може да се одреди у време креирања, или касније командом `\setupenumeration` чији је формат:

`\setupenumeration[Име] [Конфигурација]`.

Сваком набрајању можемо посебно да конфигуришемо сваки његов ниво. Тако на пример, `\setupenumeration [subExercise] [Конфигурација]` утиче на други ниво набрајања под називом „Exercise”.

Опције и вредности које могу да се конфигуришу са `\setupenumeration` су слична онима у `\setupdescription`.

12.5 Линије и оквири

У ConTeXt референтном упутству се каже да TeX поседује велики капацитет за управљање текстом, али је веома слаб у управљању графичким информацијама. Молим за разлику: тачно је да могућности система ConTeXt (уствари TeX) нису тако надмоћне као када је у питању словослагање текста. Али ићи дотле да се каже како је систем слаб у овом погледу је по мом мишљењу, помало натегнуто. Не знам ниједну функцију са линијама и оквирима које остали словослагачки системи могу да ураде, а да и ConTeXt не може да генерише. А ако ConTeXt упаримо са METAPOST, или са TikZ (ConTeXt има модул проширења за то), онда је само машта граница.

Међутим, у наредним одељцима ћу се ограничити на објашњавање начина за генерисање једноставних хоризонталних и вертикалних линија, као и оквира око речи, реченица или пасуса.

12.5.1 Просте линије

Најједноставнији начин цртања хоризонталне линије је командом `\hairline`. Она генерише хоризонталну линију која заузима комплетну ширину нормалне линије текста.

У линији текста где постоји линија генерисана командом `\hairline` не може да се нађе било какав текст. Ако је потребно да се у истој линији налазе и текст и линија, морамо да употребимо команду `\thinrule`. Ова друга команда ће искористити пуну ширину линије. Тако да ће у изолованом пасусу да има исти ефекат као и `\hairline`, док ће у супротном случају `\thinrule` да произведе исто хоризонтално простирање као и `\hfill` (погледајте одељак 10.3.3), али уместо испуњавања хоризонталног простора празнином (као што то ради `\hfill`), она га испуњава линијом.

```
Са леве\thinrule\\
\thinrule Са десне\\
Са обе\thinrule стране\\
\thinrule у центру\thinrule
```

```
Са леве_____Са десне
_____Са десне
Са обе_____стране
_____у центру_____
```

Командом `\thinrules` можемо да генеришемо неколико линија. `\thinrules[n=2]` ће на пример да генерише две узастопне линије, од којих је свака ширине нормалне линије текста. Линије генерисане командом `\thinrules` такође могу да се конфигуришу, било у самом позиву команде, навођењем конфигурације као један од њених аргумената, или уопштено са `\setupthinrules`. Конфигурација укључује дебљину линије (`rulethickness`), њену боју (`color`), боју позадине (`background`), проред између линија (`interlinespace`), итд.

Већи број опција остављам без објашњења. Читалац може да консултује `setup-en.pdf` (погледајте [одељак 3.6](#)). Неке опције се од других разликују у нијансама (тј. једва да има разлике међу њима), па мислим да је читаоцу брже да покуша *видети* разлику, него да је ја покушам пренети речима. На пример: дебљина линије коју сам управо поменуо зависи од опције `rulethickness`. Али на њу такође утичу и опције `height` и `depth`.

Мање линије могу да се генеришу командама `\hl` и `\vl`. Прва генерише хоризонталну линију, а друга вертикалну. Обе прихватају број као параметар који нам омогућава да израчунамо дужину линије. У `\hl` број је мера дужине у *емовима* (није потребно да се јединица мере наводи у команди), а у `\vl` аргумент представља текућу висину линије.

Тако `\hl[3]` генерише хоризонталну линију дужине 3 ема, а `\vl[3]` генерише вертикалну линију чија дужина одговара висини три линије текста. Упамтите да број мере мора да се уметне у велике заграде, а не у витичасте. У обе команде аргумент није обавезан. Ако се не наведе, узима се да је вредност 1.

`\fillinline` је још једна команда за креирање хоризонталних линија прецизне дужине. Она прихвата више подешавања којима можемо навести (или предодредити са `\setupfillinlines`) ширину (опција `width`) уз још неке особине.

Особеност ове команде је да ће се текст написан са њене десне стране поставити лево од генерисане линије и одвојиће се потребним празним простором тако да се заузме комплетна линија текста. На пример:

```
\fillinline[width=6cm] Име
```

ће да генерише

Име



Претпостављам да је ово чудно понашање услед чињенице да је овај макро дизајниран за писање формулара у којима постоји хоризонтална линија иза текста на којој нешто треба да се напише. У суштини, само име команде `fillinline` значи попуни у линији.

Осим ширине линије, можемо да конфигуришемо и маргину (`margin`), растојање (`distance`), дебљину (`rulethickness`) и боју (`color`).

`\fillinrules` је скоро идентична са `\fillinline`, мада нам ова команда омогућава да унесемо више од једне линије (опцијом „`n`”).

```
\fillinrules[Конфигурација] {Текст} {Текст}
```

при чему сва три аргумента нису обавезна.

12.5.2 Линије везане за текст

Мада неке од команди које смо управо видели могу генерисати линије које стоје уз текст у истој линији, оне се у суштини концентришу на изглед линије. Ако желимо да цртамо линије везане за одређени текст, `ConTeXt` има команде:

- које генеришу текст између линија.
- које генеришу линије испод текста (подвлачење), изнад текста (надвлачење) или кроз текст (прецртавање).

Уобичајена команда за генерисање текста између линија је `\textrule`. Ова команда исцртава линију која пролази кроз целу ширину странице и исписује текст који јој се проследи као параметар са леве стране (али не на маргини). На пример:

```
\textrule{Пример тест}
```

— Пример тест —



Претпоставља се да `\textrule` прихвата необавезни први аргумент који може имати три вредности: `top`, `middle` и `bottom`. Али након неких тестова, нисам успео да пронађем који је ефекат те три опције.

Команди `\textrule` је слично `\starttextrule` окружење које, осим што умеће линију са текстом на почетку окружења, умеће и хоризонталну линију на крај. Формат ове команде је следећи:

```
\starttextrule[Конфигурација]{Текст на линији} ... \stoptextrule
```

И `\textrule` и `\starttextrule` могу да се конфигуришу помоћу `\setuptextrule`.

За цртање линија испод, изнад, или кроз текст, користе се следеће команде:

```
\underbar{Текст}
\underbars{Текст}
\overbar{Текст}
\overbars{Текст}
\overstrike{Текст}
\overstrikes{Текст}
```

Као што можемо видети, за сваки тип линије (испод, изнад, или кроз текст) постоје две команде. Верзија команде у једнини црта једну линију испод, изнад или кроз текст који јој се проследи као аргумент, док верзија команде у множини црта само линију изнад речи, али не и изнад размака.

Ове команде нису компатибилне међусобно, што значи да било које две од њих не могу да се примене на исти текст. Ако то ипак покушамо, примениће се она која је задата последња. С друге стране, `\underbar` може да се угнеждава, па подвлачи оно што је већ било подвучено.



Референтно упутство истиче да `\underbar` искључује растављање речи на крају реда речи које чине њен аргумент. Није ми сасвим јасно да ли се тај исказ односи само на `\underbar` или на свих шест команди које испитујемо.

12.5.3 Уоквирене речи или текстови

Када желимо да уоквиримо текст оквиrom или мрежом, користимо:

- Команде `\framed` или `\inframed` ако је текст релативно кратак и не заузима више од једне линије.
- Окружење `\startframedtext` за дуже текстове.

Разлика између `\framed` и `\inframed` је у тачки из које се исцртава оквир. `\framed` исцртава оквир навише од идеалне линије која се зове основна линија, на коју се постављају

слова, мада нека слова пролазе и испод ње. `\inframed` оквир такође исцртава навише, али од најниже могуће тачке у линији. На пример:

Овде имамо `\framed{два}` добра `\inframed{оквира}`.

Овде имамо `\framed{два}` добра `\inframed{оквира}`.

И `framed` и `inframed` текст може да се прилагоди командом `\setupframed`, а `\startframedtext` се прилагођава помоћу `\setupframedtext`. Опције прилагођавања су прилично сличне за обе команде. Оне нам омогућавају да наведемо димензије оквира (`height`, `width`, `depth`), облик углова (`framecorner`), који може бити `rectangular` (правоугаони) или заобљен (`round`), боју оквира (`framecolor`), дебљину линије (`framethickness`), поравнање садржаја (`align`), боју текста (`foregroundcolor`), боју позадине (`background` и `backgroundcolor`), итд.

За `\startframedtext` постоји и привидно чудна особина: `frame=off` која чини да се оквир не исцртава (мада је још увек ту, само невидљив). Ова особина постоји из разлога што је оквир око пасуса недељив, па је уобичајено да се цео пасус смести у `framedtext` окружење и искључи опција исцртавања оквира, чиме се обезбеђује да унутар пасуса не може да се уметне прелом странице.

Такође можемо да креирамо и прилагођене верзије ових команди помоћу `\defineframed` и `\defineframedtext`.

12.6 Остала интересантна окружења и конструкције

У систему ConTeXt постоји још доста окружења која уопште нисам ни поменуо, или сам их само упут поменуо. Ево примера:

- **buffer** *Бафери* су фрагменти текста који се чувају у меморији за каснију поновну употребу. *Бафер* се дефинише негде у документу са `\startbuffer[ИмеБафера] ... \stopbuffer` и може да се примени на неком другом месту у документу командом `\getbuffer[ИмеБафера]`, онолико пута колико је потребно.
- **chemical** Ово окружење нам омогућава да у њега сместимо хемијске формуле. Како се TeX истиче, између многих осталих ствари, у својој могућности да исправно слаже текстове са математичким формулама, од свог настанка ConTeXt је тежио да ову могућност прошири и на хемијске формуле, па поседује ово окружење у којем постоје команде и структуре прилагођене писању хемијских формула.
- **combination** Ово окружење нам омогућава да комбинујемо неколико плутајућих елемената на исту страну. Посебно је корисно за поравнање разних спољних слика које су у вези једна са другом.
- **formula** Ово је окружење намењено слагању математичких формула.
- **hiding** Текст постављен у ово окружење се неће компајлирати, па се стога неће појавити ни у финалном документу. Ово је корисно да се привремено искључи компајлирање

одређених делова изворног фајла. Иста ствар се постиже означавањем једне или више линија као коментар. Али када је фрагмент који желимо да искључимо релативно дугачак, уместо означавања десетина или стотина линија изворног фајла као коментар, боље је да се на почетак фрагмента уметне команда `\starthiding`, а на крај `\stophiding`.

- **legend** У математичком контексту, $\text{T}_{\text{E}}\text{X}$ примењује другачија правила тако да није могуће писање обичног текста. Међутим, понекад уз формулу иде опис њених елемената. За ову потребу постоји `\startlegend` окружење које нам омогућава да у математички контекст уметнемо обичан текст.
- **linecorrection** $\text{ConT}_{\text{E}}\text{Xt}$ обично на исправан начин управља вертикалним размаком између линија, али повремено линија може да садржи нешто због чега не изгледа како треба. То се углавном дешава са линијама које имају фрагменте уоквирене командом `\framed`. Ово окружење у таквим случајевима поправља проред тако да пасус изгледа како треба.
- **mode** Ово окружење је намењено за укључивање фрагмената који ће се у изворном фајлу компајлирати само ако је активан одговарајући режим. Употреба *режима* није тема овог увода, али је веома интересантан алат када је потребно да из једног изворног фајла генеришемо неколико различитих верзија са другачијим форматирањем. Уз ово окружење постоји и `\startnotmode`.
- **opposite** Ово окружење се користи за слагање текстова у којима је садржај левих и десних страница у вези један са другим.
- **quotation** Окружење врло слично са `parrower`, намењено за унос умерено дугачких дословних цитата. Окружење обезбеђује да се текст унутар њега цитира, као и да се маргине повећају тако да се пасус са цитатом визуелно истакне на страници. Али требало би напоменути да према уобичајеном стилу блок цитата у енглеском, не би требало да постоје отварајући и затварајући знаци навода – па ова команда и није толико корисна.
- **standardmakeup** Ово окружење је дизајнирано за генерисање страница са насловима документа, што је релативно уобичајено у академским документима одређене дужине, као што су докторске дисертације, мастер тезе, итд.

Ако желите да научите више о било ком од ових окружења (или осталих које нисам поменуо), предлажем следеће кораке:

1. Погледајте информације о окружењу у $\text{ConT}_{\text{E}}\text{Xt}$ референтном упутству. Оно не помиње сва окружења; али каже нешто о свакој ставки из горње листе.
2. Напишите тест документ у којем се користи окружење.
3. Потражите у $\text{ConT}_{\text{E}}\text{Xt}$ званичној листи команди (погледајте [одељак 3.6](#)) у вези са опцијама за конфигурацију окружења о којем се ради, па их затим тестирајте да откријете шта тачно раде.

Глава 13

Слике, табеле и остали плутајући објекти

Садржај: 13.1 Шта су плутајући објекти и шта они раде?; 13.2 Спољне слике; 13.2.1 Директно уметање слика; 13.2.2 Уметање слике са `\placefigure`; 13.2.3 Уметање слика интегрисаних у текст блок; 13.2.4 Конфигурација и трансформација уметнутих слика; А Уметање опција команде које врше неке трансформације слике; Б Специфичне команде за трансформисање слике; 13.3 Табеле; 13.3.1 Општи концепти о табелама и њиховом постављању у документ; 13.3.2 Просте табеле са `tabulate` окружењем; 13.4 Заједнички аспекти слика, табела и осталих плутајућих објеката; 13.4.1 Опције за уметање плутајућих објеката; 13.4.2 Конфигурисање наслова плутајућег објекта; 13.4.3 Комбиновано уметање два или више објеката; 13.4.4 Општа конфигурација плутајућих објеката; 13.5 Дефинисање додатних плутајућих објеката;

Ово поглавље се углавном бави плутајућим објектима. Али уз овај концепт, није лоше ни да се објасне два типа објекта која нису обавезно плутајућа, мада се често конфигуришу као такви: спољне слике и табеле. Ако погледа садржај овог поглавља, неко би могао помислити да је све прилично несређено: почиње се причом о плутајућим објектима, затим се говори о сликама и табелама, па се завршава поново причом о плутајућим објектима. Разлог ове несређености је *негајошке* природе: слике и табеле могу да се објасне без превише инсистирања на чињеници да су оне обично пливајуће; а ипак, када почнемо да их истражујемо, од велике помоћи је откриће да, гле чуда, већ знамо понешто о плутајућим објектима.

