

Joaquín Ataz-López

Nepříliš krátký
úvod
do ConT_EXtu Mark IV

Nepříliš krátký úvod do ConT_EXtu Mark IV

Verze 1.6 [2. ledna 2021]

Český překlad: leden 2022

© 2020–2022, Joaquín Ataz-López

Původní název: Una introducción (no demasiado breve) a ConT_EXt Mark IV

Anglický překlad: Joaquínův dobrý kamarád, který si přeje zůstat v anonymitě.

Český překlad: Tomáš Hála, Aleš Ďurčanský, Jan Janča, Natália Várošová, Robert Blaha, Tamara Kocurová, Zdeněk Svoboda

Autor tohoto textu i překladatelé české verze povolují jeho volné šíření a používání, včetně práva kopírovat a šířit tento dokument v digitální podobě, pod podmínkou, že bude zachováno uvedení autorství původního textu i překladu, a že nebude součástí žádného softwarového balíku nebo sady nebo dokumentace, jejíž podmínky používání nebo šíření nezahrnují právo příjemců na volné kopírování a šíření.

Stejně tak je povolen překlad dokumentu za předpokladu, že je uvedeno autorství původního textu (případně překladu) a že přeložený text je šířen pod licencí FDL nadace *Free Software Foundation*, licencí *Creative Commons*, která povoluje kopírování a další šíření, nebo podobnou licencí.

Bez ohledu na výše uvedené, zveřejnění, marketing nebo překlad tohoto dokumentu v listinné podobě bude vyžadovat výslovný písemný souhlas původního autora, případně i autorů překladu.

Historie verzí (výběr):

- 18. srpna 2020: Verze 1.0 (pouze ve španělštině): Původní verze.
- 21. října 2020: Verze 1.5 (pouze ve španělštině): Verze po zapracování připomínek a oprav uživatelů ConT_EXtu diskusního listu *NTG-context*.
- 2. ledna 2021: Verze 1.6: Opravy navržené nové a pečlivé revizi dokumentu v souvislosti s jeho překladem do angličtiny. První verze anglického překladu.

Obsah

I	Co je ConTeXt a jak s ním pracujeme	6
1	Příkazy a další základní koncepty ConTeXtu	6
1.1	Vyhrazené znaky ConTeXtu	7
1.2	Samotné příkazy	10
1.3	Rozsah působnosti příkazů	13
1.4	Parametry provádění příkazů	16
1.5	Shrnutí syntaxe příkazů, jejich parametrů options a použití hranatých a složených závorek při jejich volání	19
1.6	Oficiální seznam příkazů ConTeXtu	20
1.7	Definování nových příkazů	21
1.8	Další základní koncepty	25
1.9	ConTeXt pro samouky	29
2	Zdrojové soubory a projekty	32
2.1	Kódování zdrojových souborů	32
2.2	Znaky ve zdrojovém souboru (souborech)	35
2.3	Jednoduché a vícesouborové projekty	38
2.4	Struktura zdrojového souboru v jednoduchých projektech	39
2.5	Správa více souborů ve stylu T _E Xu	40
2.6	ConTeXt projekty jako takové	43
II	Dokument jako celek a jeho struktura	49
1	Stránky a dokumentové stránkování	49
1.1	Velikost stránky	50
1.2	Prvky na stránce	54
1.3	Rozvržení stránky (<code>\setuplayout</code>)	56
1.4	Číslování stránek	60
1.5	Vynucené nebo navrhované konce stránek	63
1.6	Záhlaví a zápatí	65
1.7	Vložení textových prvků na hrany a okraje stránky	68
2	Fonts and colours in ConTeXt	71
2.1	Typografická písma zahrnutá v „ConTeXt Standalone“	71
2.2	Vlastnosti písma	72
2.3	Nastavení hlavního písma dokumentu	75
2.4	Změna písma nebo některých funkcí písma	77

2.5	Další záležitosti týkající se používání některých alternativních řešení	83
2.6	Použití a konfigurace barev	85
3	Struktura dokumentu	90
3.1	Strukturální členění v dokumentech	90
3.2	Typy sekcí a jejich hierarchie	91
3.3	Syntaxe společná pro příkazy sekcí	93
3.4	Formát a konfigurace sekcí a jejich názvů	95
3.5	Definice nových příkazů sekcí	105
3.6	Makrostruktura dokumentu	105
4	Obsah, rejstříky, seznamy	107
4.1	Obsah	107
4.2	Seznamy, kombinované seznamy a obsah na základě seznamu	117
4.3	Index	121
5	Reference a hypertextové odkazy	127
5.1	Referenční typy	127
5.2	Interní reference	128
5.3	Interaktivní elektronické dokumenty	135
5.4	Hypertextové odkazy na externí dokumenty	137
5.5	Vytvoření záložek ve finálním PDF	141
III	Vybrané oblasti úpravy dokumentu	143
1	Znaky, slova, text a horizontální prostor	143
1.1	Získání znaků, které nejsou běžně dostupné z klávesnice	144
1.2	Speciální formáty znaků	152
1.3	Mezery mezi znaky a slovy	156
1.4	Složená slova	160
1.5	Jazyk textu	161
2	Odstavce, řádky a svislá mezera	168
2.1	Odstavce a jejich charakteristika	168
2.2	Svislá mezera mezi odstavci	171
2.3	Jak ConTeXt vytváří řádky tvořící odstavce	175
2.4	Meziřádkový prostor	180
2.5	Ostatní záležitosti týkající se řádkování	181
2.6	Horizontální a vertikální zarovnání	183
3	Speciální konstrukce a odstavce	187
3.1	Poznámky pod čarou a závěrečné poznámky	187
3.2	Odstavce s více sloupci	196
3.3	Strukturované seznamy	201
3.4	Popisy a výčty	208
3.5	Čáry a rámce	211
3.6	Další zajímavá prostředí a stavby	215

4	Obrázky, tabulky a další plovoucí objekty	217
4.1	Co jsou plovoucí objekty a co dělají?	217
4.2	Externí obrázky	218
4.3	Tabulky	226
4.4	Aspekty společné pro obrázky, tabulky a další plovoucí objekty	233
4.5	Definování dalších plovoucích objektů	238
 Přílohy		 241
————— Czech version —————		
TO BE DONE! —————		

I

Co je ConT_EXt a jak s ním pracujeme

————— Czech version —————

TO BE DONE! —————

————— Czech version —————

TO BE DONE! —————

Chapter 1

Příkazy a další základní koncepty ConT_EXtu

Table of Contents: 1.1 Vyhrazené znaky ConT_EXtu; 1.2 Samotné příkazy;
1.3 Rozsah působnosti příkazů; 1.3.1 Příkazy vyžadující a nevyžadující specifi-
kaci rozsahu; 1.3.2 Příkazy vyžadující výslovné označení místa; 1.4 Parametry

provádění příkazů; 1.4.1 Příkazy; 1.4.2 Příkazy upravující chování jiných příkazů (`\setupSomething`); 1.4.3 Vytváření vlastních variant konfigurovatelných příkazů (`\defineSomething`); **1.5 Shrnutí syntaxe příkazů, jejich parametrů options a použití hranatých a složených závorek při jejich volání;** **1.6 Oficiální seznam příkazů ConT_EXtu;** **1.7 Definování nových příkazů;** 1.7.1 Základní postup definování nových příkazů; 1.7.2 Vytváření nových prostředí; **1.8 Další základní koncepty;** 1.8.1 Skupiny; 1.8.2 Rozměry; **1.9 ConT_EXt pro samouky;**

Již jsme si ukázali, že ve zdrojovém souboru, stejně jako v textu, obsaženém v našeho budoucím formátovaném dokumentu, najdeme instrukce potřebné k tomu, abychom ConT_EXtu sdělili jak chceme, aby náš rukopis vypadal po zpracování. Tyto instrukce můžeme nazývat „příkazy“, „makra“ nebo „řídící sekvence“.

Z hlediska vnitřního fungování ConT_EXtu (resp. fungování T_EXu) je rozdíl mezi *primitivy* a *makry*. Primitiva jsou jednoduché instrukce, které nemohou být rozděleny na další jednodušší instrukce. Makra jsou instrukce, které lze rozdělit na další jednodušší instrukce, které lze případně rozložit na další instrukce, a tak dále a tak dále. Většina instrukcí ConT_EXtu jsou ve skutečnosti makra. Z pohledu programátora je mezi makry a primitivy důležitý rozdíl. Z pohledu uživatele však tento rozdíl tak důležitý není: v obou případech máme k dispozici instrukce, které budou provedeny, aniž bychom se museli starat o to, jak fungují na vnitřní úrovni. Proto se v dokumentaci ConT_EXtu běžně hovoří o *příkazu*, v případě kdy jde o sdělení z pohledu uživatele a o *makru* v případě, když jde o sdělení pro programátora. Jelikož v tomto návodu zohledňujeme primárně uživatelský pohled, budu používat oba termíny jako synonyma.

Pomocí *příkazů* rozkazujeme ConT_EXtu něco udělat; *řídíme* tak, s jejich pomocí, běh programu. Proto když KNUTH, otec T_EXu, hovoří o primitivech nebo makrech, používá termín *řídící sekvence*. Domnívám se, že to je nejpřesnější termín ze všech zmíněných. Budu ho používat v případech, kdy bude třeba rozlišit mezi *řídícími znaky* a *řídícími slovy*.

Instrukce ConT_EXtu jsou dvou základních typů: vyhrazené znaky a příkazy, které se tak ostatně jmenují.

1.1 Vyhrazené znaky ConT_EXtu

Když ConT_EXt čte zdrojový soubor, který je složený pouze z textových znaků (protože se jedná o čistě textový soubor), musí nějak rozlišit, co je skutečný text, který má být formátován, a co jsou instrukce, které má provést. Vyhrazené znaky ConT_EXtu umožňují toto rozlišení. ConT_EXt v zásadě předpokládá, že každý znak ve zdrojovém souboru je text, který má být zpracován, pokud se nejedná o jeden z 11 znaků rezervovaných znaků, které se považují za instrukce. Pouze 11 instrukcí? Ne. Existuje pouze 11 vyhrazených znaků; ale protože jeden z nich „\“, mění bezprostředně následující znak, nebo znaky, na instrukci, je skutečně možný počet příkazů je neomezený. ConT_EXt má přibližně 3000 příkazů (sečteme-li příkazy pouze pro Mark II, pro Mark IV a příkazy společné pro obě verze).

Vyhrazené znaky jsou následující:

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

ConTeXt je interpretuje následujícím způsobem:

- \ Tento znak je pro nás nejdůležitější ze všech: naznačuje, že to, co následuje bezprostředně za ním, nesmí být interpretováno jako text, ale jako instrukce. Říká se mu „Escape znak“ nebo „Escape sekvence“ (i když nemá nic společného s klávesou „Esc“, kterou najdete na většině klávesnic).¹
- % Říká ConTeXtu, že to, co následuje až do konce řádku, je komentář. který nesmí být zpracován ani zahrnut do konečného formátovaného souboru. Zavedení komentářů do zdrojového souboru je velmi užitečné. Komentář může pomoci vysvětlit, proč bylo něco provedeno určitým způsobem, a to je velmi užitečné v dokončených zdrojových souborech s ohledem na pozdější revize. kdy si někdy nemůžeme vzpomenout, proč jsme udělali to, co jsme udělali; nebo to také může pomoci jako připomínka něčeho, co bychom mohli potřebovat revidovat. Může být dokonce použita jako pomůcka k nalezení příčiny určité chyby ve zdrojovém souboru, protože umístěním značky komentáře na začátek řádku, vyloučíme tento řádek z kompilace a můžeme zjistit, zda to byl právě on. řádek byl příčinou chyby; lze jej také použít k uložení dvou různých verzí téhož makra, a tak získat různé výsledky. nebo zabránit zkompilování úryvku, u něhož si nejsme jisti, že ho chceme smazat, protože je možné, že jej budeme chtít do souboru vrátit... atd. Jakmile jsme otevřeli možnost, že náš zdrojový soubor obsahuje text, který neuvidí nikdo jiný než my sami, je naše použití tohoto znaku omezeno pouze naší vlastní představivostí. Přiznám se, že toto je jedn z nástrojů, které mi nejvíce chybí, když jediným prostředkem pro psaní textu je textový procesor.
- { Tento znak zahajuje skupinu. Skupiny jsou bloky textu ovlivněné určitými funkcemi. Budeme o nich hovořit v [section 1.8.1](#).
- } Tento znak ukončuje skupinu dříve otevřenou pomocí {.
- # Tento znak se používá pro definování maker. Odkazuje na argumenty makroa. Viz [section 1.7.1](#) dále kapitole.
- ~ Vloží do dokumentu nezlomitenou mezeru, což znamená, že dvě slova oddělená znakem ~ zůstanou na stejném řádku. O této instrukci a o tom, kde by měla být použita více v [section 2.3.1](#).

¹ V počítačové terminologii se klíč, který ovlivňuje interpretaci následujícího znaku, nazývá *escape znak*. Naproti tomu klávesa escape na klávesnicích se tak nazývá proto, že generuje tzv. znak 27 v kódu ASCII, který se v tomto kódování používá jako znak escape. Dnes se používá klávesy Escape spojeno spíše s myšlenkou zrušení probíhající akce.

- | Tento znak se používá ve složených slovech s oddělovačem. Tato slova mohou být rozdělena na slabiky pouze v první části složeného slova. Viz [section 1.4](#).
- \$ Tento znak je přepínačem matematického režimu. Nastavuje tento režim, pokud nebyl nastaven, a ukončuje jej, pokud nastaven byl. V matematickém režimu ConTeXt používá některá písma a pravidla, která se liší od běžných, s cílem optimalizovat zápis matematických vzorců. Přestože psaní matematiky je velmi důležitým použitím ConTeXtu, nebudu se mu v tomto úvodu věnovat. Abych byl přesný: necítím se na to.
- _ Tento znak se používá v matematickém módu a indikuje, že následující znaky jsou spodním indexem. Například, pro zápis x_1 , použijeme `x_1`.
- ^ Tento znak se používá v matematickém módu a indikuje, že následující znaky jsou spodním indexem. Například, pro zápis $(x + i)^{n^3}$ použijeme `$(x+i)^{n^3}$`.
- & V dokumentaci ConTeXtu se píše, že se jedná o vyhrazený znak, ale neuvádí se proč. V obyčejném T_EXu má tento znak v podstatě dvě použití: používá se k zarovnání sloupců v prostředí základních tabulek a v matematickém kontextu tak, aby to, co následuje, bylo považováno za normální text. Je uveden i v úvodní příručce „ConTeXt Mark IV, an Excursion“, ačkoli se neuvádí, k čemu slouží. A jsou uvedeny příklady jeho použití v matematických vzorcích, i když ne toho druhu, jakým je v používán v běžném T_EXu, ale k zarovnání sloupců v rámci komplexních funkcí. Jelikož jsem spíš spisovatel, nemám pocit, že bych mohl provádět další testy, abych zjistil, jaké je přesné použití tohoto vyhrazeného znaku a k čemu slouží.



Lze předpokládat, že při výběru vyhrazených znaků byly zvoleny ty, které jsou dostupné na většině klávesnic, ale současně nejsou často používány v běžném textu. Nicméně se může stát, byť ne příliš často, že některý z nich budeme potřebovat v textu použít. Například když budeme chtít napsat, že něco stojí 100 dolarů (\$100), nebo že ve Španělsku procento řidičů starších 65 let v roce 2018 činilo 16 %. V těchto případech nemůžeme psát vyhrazený znak přímo, ale použít *příkaz*, který vypíše ve výsledném dokumentu vyhrazený znak samotný. Příkaz pro každý z vyhrazených znaků najdete [table ??](#).

Dalším způsobem, jak dostat vyhrazené znaky do výstupu, je použití příkazu `\type`. Tento příkaz odešle to, co převezme jako argument, bez jakéhokoli zpracování či interpretace. V konečném dokumentu je text, zapsaný v příkazu `\type`, zobrazen neproporciálním písmem typickým pro počítačové terminály a psací stroje.

Normálně bychom text, který má `\type` zobrazit, zapsali do složených závorek. Pokud však tento text sám o sobě obsahuje složené závorky, musíme místo nich zapsat text mezi dva stejné znaky, které nejsou součástí textu, který je argumentem příkazu `\type`. Například: `\type*{*, or \type+}+`.

Vyhrazený znak	Příkaz, který jej vygeneruje
\	<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>

Tabulka 1.1 Psaní vyhrazených znaků

Pokud omylem použijeme některý z vyhrazených znaků přímo, jinak než k účelu, ke kterému je určen (protože jsme zapomněli, že se jedná o vyhrazený znak a nelze jej použít jako normální znak), mohou se stát tři věci:

1. Nejčastěji dojde v průběhu zpracování k chybě.
2. Výsledek bude jiný, než jsme čekali. Speciálně k tomu dochází v případě „~“ a „%“. V prvním případě bude v textu místo znaku „~“, který očekáváme, jen mezera. V druhém případě nebude v textu nic, co bylo na na stejném řádku za znakem „%“. I nesprávné použití „\“ může způsobit nečekaný výstup. To v případě, kdy za ním následuje znak nebo text, který ConT_EXt „umí“. Nicméně nesprávné použití „\“ obvykle způsobí chybu při zpracování.
3. Nedojde k problému. U vyhrazených znaků, které se používají v prostředí matematických výrazů (`_` `^` `&`), pokud jsou použity mimo toto prostředí jsou zpracovány jako běžné znaky.



Bod 3 je můj závěr. Pravdou je, že jsem nikde v dokumentaci ConT_EXtu nenašel uvedeno, že tyto tři znaky mohou být uvedeny přímo. Nicméně při svých testech jsem při jejich použití nenarazil na žádnou chybu. Narozdíl od L^AT_EXu například.

1.2 Samotné příkazy

Samotné příklady začínají vždy znakem „\“. V závislosti na tom, co následuje bezprostředně za ním, rozlišujeme:

- a. **Řídící znaky.** Řídící znak začíná escape znakem („\“) a skládá se výhradně z jiných znaků, než z písmen. Jako například „\“, „\1“, „\‘“ nebo „\%“. Jakýkoliv znak nebo symbol, který není písmenem v doslovném slova smyslu, může

být řídicími znakem. Včetně čísel, interpunkčních znamének nebo dokonce mezer. V tomto dokumentu, pokud potřebuji zviditelnit mezeru (bílý znak) tam, kde je potřeba její přítomnost zdůraznit, používám symbol \sqcup . Ve skutečnosti je „ \sqcup “ (zpětné lomítko následované mezerou) běžně používaný řídicí znak, jak brzy uvidíme.

Mezera, nebo tzv. bílý znak, je „neviditelný“ znak, což je, v dokumentu jako je tento, problém. Protože někdy potřebujeme jasně zobrazit, že ve zdrojovém souboru má být mezeru zapsána. Knuth si byl tohoto problému vědom a v jeho knize „The TeXBook“ zavedl zvyk zobrazovat významné mezery pomocí symbolu „ \sqcup “. Takže pokud je například třeba, aby ve zdrojovém souboru byla zapsána dvě slova oddělená dvěma mezerami, můžeme to zapsat jako „slovo1 \sqcup slovo2“.

- b. **Řídicí slova.** Pokud je bezprostředně za zpětným lomítkem písmeno, příkazem je *řídicí slovo*. Tato skupina příkazů je nejpočetnější. Její vlastností je, že jméno příkazu může být tvořeno pouze písmeny. Čísla, interpunkční znaménka, ani jiné symboly nejsou ve jméně příkazu povolena. Pouze malá a velké písmena. A je třeba mít na paměti, ConTeXt rozlišuje mezi malými a velkými písmeny. Což znamená, že příkazy `\mycommand` and `\MyCommand` jsou dva různé příkazy. Na druhou stranu příkazy `\MyCommand1` a `\MyCommand2` budou vyhodnoceny jako shodné, protože znaky ‚1‘ a ‚2‘ nejsou písmena a nejsou tudíž zpracovány jako součást názvu příkazu.



Referenční příručka ConTeXtu neobsahuje popis pravidel pro tvorbu jmen příkazů. Stejně tak žádný z dalších „manuálů“ zahrnutých v „ConTeXt Standalone“. To co uvádím v předchozím odstavci vychází z toho, co platí v TeXu (kde tedy, mimochodem, nejsou znaky s interpunkcí, které nejsou součástí anglické abecedy, zahrnuty mezi „písmena“). Tato pravidla také nabízí dobré vysvětlení mizení mezery za názvem příkazu.

Když ConTeXt čte zdrojový soubor a narazí na escape znak („ \backslash “) ví, že bude následovat příkaz. Pak přečte první následující znak. Pokud to není písmeno, znamená to, že příkaz je řídicí symbol a skládá se pouze z tohoto prvního symbolu. Ale na druhou stranu, pokud první znak po escape sekvenci je písmeno, pak pokračuje ConTeXt ve čtení dalších znaků do doby, než narazí na první znak, který není písmenem. V tom okamžiku ví, že jméno příkazu skončilo. To je důvod, proč názvy příkazů, které jsou řídicími slovy, nemohou obsahovat znaky, které nejsou písmeny.

Pokud je „ne-písmenný“ znak na konci příkazu mezeru předpokládá se, že není součástí jména příkazu, ale že je v kódu jenom proto, aby oddělila jméno příkazu od zbytku textu. Takže ConTeXt tuto mezeru zahodí. Výsledný efekt překvapuje ConTeXtové začátečníky. Protože ve výstupu je výsledek příkazu spojen s následujícím textem. Například, tyto dvě věty:

Znát `\TeX` pomůže, pokud se chcete naučit `\ConTeXt`.

Znát `\TeX`, jakkoliv to není nezbytné, pomůže, pokud se chcete naučit `\ConTeXt`

dopadne ve výstupu takto:

Znát `TeX` pomůže, pokud se chcete naučit ConTeXt.

Znát `TeX`, jakkoliv to není nezbytné, pomůže, pokud se chcete naučit ConTeXt.¹

Všimněte si, že v prvním případě je slovo „TeX“ spojeno s následujícím slovem. Ve druhém případě tomu tak není. Je to tím, že v prvním případě je prvním „ne-písmenným“ znakem mezera, kterou ConTeXt potlačí, protože předpokládá, že jen značí konec příkazů. Ve druhém případě je tím znakem čárka, jejíž výstup potlačen není.

A tento problém nevyřeší přidání mezery navíc. Tedy například zápis:

Znát `\TeX` pomůže, pokud se chcete naučit ConTeXt².

není řešením, protože jiné pravidlo ConTeXtu (které je vysvětleno v kapitole 2.2.1) je takové, že prázdný znak pohltí všechny následující prázdné znaky a tabelátory. Tento problém, naštěstí, nenastává příliš často a řešením je to, aby první „ne-písmenný“ znak za názvem příkazu nebyla mezera. A kandidáti na takový znak jsou dva:

- Vyhrazený znak „`{}`“. Ten, jak jsme si řekli, zahajuje skupinu. A znak „`}`“ skupinu uzavírá. Takže posloupnost znaků „`{}`“ představuje prázdnou skupinu. Ta se nijak neprojeví ve výsledném dokumentu, ale pomůže ConTeXtu odlišit konec jména příkazu. Je také možné vytvořit skupinu obsahující příkaz, například „`{\TeX}`“. Ani v jednom případě není prvním „ne-písmenným“ znakem za jménem příkazu mezera.
- Řídící znak „`\`“ (zpětné lomítko následované mezerou - viz poznámku na str. 11). Tento řídicí znak vloží do výstupu mezeru. Pro správné pochopení logiky ConTeXtu stojí za to podívat se v klidu na to, co se děje, když po sobě následuje řídicí sloco (například `\TeX`) a řídicí znak (např. „`\`“):
 - ConTeXt zaregistruje znak `\` následovaný `T`. Tudíž ví, že jde o řídicí slovo a pokračuje ve čtení dalších znaků, dokud nenarazí na „ne-písmenný“ znak. K tomu dojde, když narazí nad další znak `\` uvozující kontrolní znak.
 - Jakmile zjistí, že jméno příkazu je `\TeX`, zpracuje příkaz a vytiskne TeX do výsledného dokumentu. Poté se vrátí do bodu, kde skončil a přešle další znak následující za druhým zpětným lomítkem.

¹ **Poznámka:** v případech, kdy chci demonstrovat zapsaný kód zdrojového souboru a výsledek ve výstupu, používám v tomto dokumentu dva způsoby: buď jsou zdroj a výstup zapsány vedle sebe ve dvouslupcovém odstavci, nebo je zdrojový kód zapsán fialovou barvou, kterou je obecně používána pro zápis příkladů ConTeXtu, a výsledný výstup je zobrazen červeně.

² Pro vysvětlení symbolu „`\`“ viz poznámku na straně 11.

- Zjistí, že následuje mezeru, tedy „ne-písmenný“ znak. A to znamená, že jde o řídicí znak, který už je přečtený. Tudíž zapíše do výsledného souboru mezeru.
- Nakonec se vrátí zpět ke čtení zdrojového souboru a pokračuje dál.

Tento mechanismus jsem podrobně vysvětlil, protože vypuštění mezer často překvapuje nováčky. Je však třeba poznamenat, že tento problém je relativně málo častý. Protože řídicí slova se obvykle nezapisují do výsledného dokumentu, ale ovlivňují jeho formát a strukturu. Naproti tomu řídicí znaky obvykle do výsledného dokumentu zapisují.