13.1 Шта су плутајући објекти и шта они раде?

Ако би документ садржао само *обичан* текст, пагинација би била релативно једноставна: познајући максималну висину области текста на страници, довољно је да се измери висина различитих пасуса па да се одреди где да се уметну преломи страница. Проблем је што у многим документима постоје објекти, фрагменти или недељиви блокови текста као што су слика, табела, формула, уоквирени пасус, итд.

Понекад ови објекти могу да заузму већи део странице, што онда представља проблем да ако је потребно да се објекат уметне на неко одређено место у документу, можда неће моћи да стане на текућу странцу, па она мора нагло да се прекине, остављајући велики празан простор на дну, а објекат о коме је реч и текст након њега се померају на наредну страницу. Међутим, правила доброг словослагања говоре да на свакој страници треба да буде иста количина текста, осим на последњој страници поглавља.

Зато се не препоручује појављивање великих вертикалних празних простора; а механизам да се то и постигне су *плутајући* објекти. „Плутајући објекат” је онај који не мора да се нађе на тачно одређеном месту у документу, већ може да се *помера* или да *плута* по документу. Идеја је да се дозволи да `ConTeXt` из угла пагинације одлучи које је најбоље место за објекат, да лоцира такве објекте, па чак и да их постави на другу страницу; али увек покушавајући да га не помери предалеко од места уметања у изворном фајлу.

Стога, нема објеката који *сами њо себи* морају да буду пливајући. Али постоје објекти који ће понекада морати да буду плутајући. У сваком случају, одлука о томе је на аутору или, ако су то две различите особе, на особи одговорној за словослагање.

Несумњиво је да могућност померања тачно позиционираних недељивих објеката помаже код задатка словослагања добро уравнотежених страница; али уз то иде и проблем што у време када пишемо оригинални документ, не знамо тачно где ће објекат да заврши, па је тешко да се укаже на њега. Тако на пример, ако сам у свој документ управо поставио команду која умеће слику и у наредном пасусу желим да је опишем и напишем нешто као: „Као што можете видети из претходне слике”, када слика *џлуџа*, лако би могла да се нађе *иза* онога што сам управо написао, па резултат није доследан: читалац тражи слику *исћред* текста који указује на њу и не може да је пронађе јер је након плутања, слика завршила *иза* те референце.

Ово се исправља *нумерацијом* плутајућих објеката (након што се расподеле по категоријама), тако да уместо указивања на слику као „претходна слика” или „наредна слика”, на њу указујемо као „слика 1.3”, јер можемо користити ConTeXt механизам интерних референци (погледајте [одељак 9.2](#)). С друге стране, нумерација ове врсте објеката олакшава креирање њиховог индекса (индекс табела, графика, слика, једначина, итд.). Да бисте видели како се то ради, погледајте ([одељак 8.2](#)).

Механизам управљања плутајућим објектима у систему ConTeXt је прилично софистициран и на тренутке тако апстрактан да није баш погодан за почетнике. Зато ћу у овом поглављу почети са објашњењем два одређена случаја: слика и табела. Затим ћу покушати да то донекле генерализујем, тако да можемо разумети како да механизам проширимо и на остале врсте објеката.

13.2 Спољне слике

Као што у овом тренутку читалац већ зна (јер је објашњено у [одељку 1.5](#)), ConTeXt је савршено интегрисан са METAPOST и може да генерише слике и графику које се *програмирају* на сличан начин као што се програмирају и трансформације текста. Такође постоји и модул проширења за ConTeXt¹ који му омогућава да ради са TikZ.² Али таквим сликама се не бавимо у овом уводу (јер би то вероватно удвостручило његову дужину). Овде мислим на употребу спољних слика које се налазе у фајлу на нашем хард диску, или које ConTeXt преузима директно са интернета.

13.2.1 Директно уметање слика

За директно уметање слике (не као плутајућег објекта) користимо команду `\externalfigure` чија је синтакса

`\externalfigure[Име] [Конфигурација]`

¹ ConTeXt модули проширења обезбеђују додатне могућности, али нису део овог увода.

² Ово је графички програмски језик намењен за рад са системима базираним на Т_ЕX систему. То је „рекурзивни акроним” немачке реченице „TikZ ist keinen Zeichenprogramm” што у преводу значи: „TikZ није програм за цртање”. Рекурзивни акроними су врста програмерске шале. На страну METAPOST (који не знам како да користим), верујем да је TikZ одличан систем за програмирање графике.

где


- *Име* може да буде или име фајла у којем је слика, или веб адреса слике на Интернету, или симболичко име које смо претходно придружили слици користећи `\useexternalfigure` команду чији је формат сличан команди `\externalfigure` мада као први аргумент узима симболичко име придруженој слици о којој је реч.
- *Конфигурација* није обавезан аргумент који нам омогућава да пре уметања слике у документ применимо на њу одређене трансформације. Овај аргумент ћемо детаљније испитати у одељку 13.2.4.

Дозвољени формати слике су pdf, mps, jpg, png, jp2, jbig, jbig2, jb2, svg, eps, gif или tif. ConTeXt директно може да управља са осам од њих, док остали (svg, eps, gif или tif) морају да се конвертују спољним алатом пре отварања, који се мења у зависности од формата, па зато мора бити инсталиран на систему ако желимо да ConTeXt може манипулисати том врстом фајлова.

Међу форматима које подржава `\externalfigure` су и неки видео формати. Тачније: QuickTime (екстензија .mov), Flash Video (екстензија .flv) и MPEG 4 (екстензија .mp4). Али већина PDF прегледача не зна како да ради са PDF фајловима који у себи имају уграђени видео. О овоме не могу пуно да кажем, јер нисам обавио никакве тестове.

Нема потребе да се наводи екстензија фајла: ConTeXt ће тражити фајл са наведеним именом са једном од екстензија познатих формата слике. У случају да има неколико кандидата, ако постоји, најпре се користи PDF формат, а у његовом одсуству, MPS формат (графика генерисана са METAPOST). Ако нема ова два, редослед је следећи: jpeg, png, jpeg 2000, jbig и jbig2.

Ако се стварни формат слике не слаже са екстензијом фајла у којем је сачувана, ConTeXt не може да је отвори осим ако опцијом `method` не наведемо стварни формат слике.

Ако се слика не постави сама, ван пасуса, већ унутар текста пасуса, а њена висина је већа од прореда, линија ће се подесити тако да се спречи преклапање слике са претходним линијама, као у примеру који је дат у овој линији .

ConTeXt подразумевано претражује радни директоријум, његов родитељ директоријум и родитељ тог директоријума. Локацију директоријума који садржи слике са којима ћемо радити можемо да задамо опцијом `directory` команде `\setupexternalfigures`, која ће задати директоријум додати на путању за претрагу. Ако желимо да се претрага врши само у директоријуму са сликама, морамо да поставимо и опцију `location`. Тако на пример, ако желимо да наш документ тражи све слике у „img” директоријуму, требало би да напишемо:

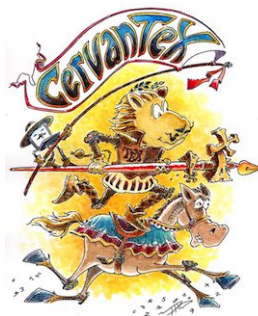
```
\setupexternalfigures
[directory=img, location=global]
```

У `directory` опцији команде `\setupexternalfigures` можемо да наведемо више директоријума, раздвајајући их запетама; али у том случају директоријуме морамо да поставимо унутар витчастих заграда. На пример „`directory={img, ~/slike}`”.

У `directory` увек користимо карактер `'/'` као граничник између директоријума; такође и у систему Microsoft Windows, који иначе као граничник директоријума користи `'\'`.

`\externalfigure` може да користи и слике које се налазе на Интернету. Тако на пример, следећи фрагмент ће у документ уметнути CervanTeX лого директно са Интернета. То је \TeX корисничка група на шпанском језику:¹

```
\externalfigure
[http://www.cervantex.es/files/
cervantex/cervanTeXcolor-small.jpg]
```



Када се први пут компајлира документ који садржи удаљени фајл, он се преузима са сервера и чува у Lua \TeX кеш директоријуму. Затим се у накнадним компајлирањима документа користи овај кеширани фајл. Удаљена слика се обично поново преузима ако је слика старија од 1 дан. Ако желите да промените ову границу, погледајте [Con \$\TeX\$ t вики](#).

Ако Con \TeX t не пронађе слику која треба да се уметне, не генерише се грешка, већ се уместо слике умеће текст блок величине слике (ако је Con \TeX t зна) или у супротном, стандардне величине. Пример за ово се налази у одељку 13.4.3.

13.2.2 Уметање слике са `\placefigure`

Слике могу и директно да се уметну. Али је пожељно да се то ради командом `\placefigure`. Ова команда чини да Con \TeX t:

- зна да слика која се умеће мора бити део листе слика у документу која, ако желимо, касније може да се употреби за прављење индекса слика.
- додели број слици, и тако омогући интерно указивање на њу.
- дода наслов слици, креирајући блок текста између слике и њеног наслова што значи да се они не могу раздвојити.
- аутоматски постави празан простор (хоризонтални и вертикални) потребан да се слика исправно види.
- постави слику на назначено место, тако да по потреби текст тече око ње.
- конвертује слику у плутајући објекат ако је то могуће, узимајући у обзир спецификацију њене величине и места.²



Синтакса ове команде је следећа:

¹ Интернет адресе су веома дугачке, па нема довољно места да се испише у примеру у две колоне. Из тог разлога сам у веб адресу уметнуо прелом реда како би могла да стане у колону. Ако неко жели да копира и налепи пример, он неће функционисати док се не обрише овај прелом линије.

² Ово друго је мој закључак, узевши у обзир да се међу опцијама постављања налазе и `force` или `split` које се косе са смислом плутајућих објеката.

`\placefigure[Опције][Лабела] {Наслов} {Слика}`

Аргументи имају следеће значење:

- *Опције* су скуп назнака које уопштено наводе где слика треба да се постави. Пошто су ове опције исте и у овој и у осталим командама, све ћу их објаснити касније (у одељку 13.4.1). За сада, користићу опцију `here` у примерима. Она систему ConTeXt говори да, колико је то могуће, постави слику тачно на позицију на којој се налази команда.
- *Лабела* је текст стринг којим се интерно указује на овај објекат, тако да можемо правити референце на њега (погледајте одељак 9.2).
- *Наслов* је текст наслова који се додаје слици.
- *Слика* је команда која умеће слику.

На пример

```
\placefigure
[here]
[fig:texknuth]
{\TeX\ лого и фотографија {\sc Кнута}}
{\externalfigure[https://i.ytimg.com/vi/8c5Rrfabr9w/maxresdefault.jpg]}
```



Слика 13.1 Т_ΕX лого и фотографија КНУТА

Као што можемо видети у овом примеру, уметањем слике (које је, иначе урађено директно са Интернета), долази до неких промена у односу на оно што се дешава када се команда `\externalfigure` користи директно. Додаје се вертикални простор, слика се центрира и додаје се наслов. То су *свољне* измене, видљиве на први поглед. Са интерне тачке гледишта, команда је такође произвела не мање важне ефекте:

- Пре свега, слика је стављена у „листу слика” објеката уметнутих у документ, коју ConTeXt интерно одржава. Ово затим значи да ће се слика појавити у индексу слика који може да се генерише помоћу `\placelist[figure]` (погледајте одељак 8.2), мада постоје и две специфичне команде за генерисање индекса слика `\placelistoffigures` или `\completelistoffigures`.
- Друго, слика је повезана са лабелом која је наведена као други аргумент команде `\placefigure`, што значи да од сада можемо да наводимо интерне референце на слику користећи ту лабелу (погледајте одељак 9.2).
- Коначно, слика је постала плутајући објекат, што значи да се у случају словослагачких потреба (пагинације) може померити, ConTeXt ће изменити њену позицију.

Уствари, `\placefigure` се, упркос свом имену, не користи само за уметање слика. Помоћу ње можемо да уметнемо било шта, укључујући и текст. Међутим, текст и остале ствари које се уметну у документ командом `\placefigure`, ће се третирати *као да су слика*, чак и ако то нису; додаће се на листу слика коју ConTeXt интерно одржава и можемо применити трансформације сличне онима које користимо за слике, као што су скалирање или ротирање, уоквиравање, итд. Тако се наредни пример:



Слика 13.2 Употреба `\placefigure` за трансформацију текста

постиже на следећи начин:

```
\placefigure
[here, force]
[fig:testtext]
{Употреба \backslash placefigure за трансформацију текста}
{\rotate[rotation=180]{\framed{\tfd Тест текст}}}
```

13.2.3 Уметање слика интегрисаних у текст блок

Осим за веома мале слике које могу да се интегришу у линију без превеликог ометања прореда, слике се обично умећу у пасус који садржи само ту слику (или другим речима, слика може да се посматра као посебан пасус). Ако се слика уметне са `\placefigure` и њена величина то дозвољава, у зависности од тога шта смо навели за место на које треба да се постави (погледајте одељак 13.4.1), ConTeXt ће дозволити да текст из претходних и наредних пасуса тече око слике. Међутим, ако желимо обезбедити да се одређена слика не одваја од одговарајућег текста, можемо да употребимо `figuretext` окружење, чија је синтакса:

```
\startfiguretext
[Опције]
[Лабела]
{Наслов}
{Слика}

... Текст

\stopfiguretext
```

Аргументи окружења су потпуно исти као за `\placefigure` и имају исто значење. Али опције овде више нису за постављање плутајућег објекта, већ назнаке за интеграцију слике у пасус; тако на пример, „left” овде значи да ће се слика поставити лево, док ће текст тећи десно, а „left, bottom” значи да слика мора да се постави на доњу леву страну текста којем је придружена.

Текст који се напише унутар окружења ће тећи око слике.

13.2.4 Конфигурација и трансформација уметнутих слика

А. Уметање опција команде које врше неке трансформације слике

Последњи аргумент команде `\externalfigure` нам омогућава да изведемо одређена подешавања слике која се умеће. Доступне су следеће врсте подешавања:

- У општем случају за све слике које се умећу у документ; или за све слике које се умећу од одређеног места. У овом случају подешавање радимо командом `\setupexternalfigures`.
- За одређену слику коју желимо да уметнемо неколико пута у документ. У овом случају подешавање се ради у последњем аргументу `\useexternalfigure` команде који слици придружује симболичко име.
- У тренутку уметања одређене слике. У овом случају се подешавање врши самом `\externalfigure` командом.

Измене слике које могу да се постигну на овај начин су:

Промена величине слике. Можемо да урадимо следеће:

- *Навођење прецизне ширине или висине*, ради се редом опцијама `width` и `height`; ако се измени само једна од две вредности, она друга се аутоматску подешава тако да се одржи пропорција.

Можемо да доделимо прецизну висину или ширину, или да је наведемо као проценат висине или ширине странице. На пример:

```
width=.4\textwidth
```

ће обезбедити да је ширина слике 40% ширине линије.

- *Скалирање слике*: опција `xscale` ће скалирати слику по хоризонтали; `yscale` ће то урадити по вертикали, а `scale` ће је скалирати и по хоризонтали и по вертикали. Ове три опције очекују број који представља фактор скалирања помножен са 1000. Дакле: `scale=1000` ће оставити слику у њеној оригиналној величини, док ће је `scale=500` умањити на половину, а `scale=2000` ће јој дуплирати величину.

Условно скалирање се примењује само ако слика прелази одређене димензије, а добија се опцијама `maxwidth` и `maxheight` које прихватају димензију као вредност. На пример `maxwidth=.2\textwidth` ће скалирати слику само ако се испостави да је већа од 20% ширине линије.

Ротирање слике. За ротирање слике користимо опцију `orientation`. Она као вредност узима број који представља угао у степенима за који ће се слика ротирати. Ротација се врши у смеру обрнутом од смера казаљке на сату.

Вики наводи да ротације које се раде овом опцијом морају бити умношци од 90 степени (90, 180 или 270), а ако је потребно да се добије другачија вредност, треба да се употреби команда `\rotate`. Међутим, ни сам имао никаквих проблема да на слику применим ротацију од 45 степени помоћу `orientation=45`, без потребе да се употреби `\rotate` команда.

Постављање оквира око слике. Око слике такође можемо да поставимо и оквир опцијом `frame=on`, да конфигуришемо његову боју са `framecolor`), растојање од оквира до слике са `frameoffset`), дебљину линије оквира са `rulethickness`) или облик његових углова са `framecorner`) који могу бити заобљени `round`) или угласти.

Остали аспекти слике који могу да се конфигуришу. Уз аспекте које смо већ видели, који се тичу трансформисања слике која треба да се уметне, командом `\setupexternalfigures` можемо да конфигуришемо и остале аспекте, као што је место на коме се тражи слика (опција `directory`), да ли би слика требало да се тражи само у наведеном директоријуму (`location=local`), или би требало да се у претрагу укључе и текући директоријум и његови родитељи (`location=global`), као и да ли ће слика бити интерактивна или не (`interaction`), итд.

Б. Специфичне команде за трансформисање слике

ConTeXt поседује три команде које врше неке трансформације слике. Оне могу да се користе у комбинацији са `\externalfigure`. То су:

- *Осна симетрија по вертикали:* постиже се командом `\mirror`.
- *Одсецање:* ово се постиже командом `\clip` када се задају ширина (`width`), висина (`height`), хоризонтални померај (`hoffset`) и вертикални померај (`voffset`). На пример:

```
\clip
[width=2cm, height=1cm, hoffset=3mm, voffset=5mm]
{\externalfigure[logo.pdf]}
```

- *Ротирација.* Трећа команда која на слику може применити трансформацију је `\rotate` команда. Може да се користи заједно са `\externalfigure` али то обично није потребно јер ова друга, као што смо видели, има опцију `orientation` која даје исти резултат.

Типична употреба ових команди је за слике, али оне уствари раде над *кућијама*. Због тога можемо да их применимо на било који текст, само ако га поставимо унутар кутије (што сама команда ради аутоматски), па ћемо добити неке прилично занимљиве ефекте као што су следећи:

```
\mirror{Тест текст}\
\rotate[rotation=20] {Тест текст}
```

тэжэт тээТ
Тест текст

13.3 Табеле

13.3.1 Општи концепти о табелама и њиховом постављању у документ

Табеле су структурни објекти који садрже текст, формуле, или чак слике поређане у низове *ћелија* које нам омогућавају да графички представимо везу између садржаја сваке ћелије. Да

бисмо то урадили, информације организујемо у редове и колоне: сви подаци (или ставке) у истом реду су у одређеној међусобној вези, исто тако и сви подаци (или ставке) у истој колони. Ћелија је пресек реда и колоне, као што је приказано на [слици 13.3](#).

Табеле су идеалне за приказ текста или података који су у међусобној вези, јер пошто је сваки од њих везан за сопствену ћелију, чак и ако се њен садржај или садржај преосталих ћелија измени, релативна позиција једне ћелије у односу на остале се неће променити.

Од свих задатака везаних за слово-слагање текста, креирање табела је, по мом мишљењу, једноставније урадити у графичком програму (типа текст процесора) него у систему ConTeXt. Јер табелу је једноставније *нацртати* (а то је управо оно што ради текст процесор) него *описати*, а управо то радимо када је креирамо у систему ConTeXt.

Свака промена реда и промена колоне захтева присуство команде, а то значи да имплементација табеле траје много дуже од простог навођења жељеног броја редова и колоне.

ConTeXt поседује три различита механизма за прављење табела; *tabulate* окружење, које се препоручује за једноставне табеле и које је најдиректније инспирисано \TeX табелама; такозване *природне табеле*, инспирисане HTML табелама, погодне за табеле са посебним потребама дизајна где, на пример, сви редови немају исти број колоне; и такозване *екстремне табеле*, јасно базиране на XML и препоручене за табеле средње или велике дужине које се простиру на више од једне странице. Од ове три, ја ћу објаснити само прву. Природне табеле су прилично добро објашњене у „ConTeXt Mark IV an excursion”, а о *екстремним табелама* постоји документ у „ConTeXt Standalone” документацији.

Код табела се дешава нешто слично као са сликама: можемо једноставно да напишемо потребне команде на неком месту у документу и тако генеришемо табелу која ће се уметнути тачно на то место, или за уметање табеле можемо да употребимо команду `\placetable`. Ово има неке предности:

- ConTeXt нумерише табелу и додаје је на листу табела чиме се омогућава интерна референца на табелу (преко њене нумерације), укључујући је у евентуални индекс табела.
- Добићемо флексибилност у позиционирању табеле унутар документа, па тако олакшавамо задатак пагинације.

Формат команде `\placetable` је сличан ономе што смо видели код `\placefigure` (погледајте [одељак 13.2.2](#)):



Слика 13.3 Слика просте табеле

`\placetable[Опције][Лабела] {Наслов} {табела}`

Указујем на одељке 13.4.1 и 13.4.2 у вези са опцијама које се тичу постављања табеле и конфигурације њеног наслова. Међутим, међу опцијама постоји и једна која изгледа да је дизајнирана само за табеле. То је „split” опција, која када се постави, дозвољава систему ConTeXt да табелу распореди на две или више страница, па у том случају табела више не може да буде плутајући објекат.

Уопштену конфигурацију табела можемо да поставимо командом `\setuptables`. Исто као са сликама, такође је могуће да се генерише индекс табела командом `\placelistoftables` или `\completelistoftables`. У вези овога, погледајте одељак 8.2.2.

13.3.2 Просте табеле са *tabulate* окружењем

Најједноставније табеле су оне које се добијају *tabulate* окружењем, чији је формат:

```
\starttabulate[Распоред колона табеле]
... % Садржај табеле
...
...
\stoptabulate
```

где аргумент који се поставља у велике заграде описује (кодирано) број колона који ће имати табела, и (понекад индиректно) наводи њихову ширину. Кажем да аргумент описује дизајн *кодирано*, јер на први поглед изгледа врло мистериозно: састоји се од низа карактера, од којих сваки има специјално значење. Објаснићу ово мало по мало и у корацима, јер мислим да је тај начин лакши за разумевање.

Ово је типичан случај у којем огроман број аспеката које можемо да конфигуришемо значи да је потребно много текста да се све опише. Чини се да је ово ђаволски тешко. Уствари, за већину табела које се састављају у пракси, довољне су тачке 1 и 2. Остатак представљају додатне могућности за које је корисно знати да постоје, али нису од суштинског значаја за словослагање табеле.

1. **Граничник колона:** за границу између колона табеле се користи карактер „|”. Тако на пример, „`[|lT|rB|]`” описује табелу са две колоне, од којих једна има карактеристике придружене индикаторима „l” и „T” (чије значење ћемо видети врло брзо), а друга колона има карактеристике придружене индикаторима „r” и „B”. На пример, проста табела са три лево поравнате колоне би се описала са „`[|l|l|l|]`”.
2. **Одређивање основне природе ћелија у колони:** прва ствар која се одређује када изграђујемо нашу табелу је да ли желимо да се садржај сваке ћелије исписује у једном реду, или желимо да се сувише дугачак текст сваке колоне распореди у две или више линија. У *tabulate* окружењу се на ово питање не одговара за сваку ћелију посебно, већ се посматра као карактеристика колона.
 - а. *Једнолинијске ћелије:* ако желимо да се садржај ћелија у колони, без обзира на дужину, пише у једној линији, морамо задати поравнање текста у колони, које може бити лево („l”, од *left*), десно („r”, од *right*) или центрирано („c”, од *center*).

У принципу, ове колоне ће бити широке онолико колико је потребно да стане садржај најшире ћелије. Али ширину колоне можемо ограничити кључем „w (Ширина)”. На пример, „`[|tw(2cm)|c|c|]`” описује табелу са три колоне, од којих је прва десно поравната, тачне ширине 2 центиметра, а друге две центриране и без ограничења ширине.

Треба запазити да ограничење ширине у једнолинијским колонама за последицу има да се текст једне колоне преклапа са текстом наредне колоне. Тако да је мој савет да када су нам потребне колоне фиксне ширине увек користимо колоне са вишелинијским ћелијама.

- б. *Ћелије које њо њоћреди моју заузећи више од једне линије:* кључ „r” генерише колоне у којима ће текст у свакој ћелији заузимати потребан број линија. Ако једноставно наведемо „r”, ширина колоне ће бити пуна доступна ширина. Али такође је могуће да се наведе „r(Ширина)”, па ће онда колона бити наведене ширине. Тако имамо следеће примере:

```
\starttabulate[|l|r|p|]
\starttabulate[|l|p(4cm)|]
\starttabulate[|r|p(.6\textwidth)|]
\starttabulate[|p|p|p|]
```

Први пример ће креирати табелу са три колоне, прву и другу у једној линији, редом лево и десно поравнате, а трећа ће заузимати остатак ширине и висине неопходан да се смести сав садржај. У другом примеру, друга колона ће бити широка тачно четири центиметра, без обзира шта садржи; али ако садржај не стаје у тај простор, поставиће се у више линија. Трећи пример израчунава ширину друге колоне у односу на максималну ширину линије, а у последњем примеру постоје три колоне које ће међу собом поделити доступну ширину на једнаке делове.

Имајте на уму да је у реалности ћелија четворострана, оно што кључ „r” ради је да дозвољава променљиву висину ћелија у колони, зависно од дужине текста у њима.

3. **Додавање назнака стила и варијанте фонта у опис колоне:** када се одлучи основна природа колоне (ширина и висина ћелија, аутоматска или фиксна), у опис садржаја колоне још можемо додати и карактер који поставља *формат* у којем се садржај исписује. Ови карактери могу бити „B” за црни слог, „I” за курзив, „S” за коси слог, „R” за испис у римском стилу, или „T” за испис у стилу *писаће машине*.

4. **Остали додатни аспекти који могу да се наведу у опису колона табеле:**

- *Колоне са математичким формулама:* кључеви „m” и „M” укључују математички режим у колони без потребе да се он наводи за сваку од ћелија у колони. Ћелије у овој колони неће моћи да садрже обичан текст.

Мада је TeX, претходник система ConTeXt, креиран за словослагање било које врсте математике, све до сада нисам рекао скоро ништа о писању математике. У математичком режиму (који нећу објашњавати) ConTeXt мења уобичајена правила, па чак користи и другачије фонтове. Математички режим долази у две варијанте: један можемо назвати *линеарни* у којем је формула обухваћена линијом која садржи нормални текст (кључ „m”) и *комплетни математички режим* који формуле исписује у окружењу у којем нема обичног текста. Главна разлика између ова два режима у табели је у основи величина у којој ће се исписати формула, као и у хоризонталном и вертикалном простору око ње.

- *Додавање догађајној хоризонталној празној простора око садржаја ћелија у колони:* кључевима „in”, „jn” и „kn” можемо додати још празног простора лево од садржаја

колоне („in“), десно („jn“) или са обе стране („kn“). У сва три случаја, „n“ представља број којим се множи празан простор који и иначе постоји када се ови кључеви не наведу (просек је подразумевано један *ема*). Тако на пример, „|j2r|“ наводи да се креира колона која ће бити десно поравната и у којој желимо празан простор ширине 1 *ема*.