Třetím způsobem, jak se vyhnout problému s prázdným znakem. A tím je definice vlastního příkazu, který již bude obsahovat „ne-písmenný“ znak na konci názvu. Například zápis:

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

vytvoří příkaz nazvaný `\txt`, který udělá to stejné, jako příkaz `\TeX` a funguje správně jen v případě, že je volán s pomlčkou na konci `\txt-`. Pomlčka není, technicky, součástí jména příkazu, ale ten nebude fungovat, pokud nebude uvedena. Má to co dělat s mechanismem definice maker v T_EXu a ta je příliš komplikovaná na to, abych ji vysvětloval zde. Ale funguje to: jakmile je příkaz definován, pokaždé, když použijeme `\txt-`, ConT_EXt jej nahradí `\TeX` a odstraní pomlčku, ale vnitřně podle ní pozná konec jména příkazu. Takže následující mezeru nebude vynechán.

Tento „trik“ nebude správně fungovat v příkazu `\define`, což je specifický příkaz ConT_EXtu pro definici maker.

1.3 Rozsah působnosti příkazů

1.3.1 Příkazy vyžadující a nevyžadující specifikaci rozsahu

Mnoho příkazů ConT_EXtu, zejména ty, které ovlivňují formátování (tučné písmo, kurzíva, malá písmena atd.), nastavují určité chování do doby, dokud se neobjeví jiný příkaz, který ji zakáže nebo který nastaví jinou funkci, která s ní není kompatibilní. Například příkaz `\bf` povoluje tučné písmo a zůstane aktivní, dokud nenarazí na *nekompatibilní* příkaz, jako např. `\tf` nebo `\it`.

Tyto typy příkazů nepotřebují žádné argumenty, protože nejsou určeny pouze pro určitý text. Jejich funkce je limitována pouze na *nastavení* nějaké vlastnosti (tučné písmo, kurzíva, bezpatkové písmo, určitá velikost písma atd.).

When these commands are executed within a *group* (see [section 1.8.1](#)), they also lose their effectiveness when the group they are executed in is closed. Therefore, often in order to make these commands affect only a portion of text, what is done is to generate a group containing that command and the text we want it to affect.

A group is created by enclosing it between curly brackets. Therefore, the following text

Pokud jsou tyto příkazy prováděny v rámci skupiny (viz [kapitolu 1.8.1](#)), končí jejich účinek také v okamžiku, kdy skončí skupina, v níž jsou prováděny. Proto se často, aby tyto příkazy ovlivnily pouze část textu, dělá to, že se vytvoří skupina, která obsahuje daný příkaz. Takto text

```
V {\it The \TeX Book}, {\sc Knuth}
vysvětluje vše, co potřebujete
vědět o \TeX{}u.
```

V *The T_EX Book*, KNUTH vysvětluje vše, co potřebujete vědět o T_EXu.

vytváří dvě skupiny. Jedna definuje rozsah příkazu `\it` (kurzíva) a druhá příkazu `\sc` (kapitálky).

Naopak potom existují příkazy, které kvůli svému účinku nebo z jiných důvodů vyžadují výslovné uvedení toho, na jaký text mají být aplikovány. V těchto případech je text, který má být příkazem ovlivněn, uveden ve složených závorkách *bezprostředně za příkazem*. Jako příklad můžeme uvést příkaz `\framed`: tento příkaz vykreslí rámeček kolem textu, který je uveden jako argument. Například:

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

vygeneruje

Tweedledum and Tweedledee

Všimněte si, že ačkoli se v první skupině příkazů (těch, které vyžadují argument) někdy k určení rozsahu působnosti používají složené závorky, nejsou nutné k tomu, aby příkaz fungoval. Příkaz je určen k použití od místa, kde se objeví. Při určování jeho rozsahu působnosti pomocí je příkaz umístěn *uvnitř těchto závorek*. V druhém případě závorky rámuji text, na který se má příkaz vztahovat, a jsou uvedeny až *za příkazem*.

U příkazu `\framed` je zřejmé, že efekt, který způsobí, potřebuje argument – text, na který má být aplikován. V principu ale záleželo na programátorovi, o který typ příkazu půjde. Například příkazy `\it` a `\color` dělají podobnou věc: nastavují nějakou vlastnost (formát nebo barvu. Ale z rozhodnutí vývojářů se první používá bez argumentu a druhý s argumentem.

1.3.2 Příkazy vyžadující výslovné označení místa

Existují určité příkazy, které určují svůj rozsah přesným uvedením bodu, ve kterém začíná jejich platnost, a bodu, ve kterém končí. Tyto příkazy se tedy vyskytují ve dvojicích: jeden označuje začátek platnosti a druhý konec. „start“, za kterým

následuje názvem příkazu, se používá k označení začátku akce, a „stop“, následovaný názvem příkazu, k označení konce. Takže například příkaz „`itemize`“ je představován `\startitemize` na začátku platnosti a `\stopitemize` označujícím její konec.

V oficiální dokumentaci ConTeXtu nemají tyto párové příkazy žádné speciální označení. Referenční příručka je označuje jednoduše jako „start ... stop“. Někdy jsou nazývány jako *prostředí*, což je ale název, který používá L^AT_EX pro podobnou konstrukci. Nevýhodou tohoto označení je také to, že v ConTeXtu je termín „prostředí“ používán pro něco jiného (speciální druh souboru, o kterém bude řeč u vícesouborových projektů v kapitole 2.6). Přesto, jelikož pojem prostředí je jasný a srozumitelný, a z kontextu bude vždy zřejmé, zda je řeč o *příkazech prostředí* ne *souborech prostředí*, budu toto označení používat.

Prostředí se tedy skládají z příkazu, který je otevírá nebo spouští, a příkazu, který je uzavírá nebo ukončuje. Pokud zdrojový soubor obsahuje příkaz pro otevření prostředí, který není později uzavřen, bude obvykle vygenerována chyba.¹ Tento druh chyb obtížnější nalézt, protože chyba může vzniknout daleko za místem, kde se příkaz k otevření vyskytuje. Někdy lze v „`.log`“ souboru nalézt řádek, na kterém začíná nesprávně uzavřené prostředí. Jindy se ale chyba neuzavření prostředí znamená, že ConTeXt špatně interpretuje určitou pasáž a ne v tomto chybném prostředí, což znamená, že „`.log`“ soubor v tomto případě nepomůže zjistit, kde je problém.

Prostředí mohou být vnořená, což znamená, že další prostředí může být otevřeno v rámci existujícího prostředí. Je však nutné dodržet hierarchii a vnořené prostředí musí být uzavřeno uvnitř prostředí, ve kterém bylo otevřeno. Jinými slovy pořadí uzavírání prostředí musí odpovídat pořadí jejich otevírání (v opačném pořadí). Domnívám se, že by to mělo být zřejmé z následujícího příkladu:

```
\startSomething
...
\startSomethingElse
...
  \startAnotherSomethingElse
  ...
  \stopAnotherSomethingElse
\stopSomethingElse
\stopSomething
```

V příkladu je vidět, že prostředí „`AnotherSomethingElse`“ bylo otevřeno uvnitř prostředí „`SomethingElse`“ a musí v něm být také uzavřeno. V opačném případě vznikne při zpracování chyba.

¹ I když ne vždy; záleží na prostředí a na zbytku dokumentu. ConTeXt se v tomto liší od L^AT_EXu, který je při kontrole mnohem striktnější in this regard, which is much stricter.

Obecně lze říci, že příkazy koncipované jako prostředí jsou takové, které jsou určeny k definici nějaké vlastnosti na text o velikosti minimálně odstavce. Například prostředí „`narrower`“, které mění okraje, má smysl pouze v případě, že se jedná o odstavec. Nebo prostředí „`framedtext`“, které orámuje jeden nebo více odstavců. Toto druhé prostředí nám může pomoci pochopit, proč některé příkazy jsou navrženy jako prostředí a jiné jako jednotlivé příkazy: pokud chceme orámovat jedno nebo více slov na stejném řádku, použijeme příkaz `\framed`, ale pokud chceme orámovat celý odstavec (nebo několik odstavců), použijeme příkaz `\framedtext`.

Dále je třeba pochopit, že text umístěný v nějakém prostředí, současně vytváří skupinu (viz [kapitolu 1.8.1](#)). A tedy pokud je uvnitř prostředí aktivován nějaký příkaz, aplikuje se pouze na zbývající část tohoto prostředí a jeho účinek končí s koncem prostředí. A ConT_EXt má, de facto, nepojmenované prostředí začínající příkazem `\start` (bez dalšího textu, prostě *start*. Proto ho nazývám *nepojmenované prostředí*) a končí příkazem `\stop`. Předpokládám, že jeho jediná funkce je vytvořit skupinu.



V dokumentaci ConT_EXt jsem se nikde nedočel, že by jedním z efektů prostředí bylo i skupinování jejich obsahu. Ale je to výsledek mých testů z řadou předdefinovaných prostředí, byť musím přiznat, že nebyly zdaleka vyčerpávající. Zkrátka jsem vyzkoušel nějaká náhodně vybraná prostředí. Moje testy nicméně ukázaly, že je tato skutečnost platná jen pro některá předdefinovaná prostředí. Prostředí vytvořená příkazem `\definestartstop` (popsaná v [kapitole 1.7.2](#)) nevytváří skupiny, pokud do vytvoření prostředí nezahrneme i vytvoření skupiny (viz [kapitolu 1.8.1](#)).

Také chování prostředí, které nazývám *nepojmenované* (`\start`) je pouze mým předpokladem. Skupinu vytváří, ale jestli má i jiné využití nevím. Je to jeden z příkazů, nezdokumentovaných v referenční příručce.

1.4 Parametry provádění příkazů

1.4.1 Příkazy

Řada příkazů může fungovat více než jedním způsobem. V takových případech existuje vždy výchozí chování, který lze změnit pomocí parametrů požadovné operace uvedených v hranatých závorkách za názvem příkazu.

Dobrým příkladem toho, co jsem právě uvedl, je `\framed`, zmíněném v předchozí kapitole. Tento příkaz vykreslí rámeček kolem textu, který bere jako argument. Ve výchozím nastavení má rámeček výšku a šířku dle textu, na který je aplikován. Můžeme však uvést i jinou výšku a šířku. Můžeme tak vidět rozdíl mezi tím, jak funguje ve výchozím nastavení `\framed`:


```
\framed{Tweedledum}
```



a při upraveném spuštění:

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



Ve druhém příkladu jsme v hranatých závorkách uvedli konkrétní šířku a výšku rámečku, který obklopuje text v předaném argumentu. Uvnitř závorek jsou uvedeny různé možnosti odděleny čárkou. Prázdné znaky a dokonce i zalomení řádků (pokud nejde o dvojité odřádkování) nejsou zohledňovány. Takže například další čtyři varianty příkazu vedou k naprosto stejnému výsledku:

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

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

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

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

Je mi jasné, že poslední varianta je nejlépe čitelná. Na první pohled vidíme kolik parametrů existuje a jak se používají. V příkladu, jako je tento, s pouze dvěma možnostmi, by se to možná nemuselo zdát tak důležité, ale v případech, kdy existuje dlouhý seznam parametrů, je pro uživatele pochopit, co má být výsledkem v ConTeXtu, pokud je každý zapsán na vlastní řádek. I odhalení případné chyby je v takovém případě snazší. Proto je tento formát (nebo podobný) pro zápisu příkazů uživateli "preferován".

Pokud jde o syntaxi konfiguračních voleb, podívejte se dále do [kapitoly 1.5](#).

1.4.2 Příkazy upravující chování jiných příkazů (`\setupSomething`)

Ukázali jsme si, že příkazy, které podporují parametry, mají vždy definovány jejich výchozí hodnoty, tj. své výchozí chování. Pokud jeden z těchto příkazů voláme několikrát a chtěli bychom změnit toto výchozí chování pro všechna volání místo toho, abychom jej měnili při každém volání, existuje snazší a eketivnější způsob,

jak to udělat. K tomuto účelu je skoro vždy k dispozici příkaz, jehož název začíná `\setup`, a následuje název přílazu, jehož výchozí nastavení chceme změnit.

I v tomto případě můžeme jako dobrý příklad použít příkaz `\framed`. Pokud tedy máme v dokumentu spoustu rámečků, u nichž potřebujeme použít přesný rozměr, můžeme nastavit chování příkazu `\framed` pomocí příkazu `\setupframed`. Takže

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

zajistí, že pokaždé, když použijeme příkaz `\framed` bez dalších parametrů, vygeneruje rámeček široký 3 cm a vysoký 1 cm, bez toho, abychom mu to muselo pokaždé nastavovat.

ConTeXt obsahuje okolo 300 příkazů umožňujících definovat chování jiných příkazů. Můžeme tak konfigurovat výchozí chování rámečků (`\framed`), výčtů („itemize“), názvů kapitol (`\chapter`), částí (`\section`), atd.

1.4.3 Vytváření vlastních variant konfigurovatelných příkazů (`\defineSomething`)

Když bychom pokračovali s příkladem příkazu `\framed`, tak je obvyklé, že v dokumentu používáme různé typy rámečků a každý má jiné rozměry. Bylo by tedy ideální, kdybychom si mohli *předdefinovat* různé varianty a nastavení `\framed` a pojmenovat je. A následně používat tyto pojmenované varianty. To lze v ConTeXtu provést pomocí příkazu `\defineframed` se syntaxí:

```
\defineframed[Jmeno][Nastaveni]
```

kde *Jmeno* je naše vlastní pojmenování konkrétní varianty nastavení daného typu rámečku a *Nastaveni* je konkrétní nastavení parametrů pro tuto variantu.

Výsledkem bude, že uvedená konfigurace bude spojena se jménem, které jsme vytvořili a bude fungovat tak, jako by to byl nový příkaz. Přičemž jej můžeme použít v jakémkoli místě, kde bychom mohli použít původní příkaz `\framed`.

Tato možnost samozřejmě existuje nejen konkrétně pro příkaz `\framed`, ale pro spoustu příkazů, které mají svou `\setup` variantu. Kombinace `\defineSomething` + `\setupSomething` je mechanismus který dává ConTeXtu jeho extrémní sílu a flexibilitu. Pokud bychom podrobně zkoumali, co přesně vykonání příkazu `\defineSomething` dělá, zjistilo bychom následující:

- Nejprve vytvoří kopii konkrétního příkazu se všemi jeho dostupnými parametry.
- Následně spojí tuto kopii s požadovaným jménem.
- A nakonec nastaví požadované výchozí chování, odlišné od původního příkazu.

V příkladu, který jsme si uvedli, jsme konfigurovali náš speciální rámeček současně s jeho definicí. Můžeme jej však také nejprve vytvořit a nakonfigurovat později. Protože, jak jsem již uvedl, jakmile je klon vytvořen, lze jej použít všude tam, kde by mohl být použit původní příkaz. Pokud jsme tedy například vytvořili rámeček s názvem „MySpecialFrame“, můžeme jej nakonfigurovat pomocí `\setupframed` s uvedením konkrétního jména rámečku. V tomto případě příkaz `\setup` přijme nový argument s názvem rámečku, který má být nakonfigurován:

```
\defineframed[MySpecialFrame]

\setupframed
  [MySpecialFrame]
  [ ... ]
```

1.5 Shrnutí syntaxe příkazů, jejich parametrů options a použití hranatých a složených závorek při jejich volání

Shrňme-li si to, co jsme dosud viděli, vidíme, že v ConT_EXtu

- Příkazy, jako takové, začínají vždy znakem „\“.
- Některé příkazy mohou mít jeden nebo více argumentů.
- Parametry, které sdělují příkazu *jak* má proběhnout nebo jinak ovlivňují to, jak se chová, se uvádějí v hranatých závorkách.
- Parametry určující na kterou část textu má být příkaz aplikován se uvádějí ve složených závorkách.

Pokud se příkaz provádí pouze na jednom znaku, jako například v případě příkazu `\buildtextcedilla` (jen pro vysvětlení – the ‚ç‘ se často užívá v katalánštině), složené závorky kolem parametru (textu) můžeme vynechat: příkaz se provede na prvním znaku, který není mezera.

- Některé parametry mohou být nepovinné, v takovém případě je můžeme vynechat. Nikdy však nemůžeme měnit pořadí argumentů, které příkaz očekává.

Parametry předávané v hranatých závorkách mohou být několika typů. Zejména:

- Mohou nabývat pouze jediné hodnoty, kterou bude téměř vždy jedno slovo nebo fráze
- Mohou nabývat více hodnot. V takovém případě může jít o:

- Jedno slovo - symbolické označení (jehož význam ConTeXt zná), míru nebo rozměr, číslo, název jiného příkazu apod.
- Názvy proměnných, které musí mít přiřazenu hodnotu. V takovém případě definice příkazu (viz [kapitolu 1.6](#)) vždy uvádí, jaký typ hodnoty každý parametr očekává.
 - ★ Pokud je očekávanou hodnotou text, může obsahovat i prázdné znaky nebo další příkazy. V těchto případech je někdy vhodné zapsat hodnotu ve složených závorkách.
 - ★ Pokud je očekávanou hodnotou parametru příkaz, můžeme obvykle jako hodnotu uvést více než jeden příkaz. Někdy je potřeba všechny takto předávané příkazy uzavřít do složených závorek. Do složených závorek je třeba pamatovat i v případě, že předávaný příkaz obsahuje parametry v hranatých závorkách.

Pokud je do parametru předáváno více hodnot, vždy se oddělují čárkami. Bílé znaky a konce řádků (ale ne dvojnásobné) jsou pro zpracování vynechávány. Stejně tak jsou vynechávány prázdné znaky a jednoduché konce řádků mezi jednotlivými parametry.

- Závěrem je dobré říci, že v ConTeXtu se nikdy nestane, že by příkaz měl parametry jak typu jednoho slova, tak s očekávanou hodnotou nebo proměnnou. Jinými slovy můžeme najít příkazy jako

```
\command[Option1, Option2, ...]
```

a jiné jako

```
\command[Variable1=value, Variable2=value, ...]
```

ale nikdy nenajdeme kombinaci

```
\command[Option1, Variable1=value, ...]
```

1.6 Oficiální seznam příkazů ConTeXtu

Mezi dokumentací ConTeXtu je zvláště důležitý dokument se seznamem všech příkazů, kde je u každého z nich uvedeno, kolik argumentů očekávají a jakého druhu, a také různé předpokládané volby a jejich povolené hodnoty. Tento dokument se nazývá „`setup-en.pdf`“ a je automaticky generován pro každou novou verzi ConTeXtu. Najdete ho v adresáři „`tex/texmf-context/doc/context/documents/general/qrcs`“.

Ve skutečnosti má „qrc“ sedm verzí tohoto dokumentu, jednu pro každý z jazyků, které mají rozhraní ConT_EXtu: německy, česky, francouzsky, holandsky, anglicky, italsky a rumunsky. Pro každý z těchto jazyků jsou v adresáři dva dokumenty: jeden s názvem „setup-LangCode“ (kde LangCode je písmenný kód pro identifikaci dvou mezinárodních jazyků) a druhý dokument s názvem „setup-mapping-LangCode“. Tento druhý dokument obsahuje seznam příkazů v abecedním pořadí a uvádí pouze prototyp příkazu bez dalších informací o hodnotách argumentů.

Tento dokument je zásadní pro učení se ConT_EXtu, protože v něm můžeme zjistit, zda určitý příkaz existuje, nebo ne. Speciálně je pak dobré mít na paměti kombinaciCOMMAND (or ENVIRONMENT) + setup COMMAND + defineCOMMAND. Například pokud víme, že se prázdný řádek vloží příkazem `\blank`, můžeme v dokumentu snadno zjistit, zda existuje i příkaz `\setupblank`, kterým můžeme modifikovat jeho výchozí chování a příkaz `\defineblank`, kterým můžeme vytvořit jeho pojmenované varianty.

„setup-en.pdf“ je tedy základem pro učení se ConT_EXtu. Ale já bych byl opravdu raději, kdyby nám především řekl, zda daný příkaz funguje pouze v Mark II nebo Mark IV a zejména pak, kdyby místo pouhého výčtu parametrů a jejich typů obsahoval i informace, k čemu tyto parametry slouží. Tím by se značně omezily nedostatky dokumentace ConT_EXtu. Existují i příkazy, které umožňují použití volitelných argumenty, které ale v tomto úvodu ani nezmiňuji, protože nevím, k čemu slouží. A jelikož jsou nepovinné, není nutné se o nich zmiňovat. A to je nesmírně frustrující.

1.7 Definování nových příkazů

1.7.1 Základní postup definování nových příkazů

We have just seen how, with `\defineSomething` we can clone a pre-existing command and develop a new version of it from there, that will function, to all intents and purposes, as a new command. Již jsme si ukázalo, jak můžeme pomocí `\defineSomething` klonovat existující příkazy a vytvořit z nich novou verzi, která bude fungovat, ve všech ohledech, jako nový příkaz.

Vedle této možnosti, která je dostupná pouze pro některé specifické příkazy (docela dost, ale ne všechny), má ConT_EXt obecný mechanismus pro definování nových příkazů, který je mimořádně výkonný. I když v některých případech použití také poměrně složitý. V textu, který je jako je tento určený začátečníkům, považuji za nejlepší představit jej tím, že začneme některými z jeho jednodušších použití. Nejjednodušší ze všech je spojení nějakého slova (názvu) s úryvkem textu. Takže pokaždé, když použijete vybrané slovo, bude nahrazeno textem, který je s ním spojen. To nám na jedné straně umožní ušetřit spoustu času při psaní, a na druhé straně to, jako další výhoda, snižuje možnost udělat při psaní chyby. Současně tím dosáhneme toho, že bude daný text zapsán vždy stejným způsobem.

Představme si například, že píšeme pojednání o aliteracích v latinských textech, kde často citujeme latinskou větu „*O Tite tute Tati tibi tanta tyranne tulisti*“ (O Tite Tati, ty tyrane, tolik jsi toho na sebe přivolal!). Je to poměrně dlouhá věta, v níž dvě slova jsou vlastními jmény a začínají velkými písmeny, a kde, přiznejme si, jakkoli můžeme mít latinskou poezii rádi, při jejím zápisu snadno "zakopneme". V tomto případě jsme mohli jednoduše vložit do úvodu našeho zdrojového souboru:

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

Na základě takové definice bude pokaždé, když se v našem zdrojovém souboru objeví příkaz `\Tite`, nahrazen uvedenou větou a bude také v uvozovkách, stejně jako v původní definici, což nám umožní zajistit, že způsob, jakým se tato věta objeví, bude vždy stejný. Mohli bychom ji také napsat kurzívou, větší velikostí písma... jak chceme. Důležité je, že ji musíme napsat jen jednou a v celém textu bude reprodukována přesně tak, jak byla napsána, tak často, jak budeme chtít. Mohli bychom také vytvořit dvě verze příkazu, nazvané `\Tite` a `\tite`, podle toho, zda je třeba větu napsat velkými písmeny, nebo ne. Náhradní text může být čistě text nebo může obsahovat příkazy nebo tvořit matematické výrazy v nichž se často chybuje (alespoň v mém případě). Například pokud se má v našem textu pravidelně objevovat výraz (x_1, \dots, x_n) , mohli bychom

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

takže vždy, když se v textu objeví `\xvec`, bude nahrazen uvedeným výrazem.

Základní syntaxe příkazu `\define` je následující:

```
\define[PocetParametru]\NazevPrikazu{TextProNahrazeni}
```

kde

- **PocetParametru** definuje, kolik parametrů příkaz má. Pokud nemá žádné, jako ve výše uvedených příkladech, můžeme tuto část vynechat.
- **NazevPrikazu** je název nového příkazu. Ten musí odpovídat obecnému pravidlu pro názvy příkazů. Tj. může jít buď o jeden znak, který není písmenem anglické abecedy, nebo o jedno nebo více písmen základní anglické abecedy (a-zA-Z).
- **TextProNahrazeni** obsahuje to, čím bude nahrazeno ve výstupu uměno příkazu, kdykoliv se ve zdrojovém textu objeví.

Možnost zadávat nové příkazy s argumenty v jejich definici dává tomuto mechanismu velkou flexibilitu, protože umožňuje nahradit text proměnnou, předanou v parametru.

Představme si, například, že chceme nadefinovat příkaz, který vytvoří úvod obchodního dopisu. Hodně jednoduchá verze může být::

```
\define\ZahlaviDopisu{
  \rightaligned{Eda Pádlo}\par
  \rightaligned{konzultant}\par
  Horní Věrolomná, \date\par
  Vážený pane,\par
}
```

but it would be preferable to have a version of the command that would write the name of the recipient in the header. This would require the use of a parameter that communicates the name of the recipient to the new command. This would require redefining the command as follows: ale bylo by vhodnější mít verzi příkazu, která by zapisovala i jméno příjemce v záhlaví. To by vyžaduje použití parametru, kterým novému příkazu předáme jméno příjemce. Příkaz můžeme předefinovat takto:

```
\define[1]\ZahlaviDopisu{
  \rightaligned{Eda Pádlo}\par
  \rightaligned{konzultant}\par
  Horní Věrolomná, \date\par
  Vážený pane #1,\par
}
```

Všimněte si, že jsme v definici provedli dvě změny. Především jsme mezi klíčové slovo `\define` a název příkazu vložili jedničku do hranatých závorek ([1]). Tím říkáme ConTeXtu, že příkaz, který definujeme, bude mít jeden argument. Dále jsme v posledním řádku definice příkazu napsali „Vážený pane #1,“, přičemž jsme použili vyhrazený znak „#“. To znamená, že v místě náhradního textu, kde se objeví „#1“, bude vložen obsah prvního argumentu. Pokud by měl příkaz dva parametry, „#1“ by odkazovalo na první parametr a „#2“ na druhý. Aby bylo možné příkaz zavolat (ve zdrojovém souboru), za názvem příkazu, musí být argumenty uvedeny ve složených závorkách (každý argument s vlastními). Příkaz, který jsme právě definovali, by tedy měl být v textu našeho zdrojového souboru volán následujícím způsobem:

```
\ZahlaviDopisu{Jméno příjemce}
```

Například tedy: `\ZahlaviDopisu{Nevěřící Tomáši}`.

Předchozí funkci bychom mohli ještě vylepšit, protože předpokládá, že dopis bude odeslán muži (vloží „Vážený pane“). Takže bychom mohli přidat další parametr, který by rozlišoval mezi oslovením muže a ženy. Například tedy:

```
\define[1]\ZahlaviDopisu{
  \rightaligned{Eda Pádlo}\par
  \rightaligned{konzultant}\par
  Horní Věrolomná, \date\par
  #1\ #2,\par
}
```

takže by potom volání vypadalo například takto

```
\ZahlaviDopisu{Vážená paní}{Liduška Muzikantová}
```


although this is not very elegant (from a programming point of view). It would be preferable for symbolic values to be defined for the first argument (man/woman; 0/1; m/f) so that the macro itself would choose the appropriate text according to this value. But explaining how to achieve this requires us to get into more depth than I think the novice reader can understand at this stage. ačkoli to není příliš elegantní (z programátorského hlediska). Vhodnější by to bylo, aby byly pro první argument definovány symbolické hodnoty (muz/zena; 0/1; m/f), takže makro by následně samo vybralo vhodný text podle této hodnoty. Vysvětlení, jak toho dosáhnout, však vyžaduje, abychom se dostali do větší hloubky než si myslím, že začínající čtenář může v této fázi pochopit.

1.7.2 Vytváření nových prostředí

K vytvoření nového prostředí nabízí ConT_EXt příkat `\definestartstop` s následující syntaxí:

`\definestartstop[Jmeno][Kongfigurace]`



V oficiální definici `\definestartstop` (viz [kapitulu 1.6](#)) je další parametr, který jsem výše nevedl, protože je nepovinný a nepodařilo se mi zjistit, k čemu slouží. Nevysvětluje ho ani úvodní ConT_EXt „Excursion“, ani neúplná referenční příručka. Předpokládal jsem, že tento argument (který se musí zadat mezi název a konfiguraci) by mohl být názvem nějakého existujícího prostředí, které by mohlo sloužit jako výchozí model pro nové prostředí. Ale mé testy ukázaly, že tento předpoklad byl mylný. Podíval jsem se do poštovní konference ConT_EXt a neviděl jsem žádné použití tohoto parametru.

kde

- **Jmeno** je jméno definovaného prostředí.
- **Konfigurace** nám dovoluje nastavit chování nového prostředí. K dispozici máme následující hodnoty:
 - **before** – Příkazy, které mají být provedeny před zahájením prostředí.
 - **after** – Příkazy, které mají být provedeny po ukončení prostředí.
 - **style** – Styl, který má být textu v prostředí nastaven.
 - **setups** – Sada příkazů vytvářená pomocí `\startsetups ... \stopsetups`. Tyto příkazy a jejich užití nebudu v tomto dokumentu vysvětlovat.
 - **color, inbetween, left, right** – Nedokumentované možnosti, které se mi nepodařilo zprovoznit. Co některé dělají se dá odhadovat z jejich názvu, například barva, ale z více testů, které jsem provedl, vyplývá, že při zadání různých hodnot tohoto parametru, nevidím v prostředí žádnou změnu.



Příklad definice prostředí:


```

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

\starttext

První dva základní zákony lidské hlouposti jednoznačně říkají že:

\startTextWithBar

\startitemize[n,broad]

\item Vždy a nevyhnutelně podceňujeme počet existujících hloupých jedinců.

\item Pravděpodobnost, že daná osoba je hloupá, je nezávislá na na jiných
vlastnostech téže osoby.

\stopitemize

\stopTextWithBar

\stoptext

```

Výsledek bude:

První dva základní zákony lidské hlouposti jednoznačně říkají že:

1. *Vždy a nevyhnutelně podceňujeme počet existujících hloupých jedinců.*
2. *Pravděpodobnost, že daná osoba je hloupá, je nezávislá na na jiných vlastnostech téže osoby.*

Pokud chceme, aby naše nové prostředí bylo současně skupinou ([kapitola 1.8.1](#)), tj. aby jakákoli změna normálního fungování ConTeXtu, ke které v něm dojde, skončila při opuštění prostředí, musíme do parametru „before“ uvést příkaz `\bgroup` a do parametru „after“ příkaz `\egroup`.

1.8 Další základní koncepty

Kromě příkazů existují i další pojmy, které jsou zásadní pro pochopení logiky fungování ConTeXtu. Některé z nich, díky své složitosti, nejsou vhodné pro úvod, a proto se jimi nebudeme zabývat. Dva z nich ale nyní prozkoumáme podrobněji: skupiny a rozměry.

1.8.1 Skupiny

Skupina je přesně ohraničený fragment zdrojového souboru, který ConTeXt používá jako *pracovní jednotku* (co to znamená, je vysvětleno níže). Každá skupina má explixitně uvedený začátek a konec. Skupina začíná

- Vyhrazeným znakem „{“ nebo příkazem `\bgroup`.
- Příkazem `\begingroup`
- Příkazem `\start`
- Zahájením některých prostředí (příkaz `\startSomething`).
- Zahájením prostředí matematiky (rezervovaným znakem «\$»).

a je ukončena

- Vyhrazeným znakem „}“ nebo příkazem `\egroup`.
- Příkazem `\endgroup`
- Příkazem `\stop`
- Ukončením prostředí (příkaz `\stopSomething`).
- Ukončením prostředí matematiky (rezervovaným znakem «\$»).

Skupiny generují automaticky také některé příkazy, například `\hbox`, `\vbox` (obecně příkazy spojené s vytvářením boxů)¹. Mimo tyto případy (skupiny automaticky generované určitými příkazy), musí být vždy způsob uzavření skupiny v souladu se způsobem jejího otevření. To znamená, že skupina která je zahájena znakem „{“, musí končit znakem „}“ a skupina zahájená příkazem `\begingroup` musí být končit příkazem `\endgroup`. Toto pravidlo má pouze jednu výjimku, a to že skupina začínající „{“ může končit pomocí `\egroup` a naopak (kombinace `\bgroup` a „}“). Což znamená, že dvojice „{“ a `\bgroup` a dvojice „}“ a `\egroup` jsou vzájemně zcela nahraditelné.

Příkazy `\bgroup` a `\egroup` byly navrženy tak, aby bylo možné definovat příkazy pro otevření skupiny a jiné pro její uzavření. Proto z vnitřních důvodů syntaxe T_EXu nebylo možné tyto skupiny otevírat a zavírat pomocí složených závorek, protože by to generovalo nepárové složené závorky a při kompilaci vždy končilo chybou.

Naproti tomu příkazy `\begingroup` a `\endgroup` nejsou zaměnitelné se složenými závorkami nebo příkazy `\bgroup` ... `\egroup`. Skupina započatá příkazem `\begingroup` musí být uzavřena pomocí `\endgroup`. Tyto poslední příkazy byly navrženy tak, aby umožňovaly důkladnější kontrolu chyb. Běžní uživatelé je obecně nemusí používat.

Můžeme mít i vnořené skupiny (skupina v jiné skupině). V takovém případě musí být pořadí uzavírání skupin odpovídat jejich otevírání. Tj. každá podskupina musí být uzavřena v rámci skupiny, ve které začala. Mohou existovat i prázdné skupiny

¹ Pojem *box* je také jeden z obecných pojmů ConTeXtu, ale jeho vysvětlení není součástí tohoto úvodu

generované pomocí „{ }“. Prázdná skupina nemá na výsledný dokument žádný vliv, ale může být užitečná například pro označení konce názvu příkazu.

Hlavním účelem skupin je zapouzdření jejich obsahu: definice, formáty a přiřazení hodnot, které jsou provedeny v rámci skupiny, jsou zpravidla „zapomenuty“, jakmile skupinu opustíme. Pokud tedy chceme, aby ConT_EXt dočasně změnil svůj běžný způsob fungování, je nejefektivnějším způsobem vytvořit skupinu a v jejím rámci toto fungování změnit. Jakmile apk skupinu opustíme, všechny hodnoty a formáty se obnoví do stavu před zahájením skupiny. Několik příkladů jsme již viděli: `\it`, `\bf`, `\sc`. To se však neděje pouze u formátovacích příkazů. Skupina určitým způsobem *izoluje* svůj obsah, takže jakákoli změna v kterékoli z mnoha vnitřních proměnných, které ConT_EXt neustále spravuje, zůstane platná jen tak dlouho, dokud se nacházíme ve skupině, v níž byla tato změna provedena. Stejně tak příkaz definovaný v rámci skupiny nebude znám mimo ni.

Takže pokud vyzkoušíme následující příklad

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

uvidíme, že při prvním spuštění příkazu `\A` odpovídá výsledek jeho původní definici (,B‘). Poté jsme vytvořili skupinu a předefinovali v ní příkaz `\A`. Spustíme-li jej nyní v rámci skupiny, příkaz poskytne výsledek dle nové definice (v uvedeném příkladu tedy ,C‘). Ale když opustíme skupinu, ve které byl příkaz ,B‘. Redefinice příkazu provedená ve skupině je skutečně ,zapomenuta‘, jakmile ji opustíme.

Další použití nachází skupin u příkazů nebo instrukcí, které jsou určeny výhradně pro znak, který je za nimi napsán. V tomto případě, pokud chceme, aby se příkaz vztahoval na více než jeden znak, musíme znaky, na které chceme, aby se příkaz nebo instrukce vztahovaly, uzavřít do skupiny. Tak například rezervovaný znak „[^]“, který, jak již víme, při použití uvnitř matematického prostředí převede následující znak na horní index. Pokud tedy napíšeme například „ 4^2x “, dostaneme „ 4^2x “. Pokud však napíšeme „ $4^{\{2x\}}$ “, „ 4^{2x} “.

A konečně: třetím použitím skupiny je případ, kdy ConT_EXtu sdělujeme, že to, co je ve skupině uzavřeno, musí být považováno za jeden celek. To je důvod, proč jem dříve ([kapitola 1.5](#)) uváděl, že v některých případech je lepší příkazy předávané jako parametr jiného příkazu uzavřít do složených závorek.

1.8.2 Rozměry

Ačkoli bychom mohli ConT_EXt používat bez obav o rozměry, nemohli bychom využít všech jeho možností, bez zvažování úráce s ním. Protože typografická dokonalost dosažená T_EXem a jeho odvozeninami spočívá, do značné míry, ve velké pozornosti, kterou systém vnitřně věnuje rozměrům. Své rozměry mají znaky, mezery mezi slovy, řádky nebo odstavci, samotné řádky, okraje, záhlaví i zápatí. Téměř pro každý prvek na stránce, který nás napasne, existuje nějaký rozměr.

V ConT_EXtu jsou rozměry specifikovány desetinným číslem následovaným jednotkou. Jednotky, které lze používat, jsou uvedeny v [tabulce 1.2](#).

Název	Označení v ConT _E Xtu	Ekvivalent
Palec	in	1 in = 2.54 cm
Centimetr	cm	2.54 cm = 1 inch
Milimetr	mm	100 mm = 1 cm
Bod	pt	72.27 pt = 1 inch
Big point	bp	72 bp = 1 inch
Scaled point	sp	65536 sp = 1 point
Pica	pc	1 pc = 12 points
Didot	dd	1157 dd = 1238 points
Cicero	cc	1 cc = 12 didots
	em	
	ex	

Tabulka 1.2 Rozměrové jednotky použitelné v ConT_EXtu

První tři jednotky v [tabulce 1.2](#) jsou standardní délkové míry. První z nich je používána v některých částech anglicky mluvícího světa, zbylé mimo něj. Zbývající jednotky pocházejí ze světa typografie. Poslední dvě, pro které jsem neuvedl žádný ekvivalent, jsou relativní měrné jednotky založené na základě aktuálního písma. 1 „em“ se rovná šířce písmene „M“ a „ex“ výšce písmene „x“. Použití měr vztahujících se k velikosti písma umožňuje vytvářet makra, která vypadají stejně dobře, ať už je v daném okamžiku použit jakýkoli zdroj. Proto se obecně doporučuje jejich používání.

Až na velice málo výjimek můžeme použít jakoukoliv z uvedených měrných jednotek a ConT_EXt si je vnitřně převede. Kdykoli však uvádíme rozměr, je to povinně uvést i měrnou jednotku. A to i v případě, kdy jde o rozměr velikosti „0“. Musíme uvést např. „0pt“ nebo „0cm“. Mezi číslem a názvem jednotky můžeme, ale nemusíme nechat prázdné místo. Pokud má jednotka desetinnou část, můžeme použít oddělovač desetinných míst, buď (.), nebo čárku (,).

Rozměry a míry se obvykle používají jako volitelné parametry příkazů. Můžeme však také přímo přiřadit hodnotu nějaké interní míře ConT_EXtu, pokud známe

její název. Například odsazení prvního řádku běžného odstavce je interně řízeno ConT_EXtem pomocí proměnné nazvané `\parindent`. Přiřazením hodnoty do této proměnné změníme rozměr, který ConT_EXt bude od tohoto okamžiku používat. A tak například, pokud chceme odstranit např. odsazení prvního řádku stačí, když do našeho zdrojového souboru zapíšeme text:

```
\parindent=0pt
```

Můžeme také psát `\parindent Opt` (bez rovnítka) nebo `\parindentOpt` bez meze mezi názvem rozměru a přiřazovanou hodnotou.

However, assigning a value directly to an internal measure is considered „inelegant“. In general, it is recommended to use the commands that control that variable, and to do so in the preamble of the source file. The opposite results in source files that are very difficult to debug because not all the configuration commands are in the same place, and it is really difficult to obtain a certain consistency in typographical characteristics.

Nicméně přiřazení hodnoty přímo internímu rozmětu je považováno za „neelegantní“. Obecně se doporučuje používat příkazy, které danou proměnnou ovládají, v úvodu zdrojového souboru. Nedodržení tohoto pravidla vede ke zdrojovém kódu, který se velmi obtížně ladí, protože ne všechny konfigurační příkazy jsou na stejném místě, a pak hodně obtížné dosáhnout konzistence v ve vztahu k typografickým vlastnostem.

Některé rozměry používané ConT_EXtem jsou „pružné“. To znamená, že v závislosti na situaci mohou nabývat různých hodnot. Tyto rozměry se nastavují s pomocí následující syntaxe:

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

Například

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

Tímto příkazem říkáme ConT_EXtu, že *normální* hodnota `\parskip` (definující vertikální vzdálenost mezi odstavci) je 3 body. Ale pokud to aktuální situace na stránce vyžaduje, může být až 5 bodů (3 plus 2) nebo jen 2 body (3 minus 1). Výsledné rozhodnutí tak necháváme na ConT_EXtu s tím, že víme, že výsledná vzdálenost bude mezi 2 a 5 body.

1.9 ConT_EXt pro samouky

Obrovské množství příkazů a možností ConT_EXtu se ukazuje jako skutečně ohromující a může v nás vyvolat pocit, že se s ním nikdy nenaučíme dobře pracovat.

Tento dojem je ale mylný, protože jednou z výhod ConTeXtu je jednotný způsob, jakým pracuje se všemi svými strukturami. Stačí se tak dobře naučit několik málo konstrukcí a víceméně víme, k čemu slouží ty zbývající. A když budeme potřebovat použít nějakou novou vlastnost, bude relativně snadné naučit se ji používat. Proto chápu tento úvod, jako jakési školení, které vás připraví na to, abyste mohli vyrazit na vlastní průzkum.

Abyste mohli vytvořit dokument v ConTeXtu, mělo by vám stačit znát těchto pět základních věcí (můžeme je nazývat ConTeXtí *Top Five*):

1. Vědět, jak vytvořit zdrojový soubor nebo projekt. Popsáno je to v [kapitole 2](#).
2. Umět nastavit hlavní font a znát základní příkazy pro změnu fontu a barvy - viz ([kapitolu 2](#)).
3. Znat základní příkazy pro definici struktury dokumentu, jako jsou části, kapitoly, podkapitoly pod. To vše vysvětluje [kapitola 3](#).
4. Možná také vědět něco o tom, jak pracovat s prostředím *itemize*, což vysvětluje do detailů [kapitola 3.3](#).
5. ...a něco málo dalšího.

For the rest, all we need to know is that it is possible. Certainly no one will use a utility if they do not know that it exists. Many of them are explained in this introduction; but, above all, we can repeatedly watch how ConTeXt always acts when faced with a certain type of construction:

O ostatním stačí vědět, že je to možné. Určitě nikdo nebude používat nástroje, o kterých neví, že existují. Mnoho z nich je vysvětleno v tomto úvodu, především však můžeme opakovaně vidět, jak se ConTeXt chová u většiny těchto nástrojů:

- Za prvé bude existovat příkaz, který něco umožní.
- Za druhé, skoro vždy zde bude i příkaz, který nám umožní nastavit a definovat výchozí chování v dané situaci. Tento příkaz začíná `\setup` a obvykle odpovídá názvu nastavovaného příkazu.
- A nakonec je obvykle i možnost nadefinovat si nový příkaz pro spouštění dané úlohy s různými nastaveními.

Chcete-li zjistit, zda tyto příkazy existují, vyhledejte si oficiální seznam příkazů (viz [kapitolu 1.6](#)), kde najdete také parametry, které příkaz má. A přestože se na první pohled mohou zdát názvy těchto parametrů poněkud záhadné, brzy zjistíte, že jsou parametry, které se opakují v mnoha dalších příkazech a ve všech fungují stejně nebo velmi podobně. Pokud máte pochybnosti o tom, co některý parametr dělá nebo jak funguje, stačí si vygenerovat dokument a vyzkoušet jej. Můžete se

také podívat do bohaté dokumentace ConT_EXtu. Jak je ve světě svobodného softwaru obvyklé, „ConT_EXt Standalone“ obsahuje, mimo jiné, zdrojové kódy téměř veškeré své dokumentace. Nástroj jako „**grep**“ (pro systémy GNU Linux) vám může pomoci vyhledat, zda příkaz nebo volba, u které máte pochybnosti, je použit v některém z těchto zdrojových souborů. Takže můžete mít k dispozici i příklad použití.

Takto jsem výuku ConT_EXtu koncipoval: v tomto „úvodu“ je do podrobnosti vysvětleno pět (ve skutečnosti čtyři) základních konceptů, na které kladu důraz, a řada dalších. Při čtení by se vám měl v hlavě poskládat jasný obraz posloupnosti: *příkaz, který něco provádí – příkaz, který jej konfiguruje – příkaz který nám umožní vytvořit podobný příkaz*. Naučíme se také některé ještě něco o hlavní struktuře ConT_EXtu a ukážeme si, k čemu slouží.

Chapter 2

Zdrojové soubory a projekty

Table of Contents: **2.1 Kódování zdrojových souborů;** **2.2 Znaky ve zdrojovém souboru (souborech);** 2.2.1 Prázdné mezery (prázdné místo) a tabulátory; 2.2.2 Konce řádků; 2.2.3 Pravidla/pomlčky; **2.3 Jednoduché a vícesouborové projekty;** **2.4 Struktura zdrojového souboru v jednoduchých projektech;** **2.5 Správa více souborů ve stylu \TeX ;** 2.5.1 Příkaz `input`; 2.5.2 `\ReadFile` a `\readfile`; **2.6 Con \TeX t projekty jako takové;** 2.6.1 *Environment* soubory; 2.6.2 Komponenty a produkty; 2.6.3 Projekty jako takové; 2.6.4 Společné aspekty prostředí, komponenty, produkty a projekty;

Jak již víme, při práci s Con \TeX tem vždy začínáme textovým souborem ve kterém je spolu s obsahem výsledného dokumentu již zahrnuta řada pokynů, které informují Con \TeX t o transformacích, které musí použít k vygenerování našeho finálně správně naformátovaného dokumentu v PDF ze zdrojového souboru.

Myslíme na čtenáře, kteří dosud uměli pracovat pouze se slovem procesory, myslím, že stojí za to strávit nějaký čas se zdrojovým souborem samotným. Nebo spíše se zdrojovými soubory, protože jsou chvíle, kdy existuje pouze jeden zdrojový soubor a další, když k průchodu použijeme více zdrojových souborů u závěrečného dokumentu. V tomto posledním případě můžeme mluvit o „vícenásobné projekty“.

2.1 Kódování zdrojových souborů

Zdrojové soubory musí být textové soubory. V počítačové terminologii je to pak název souboru, který obsahuje pouze text čitelný pro člověka, který neobsahuje binární kód. Tyto soubory se také nazývají *jednoduché textové* nebo *prosté textové* soubory.

Protože interně počítačové systémy zpracovávají pouze binární čísla, textový soubor se ve skutečnosti skládá z čísel, která představují *znaky*. A *tabulka* se používá ke spojení čísla se znakem. Pro textové soubory tam je několik možných tabulek. Termín *kódování textového souboru* odkazuje na konkrétní znak-odpovídající tabulce, kterou konkrétní textový soubor používá.

Existence různých kódovacích tabulek pro textové soubory je důsledek historie samotné informatiky. Na začátku vývojové fáze, kdy paměť a úložná kapacita počítače zařízení byla vzácná, bylo rozhodnuto použít tabulku nazvanou ASCII (což znamená „*Americký standardní kód pro výměnu informací*“) který umožňoval pouze 128 znaků a byl založen v roce 1963 v USA výbor pro standardy. Je zřejmé, že 128 znaků je málo pro představu všech znaků a symbolů používaných ve všech jazycích světa; ale na reprezentaci angličtiny to bylo víc než dost. Západní jazyk, ten, který má méně znaků, protože nepoužívá žádnou diakritiku (přízvuky a jiné značky nad nebo pod nebo skrz jiná písmena). Výhodou použití ASCII bylo, že textové soubory zabírají velmi málo místa, protože 127 (nejvyšší číslo v tabulce) může být reprezentováno 7místným binárním číslem a prvními použitými počítači používající byte jako jednotku pro měření paměti, 8místné binární číslo. Žádný znak v tabulce by se nevešel do jednoho bajtu. Protože bajt má 8 číslice a ASCII používaly pouze 7 číslic, zbylo dokonce místo na přidání některých dalších znaků reprezentujících jiné jazyky.

Ale když se používání počítačů rozšířilo, ASCII se stala nedostatečnou a bylo nutné vyvinout *alternativní* tabulky, které budou zahrnovat znaky neznámé anglické abecedě, jako je španělština „ñ“, samohlásky s diakritikou, katalánština nebo francouzština „ç“ atd. Na druhé straně neexistovala žádná počáteční shoda ohledně toho, co tyto *alternativní tabulky* ASCII by měly být, takže jiný specializované počítačové firmy se s problémem postupně vypořádaly samy. Proto nejen byly vytvořeny specifické tabulky pro různé jazyky nebo skupiny jazyků, ale i různé tabulky podle firmy, které je vytvořili (Microsoft, Apple, IBM atd.).

Bylo to pouze s nárůstem paměti počítače a nižšími náklady úložných zařízení a tomu odpovídajícímu zvýšení kapacity, že myšlenka vytvoření jedné tabulky, která by mohla být použita pro všechny jazyky. Ale znovu, ve skutečnosti to nebyla jediná tabulka obsahující všechny znaky, které byly vytvořeny, ale standardní kódování (nazývané Unicode) spolu s různými způsoby jeho znázornění (UTF-8, UTF-16, UTF-32 atd.) Ze všech těchto systémů se nakonec stal de facto standardem jen UTF-8, což umožňuje reprezentovat prakticky jakýkoli živý jazyk a mnoho již zaniklých jazyků, stejně jako další čtenné symboly, všechny používající čísla proměnné délky (mezi 1 a 4 byty), což zase pomáhá optimalizovat velikost textových souborů. Tato velikost se nezvětšila *příliš* ve srovnání se soubory pomocí čistého ASCII.

Než se objevil $\text{X}\text{\TeX}$ systémy založené na $\text{T}\text{\TeX}$ u – který se také zrodil v USA, a proto má angličtinu jako svůj rodný jazyk – předpokládá se, že kódování bylo v čistém ASCII; takže pro použití jiného kódování jste to měli nějak indikovat ve zdrojovém souboru.

Con $\text{T}\text{\TeX}$ Mark IV předpokládá, že kódování bude UTF-8. V méně aktuálních počítačových systémech však může být jiné kódování stále používané ve výchozím nastavení. Nejsem si příliš jistý výchozím kódováním které Windows používá, vzhledem k tomu, že strategie Microsoftu pro oslovování širší veřejnosti spočívá ve skrytí složitosti (ale i když je skrytá, je neznámeno, že by zmizela!). K dispozici není mnoho informací (nebo jsem je nemohl najít) ohledně kódovacího systému, který používá ve výchozím stavu.