- *Додавање текста испред или иза садржаја сваке ћелије у колони.* Кључеви `b{Текст}` и `a{Текст}` наводе да се текст унутар витичастих заграда исписује испред („b“, од *before*) или иза („a“, од *after*) садржаја ћелије.
- *Примењивање команде формирања на комплећну колону.* Кључеви „B“, „I“, „S“, „R“ „T“ које смо већ поменули не покривају све могућности формирања: нпр. нема кључа за капитал, или за *линеарно исмо*, или онај који утиче на величину фонта. Кључем „f\Команда“ можемо да наведемо команду формата које ће се аутоматски применити на све ћелије у колони. На пример, „|lf\sc|“ ће садржај колоне сложити капиталом.
- *Примењивање било које команде на све ћелије у колони.* Коначно, кључ „h\Команда“ ће применити наведену команду на све ћелије у колони.

У [табели 13.1](#) су приказани примери неких стрингова за спецификацију формирања табеле.

Спецификатор формата	Значење
l	Генерише колону чија је ширина аутоматски лево поравната.
rB	Генерише колону чија је ширина аутоматски десно поравната, црним слогом.
cIm	Генерише колону за математички садржај. Центриран, у курзиву.
j4cb{-}	Садржај ове колоне ће бити центриран, почињаће ем цртом (—) и додаваће се 2 <i>ема</i> празног простора на десну страну.
l p(.7\textwidth)	Генерише две колоне: прва је лево поравната, аутоматске ширине. Друга заузима 70% укупне ширине линије.

Табела 13.1 Неки примери начина за навођење формата колоне у *tabulate*

Када се табела дизајнира, потребно је унети њен садржај. Објашњење тог процеса почињемо описом начина попуњавања табеле када имамо линије које раздвајају редове и колоне:

- Увек почињемо тако што цртамо хоризонталну линију. У табели се то ради командом `\HL` (од *Horizontal Line*).
- Затим пишемо прву линију: на почетку сваке ћелије морамо назначити да она почиње, и да је потребно исцртати вертикалну линију. Ово се ради командом `\VL` (од *Vertical Line*). Дакле, почињемо овом командом и пишемо садржај сваке ћелије. Сваки пут када прелазимо на наредну ћелију, понављамо команду `\VL`.

- На крају реда експлицитно наводимо да почиње нови ред командом `\NR` (од *Next Row*). Након ње понављамо `\HL` да исцртамо нову хоризонталну линију.
- И тако, један по један, пишемо две редове табеле. Када завршимо, умећемо као додатак `\NR` команду у још једну `\HL` да затворимо мрежу доњом хоризонталном линијом.

Ако не желимо да цртамо мрежу табеле, уклањамо `\HL` команде и мењамо `\VL` команде са `\NC` (од *New Column*).

Није тако тешко када се навикнемо, иако кад погледамо у изворни кôд табеле тешко можемо стећи идеју како ће она да изгледа. У [табели 13.2](#) видимо команде које могу (и морају) да се користе унутар табеле. Постоје још неке које нисам објаснио, али мислим да је ово што сам објаснио довољно.

Команда	Значење
<code>\HL</code>	Умеће хоризонталну линију
<code>\NC</code>	Започиње нову колону
<code>\NR</code>	Започиње нови ред
<code>\VL</code>	Умеће вертикалну линију која одваја колону (користи се уместо <code>\NC</code>)
<code>\NN</code>	Започиње нову колону у математичком режиму (користи се уместо <code>\NC</code>)
<code>\TB</code>	Умеће додатни вертикални простор између два реда
<code>\NB</code>	Назначава да наредни ред започиње недељиви блок унутар којег не сме да се нађе прелом странице

Табела 13.2 Команде које се користе у табели

А сада ћу, као пример, написати кôд који генерише [табелу 13.2](#).

```
\placetable
[here, force]
[tbl:tablecommands]
{Команде које се користе у табели}
{\starttabulate[1|p(.6\textwidth)]}
\HL
\NC {\bf Команда}
\NC {\bf Значење}
\NR
\HL
\NC \tex{HL}
\NC Умеће хоризонталну линију
\NR
\NC \tex{NC}
\NC Започиње нову колону
\NR
\NC \tex{NR}
\NC Започиње нови ред
\NR
\NC \tex{VL}
\NC Умеће вертикалну линију која одваја колону (користи се уместо \tex{NC})
\NR
```



```

\NC \tex{NN}
\NC Започиње нову колону у математичком режиму (користи се уместо \tex{NC})
\NR
\NC \tex{TB}
\NC Умеће додатни вертикални простор између два реда
\NR
\NC \tex{NB}
\NC Назначава да наредни ред започиње недељиви блок унутар којег не сме да се нађе прелом
странице
\NR
\HL
\stoptabulate}

```

Читалац ће приметити да сам у општем случају користио једну (или две) линије текста за сваку ћелију. У стварном изворном фајлу би користио само једну линију текста за сваку ћелију; у примеру сам поделио сувише дугачке линије. Лакше ми је да пишем табелу једну линију по ћелији, јер уствари пишем садржај сваке ћелије, без команди за поделу колона или редова. Када се све напише, изаберам текст табеле и затражим од мог текст едитора да уметне „\NC ” на почетак сваке линије. Након тога, после сваке две линије (јер табела има две колоне) умећем линију која додаје \NR команду, јер после сваке две колоне почиње нови ред. Коначно, ручно умећем \HL команде на местима на којима желим да се види хоризонтална линија. Скоро да ми је потребно више времена да ово опишем, него да га изведем!

Али запазите да унутар табеле можемо користити и уобичајене ConTeXt команде. У овој табели константно користимо \tex која је објашњена у одељку 10.2.3.

13.4 Заједнички аспекти слика, табела и осталих плутајућих објеката

Већ знамо да слике и табеле не морају бити плутајући објекти, али су добри кандидати да то буду, мада онда у документ морају да се уметну командом \placefigure или \placetable. Уз ове две команде, и уз исту структуру, у систему ConTeXt имамо и \placechemical команду (за уметање хемијских формула), команду \placegraphic (за уметање графика) и команду \placeintermezzo за уметање структуре коју ConTeXt назива *Инијермецо*, која претпостављам да се односи на уоквирене фрагменте текста. Све ове команде су уствари примене општије команде \placefloat чија је синтакса:

```
\placefloat[Име] [Опције] [Лабела] {Наслов} {Садржај}
```

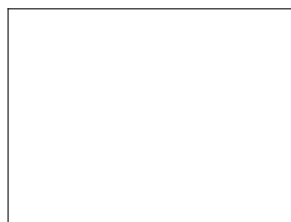
Запазите да је \placefloat идентична са \placefigure и \placetable осим што први аргумент у \placefloat представља име плутајућег објекта. То је зато што *сваки њиш њлу-џајуће објектџа може да се уметне у документџи љомоћу две различитџе команде*: \placefloat[ИмеТипа] или \placeИмеТипа. Другим речима: \placefloat[figure] и \placefigure су потпуно иста команда, као што је \placefloat[table] исто што и \placetable.

Зато ћу од сада да говорим о \placefloat, али имајте на уму да се све што кажем такође односи и на \placefigure или \placetable, јер су то специфичне примене наведене команде.

Аргументи команде `\placefloat` су следећи:

- *Име.* Односи се на дати плутајући објекат. Може бити неки предодређени плутајући објекат (`figure`, `table`, `chemical`, `intermezzo`) или плутајући објекат који сами генеришемо командом `\definefloat` (погледајте одељак 13.5).
- *Опције.* Низ симболичких речи које систему ConTeXt говоре како би требало да уметне објекат. Велика већина њих се односи на *месџо* уметања. То ћемо видети у наредном одељку.
- *Лабела.* Лабела за будућа интерна указивања на овај објекат.
- *Наслов.* Текст наслова који треба да се дода објекту. Што се тиче његове конфигурације, погледајте одељак 13.4.2.
- *Садржај.* Ово наравно зависи од врсте објекта. За слике је обично команда `\externalimage`; за табеле, команде које ће креирати табелу; за *интермеца*, уоквирени фрагмент текста; итд.

Прва три аргумента која се наводе унутар великих заграда нису обавезна. Последња два (унутар витичастих заграда) су обавезна, мада могу бити празни. Тако ће, на пример: `\placefloat{ }{ }` да уметне у документ



Слика 13.4



Напомена: видимо да је ConTeXt претпоставио да је уметнути објекат слика, јер га је нумерисао као слику и поставио на листу слика. Због тога претпостављам да су слике подразумевано плутајући објекти.

13.4.1 Опције за уметање плутајућих објеката

Аргумент *Опције* у `\placefigure`, `\placetable` и `\placefloat` контролише различите аспекте који се тичу уметања ових типова објеката. Углавном место на страници на које ће се објекат поставити. Ево неколико подржаних вредности, свака различите природе:

- Нека места за уметање се успостављају релативно у односу на елементе странице (`top`, `bottom`, `inleft`, `inright`, `inmargin`, `margin`, `leftmargin`, `rightmargin`, `leftedge`, `rightedge`, `innermargin`, `inneredge`, `outeredge`, `inner`, `outer`). То наравно мора бити објекат који може да стане у простор који му намењујемо и у распореду странице мора бити резервисан простор за тај елемент. Што се тиче овога, погледајте одељке 5.2 и 5.3.
- Остала могућа места за уметање се више тичу текста који се налази око објекта, и представљају назнаку места на које би требало поставити објекат тако да текст тече око њега. У основи, то су вредности `left` и `right`.

- Опција `here` се интерпретира као препорука да се објекат задржи на месту на којем је наведен у изворном фајлу. Ова *препорука* се неће поштовати ако то не дозвољавају потребе пагинације. Ова ставка се форсира ако додамо и опцију `force` која значи управо то: форсирано уметање објекта на одређено место. Имајте на уму да форсирањем уметања објекта на одређено место, он престаје да буде плутајући објекат.
- Остале могуће опције су у вези са страницом на коју објекат треба да се уметне: „`page`” умеће на нову страницу; „`opposite`” умеће на страницу супротну од текуће; „`leftpage`” на парну страницу; „`rightpage`” на непарну.

Постоје неке опције које нису у вези са локацијом објекта. То су између осталих:

- `none`: ова опција уклања наслов.
- `split`: ова опција дозвољава да се објекат простира на више од једне странице. Наравно, то мора бити природно дељив објекат, као што је табела. Када се ова опција употреби и објекат се подели, више се не може рећи да је он плутајући.

13.4.2 Конфигурисање наслова плутајућег објекта

Осим ако у `\placefloat` не употребимо опцију „`none`”, плутајућим објектима се подразумевано додељује наслов који се састоји из три елемента:

- Име врсте објекта о којем је реч. Ово име је потпуно исто као назив типа објекта; тако да ако, на пример, дефинишемо нови плутајући објекат под називом „`sekvenca`” па уметнемо „`sekvenca`” као плутајући објекат, наслов ће бити „`Sekvenca 1`”. Једноставно се име објекта испише првим великим словом.

Упркос овоме што сам управо рекао, ако главни језик документа није енглески, енглеска имена предефинисаних објеката, као на пример „`figure`” или „`table`” ће бити преведени; тако се на пример, „`figure`” објекат у документима на српском назива „Слика”, док се „`table`” објекат назива „Табела”. Ова предефинисана српска имена могу да се промене командом `\setuplabeltext` онако како је објашњено у одељку 10.5.3.

- Његов број. Објекти се подразумевано нумеришу по поглављима, тако да је прва табела у глави 3 табела '3.1'.
- Њен садржај. Уводи се као аргумент команде `\placefloat`.

Помоћу `\setupcaptions` или `\setupcaption[Објекат]` можемо да изменимо систем нумерисања и изглед самог наслова. Прва команда ће утицати на све наслове свих објеката, а друга само на наслов одређене врсте објекта:

- Што се тиче система нумерисања, он се контролише опцијама `number`, `way`, `prefixsegments` и `numberconversion`:
 - `number` може имати вредност `yes`, `no` или `none` и контролише да ли ће постојати број или неће.
 - `way` наводи да ли ће нумерација бити редом кроз цео документ (`way=bytext`), или ће поново кренути из почетка када се промени поглавље (`way=bychapter`) или одељак

(way=bysection). У случају поновног почетка, згодно је да се вредност ове опције усклади са опцијом prefixsegments.

- prefixsegments наводи да ли ће број имати *префикс*, и шта ће он бити. Тако да prefixsegments=chapter као резултат има бројеви објеката увек почињу са бројем поглавља, док ће prefixsegments=section да испред број објекта постави број одељка.
- numberconversion контролише врсту нумерације. Вредности ове опције могу бити: арапски бројеви („numbers”), мала слова („a”, „characters”), велика слова („A”, „Characters”), капитал „KA”), римски бројеви исписани великим словима („I”, „R”, „Romannumerals”), малим словима („i”, „r”, „romannumerals” или капиталом („KR”).
- Изглед самог наслова контролишу бројне опције. Ја ћу их овде навести, али за детаљно објашњење значења сваке од њих, упућујем на [одељак 7.4.4](#) у којем је објашњена контрола изгледа команди за поделу, јер су опције углавном исте. То су:
 - За контролу формата свих елемената наслова style, color, command.
 - Само за контролу формата врсте објеката према имену: headstyle, headcolor, headcommand, headseparator.
 - Само за контролу формата нумерисања: numbercommand.
 - Само за контролу формата самог наслова: textcommand.
- Такође можемо да контролишемо и остале аспекте као што су растојање између појединих елемената који чине наслов, ширину наслова, позиционирање наслова у односу на објекат, итд. Овде упућујем на информације из [ConTeXt викија](#) у вези са опцијама које конфигуришу ову команду.