V každém případě, bez ohledu na výchozí kódování, jakýkoli textový editor vám umožní uložit soubor v požadovaném kódování. Zdrojové soubory které mají být zpracované Con $\text{T}\text{\TeX}$ Mark IV musí být uloženy v UTF-8, pokud ovšem existuje velmi dobrý důvod pro použití jiného kódování (i když já si nedokážu představit, jaký by mohl být tento důvod).

Pokud chceme zapsat soubor napsaný v jiném kódování (možná starém soubor) můžeme

- a. Převeďte soubor na UTF-8, doporučené možnosti, a existující různé nástroje k tomu; v Linuxu například příkazy `iconv` nebo `recode`.
- b. Řekněte ConT_EXtu ve zdrojovém souboru, že kódování není UTF-8. Na to musíme použít příkaz `\enableregime`, jehož syntaxe je:

```
\enableregime[Encoding]
```

kde *Encoding* odkazuje na jméno, pod kterým ConT_EXt zná skutečné kódování daného souboru. V tabce ?? musíte najít různá kódování a názvy, pod kterými je ConT_EXt zná.

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

Tabulka 2.1 Main encodings in ConT_EXt

ConT_EXt MkIV důrazně doporučuje použití UTF-8. Souhlasím s tímto doporučením. Od tohoto okamžiku v tomto úvodu můžeme předpokládat, že kódování je vždy UTF-8.



Spolu s `\enableregime` ConT_EXt obsahuje příkaz `\useregime` což nám umožňuje používat kód jednoho nebo jiného kódování jako argument. Nenašel jsem žádné informace o tomto příkazu ani to, jak se liší od `\enableregime`, pouze některé příklady jeho použití. Mám podezření, že `\useregime` je navržen pro složité projekty, které používají mnoho zdrojových souborů, s očekáváním, že ne všechny budou mít stejné kódování. Ale je to jen odhad.

2.2 Znaky ve zdrojovém souboru (souborech)

Speciální znaky je označení, které dám skupině znaků, které se liší od *vyhrazených znaků*. Jak je vidět v kapitole ?? jsou takové, které mají speciální význam pro ConTeXt a nelze je tedy přímo použít jako znaky v zdrojových souborech. Spolu s nimi existuje další skupina znaků, ačkoliv s nimi ConTeXt zachází, když je najde ve zdroji, podle zvláštních pravidel. Tato skupina obsahuje prázdné mezery (bílá místa), tabulátory, zalomení řádků a pomlčky.

2.2.1 Prázdné mezery (prázdné místo) a tabulátory

S tabulátory a mezerami se ve zdrojovém souboru pro všechny zachází se stejnými záměry a účely. Znak tabulátoru (klávesa Tab na klávesnici) bude transformován na bílé místo pomocí ConTeXtu. A prázdná místa jsou pohlcena do jakéhokoli jiného prázdného místa (nebo karty) bezprostředně za nimi. Tedy to není absolutně žádný rozdíl ve zdrojovém souboru k zápisu

`Tweedledum a~Tweedledee.`

nebo

`Tweedledum a~Tweedledee.`

ConTeXt považuje tyto dvě věty za naprosto stejné. Proto, pokud chceme vložit další mezeru mezi slova, je potřeba použít nějaké ConTeXt příkazy, které to dělají. Normálně to bude fungovat s „\ “, což znamená \ znak následovaný mezerou. Existují ale i jiné postupy na které se podíváme v kapitole ?? týkající se horizontálních prostorů.

Absorpce po sobě jdoucích prázdných míst nám umožňuje napsat zdrojový soubor vizuálně zvýrazněním částím, které bychom chtěli zvýraznit, jednoduše tím zvýšíme nebo snížíme použité odsazení, s klidem na duši s vědomím, že to žádným způsobem neovlivní konečný dokument. Tady v následujícím příkladu

```
Hudební skupina z~Madridu na konci sedmdesátých let {\em La Romántica
  Banda Local} napsala písně eklektického stylu, které bylo velmi obtížné
kategorizovat. O~svém synovi „El Egipcio“ například řekli:
\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}}, míchání slov, frázi prostě,
mají vnitřní rytmus (comedia-histeria-seria, gitano-mano).
```

můžete vidět, jak jsou některé řádky mírně odsazeny doprava. Tyto řádky, které jsou součástí bitů, se zobrazí kurzívou. Mít tyto odsazení pomáhá (autorovi) vidět, kde končí kurzíva.

Někdo by si mohl myslet, jaký nepořádek! Musím se obtěžovat s odsazením řádků? Pravdou je, že toto speciální odsazení se u mého textu děje automaticky pomocí editoru (GNU Emacs), když upravuje zdrojový soubor `ConTEXtu`. Tohle je taková malá nápověda, díky které se rozhodnete pracovat s tímto určitým textovým editorem a ne s jinými.

Pravidlo, že prázdná místa jsou absorbována, platí výhradně pro po sobě jdoucí prázdná místa ve zdrojovém souboru. Pokud tedy prázdná skupina („{}“), je umístěna ve zdrojovém souboru mezi dvě prázdná místa, ačkoli prázdná skupina ve výsledném souboru nic nevytvoří, jeho přítomnost zajistí, že tyto dva polotovary nebudou po sobě jdoucí. Například, pokud napíšeme „`Tweedledum {}`“ a `Tweedledee`, dostaneme „`Tweedledum a Tweedledee`“, kde, když se podíváte dostatečně blízko, uvidíte dvě po sobě jdoucí mezery mezi prvními dvěma slovy.

Totéž se však stane s vyhrazeným znakem „~“, jeho efektem je generování prázdného místa, i když tady ve skutečnosti není: mezera následovaná znakem ~ toto nepohlítí a prázdné místo za ~ také nebude absorbováno.

2.2.2 Konce řádků

Ve většině textových editorů, když řádek překročí maximální šířku, dojde automaticky k zalomení řádku. Můžeme také výslovně vložit zalomení řádku stisknutím klávesy „Enter“ nebo „Return“.

ConT_EXt aplikuje na zalomení řádků následující pravidla:

- a. Zalomení jednoho řádku se ve všech směrech rovná prázdnému místu. Pokud tedy bezprostředně před nebo po zalomení řádku existuje jakékoli prázdné místo nebo tabulátor, budou pohlceny zalomením řádku nebo první prázdné místo a v konečném dokumentu bude jednoduché prázdné místo vloženo.
- b. Dva nebo více po sobě jdoucích zalomení řádku vytvoří zalomení odstavce. Pro tyto dva konce řádků jsou považovány za po sobě jdoucí, pokud tam není mezera nebo tabulátor mezi prvním a druhým zalomením řádku (protože ty jsou pohlceny zalomením prvního řádku); což ve zkratce znamená jeden nebo více po sobě jdoucích řádků, které jsou ve zdrojovém souboru zcela prázdné (bez jakéhokoli znaku nebo pouze s prázdnými mezerami nebo tabulátory) se stanou koncem odstavce.

Všimněte si, že jsem řekl „dva nebo více po sobě jdoucích zalomení řádku“ a pak „jeden nebo více prázdných po sobě jdoucích řádků“, což znamená, že pokud chceme zvětšit separaci mezi odstavci, neděláme tak jednoduše vložení dalšího

konce řádku. K tomu musíme použít příkaz který zvětšuje vertikální prostor. Pokud chceme jen jednu linii oddělení navíc, můžeme použít příkaz `\blank`. Ale jsou i další postupy pro zvětšení vertikálního prostoru. odkazují na [section 2.2](#).

V některých případech, když se konec řádku stane prázdným místem, můžeme skončit s nějakým nežádoucím a neočekávaným bílým místem. Zvlášť když jsme napsali makro, kde je snadné pro prázdné místo „se propašovat v“ aniž bychom si to uvědomovali. Abychom tomu zabránili, můžeme použít vyhrazený znak „%“, který, jak víme, způsobí že řádek, kde se objeví nezpracovat, znamená, že konce řádku také nebudou zpracovány. Tedy například příkaz

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

který píše svůj první argument kurzívou, druhý tučně a třetí malými písmeny, vloží mezi každý z těchto argumentů mezeru, zatímco

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

nevloží mezi ně žádné prázdné místo, protože vyhrazený znak % zabrání zpracování zalomení řádků a stane se pouze prázdným místem.

2.2.3 Pravidla/pomlčky

Pomlčky jsou dobrým příkladem rozdílu mezi počítačovou klávesnicí a tištěný text. Na normální klávesnici je obvykle pouze jeden znak pomlčky (nebo pravidlo, typograficky), kterou nazýváme pomlčka nebo („–“); ale tištěný text používá až čtyři různé délky pravidla:

- Krátká pravidla (pomlčky), jako jsou ta, která se používají k oddělení slabik dělení slov na konci řádku (–).
- Středně velká pravidla (en pomlčky nebo en pravidla), o něco delší než předchozí (–). Mají řadu použití, včetně některých Evropských jazyků (méně v angličtině) začátek řádku dialogu, nebo pro oddělení menších od větších číslic v rozsahu dat nebo stránek; „pp. 12–33“.
- Dlouhá pravidla (em pomlčky nebo em pravidla) (—), používané jako závorky zahrnout jednu větu do druhé.
- Znaménko mínus (–) představuje odečítání nebo záporné číslo.

Dnes jsou všechny výše uvedené a další kromě toho dostupné v kódování UTF-8. Ale protože je nelze všechny vygenerovat jedinou klávesou na klávesnici, není je tak snadné vytvořit ve zdrojovém souboru. Naštěstí \TeX vidí do našeho konečného

dokumentu a musí zahrnout více pravidel/pomlček, než by mohlo být vytvořené klávesnicí a jednoduchý postup k tomu navržený. ConT_EXt doplnil tento postup také přidáním příkazů, které vytváří tyto různé druhy pravidel. Můžeme použít dva přístupy generování čtyř druhů pravidel: buď běžným způsobem ConT_EXt příkazem nebo přímo z klávesnice. Tyto postupy jsou uvedeny v [table 2.2](#):

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

Tabulka 2.2 Rules/dashes in ConT_EXt

Názvy příkazů `\hyphen` a `\minus` jsou normálně používány v angličtině. Zatímco mnozí v polygrafickém průmyslu jim říkají ‚pravidla‘, Termíny T_EXu, jmenovitě `\endash` a `\emdash` jsou také běžné v sázecí terminologii. „en“ a „em“ jsou názvy měrných jednotek používaných v typografii. „cs“ představuje šířku ‚n‘, zatímco ‚em‘ je šířka ‚m‘ v používaném písmu.

2.3 Jednoduché a vícesouborové projekty

V ConT_EXtu můžeme použít pouze jeden zdrojový soubor, který obsahuje úplně všechnen obsah našeho konečného dokumentu, jakož i všechny související podrobnosti v tom případě mluvíme o „jednoduchých projektech“, nebo, bychom oproti tomu mohli použít řadu zdrojových souborů, které sdílejí obsah našeho závěrečného dokumentu, o kterém v tomto případě mluvíme jako o „vícenásobných projektech“.

Scénáře, kdy je typické pracovat s více než jedním zdrojovým souborem jsou následující:

- Pokud píšeme dokument, který má několik spolupracujících autorů, každý se svou částí odlišnou od ostatních; například, pokud píšeme *festschrift* s příspěvky od různých autorů, nebo číslo časopisu atd.
- Pokud píšeme dlouhý dokument, kde má každá část (kapitola) relativní autonomii, tak jejich konečné uspořádání umožňuje několik možností a bude rozhodnuto na konci. K tomu dochází s relativní četností pro mnoho akademických textů (příručky, úvody a podobné), kde se pořadí kapitol může lišit.
- Pokud píšeme řadu souvisejících dokumentů, které sdílejí nějaké stylové vlastnosti.

- Pokud, zjednodušeně řečeno, dokument, na kterém pracujeme, je velký, např počítač se zpomalí buď při jeho editaci nebo kompilaci; v tomto případě, rozdělení materiálu do několika zdrojových souborů značně urychlí kompilaci pro každou z nich.
- Také, pokud jsme napsali řadu maker, která chceme použít některé (nebo všechny) na naše dokumenty, nebo pokud jsme vygenerovali šablonu, která ovládá nebo upravuje naše dokumenty a my je na ně chceme aplikovat atd.

2.4 Struktura zdrojového souboru v jednoduchých projektech

V jednoduchých projektech vyvinutých v jediném zdrojovém souboru je struktura velmi jednoduchá a točí se kolem prostředí „textu“, který se musí v podstatě objevit ve stejném souboru. Rozlišujeme následující části tohoto souboru:

- **Preamble dokumentu:** vše od prvního řádku v souboru až na začátek prostředí „textu“ (`\starttext`).
- **Tělo dokumentu:** toto je prostředí dokumentu „text“; nebo jinými slovy všechno mezi `\starttext` a `\stoptext`.

```
% První řádek dokumentu

% oblasti preamble:
% Obsahuje globální konfiguraci
% příkazy pro dokument

\starttext % Zde začíná tělo dokumentu

...
... % Obsah dokumentu
...

\stoptext % Konec dokumentu
```

Obrázek 2.1 soubor obsahující jednoduchý projekt

V [figure 2.1](#) vidíme velmi jednoduchý zdrojový soubor. Absolutně vše před příkazem `\starttext` (který je v obrázku na řádku 5, počítají se pouze ty s nějakým textem), tvoří preamble; vše mezi `\starttext` a `\stoptext` tvoří tělo dokumentu. Cokoli po `\stoptextu` bude ignorováno.

Preamble se používá k zahrnutí příkazů, které mají ovlivnit dokument jako celek, tedy ty, které určují jeho celkovou konfiguraci. Tady není podstatné psát žádný

příkaz do preamble. Pokud žádný není, ConTeXt převezme výchozí konfiguraci, která není příliš vyvinutá, ale může to udělat pro mnoho dokumentů. V dobře naplánovaném dokumentu preamble bude obsahovat všechny příkazy ovlivňující dokument jako celek, např makra a přizpůsobené příkazy pro použití ve zdrojovém souboru. V typické preamble by to mohlo zahrnovat následující:

- Označení hlavního jazyka dokumentu (viz [section 1.5](#)).
- Označení velikosti papíru ([section 1.1](#)) a rozložení stránky ([section 1.3](#)).
- Vlastnosti hlavního písma dokumentů ([section 2.3](#)).
- Přizpůsobení příkazů sekce, které mají být použity ([section 3.4](#)) a v případě potřeby definice nové příkazy sekce ([section 3.5](#)).
- Rozložení záhlaví a zápatí ([section 1.6](#)).
- Nastavení pro naše vlastní makra ([section 1.7](#)).
- atd.

Preamble je určena pro celkovou konfiguraci dokumentu; tedy nic, co má společného s *obsahem* dokumentu, popř měl by tam být zpracovatelný text. Teoreticky zahrnuje jakýkoli zpracovatelný text v preambuli bude ignorován, i když někdy, pokud tam je, způsobí chybu při kompilaci.

Tělo dokumentu, zarámované mezi `\starttext` a příkazy `\stoptext` zahrnují skutečný obsah, což znamená zpracovatelný text spolu s příkazy ConTeXtu které by neměly ovlivnit celek dokumentu.

2.5 Správa více souborů ve stylu T_EXu

Aby bylo možné pracovat s více než jedním zdrojovým souborem, T_EX obsahoval prvotní soubor nazvaný `\input`, která také funguje v ConTeXtu, ačkoliv později obsahuje dva specifické příkazy, které do jisté míry zdokonalují způsob funkce `\input`.

2.5.1 Příkaz input

Příkaz `\input` vloží obsah souboru, který označuje. Jeho formát je:

`\input Název souboru`

kde *FileName* je název souboru, který se má vložit. Všimněte si, že není nutné, aby byl název souboru uzavřen ve složených závorkách, i když to nevyvolá chybu,

pokud se to udělá. Nemělo by však nikdy umístit do hranatých závorek. Pokud je přípona souboru „.tex“, lze jej vynechat.

Když ConTeXt kompiluje dokument a najde příkaz `\input`, vyhledá to uvedený soubor a pokračuje v kompilaci, jako by tento soubor byl část souboru, který jej nazval. Po dokončení kompilace se vrátí do původního souboru a pokračuje od místa, kde skončil; praktickým výsledkem je tedy, že obsah souboru volaný pomocí `\input` se vloží do bodu, kde se to volá. Soubor nazvaný s `\input` musí mít platný název v našem operačním systému a ne prázdná místa v názvu. ConTeXt to bude hledat v pracovním adresáři, a pokud jej tam nenajde, vyhledá jej v adresáři obsaženém v proměnné prostředí TEXROOT. Pokud soubor nebyl nakonec nalezen, způsobí chybu kompilace.

Nejběžnější použití příkazu `\input` je následující: soubor je nazvěme to „principal.tex“ a toto bude použito jako kontejner pro volání různých souborů pomocí příkazu `\input` které tvoří náš projekt. To je znázorněno na následujícím příkladu:

```
% Obecné konfigurační příkazy:

\input MyConfiguration

\starttext

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

...

\stoptext
```

Všimněte si, jak jsme pro obecnou konfiguraci dokumentu nazvali soubor „My-Configuration.tex“, o kterém předpokládáme, že obsahuje globální příkazy, které chceme použít. Poté mezi příkazy `\starttext` a `\stoptext` nazýváme několik souborů, které obsahují obsah souboru různé části našeho dokumentu. Pokud v danou chvíli pro urychlení kompilačního procesu, chceme vynechat kompilaci některých souborů, vše, co potřebujeme udělat, je umístit značku komentáře na začátek řádku volajícího nebo ty soubory. Například když píšeme třetí kapitolu a chceme zkompilovat jej jednoduše, abychom zkontrolovali, že v něm nejsou žádné chyby, nepotřebujeme zkompilovat zbytek, a proto můžeme psát:

```
% Obecné konfigurační příkazy:
```

```
\input MyConfiguration
```

```
\starttext
```

```
% \input PageTitle
```

```
% \input Preface
```

```
% \input Chap1
```

```
% \input Chap2
```

```
\input Chap3
```

```
...
```

```
\stoptext
```

a bude zkompilována pouze kapitola 3. Všimněte si, jak se naopak mění pořadí kapitol je stejně jednoduché jako změna volání pořadí řádků.

Když vyloučíme soubor ve vícesouborovém projektu z kompilace, získáme na rychlosti zpracování, ale v důsledku toho všechny reference, které část, která je zkompilována, převede na jiné části, které ještě nebyly zkompilovány již nebude fungovat. Viz section ??.

Je důležité, aby bylo jasné, že když pracujeme pouze s `\input` hlavního souboru, ten, který volá všechny ostatní, musí obsahovat `\starttext` a `\stoptext` příkazy, protože pokud ostatní soubory jej zahrnují, dojde k chybě. To na druhou stranu znamená, že nemůžeme přímo kompilovat různé soubory, které tvoří projekt, ale musí je nutně zkompilovat z hlavního souboru, což je ten, který obsahuje základní strukturu dokumentu.

2.5.2 `\ReadFile` a `\readfile`

Jak jsme právě viděli, pokud ConTeXt nenalezne soubor volaný s `\input`, vygeneruje chybu. Pro situaci, kdy chceme vložit soubor, pouze pokud existuje, ale s možností, že existovat nemusí, ConTeXt nabízí variantu příkazu `\input`. To je

```
\ReadFile{FileName}
```

Tento příkaz je ve všech ohledech podobný `\input` s jedinou výjimkou, že pokud soubor, který se má vložit, není nalezen, bude pokračovat kompilace bez generování jakékoli chyby. Také se liší od `\input` ve své syntaxi, protože víme, že s `\input` tomu není nutné vložit název souboru, který má být vložen mezi složené závorky. Ale s `\ReadFile` to je nutné. Pokud nepoužijeme složené závorky, ConTeXt si bude myslet, že název hledaného souboru je stejný jako první znak, který následuje za příkazem `\ReadFile`, následuje přípona `.tex`. Pokud tedy například píšeme

```
\ReadFile MyFile
```

ConT_EXt pochopí, že soubor ke čtení je volán „**M.tex**“, protože znak bezprostředně za příkazem `\ReadFile` (s výjimkou prázdných mezer, které jsou, jak víme, na konci názvu příkazem ignorovány) je ‚M‘. Protože ConT_EXt nemůže normálně najít soubor s názvem „**M.tex**“ a `\ReadFile` pokud soubor nenajde, vygeneruje chybu, ConT_EXt bude pokračovat kompilace za ‚M‘ v „**MyFile**“ a vloží text „**yFile**“.

Jemnější verze `\ReadFile` je `\readfile`, jejíž formát je stejný

```
\readfile{Název souboru}{TextIfExists}{TextIfNotExists}
```

První argument je podobný `\ReadFile`: název souboru uzavřeno mezi složené závorky. Druhý argument obsahuje text který má být zapsán, pokud soubor existuje, před vložením obsahu souboru. Třetí argument obsahuje text, který má být zapsán, pokud jde o nenalezený soubor. To znamená, že v závislosti na tom, zda byl či nebyl soubor zadán jak je nalezen první argument, druhý argument (pokud soubor existuje) nebo třetí (pokud soubor neexistuje) bude spuštěn.

2.6 ConT_EXt projekty jako takové

Třetí mechanismus, který ConT_EXt nabízí pro vícesouborové projekty, je více komplexní a kompletní: začíná rozlišováním mezi soubory projektu, soubory produktu, soubory součástí a soubory prostředí. Pro pochopení vztahy a fungování každého z těchto typů souborů, myslím, že ano nejlépe je vysvětlit každý zvlášť:

2.6.1 *Environment* soubory

Soubor prostředí je soubor, který ukládá makra a konfigurace specifického styl, který má být aplikován na několik dokumentů, zda se jedná o zcela nezávislé dokumenty nebo součásti komplexu dokumentů. Soubor prostředí tedy může obsahovat vše, co bychom chtěli normálně napsat před `\starttext`; to je: obecná konfigurace dokumentu.

Pro tyto druhy jsem zachoval termín „soubory prostředí“, abychom se neodchýlili od oficiální terminologie ConT_EXtu dokonce i když věřím, že lepší termín by byl pravděpodobně „format soubory“ nebo „globální konfigurační soubory“.

Stejně jako všechny zdrojové soubory ConT_EXtu jsou soubory prostředí textové soubory a předpokládejme, že rozšíření bude „**.tex**“, i když pokud bychom chtěli můžeme to změnit, třeba na „**.env**“. Obvykle to tak není však provedeno v ConT_EXtu. Nejčastěji je soubor prostředí identifikován tak, že název začíná nebo končí ‚env‘. Pro příklad: „**MyMacros.env.tex**“ nebo „**env_MyMacros.tex**“. Uvnitř takového souboru prostředí by vypadalo nějak takto:

```
\startenvironment MyEnvironment

\mainlanguage[en]

\setupbodyfont
[modern]

\setupwhitespace
[big]

...

\stopenvironment
```

Nebo jinými slovy, definice a konfigurační příkazy jsou součástí `\startenvironment` a `\stopenvironment`. Okamžitě následuje `\startenvironment` napíšeme jméno, kterým chceme identifikovat prostředí a poté zahrneme všechny příkazy, které bychom chtěli v našem prostředí, ze kterého se má skládat.

S ohledem na název prostředí podle mých testů název přidáný bezprostředně za `\startenvironment` je pouze orientační a kdybychom to nepojmenovali, pak se nic (špatného) nestane.

Soubory prostředí byly určeny pro práci s komponentami a produkty (vysvětleno v další části). To je důvod, proč může jedno nebo více prostředí být voláno z komponenty nebo produktu pomocí `\prostředí` příkazu. Tento příkaz ale také funguje, pokud je použit v konfiguraci oblast (preamble) libovolného zdrojového souboru ConTeXtu i když to není zdrojový soubor určený k sestavení po částech.

Příkaz `\environment` lze volat pomocí kteréhokoli z těchto dvou následující formátů:

```
\environment soubor
```

```
\environment[Soubor]
```

V obou případech bude účinek tohoto příkazu spočívat v načtení obsahu souboru brán jako argument. Pokud tento soubor není nalezen, bude pokračovat kompilace normálním způsobem bez generování jakékoli chyby. Pokud přípona souboru je „`.tex`“, lze ji vynechat.

2.6.2 Komponenty a produkty

Pokud si představíme knihu, kde je každá kapitola v jiném zdrojovém souboru, pak bychom řekli, že kapitoly jsou *komponenty* a kniha je *produkt*. To znamená, že *komponenta* je autonomní součást *produktu*, která může mít svůj vlastní styl

a může být sestavena nezávisle. Každá komponenta bude mít jiný soubor a navíc bude soubor produktu, který spojí všechny součásti dohromady.

Typický soubor součásti by byl následující

```
\environment MyEnvironment
\environment MyMacros

\startcomponent Chapter1

  \startchapter[title={Chapter 1}]

  ...

\stopcomponent
```

A soubor produktu by vypadal následovně:

```
\environment MyEnvironment
\environment MyMacros

\startproduct MyBook

  \component Chapter1
  \component Chapter2
  \component Chapter3

  ...

\stopproduct
```

Upozorňujeme, že skutečný obsah našeho dokumentu bude distribuován mezi různé ‚component‘ soubory a soubor produktu je omezen na stanovení pořadí komponenty. Na druhou stranu, (jednotlivé) komponenty a produkty lze sestavit přímo. Kompilace produktu vygeneruje soubor PDF obsahující všechny komponenty tohoto produktu. Pokud je naopak jedna z komponent sestavena jednotlivě vygeneruje soubor pdf obsahující pouze zkompilované komponenty.