13.4.3 Комбиновано уметање два или више објеката

Ако у документ желимо да уметнемо два или више различитих објеката, тако да их ConTeXt држи заједно као један објекат, на располагању нам је `\startcombination` окружење, чија је синтакса:

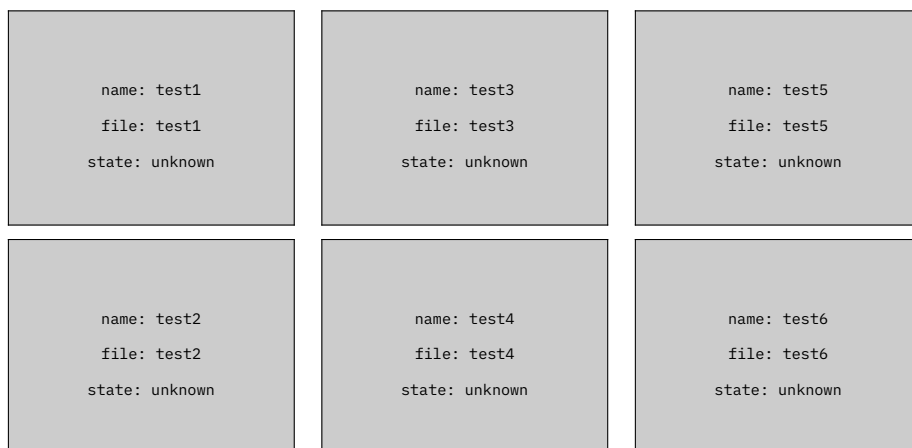
```
\startcombination[Редослед] ... \stopcombination
```

где *Редослед* назначаваче како би објекти требало да се распореде: ако сви треба да се поређају хоризонтално, *Редослед* само означава број објеката који се комбинују. Али ако желимо да их комбинујемо у два или више редова, морамо да назначимо колико објеката по реду, па затим у колико редова, раздвајајући то карактером *. На пример:

```
\startcombination[3*2]
{\externalfigure[test1]}
{\externalfigure[test2]}
{\externalfigure[test3]}
{\externalfigure[test4]}
{\externalfigure[test5]}
```

```
{\externalfigure[test6]}
\stopcombination
```

ће направити следећи распоред слика:



У претходном примеру слике које сам комбиновао не постоје, па је то разлог што је ConTeXt уместо слика генерисао текст кутије са информацијама о сликама.

С друге стране, запазите да је сваки елемент који треба да се комбинује унутар `\startcombination`, унутар витчастих заграда.

Иначе, `\startcombination` нам дозвољава не само да спојимо и поравнамо слике, већ било коју врсту *кућије* као што су текстови унутар `startframedtext` окружења, табеле, итд. За конфигурисање комбинације можемо употребити команду `\setupcombination`, а командом `\definecombination` такође можемо да креирамо и преконфигурисане комбинације.

13.4.4 Општа конфигурација плутајућих објеката

Већ смо видели да командом `\placefloat` можемо контролисати место уметања плутајућег објекта, као и још неке детаље. Такође се могу конфигурисати и:

- Глобалне карактеристике одређеног типа плутајућих објеката. Ово се ради командом `\setupfloat[Име типа плутајућег објекта]`.
- Глобалне карактеристике свих плутајућих објеката у нашем документу. Ово се ради командом `\setupfloats`.

Имајте на уму да исто као што је `\placefloat[figure]` еквивалентно са `\placefigure`, тако је и `\setupfloat[figure]` еквивалентно са `\setupfigures`, а `\setupfloat[table]` са `\setuptables`.

Што се тиче опција које су доступне за конфигурисање ових команду, указујем на ConTeXt званичну листу команду (одељак 3.6).

13.5 Дефинисање додатних плутајућих објеката

Команда `\definefloat` нам омогућава да дефинишемо наш сопствени пливајући објекат. Њена синтакса је:

```
\definefloat[Име у јединици] [Име у множини] [Конфигурација]
```

где *Конфигурација* није обавезан аргумент који нам омогућава да већ у време креирања овог новог објекта наведемо и његову конфигурацију. То такође можемо и касније да урадимо командом `\setupfloat[Име у јединици]`.

Пошто овим одељком завршавамо наш увод, искористићу прилику да мало детаљније зађем у привидну *цунџу* `ConTeXt` команди, које једном када се разумеју, и нису баш тако густа *цунџа*, већ су у суштини прилично разумне.

Хајде да почнемо тако што ћемо себе упитати шта плутајући објекат заиста представља за систем `ConTeXt`. Одговор је да је то објекат са следећим карактеристикама:

- Поседује одређени степен слободе што се тиче његовог места на страници.
- Има придружену *лисџу*. Она омогућава да се ова врста објеката нумерише, па да се по потреби генерише индекс ових објеката.
- Има наслов
- Када објекат заиста може да плута, мора да се третира као недељива јединица, што значи (у `TeX` терминологији) *унушар куџије*.

Другим речима, плутајући објекат је уствари састављен од три елемента: самог објекта, придружене листе и наслова. За контролу самог објекта нам је потребна само једна команда којом постављамо његову локацију и још једна којом објекат умећемо у документ; за постављање аспеката листе, довољне су уопштене команде за контролу листи, а за постављање аспеката наслова опште команде за контролу наслова.

И овде долазимо до места на којем се истиче генијалност система `ConTeXt`: било би довољно да је дизајнирана једноставна команда за контролу плутајућих објеката (`\setupfloats`) и једноставна команда за уметање плутајућих објеката: `\placefloat:` али `ConTeXt` ради следеће:

1. Дизајнира команду која одређеној конфигурацији плутајућег објекта додељује име. То је команда `\definefloat`, која уствари не повезује само једно име, већ једно у јединици и једно у множини.
2. Креира, заједно са глобалном конфигурационом командом за плутајуће објекте, и команду која нам омогућава да конфигуришемо само одређени тип објекта: `\setupfloat[Објекат]`.

3. Команди за постављање плутајућег објекта, (`\placefloat`), додаје аргумент који нам омогућава да направимо разлику између типова: (`\placefloat[Објекат]`).
4. Креира команде, укључујући и име објекта, за све акције над плутајућим објектом. Неке од ових команди (које су уствари клонови уопштенијих команди) ће користити име објекта у једнини, а неке у множини.

Дакле, када креирамо нови плутајући објекат и систему ConTeXt кажемо које је његово име у једнини и множини, ConTeXt:

- Резервише у меморији простор за чување одређене конфигурације тог типа објекта.
- Креира нову листу са именом типа објекта у једнини, јер се плутајући објектима придружује листа.
- Креира нову врсту „title” везаног за овај нови тип објекта, како би могла да се одржава прилагођена конфигурација ових наслова.
- И коначно, креира групу команди које су специфичне за овај нови тип објекта. Њихова имена су уствари синоними уопштенијих команди.

У [табели 13.3](#) можемо да видимо команде које се аутоматски креирају када дефинишемо нови тип плутајућег објекта, као и уопштеније команде чији су то синоними:

Команда	Синоним за	Пример
<code>\completelistof<ИмеУМножини></code>	<code>\completelist[ИмеУМножини]</code>	<code>\completelistoffigures</code>
<code>\place<ИмеУЈеднини></code>	<code>\placefloat[ИмеУЈеднини]</code>	<code>\placefigure</code>
<code>\placelistof<ИмеУМножини></code>	<code>\placelist[ИмеУМножини]</code>	<code>\placelistoffigures</code>
<code>\setup<ИмеУМножини></code>	<code>\setupfloat[ИмеУМножини]</code>	<code>\setupfigure</code>

Табела 13.3 Команде које се аутоматски креирају када се креира нови плутајући објекат

Уствари, креирају се још неке команде које су синоними ових претходних и пошто их ја нисам објаснио у овом поглављу, нисам их поставио ни у [табелу 13.3](#): `\start<ИмеУЈеднини>`, `\start<ИмеУЈеднини>text` and `\startplace<ИмеУЈеднини>`.

Као пример команди које се креирају када се дефинише нови плутајући објекат сам употребио команду за слике; то сам урадио јер су слике, као и табеле и остали предефинисани плутајући објекти у систему ConTeXt, конкретни случајеви команде `\definefloat`:

```
\definefloat[chemical][chemicals]
\definefloat[figure][figures]
\definefloat[table][tables]
\definefloat[intermezzo][intermezzi]
\definefloat[graphic][graphics]
```

Коначно, оно што видимо је да у ствари ConTeXt ни на који начин не контролише материјал који је део сваког одређеног плутајућег објекта; он претпоставља да је то посао аутора. Из тог разлога командом `\placefigure` или `\placetable` такође можемо да уметнемо и текст. Међутим, текст који се уметно командом `\placefigure` постаје део листе слика, а ако се уметне са `\placetable`, листе табела.

Додаци

Додатак А

Инсталација, конфигурација и ажурирање система ConT_EXt

Главне дистрибуције система T_EX (T_EX Live, t_ET_EX, M_ikT_EX, MacT_EX, итд.) садрже и верзију система ConT_EXt. Међутим, то није најновија верзија. У овом додатку ћу објаснити две процедуре за инсталирање две различите верзије система ConT_EXt; прва инсталира и ConT_EXt Mark II и Mark IV, а друга само ConT_EXt Mark IV.

Процедура инсталације следи исте кораке на било ком оперативном систему; али се детаљи разликују од система до система. Ипак, ствари можемо да поједноставимо тако што ћу у наредним редовима направити разлику између две велике групе система:

- **Системи Unix типа:** укључују сам Unix, као и GNU Linux, Mac OS, FreeBSD, OpenBSD или Solaris. Процедура је у суштини иста за све ове системе; постоје неке заиста мале разлике које ћу нагласити на одговарајућем месту.
- **Windows системи** подразумевају различите верзије тог оперативног система: Windows 10 (ваљда најновија верзија), Windows 8, Windows 7, Windows Vista, Windows XP, Windows NT, итд.

Важна напомена о процесу инсталације на Microsoft Windows системима:

ConT_EXt је, као и сви T_EX системи, дизајниран за рад из терминала; такође и програми и процедуре за инсталацију. Ово је такође савршено могуће у систему Windows и не би требало да представља неку велику тешкоћу. Проблем је што, с једне стране Windows корисници нису увек навикли да то раде, а са друге, пошто је Windows запао у *шлузију* (погрешну) да би све у компјутерском систему могло да се уради графички, у општем случају верзије тог оперативног система не *објављују* превише о начину употребе терминала. А такође је и уобичајено да се у свакој верзији овог оперативног система промени име програма које покреће терминал, као и начин за његово покретање. Колико ја знам, Windows програму за емулацију терминала је дато доста имена: „DOS прозор”, „Командна линија”, „cmd”, итд. Место овог програма у Windows менију са апликацијама се такође мења зависно од дате Windows верзије.

Windows системе сам престао да користим 2004. године, тако да не могу бити од помоћу читаоцу. Он или она ће самостално морати да открију како да отворе терминал у својој верзији оперативног система; што не би требало да је превише тешко.

1 Инсталирање и конфигурирање „ConT_EXt Standalone”

ConT_EXt дистрибуција позната као „Standalone”, такође позната под именом „ConT_EXt Suite”, је комплетна и ажурна дистрибуција система ConT_EXt, која са Интернета преузима потребне фајлове, не заузима превише простора на диску, једноставно се ажурира и изнад свега

— отуда и име *Standalone (самостални)* — садржи се у једном директоријуму који може да се налази на било ком месту на хард диску. Чак је могуће и да један компјутер поседује неколико ConTeXt верзија, сваку у свом директоријуму. Ова дистрибуција садржи фонтове, бинарне фајлове и документацију неопходну за покретање ConTeXt Mark II (што подразумева T_EX, pdfT_EX и X_YT_EX машине), као и ConTeXt Mark IV (што подразумева LuaT_EX машину).

За више информација у вези са T_EX машинама, погледајте [одељак 1.4.1](#); а о T_EX машинама у вези са системом ConTeXt, као и верзијама познатим под именом Mark II и Mark IV, [одељак 1.5.1](#).

Оно што следи објашњава како се на наш систем инсталира, покреће, ажурира и обнавља „ConTeXt Standalone”. Подаци и процедуре које су овде наведене су сажетак много детаљнијих информација са [ConTeXt викија](#), а којима сам додао још неке детаље преузете из вики-књиге о ConTeXt која се налази на сајту [wikibooks](#). Ако постоји било какав проблем са инсталацијом, или ако желите да проширите било који детаљ, требало би да директно консултујете нешто од ових извора (мада је овај последњи на француском језику).

1.1 Инсталација

Инсталација „ConTeXt Standalone” подразумева да имате Интернет везу, и следеће кораке:

1. Креирање директоријума у којем ће се инсталирати ConTeXt.
2. Преузимање *скрипте* за инсталацију у овај директоријум.
3. Покретање ове *скрипте* уз жељене опције.
4. Обављање неколико завршних подешавања.

Корак 1: Креирање директоријума за инсталацију

У суштини, ово нема никакве везе са системом ConTeXt и морамо претпоставити да сваки корисник зна како се то ради. У Windows системима се то обично ради из фајл менаџера. На Unix системима може да се уради из фајл менаџера или из терминала. Међутим, важно је имати на уму да се не препоручује да путања инсталационог директоријума има у себи размаке. Ја лично избегавам употребу имена директоријума која нису на енглеском и која имају ствари као што су акцендована слова.

Од сада ћу претпоставити да је инсталациони директоријум, на Unix системима, „~/context/” и на Windows, „C:\Programs\context”.

Корак 2: Преузимање *скрипте* за инсталацију у инсталациони директоријум

Скрипта за инсталацију се разликује у зависности од оперативног система на који се инсталира:

- На Unix системима може да се преузме веб прегледачем, или из терминала командом као што је „wget” или „rsync”:

```
wget http://minimals.contextgarden.net/setup/first-setup.sh
rsync rsync://minimals.contextgarden.net/setup/first-setup.sh
```

- На Windows системима, колико је мени познато, не постоји стандардни алат за преузимање из конзоле. Мора да се уради из веб прегледача. Адреса за преузимање може бити било која од следећих:

<http://minimals.contextgarden.net/setup/context-setup-mswin.zip>

<http://minimals.contextgarden.net/setup/context-setup-win64.zip>

Када се преузме, на Windows систему морате да распакујете архиву.

Корак 3: Покретање *скрипте* за инсталацију

Инсталациона *скрипта* мора да се покрене из терминала. На Unix системима, име *скрипте* је „first-setup.sh” и може да се изврши помоћу `bash` или `sh`. На Windows системима, *скрипта* се зове „first-setup.bat” и извршава се просто куцањем њеног имена у системској конзоли или MS-DOS прозору из инсталационог директоријума.

Инсталациона *скрипта* прихвата следеће опције:

- `-context`: ова опција одређује верзију система ConTeXt која ће се инсталирати, или најсвежија развојна верзија („`-context=latest`”) или последња стабилна верзија („`-context=beta`”). Подразумевана вредност је „beta”.
- `-engine`: омогућава нам да наведемо да ли желимо да инсталирамо Mark IV („`-engine=luatex`”, подразумевана вредност) или Mark II.
- `-modules`: такође се инсталирају и ConTeXt модули проширења који нису саставни део дистрибуције, али пружају интересантне додатне могућности. Ако то желимо, потребно је на наведемо „`-modules=all`”.

Што се тиче опција инсталације, верујем да су информације на викију сада застареле. Тамо се каже да ако желимо да инсталирамо само Mark IV, морамо експлицитно да наведемо опцију „`-engine=luatex`”, а да опција „`-context=latest`” инсталира последњу стабилну верзију, а не развојну верзију. Међутим, од половине 2020. године, садржај фајла `first-setup.sh` се променио, па сам након читања фајла сазнао да је за инсталирање најновије верзије потребно да се експлицитно наведе „`-context=latest`”, а да је „`-engine=luatex`” подразумевано укључено.

Француска вики-књига коју сам поменуо на почетку овог додатка додаје још две опције оних које сам управо навео (документоване на ConTeXt викију): „`-fonts=all`” и „`goodies=all`”. ConTeXtgarden их не помиње, али не смета да се додају команди за инсталацију. Зато вас саветујем да инсталациону скрипту покренете са следећим опцијама (у зависности од тога да ли то радите на Unix или Windows системима):

- 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`

У зависности од брзине ваше Интернет везе, ово може да потраје, али не предуго.

Подешавање проксија

Инсталациона скрипта користи `rsync` да дође до потребних фајлова. Зато, ако се налазите иза прокси сервера, потребно је да програму `rsync` наведете његове детаље. Најлакши начин да се то уради је у терминалу помоћу променљиве `RSYNC_PROXY`, или унутар ваше *скрипте* која се извршава приликом покретања (`.bashrc` или

одговарајући фајл за сваку љуску). Замените кориме, лозинка, проксихост и проксипорт са исправним информацијама. Ово се на већини Unix система ради командом „export”, а у Windows системима, командом „set”. На пример:

```
export RSYNC_PROXY=кориме:лозинка@проксихост:проксипорт
```

Понекада, када се налазимо иза мрежне баријере, може бити да је одлазни порт 873 затворен за TCP конекције. Ако је отворен порт 22 за ssh конекције, трик који би могао да се употреби је да се повежете на неки компјутер ван мрежне баријере и да тунелујете у порт 873 (програмом nc).

```
export RSYNC_CONNECT_PROG='ssh тунелхост nc %H 873'
```

где је 'тунелхост' машина ван мрежне баријере на коју имамо приступ. Наравно, ова машина мора да има програм nc и отворен порт 873 за одлазне TCP конекције.

Након покретања „first-setup” у инсталационом директоријуму ће се појавити два нова директоријума, „bin” и „tex”.

Корак 4: Коначна подешавања (само на GNU Linux)

На GNU Linux системима постоји много директоријума у којима могу да се инсталирају фонтови. Ако желимо да ConTeXt користи те фонтове, морамо да му кажемо где их може пронаћи. Да бисмо то урадили, морамо да додамо следећу линију у фајл „tex/setuptex” креиран након инсталације:

```
export OSFONTDIR="/usr/share/fonts:/usr/share/texmf/fonts/opentype/"
```

којом се учитава променљива окружења OSFONTDIR са три директоријума у којима се обично налазе фонтови инсталирани у систему.

/usr/share/texmf/fonts/ ће постојати само ако на вашем оперативном систему постоји нека друга инсталација T_EX или система базираног на њему; у овом случају би требало да буде део OSFONTDIR путање тако да можемо користити OpenType фонтове који су можда део те инсталације. Ако имате било које комерцијалне фонтове које желите да ConTeXt користи, проверите да ли је путања до њих у OSFONTDIR, у супротном, додајте је у ову променљиву. Видео сам да су, на пример, неки фонтови инсталирани у /usr/local/fonts уместо у /usr/share/fonts.

Коначно, није лоша идеја да ConTeXt генерише базу података са свим фајловима неопходним за извршавање. То ће се урадити извршавањем следеће три команде у терминалу:

```
. ~/context/tex/setuptex
context -generate
context -make
```

Прва наредба је тачка. То је скраћеница за интерну bash команду source. Ако нам више одговара, можемо наравно да извршимо и *source*.

1.2 Извршавање „ConTeXt Standalone”

„ConTeXt Standalone” је дизајнирана тако да може постојати заједно уз остале инсталације T_EX система, што је предност која нам омогућава да имамо неколико различитих верзија инсталираних на исти оперативни систем; али да бисмо искористили ту предност, веома је важно да се променљиве окружења које су потребне за извршавање система ConTeXt не поставе за стално, јер сваки пут када покренемо терминал да у њему покренемо „context”,

мораћемо почети тако што у меморију учитамо ове променљиве окружења. Оне се налазе у фајлу „tex/setuptex” (Unix) или „tex/setuptex.bat” (Windows). То се ради овако:

- На Unix системима, након отварања терминала у којем желимо да покренемо „context”, извршавањем било које од следеће две команде:

```
source ~/context/tex/setuptex
. ~/context/tex/setuptex
```

(под претпоставком да је директоријум са верзијом програма „context” коју желимо да користимо у „~/context”).

- На Windows системима, извршавањем команде `tex\setuptex.bat` из инсталационог директоријума у терминалу из којег ћемо користити ConTeXt.

Ако на нашем систему нема ниједне друге инсталације система TeX или било којег од система изведених из њега, ово можемо да аутоматизујемо сваки пут када се отвори терминал:

- На Unix системима то се ради тако што се команде уметну у фајл *скрипте* која садржи опште команде које се извршавају приликом покретања терминала (обично је то „.bashrc”).

Конфигурациони фајл терминала зависи од програма љуске коју терминал подразумевано користи. Ако је то bash (најчешће коришћен на GNU Linux системима), фајл који се читава на почетку је .bashrc. sh и ksh љуске користе фајл под именом .profile, zsh користи .zshenv, a tcsh или csh читају .cshrc фајл. Неке специфичне имплементације могу променити имена ових фајлова, па тако на пример, .bashrc се понекад назива .bash_profile.

- На Windows системима можемо креирати пречицу на радној површини која покреће cmd.exe па да је затим уредимо тако што као команду коју покреће када над пречицом извршимо двоструки клик поставимо:

```
C:\WINDOWS\System32\cmd.exe /k C:\Programs\context\tex\setuptex.bat
```

Још једна могућност, ако не желимо да покрећемо скрипту сваки пут када желимо да користимо ConTeXt, а ни да за стално поставимо потребне променљиве окружења, је да то урадимо из самог едитора текста, уместо да ConTeXt извршавамо из терминала. Како се то ради, зависи од конкретног текст едитора који користите. ConTeXt вики пружа информације о начину подешавања разних уобичајених текст едитора: LEd, Notepad + +, Scite, TeXnicCenter, TeXworks, vim и још неких.

1.3 Ажурирање верзије „ConTeXt Standalone” или враћање на ранију верзију

Mark IV се још увек развија, тако да се „ConTeXt Standalone” често ажурира. Ако желите да ажурирате своју инсталацију, једноставно поновите процес: преузмите нову верзију „first-setup.sh” и покрените је.

Ако из било ког разлога желимо да се вратимо на претходну верзију „ConTeXt Standalone”, једноставно покренемо „first-setup” са опцијом „-context=датум”, где је *датум* датум

одговарајуће верзије коју желимо да обновимо. Имајте на уму да се датум уноси у америчком формату месец-дан-година.

Комплетна листа ConTeXt верзија и датума који су им придружени може да се пронађе на [овој адреси](#).

Коначно, имајте на уму да након реинсталације система, било да је то ажурирање или враћање на претходну верзију, на GNU Linux системима морате поново да извршите корак 4 инсталације, који сам назвао „Коначна подешавања”.

2 Инсталација LMTX

Ако планирамо да користимо само ConTeXt Mark IV, а своје пројекте не желимо да компајлирамо директно са LuaTeX већ са LuaMetaTeX, упрошћеном LuaTeX машином која користи мање системских ресурса и која може да ради на *мање моћним* системима, уместо „ConTeXt Standalone” треба да инсталирамо LMTX. То је најновија верзија система ConTeXt. Име је акроним T_EX машине која се користи: LuaMetaTeX. Ова верзија је лансирана у 2019. години и почевши од оквирно маја 2020. је препоручена подразумевана ConTeXt дистрибуција као што се сугерише у [ConTeXt викију](#).

Тренутни развој LMTX је интензиван, а бета верзија може да се промени неколико пута недељно. Још важније, нека измена привремено може да унесе одређене некомпатибилности са Mark IV, па тако на пример, док пишем ове линије, последња LMTX верзија (4 август 2020) прави грешке са `\Caps` командом. Зато бих почетницима за сада саветовао да раде са „ConTeXt Standalone”.

2.1 Сама инсталација

Инсталација је једноставна:

- **Корак 1:** одлучите у који директоријум желите да инсталирате LMTX, и ако је потребно, креирајте га. Претпоставићу да се инсталација врши у директоријум под именом „context” који се налази у нашем почетном директоријуму.
- **Корак 2:** преузмите (у инсталациони директоријум) zip фајл са [ConTeXt викија](#) који одговара вашем оперативном систему и процесору. Може бити било шта од следећег:
 - GNU/Linux
 - ★ X86 процесор
 - ▷ [32 битна верзија](#).
 - ▷ [64 битна верзија](#).
 - ★ ARM процесор
 - ▷ [32 битна верзија](#).
 - ▷ [64 битна верзија](#).
 - Microsoft Windows
 - ★ [32 битна верзија](#)
 - ★ [64 битна верзија](#)
 - Mac OS, [64 битна верзија](#)
 - FreeBSD
 - ★ [32 битна верзија](#).

- ★ 64 битна верзија.
- OpenBSD 6.6
 - ★ 32 битна верзија.
 - ★ 64 битна верзија.
- OpenBSD 6.7
 - ★ 32 битна верзија.
 - ★ 64 битна верзија.

Ако не знате да ли је ваш систем 32-битни или 64-битни, вероватно је – осим ако ваш компјутер није баш стар – 64-битни. Ако не знате да ли је ваш процесор X86 или ARM, највероватније је X86.

- **Корак 3:** распакујте у инсталациони директоријум фајл који сте преузели у претходном кораку. Креираће се директоријум „bin” и два фајла, један под називом „installation.pdf”, који садржи детаљније информације о инсталацији, и други који је конкретни програм за инсталацију и назива се „install.sh” (на Unix системима) или „install.bat” (на Windows системима).
- **Корак 4:** покрените инсталациони програм („install.sh” или „install.bat”). Потребна је Интернет веза, јер инсталациони програм са веба преузима потребне фајлове.
 - На Unix системима се инсталациони програм из инсталационог директоријума покреће из терминала, било са bash, било са sh. Нису потребне администраторске привилегије, осим ако се инсталациони директоријум налази ван корисничког „home” (почетног) директоријума.
 - На Windows системима морате да отворите терминал, пређете у инсталациони директоријум и из терминала покренете install.bat. Ни овде није потребно да се инсталациони програм покрене као системски администратор, али се то препоручује да би се употребили симболички линкови фајлова, и тако сачува простор на диску.
- **Корак 5** обавестите систем о путањи до LMTX:

На Windows системима, инсталациони програм генерише фајл под називом „setpath.bat” који ажурира све конфигурационе фајлове неопходне да Windows зна да сте на систем инсталирали LMTX и где сте то урадили. На GNU Linux системима, FreeBSD или Mac OS не генерише се *скрипћа* која аутоматизује овај посао, тако да сами морамо унети адресу ConTeXt бинарних фајлова у променљиву PATH система, што можемо урадити ако из инсталације извршимо у терминалу:

```
export PATH="ИнсталациониДир/tex/texmf-Платформа/bin:$PATH"
```

где је *ИнсталациониДир* директоријум инсталације (на пример, „/home/user/context”) а *texmf-Платформа* ће варирати у зависности од верзије LMTX коју смо инсталирали. На пример, за инсталацију на 64-битни Linux систем, *texmf-Платформа* ће бити „texmf-linux-64”. Дакле, требало би да у терминалу извршимо следећу команду:

```
export PATH="/home/user/context/texmf-linux-64/bin:$PATH"
```


Ова команда ће LMTX поставити на системску путању, само док је отворен терминал из којег је покренута. Ако желимо да се то аутоматски ради сваки пут када се терминал отвори, команду морамо да уметнемо у конфигурациони фајл платформе програма *љуске* коју систем подразумевано користи. Име овог фајла се мења зависно од програма *љуске*: `bash`, `sh`, `zsh`, `ksh`, `tcsh`, `csh`... На већини Linux система, који користе, фајл се назива „`.bashrc`”, тако да би из нашег почетног директоријума требало да покренемо следећу команду:

```
echo 'export PATH="/home/user/context/texmf-linux-64/bin:$PATH' >> .bashrc
```

Важна напомена: када извршимо овај корак, искључићемо могућност употребе других ConTeXt верзија на нашем систему, као што је она која је део TeX Live или „ConTeXt Standalone”. Ако желимо обе верзије буду компатибилне, боље је да користимо процедуру описану у [одељку 3](#).