V souboru komponentů a před příkazem `\startcomponent` se může volat jeden nebo více souborů prostředí pomocí `\environmentNázev prostředí`. Totéž můžeme udělat dříve v souboru produktu `\startproduct`. Několik souborů prostředí lze načíst současně. Můžeme mít například naši oblíbenou sbírku maker a různé styly, které aplikujeme na naše dokumenty, všechny v různých souborech. Prosím všimněte si však, že když používáme dvě nebo více prostředí, ta se načítají v pořadí, ve kterém jsou volány, takže pokud je stejná konfigurace, příkaz byl zahrnut ve více než jednom prostředí a také ano jiné hodnoty, budou platit hodnoty naposledy načteného prostředí. Na druhou stranu jsou soubory prostředí načteny pouze jednou, takže v předchozích příkladech, ve kterých je prostředí voláno ze souboru produktu a konkrétní dílčí soubory, pokud zkompilujeme produkt, to

je čas, kdy prostředí se načtou a v uvedeném pořadí; když prostředí je voláno z kterékoli komponenty, ConTeXt zkontroluje, zda toto prostředí je již načteno, v takovém případě neudělá nic.

Název komponenty, která je volána z produktu, musí být názvem souboru, který obsahuje danou komponentu, ačkoli pokud je přípona tohoto souboru „.tex“, lze ji vynechat.

2.6.3 Projekty jako takové

Rozlišení mezi produkty a komponenty je ve většině případů dostatečné. Stejně tak ConTeXt má ještě vyšší úroveň, kde můžeme seskupit řadu produktů: toto je *project*.

Typický soubor projektu by vypadal víceméně následovně

```
\startproject MyCollection

\environment MyEnvironment
\environment MyMacros

\product Book01
\product Book02
\product Book03

...

\stopproject
```

Scénář, kdy bychom potřebovali projekt, by byl například kde potřebujeme upravit sbírku knih, všechny ve stejném formátu specifikace; nebo kdybychom editovali časopis: sbírku knih, nebo časopis jako takový by byl projekt; každá kniha nebo každý deník, problém by byl produkt; a každá kapitola knihy nebo každý článek v deníku by byl komponentou.

Na druhou stranu projekty nejsou určeny k přímému sestavování. Vezměte v úvahu, že podle definice každý produkt patříící do projektu (každý kniha ve sborníku nebo každé číslo časopisu) může být kompilován samostatně a vygenerovat si vlastní PDF. Proto příkaz `\product` je zahrnut v něm, aby bylo uvedeno, jaké produkty skutečně patří do projektu, nedělá nic: je to pouze připomínka pro autora.

Je jasné, že by se někteří mohli ptát, proč máme projekty, když je nelze zkompileovat: odpověď je, že soubor projektu propojuje určitá prostředí s projektem. To je důvod, proč, pokud zahrneme projekt `\projektProjectName` v souboru komponenty nebo produktu, ConTeXt přečte soubor projektu a automaticky načte prostředí s ním spojená. Proto musí následovat příkaz `\environment` v projektech

`\startproject`; nicméně v produktech a komponentách `\prostředí` musí přijít dříve `\startproduct` nebo `\startcomponent`

Stejně jako u příkazů `\environment` a `\component`, i příkaz `\project` nám umožňuje zadat název projektu buď uvnitř hranaté závorky nebo hranaté závorky nepoužívat vůbec. Tohle znamená tamto `\project FileName` a `\Project[FileName]` jsou ekvivalentní příkazy.

Souhrn různých způsobů načítání prostředí

Z výše uvedeného vyplývá, že prostředí může být načteno kterýmkoli z následujících postupů:

- Před vložením příkazu `\environment EnvironmentName \starttext` nebo `\startcomponent`. Tím se zatíží prostředí pro pouze kompilace příslušného souboru.
- Vložením příkazu `\environment EnvironmentName` do souborů produktu před `\startproduct`. Tím dojde k zatížení prostředí, když produkt je zkompilován, ale ne, pokud jsou zkompilovány jeho součásti jednotlivě.
- Vložením příkazu `\project` do produktu nebo prostředí: tím se načtou všechna prostředí spojená s projektem (v projektu soubor).

2.6.4 Společné aspekty prostředí, komponenty, produkty a projekty

Názvy prostředí, komponent, produktů a projektů: Již jsme viděli, že u všech těchto prvků po `\start` příkazu, který spouští určité prostředí, komponentu, produkt nebo projekt, se jeho název zadává přímo. Toto jméno se zpravidla musí shodovat s názvem souboru obsahujícího prostředí, komponentu nebo produkt protože například když ConTeXt kompiluje produkt a do souboru produktu se musí načíst prostředí nebo komponenta, nemáme žádnou možnost vědět, který soubor je toto prostředí nebo komponenta, pokud soubor nemá stejný název jako prvek, který se má načíst.

Jinak podle mých testů název napsaný za `\startproduct` nebo `\startenvironment` v souborech produktu a prostředí je pouze orientační. Pokud je vynechán nebo se neshoduje s názvem souboru, nic zlého se neděje. V případě komponentů je však důležité, že název komponenty odpovídá názvu souboru, který to obsahuje.

Struktura adresářů projektu: Víme, že ConTeXt standardně hledá soubory v pracovním adresáři a v cestě označené proměnnou `TEXROOT`. Když však

použijeme příkazy `\project`, `\product`, `\component` nebo `\environment` předpokládá se, že projekt má adresářovou strukturu, ve které se společné prvky nacházejí v nadřazeném adresáři a ty specifické v některých podřízených adresářích. Pokud tedy soubor uvedený v pracovním adresáři není nalezen, bude vyhledán v nadřazeném adresáři a pokud není také tam, pak v adresáři rodičů tohoto adresáře a tak dále.

II

Dokument jako celek a jeho struktura

Chapter 1

Stránky a dokumentové stránkování

Table of Contents: **1.1 Velikost stránky;** 1.1.1 Nastavení velikosti stránky;
1.1.2 Používání nestandardních velikostí stránek; 1.1.3 Změna velikosti stránky v libovol-
ném bodě dokumentu; 1.1.4 Přizpůsobení velikosti stránky jejímu obsahu; **1.2 Prvky
na stránce;** **1.3 Rozvržení stránky (\setuplayout);** 1.3.1 Přiřazení velikosti růz-
ným komponentám stránky; 1.3.2 Přizpůsobení rozvržení stránky; 1.3.3 Použití více
rozložení stránky; 1.3.4 Další záležitosti související s rozložením stránky; A Rozlišení
mezi lichými a sudými stránkami; B Stránky s více než jedním sloupcem; **1.4 Čís-
lování stránek;** **1.5 Vynucené nebo navrhované konce stránek;** 1.5.1 Pří-
kaz \page; 1.5.2 Spojení určitých řádků nebo odstavců; **1.6 Záhloví a zápatí;**
1.6.1 Příkazy pro stanovení obsahu záhlaví a zápatí; 1.6.2 Formátování záhlaví a zá-
patí; 1.6.3 Definování konkrétních záhlaví a zápatí a jejich propojení s příkazy sekce;
1.7 Vložení textových prvků na hrany a okraje stránky;

ConTEXt transformuje zdrojový dokument na správně formátovaný *pages*. To, jak tyto stránky vypadají, jak je rozmístěn text a prázdná místa a jaké prvky obsahují,

to vše je zásadní pro dobrou sazbu. Tato kapitola je věnována všem těmto otázkám a některým dalším záležitostem souvisejícím se stránkováním.

1.1 Velikost stránky

1.1.1 Nastavení velikosti stránky

ConTeXt standardně předpokládá, že dokumenty budou velikosti A4, což je evropský standard. Můžeme nastavit jinou velikost pomocí `\setuppapersize`, což je typický příkaz nalezený v preambule dokumentu. Normální syntaxe tohoto příkazu je:

```
\setuppapersize [LogicalPage] [PhysicalPage]
```

kde oba argumenty mají symbolická jména.¹ První argument, který jsem nazval *LogicalPage*, představuje velikost stránky, kterou je třeba vzít v úvahu při sazbě; a druhý argument, *PhysicalPage*, představuje velikost stránky, na kterou se bude tisknout. Normálně jsou obě velikosti stejné a druhý argument lze pak vynechat; v některých případech se však mohou tyto dvě velikosti lišit, jako například při tisku knižních listů o 8 nebo 16 stranách (běžná tisková technika, zejména u akademických knih přibližně do 60. let 20. století). V těchto případech nám ConTeXt umožňuje rozlišit obě velikosti; a pokud je myšlenka, že se má na jeden list papíru vytisknout několik stránek, můžeme také určit schéma skládání, které se má dodržet, pomocí příkazu `\setuparranging` (který nebude v tomto úvodu vysvětlen).

Pro velikost sazby můžeme uvést kteroukoli z předdefinovaných velikostí používaných papírenským průmyslem (nebo námi). To zahrnuje:

- Jakákoli řada A, B a C definovaná normou ISO-216 (od A0 do A10, od B0 do B10 a od C0 do C10), obecně používaná v Evropě.
- Jakákoli ze standardních velikostí v USA: dopis, účetní kniha, bulvární tisk, právní, folio, výkonný.
- Jakákoli z velikostí RA a RSA definovaných standardem ISO-217.
- Velikosti G5 a E5 definované švýcarským standardem SIS-014711 (pro doctorské práce).
- Pro obálky: libovolná z velikostí definovaných severoamerickou normou (obálka 9 až obálka 14) nebo normou ISO-269 (C6/C5, DL, E4).

¹ Připomeňme, že v [section 1.5](#) jsem uvedl, že volby používané ConTeXtovými příkazy jsou v zásadě dvojího druhu: symbolická jména, jejichž význam je ConTeXtu již známý, neboli proměnná, které musíme explicitně přiřadit nějakou hodnotu.

- CD, pro obaly CD.
- S3 – S6, S8, SM, SW pro velikosti obrazovky v dokumentech, které nejsou určeny k tisku, ale k zobrazení na obrazovce.

Spolu s velikostí papíru můžeme pomocí `\setuppapersize` označit orientaci stránky: „portrait“ (vertikální) nebo „landscape“ (horizontální).

Další možnosti, které `\setuppapersize` umožňuje, podle wiki ConT_EXt, jsou „rotated“, „90“, „180“, „270“, „mirrored“ a „negative“. V mých vlastních testech jsem si všiml pouze některých znatelných změn u „rotated“, který invertuje stránku, i když to není přesně inverze. Číselné hodnoty mají vytvářet ekvivalentní stupeň rotace, samy o sobě nebo v kombinaci s „rotated“, ale nepodařilo se mi je přimět fungovat. Ani jsem přesně nezjistil, k čemu slouží „mirrored“ a „negative“.



Druhý argument `\setuppapersize`, který jsem již řekl, lze vynechat, pokud se velikost tisku neliší od velikosti sazby, může nabývat stejných hodnot jako první, což znamená velikost papíru a orientaci. Může také vzít „oversized“ jako hodnotu, která – podle ConT_EXt wiki – přidá centimetr a půl do každého rohu papíru.

Podle wiki existují další možné hodnoty pro druhý argument: „undersized“, „doublesized“ a „doubleoversized“. Ale v mých vlastních testech jsem nezaznamenal žádnou změnu po použití některého z nich; ani oficiální definice příkazu (viz section ??) tyto možnosti nezmiňuje.

1.1.2 Používání nestandardních velikostí stránek

Pokud chceme použít nestandardní velikost stránky, můžeme udělat dvě věci:

1. Použít alternativní syntaxi `\setuppapersize`, která nám umožňuje zavést výšku a šířku papíru jako rozměry.
2. Definovat naši vlastní velikost stránky, přiřadit jí název a použít ji, jako by to byla standardní velikost papíru.

Alternativní syntaxe `\setuppapersize`

Kromě syntaxe, kterou jsme již viděli, nám `\setuppapersize` umožňuje použít tuto jinou:

```
\setuppapersize[Name][Options]
```

kde *Name* je volitelný argument, který představuje jméno přiřazené velikosti papíru pomocí `\definepapersize` (na což se podíváme dále), a *Options* jsou toho druhu, kde přiřazujeme explicitní hodnotu. Mezi povolenými možnostmi můžeme zdůraznit následující:

- **width, height**, které reprezentují šířku a výšku stránky.
- **page, paper**. První odkazuje na velikost stránky, která má být vysázena, a druhá na velikost stránky, na kterou se má fyzicky tisknout. To znamená, že „page“ je ekvivalentní prvnímu argumentu `\setuppapersize` ve své normální syntaxi (vysvětleno výše) a „paper“ druhému argumentu. Tyto možnosti mohou nabývat stejných hodnot, které byly uvedeny dříve (A4, S3 atd.).
- **scale**, označuje faktor měřítka pro stránku.
- **topspace, backspace, offset**, další vzdálenosti.

Definování přizpůsobené velikosti stránky

K definování přizpůsobené velikosti stránky používáme příkaz `\definepapersize`, jehož syntaxe je

```
\definepapersize [Name] [Options]
```

kde *Name* odkazuje na název přidělený nové velikosti a *Options* může být:

- Jakákoli z povolených hodnot pro `\setuppapersize` v jeho normální syntaxi (A4, A3, B5, CD atd.).
- Měření šířky (papíru), výšky (papíru) a offsetu (posunu) nebo škálované hodnoty („scale“).

Co není možné, je smíchat hodnoty povolené pro `\setuppapersize` s měřeními nebo měřítkovými faktory. Je to proto, že v prvním případě jsou možnosti symbolická slova a ve druhém jsou proměnné s explicitní hodnotou; a v ConTeXtu, jak jsem již řekl, není možné kombinovat oba druhy voleb.

Když použijeme `\definepapersize` k označení velikosti papíru, která se shoduje s některými standardními rozměry, ve skutečnosti namísto definování nové velikosti papíru, co děláme, je definování nového názvu pro již existující velikost papíru. To může být užitečné, pokud chceme kombinovat velikost papíru s orientací. Můžeme tedy například psát

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

1.1.3 Změna velikosti stránky v libovolném bodě dokumentu

Ve většině případů mají dokumenty pouze jednu velikost stránky, a proto je `\setuppapersize` typickým příkazem, který zařazujeme do preambule a v každém dokumentu jej používáme pouze jednou. V některých případech však může

být nutné změnit velikost v určitém bodě dokumentu; například, pokud je od určitého bodu zahrnuta příloha, ve které jsou listy na šířku.

V takových případech můžeme použít `\setuppapersize` přesně tam, kde chceme, aby ke změně došlo. Ale protože by se velikost okamžitě změnila, abychom se vyhnuli neočekávaným výsledkům, normálně bychom před `\setuppapersize` vložili vynucený konec stránky.

Pokud potřebujeme změnit velikost stránky pouze pro jednotlivou stránku, místo abychom dvakrát použili `\setuppapersize`, jednou pro změnu na novou velikost a podruhé pro návrat na původní velikost, můžeme použít `\adaptpapersize`, které změní velikost stránky a o stránku později automaticky resetuje hodnotu před jejím voláním. A stejně jako jsme to udělali s `\setuppapersize`, před použitím `\adaptpapersize` bychom měli vložit vynucený konec stránky.

1.1.4 Přizpůsobení velikosti stránky jejímu obsahu

V ConTeXtu jsou tři prostředí, která generují stránky přesné velikosti pro uložení jejich obsahu. Jsou to `\startMPpage`, `\startpagefigure` a `\startTEXpage`. První generuje stránku, která obsahuje grafiku vygenerovanou MetaPostem, jazykem grafického designu, který se integruje s ConTeXtem, ale kterým se v tomto úvodu nebudu zabývat. Druhý dělá totéž s obrázkem a možná nějakým textem pod ním. Vyžaduje dva argumenty: první identifikuje soubor obsahující obrázek. Tomu se budu věnovat v kapitole věnované externím obrázkům. Třetí (`\startTEXpage`) obsahuje text, který je jejím obsahem na stránce. Jeho syntaxe je:

```
\startTEXpage[Options] ... \stopTEXpage
```

kde možnosti mohou být kterékoli z následujících:



- **strut**. Nejsem si jistý užitečností této možnosti. V terminologii ConTeXtu je *strut* znak postrádající šířku, ale s maximální výškou a hloubkou, ale nechápu, co to má společného s celkovou užitečností tohoto příkazu. Podle wiki tato možnost umožňuje hodnoty „yes“, „no“, „global“ a „local“, přičemž výchozí hodnota je „no“.
- **align**. Označuje zarovnání textu. Může to být „normal“, „flushleft“, „flushright“, „middle“, „high“, „low“ nebo „lohi“.
- **offset** k označení množství prázdného místa kolem textu. Může to být „none“, „overlay“, pokud je požadován efekt překrytí, nebo skutečný rozměr.
- **width, height** kde můžeme uvést šířku a výšku stránky nebo hodnotu „fit“, takže šířka a výška jsou ty, které požaduje text, který je součástí prostředí.
- **frame**, který je ve výchozím nastavení „off“, ale může mít hodnotu „on“, pokud chceme text na stránce ohraničit.

1.2 Prvky na stránce

ConTeXt rozpoznává různé prvky na stránkách, jejichž rozměry lze konfigurovat pomocí `\setuplayout`. Okamžitě se na to podíváme, ale předtím by bylo nejlepší popsat každý prvek stránky a uvést název, podle kterého `\setuplayout` každý z nich zná:

- **Edges:** bílé místo kolem textové oblasti. Ačkoli je většina textových procesorů nazývá „margins“, použití ConTeXtové terminologie je vhodnější, protože nám umožňuje rozlišovat mezi hranami jako takovými, kde není žádný text (ačkoli v elektronických dokumentech mohou být navigační tlačítka a podobně), a okraji, kde mohou být jisté textové prvky někdy umístěny, jako například poznámky na okraji.
 - Výška horní hrany je řízena dvěma měřeními: samotní hornou hranou („`top`“) a vzdáleností mezi horní hranou a záhlavím („`topdistance`“). Součet těchto dvou měření se nazývá „`topspace`“.
 - Velikost levé a pravé hrany závisí na měření „`leftedge`“ „`rightedge`“. Pokud chceme, aby byly obě stejně dlouhé, můžeme je nakonfigurovat současně pomocí volby „`edge`“.

U dokumentů určených pro oboustranný tisk nehovoříme o levé a pravé hraně, ale o vnitřní a vnější; první je hrana nejbližší místu, kde budou listy sešity, tj. levá hrana na lichých stránkách a pravá hrana na sudých stránkách. Vnější hrana je opakem vnitřní hrany.

- Výška spodní hrany se nazývá „`bottom`“.
- **Margins** správně nazvané. ConTeXt volá pouze postranní okraje (levý a pravý). Okraje jsou umístěny mezi hranou a hlavní textovou oblastí a jsou určeny k umístění určitých textových prvků, jako jsou například poznámky na okraji, názvy oddílů nebo jejich čísla.

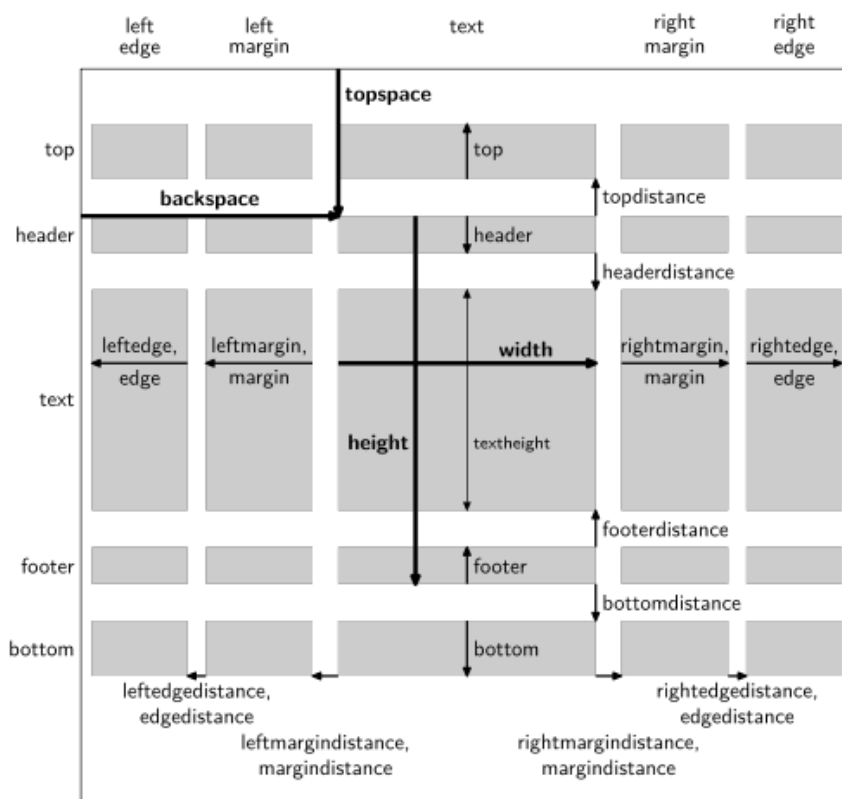
Rozměry, které řídí velikost okraje, jsou:

- **margin:** používá se, když chceme současně nastavit okraje na stejnou velikost.
- **leftmargin, rightmargin:** nastaví velikost levého a pravého okraje.
- **edgedistance:** Vzdálenost mezi hranou a okrajem.
- **leftedgedistance, rightedgedistance:** Vzdálenost mezi hranou a levým a pravým okrajem.
- **margindistance:** Vzdálenost mezi okrajem a oblastí hlavního textu.

- **leftmargindistance, rightmargindistance:** Vzdálenost mezi oblastí hlavního textu a pravým a levým okrajem.
- **backspace:** tato míra představuje prostor mezi levým rohem papíru a začátkem oblasti hlavního textu. Proto se skládá ze součtu „leftedge“ + „leftedgedistance“ + „leftmargin“ + „leftmargindistance“.
- **Header and footer:** Záhloví a zápatí stránky jsou dvě oblasti, které jsou umístěny v horní nebo dolní části psané oblasti stránky. Obvykle obsahují informace, které pomáhají text uvádět do kontextu, např. číslo stránky, jméno autora, název díla, název kapitoly nebo oddílu atd. VConTeXtu jsou tyto oblasti na stránce ovlivněny následujícími rozměry:
 - **header:** Výška záhlaví.
 - **footer:** Výška zápatí
 - **headerdistance:** Vzdálenost mezi záhlavím a hlavní textovou oblastí stránky.
 - **footerdistance:** Vzdálenost mezi zápatím a hlavní textovou oblastí stránky.
 - **topdistance, bottomdistance:** Oba výše uvedené. Jsou to vzdálenosti mezi horní hranou a záhlavím nebo dolní hranou a zápatím.
- **Main text area:** toto je nejširší oblast na stránce, která obsahuje text dokumentu. Její velikost závisí na proměnných „width“ a „textheight“. Proměnná „height“ zase měří součet „header“, „headerdistance“, „textheight“, „footerdistance“ a „footer“.

Všechny tyto oblasti můžeme vidět v [image 1.1](#) spolu s názvy přiřazenými odpovídajícím měřením a šipkami označujícími jejich rozsah. Tloušťka šipek spolu s velikostí názvů šipek má odrážet důležitost každé z těchto vzdáleností pro rozvržení stránky. Když se podíváme pozorně, uvidíme, že tento obrázek ukazuje, že stránka může být reprezentována jako tabulka s 9 řádky a 9 sloupci, nebo, pokud nebereme v úvahu separační hodnoty mezi různými oblastmi, bude tam pět řádků a pět sloupců přičemž text může být pouze ve třech řádcích a třech sloupcích. Průsečík prostředního řádku s prostředním sloupcem tvoří hlavní textovou oblast a normálně zabírá většinu stránky.

Ve fázi rozvržení našeho dokumentu můžeme vidět všechna měření stránky pomocí `\showsetups`. Chcete-li vidět hlavní obrysy distribuce textu indikované vizuálně na stránce, můžeme použít `\showframe`; a pomocí `\showlayout` můžeme získat kombinaci předchozích dvou příkazů.



Obrázek 1.1 Plochy a rozměry na stránce

1.3 Rozvržení stránky (`\setuplayout`)

1.3.1 Přiřazení velikosti různým komponentám stránky

Návrh stránky zahrnuje přiřazení konkrétních velikostí příslušným oblastem stránky. To se provádí pomocí `\setuplayout`. Tento příkaz nám umožňuje změnit kteroukoli z kót uvedených v předchozí části. Jeho syntaxe je následující:

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

kde *Name* je volitelný argument používaný pouze pro případ, kdy jsme navrhli více rozvržení (viz [section 1.3.3](#)), a možnosti jsou, kromě jiných, které uvidíme později, kterékoli z výše zmíněných měření. Mějte však na paměti, že tato měření jsou vzájemně propojená, protože součet komponent ovlivňujících šířku a těch, které ovlivňují výšku, se musí shodovat se šířkou a výškou stránky. V zásadě to bude znamenat, že při změně horizontální délky musíme upravit zbývající horizontální délky; a totéž při nastavování vertikální délky.