2.2 Инсталирање модула проширења у LMTX

ConTeXt LMTX нема процедуру за инсталирање или ажурирање ConTeXt модула проширења. Међутим, у ConTeXt викију постоји *скрипџа* која нам омогућава да инсталирамо и ажурирамо све модуле заједно са остатком инсталације.

За то нам је потребна копија [поменуте скрипџе](#), налепите је у текст фајл који треба да се налази у главном LMTX инсталационом директоријуму (онај у коме се налази `install.sh` или `install.bat`) и покрените га из терминала. Лично сам проверио да ово ради на GNU Linux систему. Нисам сигуран да ли би радило на Windows систему, јер немам ниједну верзију тог система на којој би могао да проверим.

2.3 Ажурирање LMTX

Ажурирање LMTX је просто поновно покретање инсталационог програма: он ће проверити инсталиране фајлове и упоредити их са онима на веб серверу, па ће ажурирати оне који су измењени.

Ако се веб сајт са којег су преузети фајлови променио, очигледно морамо да у *скрипџи* променимо ту адресу; мада је вероватно лакше да се у исти директоријум преузме нова верзија архиве инсталационих фајлова и да се из ње распакује нови „`install.sh`” или „`install.bat`”; или, још лакше, распакујете фајл са инсталационим програмом и реинсталирате га без потребе да уклањате старе фајлове.

2.4 Креирање фајла који у меморију учитава променљиве потребне за LMTX (само GNU/Linux системи)

Као што већ знамо, „ConTeXt Standalone” садржи („`tex/setuptex`”) фајл који у меморију учитава све променљиве које су потребне за извршавање, али LMTX нема такав фајл. Међутим, једноставно и сами можемо да га креирамо и да га сачувамо, на пример, као „`setuplmtx`” у „`tex`” директоријум. Команде које би овај фајл требало да има су:

```
export PATH="~/context/LMTX/tex/texmf-linux-64/bin:$PATH"
```



```
echo "Додавање ~/context/LMTX/tex/texmf-linux-64/bin у PATH"
export TEXROOT=~/.context/LMTX/tex
echo "Постављање ~/context/LMTX/tex као TEXROOT"
export OSFONDIR=~/.fonts:/usr/share/fonts:/usr/share/texmf/fonts/opentype/"
echo "Учитавање фонт директоријума у меморију"
alias lmtx=~/.context/LMTX/tex/texmf-linux-64/bin/context"
echo "Креирање алијаса за покретање lmtx"
```

Овиме, осим читавања у меморију путања и променљивих неопходних за извршавање LMTX, постављамо и „lmtx” команду као синоним за „context”.

Када креирамо овај фајл, пре него што будемо могли да користимо LMTX, тамо где намеравамо да га користимо, у терминалу би требало да извршимо следеће:

```
source ~/.context/LMTX/tex/setuplmtx
```

све ово претпоставља да је LMTX инсталиран у „~/context/LMTX” и да смо овај фајл назвали „setuplmtx” и сачували га у „~/context/LMTX/tex”.

Ово изнад је оно што ја радим да бих могао да LMTX користим на исти начин као што сам користио „ConTeXt Standalone”. Међутим, не искључујем могућност да у LMTX није неопходно, на пример, да се у меморију чита променљива OSFONDIR, јер сам запањен чињеницом да ConTeXt вики ништа не каже о овоме.

3 Употреба неколико верзија система ConTeXt на истом систему (само за Unix системе)

Могућност оперативног система под називом `alias` нам омогућава да различитим верзијама система ConTeXt доделимо различита имена. Тако на пример, можемо користити верзију система ConTeXt која се испоручује уз TeX Live и LMTX; или *Standalone* верзију и LMTX.

На пример, ако сачувамо верзије LMTX система преузете у јануару и августу 2020. године у различите директоријуме, могли би да напишемо следеће две инструкције у „.bashrc” (или еквивалентан фајл који се подразумевано чита када се отвара терминал):

```
alias lmtx-01="/home/user/context/202001/tex/texmf-linux-64/bin/context"
alias lmtx-08="/home/user/context/202008/tex/texmf-linux-64/bin/context"
```

Ове наредбе ће име `lmtx-01` придружити верзији LMTX инсталираној у директоријум „context/202001”, а `lmtx-08` верзији инсталираној у „context/202008”.

Додатак Б

Команде за генерисање математичких и осталих симбола

У табелама које следе можете пронаћи команде које генеришу различите симболе, које сам лично проверио један по један; већина њих (мада не и сви) се најчешће користе у математици.

Свестан сам да је начин на који су сређени отворен за побољшање. Проблем је што долазим из књижевног света, па не знам за шта се у математици користе многи од ових симбола; а често нисам ни сигуран да ли је заиста у питању симбол који се користи у математици. Зато сам направио групу симбола за које сам прилично сигуран да се не користе у математици, а што се тиче осталих, груписао сам их према одређеним препознатљивим облицима (троуглови, квадрати, звездице, ромбови, стрелице, тачке). Остатак симбола, који се *вероватно* користе у математици, поређао сам азбучно (према команди која их генерише).

Још важније, табеле нису потпуне. Сигуран сам да постоји још много симбола које ConTeXt може да генерише, мада нисам пронашао никакав документ или веб страницу која их све скупља. Они који су овде представљају у највећој мери симболе који раде у TeX или у L^ATeX, а које сам проверио да такође раде и у ConTeXt. Многи од ових симбола раде само у математичком режиму система L^ATeX (ако се поставе унутар '\$'). У систему ConTeXt, као што се лако види из табела које следе, то је неопходно само у малом броју случајева.

Симболи валута и за правну употребу

© \copyright	® \registered	€ \textcent
Ⓢ \textcircledP	¤ \textcurrency	\$ \textdollar
Ⓓ \textdong	€ \texteuro	f \textflorin
£ \textsterling	¥ \textyen	™ \trademark

Троуглови, кругови, квадрати и остали облици

△ \triangle	○ \bigcirc	□ \square
◁ \triangleleft	◦ \circ	■ \blacksquare
▷ \triangleright	• \bullet	◻ \boxdot
▽ \triangledown	⊗ \circledast	⊖ \boxminus
▼ \blacktriangledown	⊙ \circledcirc	⊕ \boxplus
◀ \blacktriangleleft	⊖ \circleddash	⊗ \boxtimes
▶ \blacktriangleright	⊕ \bigoplus	* \ast
▲ \blacktriangle	⊗ \bigotimes	✱ \maltese
≐ \triangleq	⊕ \oplus	★ \star
◇ \diamond	⊖ \ominus	♣ \clubsuit
◊ \lozenge	⊗ \otimes	♥ \heartsuit
◆ \blacklozenge	⌘ \oslash	♠ \spadesuit
◇ \diamondsuit	∘ \odot	∅ \varnothing

Стрелице:

\leftarrow	<code>\leftarrow, \gets</code>	\rightarrow	<code>\rightarrow, \to</code>	\leftrightarrow	<code>\leftrightharrow</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>
\nLeftarrow	<code>\nLeftarrow</code>	\nRrightarrow	<code>\nRrightarrow</code>	\leftrightharpoons	<code>\leftrightharpoons</code>
\Lsh	<code>\Lsh</code>	\Rsh	<code>\Rsh</code>	\leftrightsquigarrow	<code>\leftrightsquigarrow</code>
\mapsto	<code>\mapsto</code>	\mapsto	<code>\mapsto</code>	\nLeftarrow	<code>\nLeftarrow</code>
\longmapsto	<code>\longmapsto</code>	\longmapsto	<code>\longmapsto</code>	\nleftarrow	<code>\nleftarrow</code>
\Mapsfrom	<code>\Mapsfrom</code>	\Mapsto	<code>\Mapsto</code>	\rightharpoonleft	<code>\rightharpoonleft</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>
\leftharpoonup	<code>\leftharpoonup</code>	\rightharpoonup	<code>\rightharpoonup</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>	\rightsquigarrow	<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>

Интерпункција

\because	<code>\because</code>	\cdot	<code>\cdot</code>	\cdot	<code>\cdot</code>
\cdots	<code>\cdots</code>	\cdot	<code>\cdot</code>	\cdot	<code>\cdot</code>
\ddots	<code>\ddots</code>	\cdot	<code>\cdot</code>	\cdot	<code>\cdot</code>
\ldots	<code>\ldots</code>	\cdot	<code>\cdot</code>	\cdot	<code>\cdot</code>
\vdots	<code>\vdots</code>	\cdot	<code>\cdot</code>	\cdot	<code>\cdot</code>

Симболи углавном за математичку употребу:

\aleph	<code>\aleph</code>	\amalg	<code>\amalg</code>	\angle	<code>\angle</code>
\approx	<code>\approx</code>	\approx	<code>\approx</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>	\bigoplus	<code>\bigoplus</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</code>
\ddagger	<code>\ddagger</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</code>
\geqslant	<code>\geqslant</code>	\gg	<code>\gg</code>	\ggg	<code>\ggg</code>
\gtrapprox	<code>\gtrapprox</code>	\gneq	<code>\gneq</code>	\gnsim	<code>\gnsim</code>
\gtrapprox	<code>\gtrapprox</code>	\gtrdot	<code>\gtrdot</code>	\gtreqless	<code>\gtreqless</code>
\gtreqless	<code>\gtreqless</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>

\times	<code>\leftthreetimes</code>	\leq	<code>\leq, \le</code>	\leq	<code>\leqq</code>
\lessdot	<code>\leqslant</code>	\lessapprox	<code>\lessapprox</code>	\lessdot	<code>\lessdot</code>
\lesseqgtr	<code>\lesseqgtr</code>	\lesseqqgtr	<code>\lesseqqgtr</code>	\lessgtr	<code>\lessgtr</code>
\lesssim	<code>\lesssim</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	<code>\neg o \lnot</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>	\succnsim	<code>\succnsim</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>	$\vee o \lor$	<code>\vee o \lor</code>	\veebar	<code>\veebar</code>
\Vdash	<code>\Vdash</code>	\Vdash	<code>\Vdash</code>	$\wedge o \land$	<code>\wedge o \land</code>
\wp	<code>\wp</code>	\wr	<code>\wr</code>		

Остали симболи

\P	<code>\P</code>	\S	<code>\S</code>	$^{\circ}\text{C}$	<code>\celsius</code>
\checkmark	<code>\checkmark</code>	\O	<code>\mho</code>	Ω	<code>\ohm</code>
$^{\circ}$	<code>\textdegree</code>	\N	<code>\textnumero</code>	$\text{\hspace{1pt}}$	<code>\textvisiblespace</code>

Додатак В

Индекс команди

У индексу који следи су наведене команде које се разматрају у овом уводу. Неке су само поменуте, скоро у пролазу, па у том случају страница која се појављује у индексу означава где су поменуте. Али остале команде су тема много детаљнијих објашњења. У том случају индекс приказује само место на којем почиње детаљно објашњење, мада команда може бити цитирана и на осталим местима у уводу.

У индекс није укључено:

- `\stop` **Нешто** која затвара конструкцију претходно отворену са `\start` **Нешто**, осим ако текст не каже нешто посебно у вези са `\stop` командом, или ако се третира на месту које се разликује од оног на којем се налази одговарајућа `\start` команда.
- Команде намењене генерисању симбола, све су наведене у [додатку Б](#).
- У случају команди које генеришу дијакритик или слово, а које имају верзију за генерисање великог и верзију за генерисање малог слова, дата је само верзија за мало слово.

* * *

C

`\Cap` [164](#)

H

`\H` [158](#)

`\HL` [231](#)

N

`\NB` [232](#)

`\NC` [232](#)

`\NN` [232](#)

`\NR` [232](#)

R

`\ReadFile` [66](#)

T

`\TB` [232](#)

`\TeX` [17, 18](#)

V

`\VL` [231](#)

W

`\WORD` [164](#)

`\WORDS` [164](#)

`\Word` [164](#)

`\Words` [164](#)

a

`\aa` [158](#)

`\aacute` [157](#)

`\about` [145](#)

`\abreve` [157](#)

`\acircumflex` [157](#)

`\adaptlayout` [81](#)

`\adaptpapersize` [76](#)

`\adiaeresis` [157](#)

`\ae` [158](#)

`\aeligature` [158](#)

`\agrave` [157](#)

`\alpha` [159](#)

`\amacron` [157](#)

`\aring` [158](#)

`\at` [145](#)

`\atilde` [157](#)

`\atleftmargin` 90
`\atpage` 147
`\atrighmargin` 90

b

`\backslash` 39
`\\` 187
`\begingroup` 53
`\beta` 159
`\bf` 98
`\bfa` 99
`\bfb` 99
`\bfc` 99
`\bfd` 99
`\bfx` 99
`\bfxx` 99
`\bgroup` 52, 53
`\bi` 98
`\bia` 99
`\bib` 99
`\bic` 99
`\bid` 99
`\bigbodyfont` 101
`\bix` 99
`\bixx` 99
`\blank` 62, 181
`\bold` 98
`\bolditalic` 98
`\boldslanted` 98
`\break` 187
`\bs` 98
`\bsa` 99
`\bsb` 99
`\bsc` 99
`\bsd` 99
`\bsx` 99
`\bsxx` 99
`\buildmathaccent` 160
`\buildtextaccent` 160
`\buildtextbootomcomma` 160
`\buildtextbottomdot` 160
`\buildtextcedilla` 160
`\buildtextgrave` 160
`\buildtextmacron` 160
`\buildtexttognak` 160

c

`\c` 158
`\ca` 172
`\calligraphic` 98
`\cap` 164
`\ccedilla` 158
`\cf` 98
`\chapter` 110
`\chi` 159
`\clip` 227
`\clubpenalty` 193
`\color` 105
`\colored` 105
`\completecontent` 125
`\completelist` 134
`\completelistofchemicals` 135
`\completelistoffigures` 135, 224
`\completelistofgraphics` 135
`\completelistofintermezzi` 135
`\completelistoftables` 135, 229
`\compleindex` 139
`\crlf` 187
`\currentdate` 175

d

`\date` 175
`\de` 172
`\define` 49
`\definealternativestyle` 102
`\defineblank` 182
`\definebodyfontenvironment` 100
`\definebodyfontswitch` 102
`\definecapitals` 165
`\definecharacter` 160
`\definecharacterkerning` 169
`\definecolor` 107
`\definecombination` 237
`\definecombinedlist` 135
`\defineconversionset` 84
`\definedelimitedtext` 176
`\definedescription` 212
`\defineenumeration` 214
`\definefloat` 238
`\definefontfeature` 159
`\definefontstyle` 102
`\defineframed` 218