Ve výchozím nastavení ConTeXt provádí automatické úpravy rozměrů pouze v některých případech, které na druhou stranu nejsou v jeho dokumentaci uvedeny

žádným úplným nebo systematickým způsobem. Provedením několika testů jsem byl například schopen ověřit, že ruční zvýšení nebo snížení výšky záhlaví nebo zápatí způsobí úpravu v „`textheight`“; ruční změna některých okrajů však automaticky neupraví (podle mých testů) šířku textu („`width`“). To je důvod, proč nejúčinnějším způsobem, jak negenerovat žádnou nekonzistenci mezi velikostí stránky (nastavenou pomocí `\setuppapersize`) a velikostí jejích příslušných komponent, je:

- Pokud jde o horizontální měření:
 - Úpravou „`backspace`“ (zahrnuje „`leftedge`“ a „`leftmargin`“).
 - Úpravou „`width`“ (šířka textu) nikoli pomocí rozměrů, ale pomocí hodnot „`fit`“ nebo „`middle`“:
 - ★ `fit` vypočítá šířku textu na základě šířky zbytku vodorovných složek stránky.
 - ★ `middle` dělá totéž, ale nejprve srovná pravý a levý okraj.
- Pokud jde o vertikální měření:
 - Úpravou „`topspace`“.
 - Úpravou hodnot „`fit`“ nebo „`middle`“ na „`height`“. Ty fungují stejně jako v případě šířky. První vypočítá výšku na základě zbytku komponent a druhý nejprve srovná horní a dolní okraje a poté vypočítá výšku textu.
 - Jakmile je „výška“ upravena, úpravou výšky záhlaví nebo zápatí, je-li to nutné, s vědomím, že v takových případech bude „`textheight`“ automaticky přenastaven.
- Další možností pro nepřímé určení výšky hlavní textové oblasti je uvedení počtu řádků, které se do ní mají vejít (s ohledem na aktuální meziřádkový prostor a velikost písma). To je důvod, proč `\setuplayout` obsahuje volbu „`lines`“.

Umístění logické stránky na fyzickou stránku

V případě, že velikost logické stránky není stejná jako velikost fyzické stránky (viz [section 1.1.1](#)), `\setuplayout` nám umožňuje nakonfigurovat některé další možnosti ovlivňující umístění logické stránky na fyzické stránce:

- **location:** Tato volba určuje místo, kam bude stránka umístěna na fyzické stránce. Jeho možné hodnoty jsou `left`, `middle`, `right`, `top`, `bottom`, `singlesided`, `doublesided` nebo `duplex`.
- **scale:** Označuje faktor měřítka pro stránku před jejím umístěním na fyzickou stránku.

- **marking:** Vytiskne na stránku vizuální značky označující, kde má být papír oříznut.
- **horoffset, veroffset, clipoffset, cropoffset, trimoffset, blee-doffset, artoffset:** Série měření indikujících různá posunutí na fyzické stránce. Většina z nich je vysvětlena v referenční příručce z roku 2013.

Tyto volby `\setuplayout` musí být kombinovány s indikacemi z `\setuparranging`, které indikují, jak logické stránky mají být seřazeny na fyzickém listu papíru. Tyto příkazy v tomto úvodu vysvětlovat nebudu, protože jsem na nich neprováděl žádné testy.

Zjištění šířky a výšky textové oblasti

Příkazy `\textwidth` a `\textheight` vrací šířku a výšku textové oblasti. Hodnoty, které tyto příkazy nabízejí, nelze přímo zobrazit v konečném dokumentu, ale lze je použít pro jiné příkazy k nastavení jejich šířky nebo výšky. Takže například, abychom uvedli, že chceme obrázek, jehož šířka bude 60 % šířky řádku, musíme jako hodnotu obrázku uvést „width“: „width=0.6\textwidth“.

1.3.2 Přizpůsobení rozvržení stránky

Může se stát, že naše rozvržení stránky na konkrétní stránce produkuje nežádoucí výsledek; jako například poslední stránka kapitoly s pouze jedním nebo dvěma řádky, což není ani typograficky, ani esteticky žádoucí. K vyřešení těchto případů poskytuje ConTeXt příkaz `\adaptlayout`, který nám umožňuje změnit velikost textové oblasti na jedné nebo více stránkách. Tento příkaz je určen k použití pouze v případě, že jsme již dokončili psaní našeho dokumentu a provádíme jen malé poslední úpravy. Proto je jeho přirozené umístění v preambuli dokumentu. Syntaxe příkazu je:

`\adaptlayout [Pages] [Options]`

kde *Pages* odkazuje na číslo stránky nebo stránek, jejichž rozložení chceme změnit. Je to volitelný argument, který se má použít pouze tehdy, když je v preambuli umístěn `\adaptlayout`. Můžeme označit pouze jednu stránku nebo několik stránek, přičemž čísla oddělíme čárkami. Pokud vynecháte tento první argument, `\adaptlayout` ovlivní výhradně stránku, na které příkaz najde.

Pokud jde o možnosti, mohou to být:

- **height:** Umožňuje nám určit jako rozměry výšku, kterou by daná stránka měla mít. Můžeme uvést absolutní výšku (např. „19cm“) nebo relativní výšku (např. „+1cm“, „-0.7cm“).

- **lines:** Můžeme zahrnout počet řádků, které se mají přidat nebo odečíst. Chcete-li přidat řádky, před hodnotou je + a pro odečítání řádků je znak – (nejen pomlčka).

Veźměte v úvahu, že když změníme počet řádků na stránce, může to ovlivnit stránkování zbytku dokumentu. Proto se doporučuje použít `\adaptlayout` až na konci, kdy dokument nebude mít další změny, a to v preambuli. Poté přejdeme na první stránku, kterou si přejeme upravit, provedeme a zkontrolujeme jak to ovlivní stránky, které následují; pokud to ovlivní tak, že je třeba upravit další stránku, přidáme její číslo a zkompilujeme znovu a tak dále.

1.3.3 Použití více rozložení stránky

Pokud potřebujeme použít různá rozvržení v různých částech dokumentu, nejlepší způsob je začít definováním rozvržení *obecného* a poté různých alternativních, které mění pouze rozměry, které se musí lišit. Tato alternativní uspořádání zdědí všechny vlastnosti obecného uspořádání, které se nezmění jako součást jeho definice. Pro specifikaci alternativního rozvržení a pojmenování, kterým jej můžeme později zavolat, použijeme příkaz `\definelayouth`, jehož obecná syntaxe je:

```
\definelayouth [Name/Number] [Configuration]
```

kde *Name/Number* je název spojený s novým designem nebo číslo stránky, kde bude nové rozvržení automaticky aktivováno, a *Configuration* bude obsahovat aspekty rozvržení, které si přejeme změnit ve srovnání s celkovým rozložením.

Když je nové rozvržení spojeno s názvem, pro jeho volání na konkrétním místě v dokumentu používáme:

```
\setuplayouth [LayoutName]
```

a pro návrat k obecnému rozložení:

```
\setuplayouth [reset]
```

Na druhou stranu, pokud bylo nové rozvržení spojeno s konkrétním číslem stránky, bude automaticky aktivováno, když je stránka dosažena. Jakmile je však aktivován, pro návrat k obecnému návrhu to budeme muset výslovně uvést, i když to můžeme *semi-automatizovat*. Pokud například chceme použít rozvržení výhradně na stránky 1 a 2, můžeme do preambule dokumentu napsat:

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

Účinek těchto příkazů bude takový, že se na straně 1 aktivuje rozvržení definované v prvním řádku a na straně 3 se aktivuje další rozvržení, jehož funkcí je pouze návrat k obecnému rozvržení.

Pomocí `\definelayouth[even]` vytvoříme rozvržení, které se aktivuje na všech sudých stránkách; a s `\definelayouth[odd]` bude rozložení použito na všechny liché stránky.

1.3.4 Další záležitosti související s rozložením stránky

A. Rozlišení mezi lichými a sudými stránkami

U oboustranně tištěných dokumentů se často stává, že záhlaví, číslování stránek a boční okraje se u lichých a sudých stránek liší. Sudé stránky se také nazývají stránky levé ruky (verso) a liché stránky, pravé stránky (recto). V těchto případech je také obvyklé, že se mění terminologie týkající se okrajů a hovoříme o okrajích vnitřních a vnějších. První je umístěn v nejbližším bodě k místu, kde budou stránky sešívány, a druhý na opačné straně. Na lichých stránkách, vnitřní okraj odpovídá levému okraji a na sudých stránkách odpovídá vnitřní okraj pravému okraji.

`\setuplayout` nemá žádnou možnost, která by nám výslovně umožňovala sdělit, že chceme rozlišovat mezi rozložením pro liché a sudé stránky. Je to proto, že pro ConTeXt je rozdíl mezi oběma druhy stránek nastaven jinou volbou: `\setuppagenumbering`, kterou uvidíme v [section 1.4](#). Jakmile je toto nastaveno, ConTeXt předpokládá, že stránka popsaná pomocí `\setuplayout` byla lichá stránka, a sestaví sudou stránku použitím převrácených hodnot pro lichou stránku: specifikace platné pro lichou stránku vlevo, platí na sudé stránce vpravo; a naopak: to, co platí pro lichou stránku vpravo, platí pro sudou stránku vlevo.

B. Stránky s více než jedním sloupcem

Pomocí `\setuplayout` můžeme také vidět, že text našeho dokumentu je distribuován ve dvou nebo více sloupcích, jako to dělají například noviny a některé časopisy. To je řízeno volbou „columns“, jejíž hodnota musí být celé číslo. Pokud je zde více než jeden sloupec, je vzdálenost mezi sloupci indikována volbou „columnndistance“.

Tato možnost je určena pro dokumenty, ve kterých je veškerý text (nebo jeho většina) rozložen do více sloupců. Pokud v dokumentu, který je převážně jedno-sloupcový, chceme, aby konkrétní část měla dva nebo tři sloupce, nemusíme měnit rozvržení stránky, ale jednoduše použijeme prostředí „columns“ (viz [section 3.2](#)).

1.4 Číslování stránek

ConTeXt standardně používá pro číslování stránek arabská čísla a číslo se zobrazuje uprostřed záhlaví. Ke změně těchto vlastností má ConTeXt různé postupy, které podle mého názoru v této věci zbytečně komplikují.

Za prvé, základní charakteristiky číslování jsou řízeny dvěma různými příkazy: `\setuppagenumbering` a `\setupuserpagenumber`.

`\setuppagenumbering` umožňuje následující možnosti:

- **alternative:** Tato možnost řídí, zda je dokument navržen tak, aby záhlaví a zápatí byly na všech stránkách totožné („`singlesided`“), nebo zda rozlišují liché a sudé stránky („`doublesided`“). Když tato možnost nabývá druhé hodnoty, automaticky se ovlivní hodnoty rozvržení stránky zavedené pomocí „`setuplayout`“, takže se předpokládá, že to, co je uvedeno v „`setuplayout`“, se vztahuje pouze na liché stránky, a proto to, co je zadáno pro levý okraj, se ve skutečnosti vztahuje k vnitřnímu okraji (který je na sudých stránkách vpravo) a to, co je zadáno pro pravou stranu, se ve skutečnosti vztahuje k vnějšímu okraji, který je na sudých stránkách vlevo.
- **state:** Označuje, zda bude či nebude zobrazeno číslo stránky. Umožňuje dvě hodnoty: `start` (bude zobrazeno číslo stránky) a `stop` (čísla stránek budou potlačena). Název těchto hodnot (`start` a `stop`) by nás mohl vést k domněnce, že když máme „`state=stop`“, stránky přestanou být číslovány, a když „`state=start`“, číslování začne znovu. Ale není tomu tak: tyto hodnoty ovlivňují pouze to, zda je číslo stránky zobrazeno nebo ne.
- **location:** označuje, kde budou čísla zobrazeny. Normálně musíme v této možnosti uvést dvě hodnoty oddělené čárkou. Nejprve musíme určit, zda chceme číslo stránky v záhlaví („`header`“) nebo v zápatí („`footer`“), a potom, kde v záhlaví nebo zápatí: může to být „`left`“, „`middle`“, „`right`“, „`inleft`“, „`inright`“, „`margin`“, „`inmargin`“, „`atmargin`“ nebo „`marginedge`“. Například: pro zobrazení číslování zarovnaného vpravo v zápatí bychom měli uvést „`location={footer,right}`“. Podívejte se na druhou stranu, jak jsme tuto možnost obklopili složenými závorkami, aby ConTeXt mohl správně interpretovat oddělovací čárku.
- **style:** označuje velikost a styl písma, který se má použít pro čísla stránek.
- **color:** označuje barvu, která se má použít na číslo stránky.
- **left:** vybere příkaz nebo text, který se má provést, vlevo od čísla stránky.
- **right:** vybere příkaz nebo text, který se má provést, vpravo od čísla stránky.
- **command:** vybere příkaz, kterému bude předáno číslo stránky jako parametr.
- **width:** udává šířku, kterou zabírá číslo stránky.
- **strut:** Tím si nejsem tak jistý. V mých testech, když „`strut=no`“, číslo se vytiskne přesně na horní hranu záhlaví nebo na spodní část zápatí, zatímco když „`strut=yes`“ (výchozí hodnota) je mezi číslem a hranou mezeru.

`\setupuserpagenumber`, umožňuje tyto další možnosti:

- **numberconversion**: řídí druh číslování, které může být arabskými čísly („n“, „numbers“), malými písmeny („a“, „characters“), velkými písmeny („A“, „Characters“), kapitálkami („KA“), malými římskými čísly („i“, „r“, „romannumerals“), velkými římskými čísly („I“, „R“, „Romannumerals“) nebo kapitálkami římských čísel („KR“).
- **number**: označuje číslo, které má být přiřazeno první stránce, na základě kterého bude vypočítán zbytek.
- **numberorder**: pokud tomu přiřadíme „reverse“ jako hodnotu, číslování stránek bude v sestupném pořadí; to znamená, že poslední stránka bude 1, předposlední 2 atd.
- **way**: umožňuje nám určit, jak bude číslování probíhat. Může to být: byblock, bychapter, bysection, bysubsection atd.
- **prefix**: umožňuje nám označit předponu k číslům stránek.
- **numberconversionset**: Vysvětleno dále.

Kromě těchto dvou příkazů je nutné vzít v úvahu také ovládání čísel zahrnujících makrostrukturu dokumentu (viz [section 3.6](#)). Z tohoto pohledu nám `\defineconversionset` umožňuje označit pro každý z bloků makrostruktury jiný druh číslování. Například:

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

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

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

uvidíte, že první blok v našem dokumentu (frontmatter) je očíslován malými římskými číslicemi, centrální blok (bodymatter) arabskými čísly a přílohy velkými písmeny.

K získání čísla stránky můžeme použít následující příkazy:

- `\userpagenumber`: vrátí číslo stránky tak, jak bylo nakonfigurováno pomocí `\setuppagenumbering` a pomocí `\setupuserpagenumber`.
- `\pagenumber`: vrátí stejné číslo jako předchozí příkaz, ale stále v arabských číslech.
- `\realpagenumber`: vrátí skutečné číslo stránky v arabských číslech bez zohlednění kterékoliv z těchto specifikací.

Chcete-li získat číslo poslední stránky v dokumentu, existují tři příkazy, které jsou paralelní s předchozími. Jsou to: `\lastuserpagenumber`, `\lastpagenumber` a `\lastrealpagenumber`.

1.5 Vynucené nebo navrhované konce stránek

1.5.1 Příkaz `\page`

Algoritmus pro distribuci textu v ConTeXtu je poměrně složitý a je založen na množství výpočtů a vnitřních proměnných, které programu říkají, kde je nejlepší možný bod pro zavedení zalomení aktuální stránky z hlediska typografické správnosti. Příkaz `\page` nám umožňuje ovlivnit tento algoritmus:

- Navrhováním určitých bodů jako nejlepšího nebo nejnevhodnějšího místa pro vložení konce stránky.
 - **no**: označuje, že místo, kde se příkaz nachází, není vhodným kandidátem pro vložení konce stránky, takže pokud je to možné, zalomení by mělo být provedeno na jiném místě dokumentu.
 - **preference**: říká ConTeXtu, že místo, kde narazí na příkaz, je *dobré místo* pro pokus o zalomení stránky, i když to tam nevynutí.
 - **bigpreference**: označuje, že místo, kde se setká s příkazem, je *velmi dobré místo* pro pokus o zalomení stránky, ale také to nejde tak daleko, že by ho vynutilo.

Všimněte si, že tyto tři možnosti nevynucují ani nezabraňují zalomení stránky, ale pouze říkají ConTeXtu, že při hledání nejlepšího místa pro zalomení stránky by měl vzít v úvahu to, co je uvedeno v tomto příkazu. Nakonec však o místě, kde dojde k zalomení stránky, bude nadále rozhodovat ConTeXt.

- Vynucení přerušování stránky v určitém bodě; v tomto případě můžeme také uvést, kolik stránek by se mělo zalomit, stejně jako určité vlastnosti stránek, které mají být vloženy.

- **yes**: vynutit přerušení stránky v tomto bodě.
- **makeup**: podobně jako „yes“, ale nucené přerušení je okamžité, bez předchozího umístění jakýchkoli plovoucích objektů, jejichž umístění čeká na vyřízení (viz [section 4.1](#)).
- **empty**: vloží do dokumentu zcela prázdnou stránku.
- **even**: vlož tolik stránek, kolik je potřeba, aby byla další stránka sudá.
- **odd**: vlož tolik stránek, kolik je potřeba, aby byla další stránka lichá.
- **left, right**: podobné dvěma předchozím možnostem, ale použitelné pouze pro oboustranně tištěné dokumenty s různými záhlavími, zápatími nebo okraji v závislosti na tom, zda je stránka lichá nebo sudá.
- **quadruple**: vložte počet stránek potřebný k tomu, aby další stránka byla násobkem 4.

Kromě těchto voleb, které specificky řídí stránkování, `\page` obsahuje další volby, které ovlivňují způsob, jakým tento příkaz funguje. Zejména volba „`disable`“, která způsobí, že ConTeXt bude ignorovat příkazy `\page`, které odtamtud najde, a volba „`reset`“, která vyvolá opačný efekt a obnoví účinnost budoucích `\page` příkazů.

1.5.2 Spojení určitých řádků nebo odstavců

Někdy, pokud chceme zabránit zalomení stránky mezi několika odstavci, může být použití příkazu `\page` pracné, protože by musel být zapsán na každém místě, kde by bylo možné zalomení stránky vložit. Jednodušší postup je umístit materiál, který chceme nechat na stejné stránce do toho, co TeX nazývá *vertical box*.

Na začátku tohoto dokumentu (na page ??) jsem uvedl, že interně je vše *box* pro TeX. Pojem *box* je v TeXu základní pro jakýkoli druh *pokročilých* operací; ale jeho řízení je příliš složité na to, aby bylo zahrnuto do tohoto úvodu. To je důvod, proč se o boxech zmiňuji jen občas.

Jednou vytvořené boxy v TeXu jsou nedělitelné, což znamená, že nemůžeme vložit konec stránky, který by rozdělil box na dvě části. To je důvod, proč, pokud vložíme materiál, který chceme mít pohromadě, do neviditelného boxu, vyhneme se vložení zalomení stránky, které by tento materiál rozdělilo. Příkaz k tomu je `\vbox`, jehož syntaxe je

```
\vbox{Material}
```

kde *Material* je text, který chceme zachovat pohromadě.

Některá prostředí ConTeXtu umístí svůj obsah do boxu. Například „`framedtext`“, takže pokud zarámujeme materiál, který chceme mít pohromadě v tomto prostředí, a také uvidíme, že rám je neviditelný (což uděláme s volbou `frame=off`), dosáhneme stejné věci.

1.6 Záhloví a zápatí

1.6.1 Příkazy pro stanovení obsahu záhlaví a zápatí

Pokud jsme v rozvržení stránky přiřadili záhlaví a zápatí určitou velikost, můžeme do nich zahrnout text pomocí příkazů `\setupheadertexts` a `\setupfootertexts`. Oba příkazy jsou podobné, jediný rozdíl je v tom, že první aktivuje obsah záhlaví a druhý obsah zápatí. Oba mají jeden až pět argumentů.

1. Při použití s jedním argumentem bude obsahovat text záhlaví nebo zápatí, který bude umístěn uprostřed stránky. Příklad: `\setupfootertexts[pagenu-ber]` zapíše číslo stránky do středu zápatí.
2. Při použití se dvěma argumenty bude obsah prvního argumentu umístěn na levou stranu záhlaví nebo zápatí a obsah druhého argumentu na pravou stranu. Například `\setupheadertexts[Preface][pagenumber]` vysází záhlaví stránky, ve kterém je slovo „preface“ napsáno na levé straně a číslo stránky je vytištěno na pravé straně.
3. Pokud použijeme tři argumenty, první bude označovat *oblast*, ve které mají být vytištěny další dva. *Oblastí* mám na mysli *oblasti* stránky uvedené v [section 1.2](#), jinými slovy: hrana, okraj, záhlaví... Další dva argumenty obsahují text, který se má umístit na levou hranu nebo okraj, nebo na pravou hranu nebo okraj.

Použití se čtyřmi nebo pěti argumenty je ekvivalentní použití se dvěma nebo třemi argumenty v případech, kdy se rozlišuje mezi sudými a lichými stránkami, k čemuž dochází, jak víme, když „`alternative=doublesided`“ s nastavením `\setuppagenumbering`. V tomto případě jsou přidány dva možné argumenty, které odrážejí obsah levé a pravé strany sudých stránek.

Důležitou charakteristikou těchto dvou příkazů je, že když jsou použity se dvěma argumenty, předchozí centrální záhlaví nebo zápatí (pokud existovalo) není přepsáno, což nám umožňuje napsat jiný text do každé oblasti, pokud nejprve napíšeme centrální text (volání příkazu s jedním argumentem) a poté napíšeme texty pro obě strany (volání znovu, nyní se dvěma argumenty). Pokud tedy například napíšeme následující příkazy

```
\setupheadertexts[and]  
\setupheadertexts[Tweedledum][Tweedledee]
```

První příkaz napíše „and“ do středu záhlaví a druhý napíše „Tweedledum“ na-
levo a „Tweedledee“ napravo, přičemž středová oblast zůstane nedotčená, protože
nebyla objednána k přepsání. Výsledné záhlaví se nyní zobrazí jako

Tweedledum

and

Tweedledee



Vysvětlení fungování těchto příkazů, které jsem právě uvedl, je můj závěr po mnoha testech. Vysvětlení těchto příkazů v ConTeXt *exkurzi* je založeno na verzi s pěti argumenty; a ten v referenční příručce z roku 2013 je založen na verzi se třemi argumenty. Myslím, že můj je jasnější. Na druhou stranu jsem neviděl vysvětlení, proč volání druhého příkazu nepřepíše předchozí volání, ale takhle to funguje, když do záhlaví nebo zápatí napíšeme nejprve centrální položku a poté ty na obě strany. Pokud ale položky napíšeme prvně na jednu ze stran do záhlaví/zápatí, následné volání příkazu k zápisu centrální položky smaže předchozí záhlaví nebo zápatí. Proč? Nemám ponětí. Myslím, že tyto malé detaily představují zbytečné komplikace a měly by být jasně vysvětleny v oficiální dokumentaci.

Kromě toho můžeme jako skutečný obsah záhlaví nebo zápatí označit jakoukoli kombinaci textu a příkazů. Ale také následující hodnoty:

- **date**, **currentdate**: zapíše (kterýkoli z nich) aktuální datum.
- **pagenumber**: zapíše číslo stránky.
- **part**, **chapter**, **section...**: zapíše název odpovídající části, kapitole, sekci... nebo jakémukoli strukturálnímu členění.
- **partnumber**, **chapternumber**, **sectionnumbere...**: zapíše číslo dílu, kapitoly, sekce... nebo jakéhokoli strukturálního členění.

Pozor: Tato symbolická jména (**date**, **currentdate**, **pagenumber**, **chapter**, **chapternumber** atd.) jsou jako taková interpretována pouze tehdy, je-li samotný symbolický název jediným obsahem argumentu; ale pokud přidáme nějaký další text nebo příkaz pro formátování, budou tato slova interpretována doslovně, a tak například když napíšeme `\setupheadertexts[chapternumber]`, dostaneme číslo aktuální kapitoly; ale pokud napíšeme `\setupheadertexts[Chapter chapternumber]`, skončíme s: „Chapter chapternumber“. V těchto případech, kdy obsahem příkazu není pouze symbolické slovo, musíme:

- Pro **date**, **currentdate** a **pagenumber** nepoužívejte symbolické slovo, ale příkaz se stejným názvem (`\date`, `\currentdate` nebo `\pagenumber`).
- Pro **part**, **partnumber**, **chapter**, **chapternumber** atd. použijte příkaz `\getmarking[Mark]`, který vrátí obsah *Mark*, který je požadován. Takže například `\getmarking[chapter]` vrátí název aktuální kapitoly, zatímco `\getmarking[chapternumber]` vrátí číslo aktuální kapitoly.

Chcete-li zakázat záhlaví a zápatí na konkrétní stránce, použijte příkaz `\noheaderandfooterlines`, který funguje výhradně na stránce, kde se nachází. Pokud chceme smazat pouze číslo stránky na konkrétní stránce, musíme použít příkaz `\page[blank]`.