`\defineframedtext` 218
`\definehead` 121
`\defineinterlinespace` 189
`\defineitemgroup` 212
`\defineitems` 212
`\definelayou` 81
`\definelinenumbering` 191
`\definelines` 190
`\definelist` 134
`\definemargindata` 91
`\definenarrower` 180
`\definernote` 198
`\definepapersize` 75
`\defineparagraphs` 204
`\defineregister` 140
`\definesectionblock` 123
`\definestartstop` 51
`\definestretched` 169
`\definesymbol` 207
`\definetext` 89
`\definetype` 167
`\definotyping` 167
`\delta` 159
`\dontleavehmode` 98

e

`\eacute` 157
`\ebreve` 157
`\ecircumflex` 157
`\ediaeresis` 157
`\egrave` 157
`\egroup` 52, 53
`\em` 103
`\emacron` 157
`\emdash` 63
`\en` 172
`\enableregime` 59
`\endash` 63
`\endgraf` 178
`\endgroup` 53
`\endnote` 197
`\enskip` 169
`\environment` 67
`\epsilon` 159
`\es` 172
`\eta` 159
`\etilde` 157

`\externalfigure` 221

f

`\fillinline` 216
`\footnote` 197
`\fr` 172
`\framed` 217
`\from` 152

g

`\gamma` 159
`\getbuffer` 218
`\getmarking` 88
`\godown` 183
`\goto` 153

h

`\hairline` 215
`\handwritten` 98
`\hbox` 186
`\head` 208
`\hfill` 169
`\high` 166
`\hl` 216
`\hskip` 169
`\hw` 98
`\hyphen` 63
`\hyphenatedurl` 151
`\hyphenatedurlseparator` 152
`\hyphenation` 186

i

`\i` 158
`\iacute` 157
`\ibreve` 157
`\icircumflex` 157
`\idiaeresis` 157
`\igrave` 157
`\imacron` 157
`\in` 145
`\index` 137
`\inframed` 217
`\inner` 90
`\inneredge` 90
`\innermargin` 90
`\inleft` 90
`\inleftedge` 90

- `\inleftmargin` 90
- `\inmargin` 90
- `\inothet` 90
- `\inouter` 90
- `\inouteredge` 90
- `\inoutermargin` 90
- `\input` 65
- `\inright` 90
- `\inrightedge` 90
- `\inrightmargin` 90
- `\iota` 159
- `\it` 98
- `\ita` 99
- `\italic` 98
- `\italicbold` 98
- `\itb` 99
- `\itc` 99
- `\itd` 99
- `\item` 208
- `\items` 211
- `\itilde` 157
- `\its` 209
- `\itx` 99
- `\itxx` 99
-
- j**
- `\j` 158
-
- k**
- `\kappa` 159
- `\kcedilla` 158
-
- l**
- `\l` 158
- `\labeltext` 174
- `\lambda` 159
- `\language` 171
- `\lastpagenumber` 85
- `\lastrealpagenumber` 85
- `\lastuserpagenumber` 85
- `\lcedilla` 158
- `\leftaligned` 193
- `\letterbackslash` 152
- `\letterescape` 152
- `\letterhash` 152
- `\letterhat` 39
- `\letterpercent` 152
-
- `\lettertilde` 39
- `\linenote` 196
- `\loadinstalledlanguages` 171
- `\lohi` 166
- `\low` 166
-
- m**
- `\mainlanguage` 171
- `\mar` 209
- `\margintext` 90
- `\mediaeval` 98
- `\midaligned` 193
- `\minus` 63
- `\mirror` 227
- `\mono` 98
- `\month` 175
- `\mu` 159
-
- n**
- `\ncedilla` 158
- `\noheaderandfooterlines` 88
- `\noindentation` 179
- `\nolist` 112, 115
- `\nomarking` 112, 115
- `\normal` 98
- `\note` 197
- `\notesenabledfalse` 202
- `\notesenabledtrue` 202
- `\nowhitespace` 181
- `\nu` 159
-
- o**
- `\o` 158
- `\oacute` 157
- `\obreve` 157
- `\ocircumflex` 157
- `\odiaeresis` 157
- `\oe` 158
- `\oeligature` 158
- `\ograve` 157
- `\omacron` 157
- `\omega` 159
- `\omicron` 159
- `\os` 98
- `\otilde` 157
- `\overbar` 217
- `\overbars` 217

`\overstrike` 217
`\overstrikes` 217

p

`\page` 85
`\pagenumber` 85, 88
`\pagereference` 144
`\par` 178
`\parindent` 55
`\parskip` 56
`\part` 110
`\phi` 159
`\pi` 159
`\placebookmarks` 153
`\placechemical` 233
`\placecontent` 125
`\placefigure` 223
`\placefloat` 233
`\placegraphic` 233
`\placeindex` 139
`\placeintermezzo` 233
`\placelist` 134
`\placelistofchemicals` 135
`\placelistoffigures` 135, 224
`\placelistofgraphics` 135
`\placelistofintermezzi` 135
`\placelistoftables` 135, 229
`\placelocalfootnotes` 198
`\placenotes` 198
`\placetable` 228
`\pretolerance` 186
`\product` 68
`\project` 69, 70
`\psi` 159

q

`\qqquad` 169
`\quad` 169
`\quotation` 176
`\quote` 176

r

`\r` 158
`\ran` 209
`\rcedilla` 158
`\readfile` 66
`\realpagenumber` 85

`\ref` 146
`\reference` 144
`\regular` 98
`\rho` 159
`\rightaligned` 193
`\rm` 98
`\rma` 99
`\rmb` 99
`\rmc` 99
`\rmd` 99
`\rmx` 99
`\rmxx` 99
`\roman` 98
`\rotate` 227

s

`\sans` 98
`\sansserif` 98
`\sc` 98
`\scedilla` 158
`\section` 110
`\seeindex` 139
`\serif` 98
`\sethyphenatedurlafter` 152
`\sethyphenatedurlbefore` 152
`\sethyphenatedurlnormal` 152
`\setupalign` 191
`\setuparranging` 81
`\setupbackgrounds` 104
`\setupblank` 182
`\setupbodyfont` 95, 97
`\setupbottomtexts` 90
`\setupcapitals` 165
`\setupcaption` 235
`\setupcaptions` 235
`\setupcharacterkerning` 169
`\setupcolors` 104
`\setupcolumns` 203
`\setupcombinedlist` 126
`\setupcounter` 199
`\setupdescription` 213
`\setupendnotes` 199
`\setupenumeration` 215
`\setupexternalfigures` 226
`\setupfillinlines` 216
`\setupfloat` 237
`\setupfloats` 237

- `\setupfooter` 88
- `\setupfootertexts` 87, 89
- `\setupfootnotes` 199
- `\setupframed` 218
- `\setupframedtext` 218
- `\setuphead` 113
- `\setupheader` 88
- `\setupheadertexts` 87, 89
- `\setupheadnumber` 116
- `\setupheads` 113
- `\setupheadtext` 126
- `\setuphyphenmark` 171
- `\setupindenting` 179
- `\setupinteraction` 149
- `\setupinterlinespace` 188
- `\setuplabeltext` 174
- `\setuplanguage` 173
- `\setuplayout` 79
- `\setuplinenote` 196
- `\setuplinenumbering` 190
- `\setuplines` 190
- `\setuplist` 129
- `\setupmargindata` 91
- `\setupnarrower` 180
- `\setupnotation` 199
- `\setupnotations` 199
- `\setupnote` 199
- `\setupnotes` 199
- `\setuppagenumbering` 83
- `\setuppapersize` 73
- `\setupparagraphs` 204
- `\setupregister` 139, 140
- `\setupsectionblock` 123
- `\setupspacing` 168
- `\setupstretched` 169
- `\setuptables` 229
- `\setuptextrule` 217
- `\setuptolerance` 186, 193
- `\setuptoptexts` 90
- `\setuptype` 167
- `\setuptyping` 167
- `\setupurl` 151
- `\setupuserpagenumber` 83
- `\setupwhitespace` 180
- `\showbodyfontenvironment` 101
- `\showcolor` 106
- `\showcolorcomponents` 107
- `\showfont` 97
- `\showframe` 78
- `\showinstalledlanguages` 171
- `\showlayout` 78
- `\showsetups` 78
- `\showsymbolset` 162
- `\sigma` 159
- `\sl` 98
- `\sla` 99
- `\slanted` 98
- `\slantedbold` 98
- `\slb` 99
- `\slc` 99
- `\sld` 99
- `\slx` 99
- `\slxx` 99
- `\smalcaps` 98
- `\smallbodyfont` 101
- `\smallbold` 101
- `\smallbolditalic` 101
- `\smallboldslanted` 101
- `\smallitalicbold` 101
- `\smallslanted` 101
- `\smallslantedbold` 101
- `\somewhere` 147
- `\space` 169
- `\ss` 98, 158
- `\ssa` 99
- `\ssb` 99
- `\ssc` 99
- `\ssd` 99
- `\ssx` 99
- `\ssxx` 99
- `\start` 53
- `\startMPpage` 76
- `\startTEXpage` 76
- `\startalignment` 192
- `\startappendices` 122
- `\startbackmatter` 122
- `\startbodymatter` 122
- `\startbuffer` 218
- `\startchapter` 110
- `\startchemical` 218
- `\startcolor` 106
- `\startcolumns` 203
- `\startcombination` 218, 236
- `\startcomponent` 68

- `\startenvironment` 67
- `\startfiguretext` 225
- `\startformula` 218
- `\startframedtext` 217
- `\startfrontmatter` 122
- `\starthiding` 218
- `\startitem` 208
- `\startitemize` 207
- `\startlegend` 219
- `\startlinecorrection` 219
- `\startlinenumbering` 190
- `\startlines` 189
- `\startlocalfootnotes` 198
- `\startmode` 219
- `\startnarrower` 179
- `\startnotmode` 219
- `\startopposite` 219
- `\startpacked` 181
- `\startpagefigure` 76
- `\startpart` 110
- `\startproduct` 68
- `\startproject` 69
- `\startquotation` 219
- `\startsection` 110
- `\startsetups` 51
- `\startstandardmakeup` 219
- `\startsubject` 110
- `\startsubsection` 110
- `\startsubsubsection` 110
- `\startsubsubsubject` 110
- `\startsubsubsubsection` 110
- `\startsubsubsubsubject` 110
- `\starttabulate` 229
- `\starttext` 63
- `\starttextrule` 217
- `\starttitle` 110
- `\starttyping` 166
- `\stop` 53
- `\stoptext` 63
- `\stretched` 168
- `\structureuservariable` 113
- `\sub` 209
- `\subject` 110
- `\subsection` 110
- `\subsubject` 110
- `\subsubsection` 110
- `\subsubsubject` 110
- `\subsubsubsection` 110
- `\subsubsubsubject` 110
- `\support` 98
- `\switchtobodyfont` 97
- `\sym` 208
- `\symbol` 163
- t**
- `\tau` 159
- `\tcedilla` 158
- `\teletype` 98
- `\tex` 167
- `\textheight` 81
- `\textreference` 144
- `\textrule` 217
- `\textwidth` 81
- `\tf` 98
- `\tfa` 99
- `\tfb` 99
- `\tfc` 99
- `\tfd` 99
- `\tfx` 99
- `\tfxx` 99
- `\theta` 159
- `\thinrule` 215
- `\thinrules` 215
- `\title` 110
- `\tolerance` 186
- `\translate` 175
- `\tt` 98
- `\tta` 99
- `\ttb` 99
- `\ttc` 99
- `\ttd` 99
- `\ttx` 99
- `\ttxx` 99
- `\tx` 99
- `\txx` 99
- `\typ` 167
- `\type` 166
- u**
- `\u` 157
- `\uacute` 157
- `\ubreve` 157
- `\ucircumflex` 157
- `\udiaeresis` 157

`\ugrave` 157
`\umacron` 157
`\underbar` 217
`\underbars` 217
`\unhyphenated` 186
`\upsilon` 159
`\useURL` 150
`\usecolors` 105
`\useexternalfigure` 222
`\usemodule` 171
`\useregime` 59
`\userpagenumber` 85
`\usesymbols` 162
`\utilde` 157

v

`\varepsilon` 159
`\varkappa` 159
`\varphi` 159
`\varpi` 159
`\varrho` 159
`\varsigma` 159
`\vartheta` 159
`\vbox` 86
`\vfill` 183
`\vl` 216
`\vskip` 183

w

`\whitespace` 181

`\widowpenalty` 193
`\word` 164
`\wordright` 193
`\writebetweenlist` 132
`\writetolist` 131

x

`\xi` 159

z

`\zeta` 159

к

контролни симбол

`\,` 169

`_` 169

р

резервисани карактери

`\{` 39

`\}` 39

`\$` 39

`\#` 39

`\%` 39

`\&` 39

`_` 39

`\backslash` 39

`\letterhat` 39

`\lettertilde` 39

`\|` 39