1.6.2 Formátování záhlaví a zápatí

Konkrétní formát, ve kterém je zobrazen text záhlaví nebo zápatí, lze uvést v argumentech pro `\setupheadertexts` nebo `\setupfootertexts` pomocí odpovídajících příkazů formátu. Můžeme to však také konfigurovat globálně pomocí `\setupheader` a `\setupfooter`, které umožňují následující možnosti:

- **state**: umožňuje následující hodnoty: `start`, `stop`, `empty`, `high`, `none`, `normal` nebo `nomarking`.
- **style**, **leftstyle**, **rightstyle**: konfigurace stylu textu záhlaví a zápatí. `style` ovlivňuje všechny stránky, `leftstyle` sudé stránky a `rightstyle` liché stránky.
- **color**, **leftcolor**, **rightcolor**: barva záhlaví nebo zápatí. Může ovlivnit všechny stránky (volba `color`) nebo pouze sudé stránky (`leftcolor`) nebo liché stránky (`rightcolor`).
- **width**, **leftwidth**, **rightwidth**: šířka všech záhlaví a zápatí (`width`) nebo záhlaví/zápatí na sudých stránkách (`leftwidth`) nebo na lichých (`rightwidth`).
- **before**: příkaz, který se má provést před zápisem záhlaví nebo zápatí.
- **after**: příkaz, který se má provést po napsání záhlaví nebo zápatí.
- **strut**: pokud „yes“, vytvoří se vertikální oddělovací prostor mezi záhlavím a hranou. Když je to „no“, záhlaví nebo zápatí jde k hranám horní nebo dolní oblasti hran.

1.6.3 Definování konkrétních záhlaví a zápatí a jejich propojení s příkazy sekce

Systém záhlaví a zápatí ConTeXtu nám umožňuje automaticky změnit text v záhlaví nebo zápatí, když změníme kapitoly nebo sekce; nebo když měníme stránky, pokud jsme nastavili různá záhlaví nebo zápatí pro liché a sudé stránky. Co ale neumožňuje, je rozlišovat mezi první stránkou (dokumentu nebo kapitoly nebo oddílu) a zbytkem stránek. Abychom toho dosáhli, musíme:

1. Definovat konkrétní záhlaví nebo zápatí.
2. Propojit jej se sekcí, na kterou se vztahuje.

Definice konkrétních záhlaví nebo zápatí se provádí pomocí příkazu `\definertext`, jehož syntaxe je:

```
\definetext  
  [Name] [Type]  
  [Content1] [Content2] [Content3]  
  [Content4] [Content5]
```

kde *Name* je jméno přiřazené záhlaví nebo zápatí, se kterým se zabýváme; *Type* může být **header** nebo **footer** v závislosti na tom, který z těchto dvou definujeme a zbývajících pět argumentů obsahuje obsah, který chceme pro nové záhlaví nebo zápatí, podobně jako jsme viděli funkce `\setupheadertexts` a `\setupfootertexts`. Jakmile to uděláme, musíme propojit nové záhlaví nebo zápatí s nějakou konkrétní sekcí pomocí `\setuphead` pomocí voleb *header* a *footer* (které nejsou vysvětleny v [Chapter 3](#)).

Následující příklad tedy skryje záhlaví na první stránce každé kapitoly a jako zápatí se zobrazí vystředěné číslo stránky:

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

1.7 Vložení textových prvků na hrany a okraje stránky

Horní a spodní hrana a pravý a levý okraj obvykle neobsahují žádný text. ConTeXt tam však umožňuje umístění některých textových prvků. Pro tento účel jsou k dispozici zejména následující příkazy:

- `\setuptoptexts`: umožňuje umístit text na horní hranu stránky (nad oblast záhlaví).
- `\setupbottomtexts`: umožňuje umístit text na spodní hranu stránky (pod oblast zápatí).
- `\margintext`, `\atleftmargin`, `\atrightmargin`, `\ininner`, `\ininneredge`, `\ininnermargin`, `\inleft`, `\inleftedge`, `\inleftmargin`, `\inmargin`, `\inother`, `\inouter`, `\inouteredge`, `\inoutermargin`, `\inright`, `\inrightedge`, `\inrightmargin`: umožňují nám umístit text na boční hrany a okraje dokumentu.

První dva příkazy fungují přesně jako `\setupheadertexts` a `\setupfootertexts` a formát těchto textů lze dokonce předem nakonfigurovat pomocí `\setuptop` a `\setupbottom` podobně, jak to umožňuje `\setupheader` ke konfiguraci textů pro

`\setupheadertexts`. K tomu všemu odkazuji na to, co jsem již řekl v [section 1.6](#). Jediný malý detail, který je třeba dodat je, že text nastavený pro `\setuptoptexts` nebo `\setupbottomtexts` nebude viditelný, pokud v rozložení stránky nebylo vyhrazeno místo pro horní (`top`) nebo spodní (`bottom`) hranu. Viz [section 1.3.1](#).

Pokud jde o příkazy zaměřené na umístění textu na okraje dokumentu, všechny mají podobnou syntaxi:

`\CommandName [Reference] [Configuration] {Text}`

kde *Reference* a *Configuration* jsou volitelné argumenty; první se používá pro případné křížové odkazy a druhý umožňuje nastavit okrajový text. Poslední argument, uzavřený do složených závorek, obsahuje text, který má být umístěn na okraj.

Z těchto příkazů je obecnějším příkazem `\margintext`, protože umožňuje umístit text na jakýkoli okraj nebo boční hranu stránky. Zbývající příkazy, jak naznačuje jejich název, umístí text na samotný okraj (pravý nebo levý, vnitřní nebo vnější) nebo na hranu (pravou nebo levou, vnitřní nebo vnější). Tyto příkazy úzce souvisí s rozložením stránky, protože pokud například použijeme `\inrightrightedge`, ale nevyhradíme žádné místo v rozložení stránky pro pravou hranu, nic se nezobrazí.

Možnosti konfigurace pro `\margintext` jsou následující:

- **location**: označuje do jakého okraje bude text umístěn. Může to být `left`, `right` nebo v oboustranných dokumentech `outer` nebo `inner`. Ve výchozím nastavení je to `left` v jednostranných dokumentech a `outer` v oboustranných.
- **width**: šířka dostupná pro tisk textu. Ve výchozím nastavení se použije celá šířka okraje.
- **margin**: udává, zda bude text umístěn v okraji samotném nebo v hraně.
- **align**: zarovnání textu. Jsou zde použity stejné hodnoty jako v `\setupalign 2.6.1`.
- **line**: umožňuje nám označit počet řádků posunutí textu na okraji. Takže `line=1` přemístí text o jeden řádek níže a `line=-1` o jeden řádek výše.
- **style**: příkaz nebo příkazy pro označení stylu textu, který se má umístit na okraje.
- **color**: barva textu na okraji.
- **command**: název příkazu, kterému bude jako argument předán text, který má být umístěn na okraj. Tento příkaz bude proveden před napsáním textu. Například, pokud chceme kolem textu nakreslit rámeček, můžeme použít „`[command=\framed] {Text}`“.

Zbývající příkazy umožňují stejné možnosti, kromě `location` a `margin`. Zejména příkazy `\atrightmargin` a `\atleftmargin` umístí text zcela připojený k tělu stránky. Můžeme vytvořit oddělovací prostor pomocí možnosti `distance`, kterou jsem nezmínil, když jsem mluvil o `\margintext`, protože jsem v testech neviděl žádný vliv na tento příkaz.



Kromě výše uvedených možností tyto příkazy také podporují další možnosti (`strut`, `anchor`, `method`, `category`, `scope`, `option`, `hoffset`, `voffset`, `dy`, `bottomspace`, `threshold` a `stack`), které jsem nezmínil, protože jsou nedokumentovány a upřímně, nejsem si moc jistý, k čemu jsou. Ty s názvy jako *distance* můžeme hádat, ale zbytek? Wiki zmiňuje pouze volbu `stack` s tím, že se používá k emulaci příkazu `\marginpars` v L^AT_EXu, ale to se mi nezdá příliš jasné.

Příkaz `\setupmargindata` nám umožňuje globálně konfigurovat texty na každém okraji. Tak například

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

zajistí, že všechny texty na pravém okraji budou psány šikmo.

Můžeme také vytvořit vlastní přizpůsobený příkaz pomocí

```
\definemargindata[Name][Configuration]
```

Chapter 2

Fonts and colours in ConT_EXt

Table of Contents: **2.1 Typografická písma zahrnutá v „ConT_EXt Standalone“;** **2.2 Vlastnosti písma;** 2.2.1 Písma, *style* a varianty stylů; 2.2.2 Velikost písma; **2.3 Nastavení hlavního písma dokumentu;** **2.4 Změna písma nebo některých funkcí písma;** 2.4.1 Příkazy `\setupbodyfont` a `\switchtobodyfont`; 2.4.2 Rychlá změna stylu, alternativy a velikosti; 2.4.3 Definování příkazů a klíčových slov pro velikosti, style a alternativy písma; **2.5 Další záležitosti týkající se používání některých alternativních řešení;** 2.5.1 Kurzíva, šikmé písmo a zvýraznění; 2.5.2 Malé kapitálky a falešné malé kapitálky; **2.6 Použití a konfigurace barev;** 2.6.1 Postupy pro barevnou sazbu fragmentů textu; 2.6.2 Změna barvy pozadí a popředí dokumentu; 2.6.3 Příkazy pro obarvení jednotlivých textových fragmentů; 2.6.4 Předdefinované barvy; 2.6.5 Zobrazení dostupných barev; 2.6.6 Definování vlastních barev;

2.1 Typografická písma zahrnutá v „ConT_EXt Standalone“

Fonty ConT_EXtu nabízí mnoho možností, ale také poměrně dost složitých. Nebudu se zabývat všemi pokročilými možnostmi písma. v této příručce, ale omezím se na předpoklad, že budeme pracovat s některými z 21 písem, která jsou součástí instalace ConT_EXt Standalone, těmi, které jsou uvedeny v [tabulka 2.1](#).

Prostřední sloupec [tabulka 2.1](#) označuje název nebo jména, pod kterými ConT_EXt zná dané písmo. Pokud existují dvě jména, jedná se o synonyma. V posledním sloupci je uveden příklad písma který se používá. Co se týče pořadí, v jakém jsou písma zobrazena, první z nich je písmo, které ConT_EXt používá ve výchozím nastavení, a zbývající písma jsou v pořadí abecedním, zatímco poslední tři písma jsou speciálně navržena pro matematiku. Všimněte si, že Eulerovo písmo nemůže přímo reprezentovat písmena s diakritikou, takže dostaneme Bront's, nikoli Brontě's.

Pro čtenáře, kteří přicházejí ze světa Windows a jeho výchozích písem, uvedu, že *heros* je v systému Windows ekvivalentem Arialu, zatímco *termes* je stejné jako

Oficiální jméno	Jména v ConT _E Xtu	Příklad
Latin Modern	modern, modern-base	Emily Brontë's book
Antykwa Poltawskiego	antypol	Emily Brontë's book
Antykwa Toruńska	antypol	Emily Brontë's book
Cambria	cambria	Emily Brontë's book
DejaVu	dejavu	Emily Brontë's book
DejaVu Condensed	dejavu-condensed	Emily Brontë's book
Gentium	gentium	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	stix	Emily Brontë's book
Xits	xits	Emily Brontë's book

Tabulka 2.1 Fonts included in the ConT_EXt distribution

Times New Roman. Nejsou úplně stejná ale jsou si natolik podobné, že by bylo třeba být velmi všímavý, aby je člověk rozeznal.

Písma používaná systémem Windows nejsou *volný software* (ve skutečnosti téměř nic ve Windows není *volný software*), takže je nelze zahrnout do ConT_EXtu. Pokud je však ConT_EXt nainstalován na Windows, pak jsou tato písma již nainstalována a mohou být použita stejně, jako jakákoli jiná písma nainstalovaná v systému se spuštěným ConT_EXtem. V tomto úvodu se však nebudu zabývat tím, jak používat písma již nainstalovaná v systému. Náповědu lze nalézt na [ConT_EXt wiki](#).

2.2 Vlastnosti písma

2.2.1 Písma, *styly* a varianty stylů

Terminologie týkající se fontů je poněkud matoucí, protože někdy to, co se nazývá písmo, je ve skutečnosti rodina písem, která zahrnuje různé styly a varianty, které mají společný základní design. Nebudu se zabývat otázkou, která terminologie je správnější; chci pouze objasnit terminologii používanou v ConT_EXtu. Tam, se rozlišuje mezi písmy, styly a variantami (resp. alternativami) pro každý styl. Písma obsažená v ConT_EXtu (ve skutečnosti se jedná o rodiny písem) jsou ty, které jsme viděli v předchozí části. Nyní se podíváme na *styly* a *alternativy*.

Styly písma

DONALD E. KNUTH navrhl *Computer Modern* písmo pro T_EXa dal mu tři různé *style* zvané *roman*, *sans serif* a *teletype*. Styl *roman* je designem kde jsou znaky ozdobně rozvolněné, známé v typologickém designu, proto je tento styl písma známý také jako *serif*. Tento styl byl považován za *normalní* nebo výchozí styl. Styl *sans serif*, jak naznačuje jeho název, postrádá ozdobnost a je tedy jednodušší, stylizovanější, někdy se používá i jako font. známý pod jinými názvy, např. ve španělštině *paloseco*; toto písmo může být hlavním písmem v dokumentu, ale je také vhodné pro použití v některých částech textu, kterých hlavním písmem je *roman*. jako například nadpisy nebo záhlaví stránek. A konečně styl *teletype* byl zařazen do stylu *Computer Roman*, protože byl navržen pro psaní knih týkajících se počítačového programování, které zahrnují rozsáhlé části v počítačovém *kódu*, který je konvenčně v tištěných materiálech zobrazován monospace stylem, který napodobuje počítačových terminálů a starých psacích strojů.

Čtvrtý styl určený pro matematické fragmenty, by mohl být přidán k těmto tří *stylům*. Ale protože T_EX automaticky používá tento styl, když přejde do matematického režimu, a neobsahuje příkazy pro jeho povolení nebo zakázání, ani nemá *varianty* nebo alternativy ostatních stylů, není obvyklé o něm přemýšlet jako *ostylu*.

ConT_EXt obsahuje příkazy pro dva další možné style: ručně psaný a kaligrafický. Nejsem si úplně jistý, rozdílem mezi nimi, protože na jedné straně žádný z fontů obsažen ConT_EXtu tyto style neobsahuje. a na druhou stranu, jak vidím, kaligrafické písmo je v tomto případě také spíše ručně psané písmo. Tyto příkazy, které ConT_EXt obsahuje, pro umožnění těchto stylů, pokud budou použity s písmem, které je neimplementuje, nezpůsobí žádnou chybu při kompilaci: prostě se nic nestane.

Alternativní formy písma

Každý *styl* povoluje určitý počet alternativních forem, proto je ConT_EXt nazývá, (*alternativy*):

- Regular or normal („**tf**“, z *typeface*).
- Bold („**bf**“, z *boldface*).
- Italic („**it**“ z *italic*)
- BoldItalic („**bi**“ z *bold italic*)
- Slanted („**sl**“ z *slanted*)
- BoldSlanted („**bs**“ z *bold slanted*)
- Small caps („**sc**“ z *small caps*)
- Medieval („**os**“ z *old style*)

Tyto *alternativy*, jak už jejich název napovídá, se vzájemně ovlivňují: když je jedna z nich povolena, ostatní jsou zakázány. To je důvod, proč ConT_EXt poskytuje příkazy pro jejich povolení, ale ne pro zakázání. Protože když povolíme alternativu, zakážeme tu, kterou jsme předtím povolovali. a tak například, pokud píšeme kurzívou a povolíme tučné písmo, bude kurzíva zakázána. Pokud chceme používat

tučné písmo a kurzívu současně, nemusíme povolit jedno a pak druhé, ale spíše povolit alternativu, která zahrnuje obojí („bi“).

Na druhou stranu je třeba mít na paměti, že ačkoli ConTeXt předpokládá, že každé písmo bude mít tyto alternativy, a tudíž poskytuje příkazy, které umožňují, aby fungovaly a produkovaly nějaké vnímatelné efekty ve výsledném dokumentu, potřebují tyto příkazy, aby písmo. mělo ve svém návrhu specifické formy pro každý styl a alternativu.

Zejména mnoho písem nerozlišuje ve svém designu mezi šikmými a kurzívními písmeny, nebo neobsahují speciální tvary pro malé kapitálky.

Rozdíl mezi kurzívou a šikmými písmeny

Podobnost typografické funkce kurzívy a šikmého písma vede mnoho lidí k záměně těchto dvou alternativ. Šikmé písmeno vzniká mírným pootočením pravidelného tvaru. Ale kurzíva znamená - alespoň v některých písmech - odlišný design ve tvaru písmene. kde se zdá, že písmena jsou nakloněná, protože byla nakreslena tak, aby to tak vypadalo, ale ve skutečnosti se nejedná o skutečný náklon. To může být v následujícím příkladu, ve kterém jsme napsali stejné slovo třikrát ve stejné velikosti, aby bylo snadno rozpoznatelné. rozdíly. V první verzi je použita základná podoba, v druhém šikmá a ve třetím kurzíva:

italics – *italics* – italics

Všimněte si, že design znaků je v prvních dvou dílech stejný. ale ve třetím příkladu jsou jemné rozdíly v tazích některých písmen, což je velmi zřejmé, zejména v tom, jak vypadá ‚a‘. ačkoli rozdíly se ve skutečnosti vyskytují téměř ve všech případech. znaků.

Obvyklá použití kurzívy a šikmých písmen jsou podobná a každá osoba se rozhodne, zda použije jedno nebo druhé. Zde je volnost, i když měli bychom zdůraznit, že dokument bude lépe napsán a bude vypadat lépe, pokud je použití kurzívního a šikmého písma *konzistentní*. V mnoha písmech je navíc rozdíl v designu mezi kurzívou a šikmými písmeny zanedbatelný, takže je jedno, zda použijeme jedno nebo druhé. nebo druhou.

Na druhou stranu, jak kurzíva, tak šikmé písmo jsou alternativami písma, které znamená především dvě věci:

1. Můžeme je použít pouze tehdy, jsou-li definovány v písmu.
2. Povolením jedné z nich deaktivujeme alternativu, která která byla do té doby používána.

Spolu s příkazy pro kurzívu a šikmé písmo nabízí ConTeXt další příkazy pro *zdůraznění* určitého textu. Jeho použití znamená jemné rozdíly ve srovnání s kurzivou nebo šikmým písmem. Viz sekce ??.

2.2.2 Velikost písma

Všechna písma, se kterými ConTeXt pracuje, jsou založena na vektorové grafice. takže teoreticky mohou být zobrazena v libovolné velikosti písma, ačkoli, jak uvidíme, záleží to na skutečných instrukcích, které používáme k určení velikosti písma. Pokud není uvedeno jinak, předpokládá se, že velikost písma bude 12 bodů.

Všechna písma používaná v ConTeXtu jsou založena na vektorové grafice a Proto jsou to písma Opentype nebo Type 1, což znamená, že písma, jenž vznikly před touto technologií, byly implementovány znovu. Konkrétně se jedná o, výchozí písmo T_EXu, *Computer Modern*, které navrhl Knuth, existovalo pouze v určitých velikostech, a proto bylo znovu implementováno v designu nazvaném *Latin Modern*, který používá ConTeXt, ačkoli v mnoha dokumentech se nadále nazývá *Computer Modern* kvůli silné symbolice, kterou toto písmo stále má v systému T_EX, odkdy začaly a byly vyvinuty Knuthem spolu s dalším programem nazvaným MetaFont, jehož cílem bylo navrhnout písma, která by mohla pracovat s T_EX.

2.3 Nastavení hlavního písma dokumentu

Ve výchozím nastavení, pokud není uvedeno jiné písmo, ConTeXt použije *Latin Modern Roman* o velikosti 12 bodů jako hlavní písmo. Toto písmo bylo původně navrhl KNUTH pro implementaci v T_EXu. Je to elegantní písmo římského stylu s velkou proporční a dekorativní „kresbou“ - nazývanými *serifs* - v některých tazích, které je velmi vhodné jak pro tištěné texty, tak pro zobrazení na obrazovce; ačkoli – a to je osobní názor – není tak vhodné pro malé obrazovky, jako je *smartphone*, protože *serify* nebo ozdoby mají tendenci se hromadit, což ztěžuje čtení.

Pro nastavení jiného písma použijeme `\setupbodyfont`, který nám umožní ne jen změnit skutečné písmo, ale také jeho velikost a styl. Když chceme, aby to platilo pro celý dokument, musíme to zahrnout do položky preambuli zdrojového souboru. Pokud však chceme jednoduše změnit písmo v místě v určitém bodě, musíme do něj zahrnout to, co následuje.

Formát `\setupbodyfont` je:

`\setupbodyfont[Options]`

kde různé možnosti příkazu umožňují uvést:

- **Jméno písma**, což může být kterýkoli ze symbolických názvů písem nalezené v [tabulce 2.1](#).
- **Velikost**, která může být označena buď rozměry (s použitím bodu jako měrné jednotky) nebo určitými symbolickými názvy. Všimněte si však, že i když jsem

dříve uvedl, že písma používaná v ConTeXtu lze škálovat prakticky na libovolnou velikost, v `\setupbodyfont` se používají pouze velikosti skládající se z celých čísel mezi 4 a 12, stejně jako velikosti 14,4 a 17,3, jsou v ConTeXtu podporovány. Ve výchozím nastavení se předpokládá velikost 12 bodů.

`\setupbodyfont`, vytváří něco, co bychom mohli nazvat *základní velikost* dokumentu; jinými slovy *normální* velikost znaků. na jejímž základě se vypočítávají další velikosti, například nadpisy a poznámky pod čarou. Když změníme hlavní velikost pomocí `\setupbodyfont`, všechny ostatní velikosti vypočtené na základě hlavního písma se také změní. změní.

Kromě přímo uvedené velikosti znaků (10pt, 11pt, 12pt atd.) můžeme také použít některé symbolické názvy, které vypočítají velikost znaku pro použití na základě aktuální velikosti. Jedná se o tyto symbolické názvy, od největšího po nejmenší: `big`, `small`, `script`, `x`, `scriptscript` a `xx`. Pokud tedy například chceme nastavit text těla pomocí `\setupbodyfont`, který je větší než 12 bodů, můžeme tak učinit pomocí „`big`“.

- **Styl písma**, který, jak jsme již uvedli, může být *roman* (se serify), nebo bez serifů (*san serif*), nebo styl psacího stroje a u některých písem ručně psaný a kaligrafický styl. `\setupbodyfont` umožňuje různé symbolické názvy pro označení různých stylů. Jsou v tabulce 2.2:

Styl	Symbolické jména povoleny
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>
Styl psacího stroje	<code>tt</code> , <code>modo</code> , <code>type</code> , <code>teletype</code>
Ručně psaný	<code>hw</code> , <code>handwritten</code>
Kaligrafický	<code>cg</code> , <code>calligraphic</code>

Tabulka 2.2 Styles in `setupbodyfont`

Pokud mohu posoudit, různé názvy podporované pro jednotlivé styly jsou zcela synonymní.

Náhled

Before deciding to use a particular font in our document, we would normally want to see what it looks like. This can almost always be done from the operating system as there is usually some utility to examine the appearance of the fonts installed on the system; but for convenience, ConTeXt itself offers a utility that allows us to see the appearance of any of the fonts enabled in ConTeXt. This is `\showbodyfont`, that generates a table with examples of the font we indicate.

Než se rozhodneme použít v dokumentu určité písmo, obvykle bychom se chtěli podívat, jak vypadá. To lze téměř vždy provést z operačního systému, protože obvykle existuje nějaká utilita, která umožňuje prozkoumat vzhled písem nainstalovaných

v systému; ale v případě ConTeXtu se nabízí nástroj, který nám umožňuje vidět vzhled kteréhokoli z písem povolených v ConTeXtu. Jedná se o `\showbodyfont`, který vygeneruje tabulku s příklady písma, které jsme označili.

Formát `\showbodyfont` je následující:

`\showbodyfont[Options]`

where we can indicate as options precisely the same symbolic names as in `\setupbodyfont`. So, for example, `\showbodyfont[schola, 8pt]` will show us the table below, in which there are different examples of the schola font at a base size of 8 points:

	[schola] [schola,8pt]										\mr : Ag		
	\tf	\sc	\sl	\it	\bf	\bs	\bi	\tfx	\tfxx	\tfa	\tfb	\tfc	\tfd
\rm	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag
\ss	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag
\tt	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag

Všimněte si, že v prvním řádku a sloupci jsou určité příkazy. tabulky. Dále, když byl význam těchto příkazů vysvětlen, podíváme se ještě jednou do tabulky generované příkazem `\showbodyfont`.

Pokud chceme zobrazit kompletní rozsah znaků v určitém písmu, můžeme použít příkaz `\showfont[FontName]`. Tento příkaz zobrazí hlavní konstrukci každého ze znaků bez použití příkazů pro styly a alternativy.

2.4 Změna písma nebo některých funkcí písma

2.4.1 Příkazy `\setupbodyfont` a `\switchtobodyfont`

Pro změnu písma, stylu nebo velikosti můžeme použít stejný příkaz, kterým jsme stanovili písmo na začátku dokumentu, když nechceme použít výchozí písmo ConTeXtu: `\setupbodyfont`. Vše, co potřebujeme, je umístit tento příkaz na místo v dokumentu, kde si přejeme změnit písmo. Tím dojde k *trvalé* změně písma, což znamená, že přímo ovlivní hlavní písmo a nepřímě všechny ostatní písma, která s ním souvisejí.

Velmi podobné `\setupbodyfont` je `\switchtobodyfont`. Oba nám umožňují měnit stejné aspekty písma (písmo samotný styl a velikost), ale vnitřně fungují odlišně a jsou určeny pro různá použití. První z nich (`\setupbodyfont`) je určen pro

nastavení hlavního (a obvykle jediného) písma. v dokumentu; není to ani běžné, ani typograficky správné mít v dokumentu více než jedno hlavní písmo (proto se nazývá *hlavní* písmo). Naproti tomu `\switchtobodyfont` je určen pro psaní. některé části textu jiným písmem nebo k přiřazení určitého písma k jinému písmu. speciálnímu druhu odstavce, který chceme v dokumentu definovat.

Apart from the above – which actually affects the internal functioning of each of these two commands – from the user’s point of view there are some differences between the use of one or the other command. In particular:

Kromě výše uvedeného - který skutečně ovlivňuje vnitřní fungování každého z těchto dvou příkazů – z pohledu uživatele existují některé rozdíly mezi použitím jednoho nebo druhého příkazu. Zejména:

1. Jak již víme, `\setupbodyfont` je omezen na konkrétní rozsah velikostí, zatímco `\switchtobodyfont` nám umožňuje označit prakticky libovolnou velikost, takže pokud písmo v dané velikosti není k dispozici, bude na ni škálováno.
2. `\switchtobodyfont` nijak neovlivňuje textové prvky kromě místa, kde je použito, na rozdíl od `\setupbodyfont`, které, jak je uvedeno výše, nastavuje hlavní písmo a jeho změnou také změní písmo všech textových prvků, jejichž písmo je vypočteno na základě hlavního písma.

Na druhou stranu oba příkazy mění nejen písmo, ale i styl a také další aspekty spojené s písmem, jako například mezery mezi řádky.

`\setupbodyfont` generuje chybu při kompilaci, pokud je použito nepovolené písmo ale nevolá ji, pokud je požadována neexistující velikost písma. v takovém případě je výchozí písmo (*Latin Modern Roman*). bude povoleno. `\switchtobodyfont` se chová stejně, pokud jde o velikost, jak jsem již řekl, snaží se toho dosáhnout zmenšením písma. Existují však písma, která nelze škálovat na určité velikosti, v takovém případě by výchozí písmo bylo opět povoleno.

2.4.2 Rychlá změna stylu, alternativy a velikosti

Změna stylu a alternativy

Kromě příkazu `\switchtobodyfont` poskytuje ConTeXt sadu příkazů. které nám umožňují rychle změnit styl, alternativu nebo velikost. S pomocí ConTeXt wiki nás upozorňuje, že někdy, když se objeví na začátku odstavce, mohou způsobit některé nežádoucí vedlejší účinky, takže doporučuje, aby v takových případech příkaz předcházel příkaz `\dontleavehmode`.

Tabulka 2.3 obsahuje příkazy, které nám umožňují měnit styl, aniž by se změnil jakýkoli jiný aspekt; a tabulka 2.4 obsahuje příkazy, které nám umožňují měnit výhradně alternativu.

Styl	Příkazy, které to umožňují
Roman	<code>\rm, \roman, \serif, \regular</code>
Sans Serif	<code>\ss, \sans, \support, \sansserif</code>
Monospaced	<code>\tt, \mono, \teletype,</code>
Handwritten	<code>\hw, \handwritten,</code>
Calligraphic	<code>\cf, \calligraphic</code>

Tabulka 2.3 Příkazy pro změnu mezi jednotlivými styly

Alternative	Commands that enable it
Normal	<code>\tf, \normal</code>
Italic	<code>\it, \italic</code>
Bold	<code>\bf, \bold</code>
Bold-italic	<code>\bi, \bolditalic, \italicbold</code>
Slanted	<code>\sl, \slanted</code>
Bold-slanted	<code>\bs, \boldslanted, \slantedbold</code>
Small caps	<code>\sc, \smallcaps</code>
Medieval	<code>\os, \mediaeval</code>

Tabulka 2.4 Příkazy pro povolení konkrétní alternativy

Všechny tyto příkazy si zachovávají svou účinnost, dokud není použit jiný styl nebo alternativa výslovně povolena, nebo *skupina*, v jejímž rámci deklarován příkaz, skončí. Pokud tedy chceme, aby příkaz ovlivnil pouze část textu, musíme tuto část textu uzavřít do příkazu skupiny, jako v následujícím příkladu, kde pokaždé, když se objeví slovo *myšlenka*, pokud se jedná o podstatné jméno, nikoli o sloveso, je uvedeno kurzívou, se vytvoří pro něj skupina.

<pre>Myslel jsem si {\it myšlenku} ale {\it myšlenka}, kterou jsem si myslel nebyl {\it myšlenka}, kterou jsem si myslel, že jsem si myslel. Kdyby {\it myšlenka} kterou jsem si myslel že jsem si myslel byla {\it myšlenka}, kterou jsem si myslel Nemusel bych myslet tak moc!</pre>	<pre>Myslel jsem si <i>myšlenku</i> ale <i>myšlenka</i>, kterou jsem si myslel nebyla <i>myšlenka</i>, kterou jsem si myslel, že jsem si myslel. Kdyby <i>myšlenka</i> kte- rou jsem si myslel, že jsem si myslel byla <i>myš-</i> <i>lenka</i>, kterou jsem si myslel Nemusel bych mys- let tak moc!</pre>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Přípony pro změnu alternativ a velikosti naráz

Příkazy, které mění styl nebo alternativu ve své dvoupísmenné verzi (`\tf, \it, \bf` atd.), umožňují řadu *přípon*, které ovlivňují velikost písma. Přípony a, b, c a d zvětšují velikost písma a násobí ji o 1, 2, 1, 2² (= 1, 44), 1, 2³ (= 1, 728) nebo 1, 2⁴ (= 2, 42). Viz příklad:

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

test, test, test, test, test

přípony `x` a `xx` zmenšují velikost písma a násobí ji čísla 0,8 a 0,6. v tomto pořadí:

`\tf test, \tfx test, \tfxx test`

`test, test, test`

Přípony `,x'` a `,xx'` aplikované na `\tf` nám umožňují zkrátit příkaz, takže `\tfx` lze zapsat jako `\tx` a `\tfxx` jako `\txx`.

Dostupnost těchto různých přípon závisí na aktuálním stavu implementaci písma. Podle reference příručky ConTeXtu 2013 (určené především pro Mark II) jediná přípona zaručená pro to vždy fungovat, je `,x'` a ostatní mohou, ale nemusí být implementovány; nebo mohou být jen pro některé alternativy.

Každopádně, abychom předešli pochybnostem, můžeme použít `\showbodyfont`, o kterém jsem mluvil. (v [sekc](#)i). Tento příkaz zobrazí graf, který nám umožní nejen ocenit vzhled písma, ale také vidět, jak písmo vypadá v každém ze svých stylů a alternativách a také jaké přípony pro změnu velikosti jsou k dispozici.

Podívejme se ještě jednou na tabulku zobrazující `\showbodyfont`:

[modern]										\mr : Ag			
	\tf	\sc	\sl	\it	\bf	\bs	\bi	\tfx	\tfxx	\tfa	\tfb	\tfc	\tfd
\rm	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag
\ss	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag
\tt	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag

Pokud se na tabulku podíváme pozorněji, zjistíme, že první sloupec obsahuje styly písma (`\rm`, `\ss` a `\tt`). První řádek obsahuje vlevo alternativy (`\tf`, `\sc`, `\sl`, `\it`, `\bf`, `\bs` a `\bi`), zatímco vpravo první řádek obsahuje ostatní dostupné přípony, ačkoli na druhé straně jsou přípony pouze s pravidelnou alternativou.

Je důležité si uvědomit, že změna velikosti písma provedená některým z těchto typů přípony změní pouze velikost písma v užším slova smyslu, přičemž zůstane zachována ostatní hodnoty obvykle spojené s velikostí písma, jako je velikost řádku, zůstanou nedotčeny.

Přizpůsobení měřítka přípon

To customise the scaling factor we can use `\definebodyfontenvironment` whose format can be:

Pro přizpůsobení měřítkového faktoru můžeme použít `\definebodyfontenvironment` jehož formát může být:

```
\definebodyfontenvironment[particular size][scaled]
\definebodyfontenvironment[default][scaled]
```

V první verzi bychom nadefinovali měřítko pro určitou velikost. hlavního písma nastaveného pomocí `\setupbodyfont` nebo pomocí `\switchtobodyfont`. Například:

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

by zajistil, že když je hlavní písmo 10 bodů, přípona ,a‘ by se změnila na 12 bodů, přípona ,b‘ na 14 bodů, přípona ,c‘ by původní písmo vynásobila dvěma a přípona ,d‘ třema. Poznámka že pro a a b je uveden pevný rozměr, ale pro c a d je uveden násobek původní velikosti.

Pokud je však první argument `\definebodyfontenvironment` roven „default“, pak budeme předefinovávat hodnotu měřítka pro všechny hodnoty možných velikostí písma a jako hodnotu škálování můžeme zadat pouze hodnotu násobícího číslo. Pokud tedy například napíšeme:

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

naznačujeme, že bez ohledu na velikost hlavního písma a přípona by měla být vynásobena koeficientem 1,3, b koeficientem 1,6, c koeficientem 2 a d koeficientem 4.

Stejně jako přípony xx, x, a, b, c a d, s `\definebodyfontenvironment` můžeme přiřadit hodnotu škálování k znakům „big“, „small“, „script“ a „script-script“ klíčová slova. Tyto hodnoty jsou přiřazeny všem velikostem přidruženým k těmto klíčovým slovům v `\setupbodyfont` a `\switchtobodyfont`. Jsou také použity v následujících příkazech, jejichž užitečnost lze odvodit z jejich názvu:

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

Pokud chceme zobrazit výchozí velikosti určitého písma, můžeme použít příkaz `\showbodyfontenvironment[Font]`. Tento příkaz použitý například na písmo modern poskytně následující výsledek:

[modern]							
text	script	scriptscript	x	xx	small	big	interlinespace
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	

2.4.3 Definování příkazů a klíčových slov pro velikosti, styly a alternativy písma

Předdefinované příkazy pro změnu velikosti, stylů a variant písma jsou dostatečné. ConT_EXt nám navíc umožňuje:

1. Přidání vlastního příkazu pro změnu stylu, velikosti nebo varianty písma.
2. Přidání synonym k názvům stylů nebo variant rozpoznaných pomocí `\switch-tobodyfont`.

K tomu slouží následující příkazy:

- `\definebodyfontswitch`: umožňuje definovat příkaz pro změnu velikosti písma. Například pokud chceme definovat příkaz `\osm` (nebo `\viiiipříkaz`¹) pro nastavení velikosti 8 pt potřebujeme napsat:

```
\definebodyfontswitch[eight][8pt] nebo \definebodyfontswitch[viii][8pt]
```

- `\definefontstyle`: umožňuje nám definovat jedno nebo více slov, která lze použít v `\setupbodyfont` nebo `\switchtobodyfont` nastavit určitý styl písma; takže například, kdybychom chtěli nazvat *sans serif* něčím jiným (např. v příkazu španělsky se nazývá „paloseco“), můžeme napsat

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

¹ Zapomeňte, že s výjimkou řídicích symbolů, ConT_EXt názvy příkazů mohou sestávat pouze z písmen.

Zvláštností `\definefontstyle` je, že umožňuje použít několik slov. současně přiřazeno stejnému stylu, takže, abychom mohli pokračovat španělským příkladem:

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

- `\definealternativestyle`: umožňuje přiřadit jméno k variantě písma. Toto jméno může fungovat jako příkaz nebo být rozpoznán jako volba stylu příkazů, které nám umožňují nastavit styl, který se má použít. Takže pro příklad následující fragment

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

povolí příkaz `\strong` a klíčové slovo „strong“, které bude rozpoznáno volbou `style` příkazů, které tuto volbu umožňují. Mohli jsme říci „bold“, ale toto slovo se již v ConTeXtu používá, proto jsem zvolil výraz používaný v HTML, a to „strong“ jako alternativu.



Nevím, co znamená třetí argument `\definealternativestyle`. Není nepovinný, a proto nemůže být vynechán; ale jediné informace, které jsem o něm našel, jsou tyto v referenční příručce ConTeXt kde se o tomto třetím argumentu říká, že je relevantní pouze pro názvy kapitol a oddílů „*kde se kromě `\cap` musíme řídit i zde použitým písmem*“. (??)

2.5 Další záležitosti týkající se používání některých alternativních řešení

Mezi různými alternativami písma existují dvě, jejichž použití vyžaduje určitá upřesnění:

2.5.1 Kurzíva, šikmé písmo a zvýraznění

Kurzíva i šikmá písmena se používají hlavně pro typografické účely zvýraznění části textu, aby se na něj upozornilo. Jinými slovy, aby jej zdůraznily.

Text můžeme samozřejmě zdůraznit tím, že výslovně povolíme kurzívu nebo šikmé písmo. ConTeXt však nabízí alternativní příkaz, který je mnohem vhodnější. užitečnější a zajímavější a je určen speciálně pro zvýraznění fragmentu textu. Jedná se o příkaz `\em` ze slova *emphasis*. Na rozdíl od `\it` a `\sl`, které jsou čistě typografické příkazy, `\em` je *konceptuální* příkaz; pracuje s ním jinak, takže je univerzálnější, a to až do té míry, že ConTeXt dokumentace doporučuje používat `\em` před `\it` nebo `\sl`. Při použití těchto dvou posledních příkazů říkáme ConTeXtu, jakou alternativu písma chceme použít; ale když použijeme příkaz `\em`, říkáme mu, jaký efekt chceme vytvořit, a necháváme to na ConTeXtu, aby rozhodl, jak

toho dosáhnout. Obvykle se pro dosažení efektu zdůraznění nebo zvýraznění něčeho, povolíme kurzívu nebo šikmé písmo, ale to závisí na kontextu. Pokud tedy použijeme `\em` v příkazu textu, který je již napsán kurzívou – nebo je nakloněn –, příkaz bude zvýrazní opačným způsobem – v tomto případě svislým textem.

Proto následující příklad:

```
{\em Jedna z~nejkrásnějších  
orchidejí na světě je  
\em Thelymitra variegata}.  
neboli Jižní královna ze Sáby.}
```

Jedna z nejkrásnějších orchidejí na světě je Thelymitra variegata neboli Jižní královna ze Sáby.

Všimněte si, že první `\em` umožňuje psaní kurzívou (ve skutečnosti šikmou, ale viz. níže) a že druhý `\em` ji vypíná a místo toho vkládá slova „Thelymitra variegata“ normálním vzpřímeným stylem.

Další výhodou `\em` je to, že není alternativou, takže nezakáže alternativu, kterou jsme měli předtím, a tak například v textu který je tučně, pomocí `\em` získáme tučné šikmé písmo, aniž bychom potřebovali explicitně volat `\bs`. Podobně, pokud se příkaz `\bf` objeví v textu, který je již zvýrazněn, toto zvýraznění nepřestane.

Ve výchozím nastavení `\em` umožňuje spíše šikmou než kurzívu, ale můžeme. změnit pomocí `\setupbodyfontenvironment[default][em=italic]`.

2.5.2 Malé kapitálky a falešné malé kapitálky

Malé kapitálky jsou typografickým prostředkem, který je často mnohem lepší než použití velkých písmen. Malé kapitálky nám dávají tvar velkého písmene, ale zachovávají stejnou výšku jako malá písmena na řádku. Proto jsou malé kapitálky stylistickou variantou malých písmen. Malé kapitálky nahrazují v určitých kontextech velká písmena a jsou zvláště užitečná při psaní římských číslic nebo názvů kapitol. V akademických textech je také zvykem používat malá písmena pro psaní jmen citovaných autorů.

Problémem je, že ne všechna písmena používají malé kapitálky, a ta, která je používají, to ne vždy dělají pro některé ze svých stylů písma. Kromě toho, protože malé kapitálky jsou alternativou kurzívy, tučného písma nebo šikmého písma, v souladu s obecnými pravidly, která jsme uvedli v této kapitole, se všechny tyto typografické vlastnosti nemohly použít současně.

Tyto problémy lze vyřešit použitím *falešných malých kapitálek*, které ConTeXt umožňuje vytvořit pomocí příkazů `\cap` a `\Cap`; v tomto ohledu viz sekce ??.

2.6 Použití a konfigurace barev

ConTeXt poskytuje příkazy pro změnu barvy celého dokumentu, některých jeho prvků nebo určitých částí textu. Poskytuje také příkazy pro nahrání stovek předdefinovaných barev do paměti a pro zobrazení jejich složek.

2.6.1 Postupy pro barevnou sazbu fragmentů textu

Většina konfigurovatelných příkazů ConTeXtu umožňuje volbu nazvanou „color“, která nám umožňuje určit barvu, ve které má být zapsán příkazem ovlivněný text. Tak například pro to že názvy kapitol se mají psát modře, stačí napsat:

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

Pomocí tohoto postupu můžeme obarvit názvy, nadpisy, poznámky pod čarou, okraje, poznámky, sloupce a čáry, tabulky, názvy tabulek nebo obrázků atd. Výhodou tohoto postupu je to, že konečný výsledek bude konzistentní (všechny texty, které plní stejnou funkci, budou napsány stejným způsobem. barvou) a snadněji se globálně mění.

Část nebo fragment textu můžeme také přímo obarvit, ačkoli, abychom se vyhlí příliš pestrému použití barev, které nejsou příjemné z pohledu typografického hlediska, nebo nekonzistentnímu použití, je obecně vhodné vyhnout se přímému vybarvování a používat to, co bychom mohli nazvat *sémantické barvení*, tj. místo psaní např.

```
\color[red]{Velmi důležitý text}
```

definujeme příkaz pro velmi důležitý text, kterému je přiřazena barva. Pro například

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

2.6.2 Změna barvy pozadí a popředí dokumentu

Pokud chceme změnit barvu celého dokumentu v závislosti na tom zda chceme změnit barvu pozadí nebo barvu dokumentu, nebo zda chceme změnit barvu popředí (textu), použijeme `\setupbackgrounds` nebo `\setupcolors`. Takže například

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

Tento příkaz nastaví barvu pozadí stránek na modrou. Jako hodnota pro „backgroundcolor“ můžeme použít název některého z předdefinovaných názvů. barvy.

Globální změna barvy popředí v celém dokumentu (od od místa vložení příkazu) použijte `\setupcolors`, kde volba „textcolor“ řídí barvu textu. Například:

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

uvidíte, že barva textu je červená.

2.6.3 Příkazy pro obarvení jednotlivých textových fragmentů

Obecný příkaz pro obarvení malých částí textu je:

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

Pro větší části textu je vhodnější používat

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

Obě jsou pojmenovány podle nějaké předem definované barvy. Pokud chceme definovat barvu za běhu, můžeme použít příkaz `\colored`. Například:

```
Tři \colored[r=0.1, g=0.8, b=0.8]
{zabarvené} kočky
```

Tři **zabarvené** kočky.

2.6.4 Předdefinované barvy

ConTeXt načte nejběžnější předdefinované barvy uvedené v seznamu tabulka ??.¹

Jméno	Světlý odstín	Střední odstín	Tmavý odstín
black			
white			
gray	lightgray	middlegray	darkgray
red	lightred	middlered	darkred
green	lightgreen	midlegreen	darkgreen
blue	lightblue	middleblue	darkblue
cyan		middlecyan	darkcyan
magenta		middlemagenta	darmagenta
yellow		middleyellow	darkyellow

Tabulka 2.5 ConTeXt's predefined colours

¹ Tento seznam lze nalézt v referenční příručce a na wiki ConTeXtu, ale jsem si docela jistý, že je to neúplný seznam, protože v tomto dokumentu, aniž bychom načteli nějakou další barvu, používáme například „oranžová“ - která není v tabulce ?? - pro názvy sekcí.

Existují i další kolekce barev, které se ve výchozím nastavení nenačítají, ale které mohu načíst příkazem

`\usecolors[CollectionName]`

kde `CollectionName` může být

- „crayola“, 235 barev imitujících odstíny fixů.
- „dem“, 91 barev.
- „ema“, 540 definice barev založené na barvách používaných v systému Emacs..
- „rainbow“, 91 barvy pro použití v matematických vzorcích.
- „ral“, 213 definice barev ze *Deutsches Institut für Gütesicherung und Kennzeichnung* (Německý institut pro zajišťování kvality a označování).
- „rgb“, 223 barev.
- „solarized“, 16 barev na základě solárního schématu.
- „svg“, 147 barev.
- „x11“, 450 standardních barev pro X11.
- „xwi“, 124 barev.

Soubory s definicemi barev jsou obsaženy v „context/base/mkiv“ adresář distribuce a jeho název odpovídá schématu „colo-imp-NOMBRE.mkiv“. Informace o různých kolekcích předdefinovaných barev, které jsem právě uvedl jsou založeny na mé konkrétní distribuci. Konkrétní kolekce, resp. počet barev v nich definovaných, se mohou změnit v budoucích verzích.

Abychom zjistili, jaké barvy obsahuje každá z těchto kolekcí, můžeme použít příkaz `\showcolor[CollectionName]` příkaz popsáný v kapitole 2. v následujícím textu. Abychom mohli některé z těchto barev použít, musíme je nejprve načíst do paměti pomocí příkazu (`\usecolors[CollectionName]`) a poté musíme příkazem `\color` nebo `\startcolor` přikázat název barvy. Například následující sekvence:

```
\usecolors[xwi]
\color[darkgoldenrod]{Tweedledum a~Tweedledee}
```

napíše

Tweedledum a Tweedledee

2.6.5 Zobrazení dostupných barev

Příkaz `\showcolor` zobrazí seznam barev, ve kterých můžete vidět vzhled barvy, její vzhled, když je barva použita v stupnici šedi, červené, zelené a modré a název, pod kterým ji ConTeXt zná. Používá se bez argumentu `\showcolor` zobrazí barvy použité v aktuálním dokumentu. Ale jako argument můžeme uvést libovolnou z předdefinovaných kolekcí barev, které byly popsány v [sekcí 2.6.4](#), a takže například `\showcolor[solarized]` nám zobrazí 16 barev. solarizovaných v této kolekci:

		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

Pokud chceme zobrazit složky rgb určité barvy, můžeme použít příkaz `\showcolorcomponents[ColourName]`. To je užitečné, pokud se snažíme definovat určitou barvu, abychom viděli složení nějaké barvy, která je jí blízká. Například, `\showcolorcomponents[darkgoldenrod]` nám ukáže:

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

2.6.6 Definování vlastních barev

`\definecolor` nám umožňuje buď klonovat existující barvu, nebo definovat novou barvu. Klonování existující barvy je stejně jednoduché jako vytvoření jejího alternativního názvu. K tomu je třeba napsat:

```
\definecolor[Nová barva][Stará barva]
```

Tím se zajistí, že "Nová barva" bude mít přesně stejnou barvu jako "Stará barva".

Hlavní použití `\definecolor` však spočívá ve vytváření nových barev. Chcete-li provést je třeba příkaz použít následujícím způsobem:

```
\definecolor[ColourName][Definition]
```

kde *Definition* může být prováděna použitím až šesti různých barevných schémat:

- **RGB barvy:** Definice barev RGB je jedním z nejrozšířenější; vychází z myšlenky, že barvy je možné znázornit smícháním tří základních barev sčítáním:

červené (*r*‘ pro *červená*), zelené (*g*‘ pro *zelená*) a modré (*b*‘ pro *modrá*). Každá z těchto složek je označena jako desetinné číslo mezi 0 a 1.

```
\definecolor[lime 1][r=0.75, g=1, b=0]: Text in “lime 1”.
```

- **Hex barvy:** Tento způsob reprezentace barev je také založen na schématu RGB, ale červená, zelená a modrá složka jsou označeny jako tříbajtové hexadecimální číslo, ve kterém první bajt představuje hodnotu červené barvy, druhý hodnotu zelené barvy a třetí hodnotu modré barvy. Například:

```
\definecolor[lime 2][x=BFFF00]: Text in “lime 2”.
```

- **Barvy CMYK:** Tento model generování barev je to, co se nazývá „subtraktivní model“ a je založen na směsi pigmentů těchto barev: azurová (*c*‘), purpurová (*m*‘), žlutá (*y*‘, z *yellow*) a černá (*k*‘, z *key*). Každá z těchto složek je označena jako desetinné číslo. mezi 0 a 1:

```
\definecolor[lime 3][c=0.25, m=0, y=1, k=0]: Text in “lime 3”.
```

- **HSL/HSV:** Tento barevný model je založen na měření odstínu. (*h*‘, z *hue*), sytosti (*s*‘) a luminiscence (*l*‘ nebo někdy *v*‘, z *value*). Odstín odpovídá číslu v rozmezí 0 až 360; sytost a luminiscence musí odpovídat číslu desetinným číslem mezi 0 a 1. Například:

```
\definecolor[lime 4][h=75, s=1, v=1]: Text in “lime 4”
```

- **HWB barvy:** Model HWB je navrhovaným standardem pro CSS4. který měří odstín (*h*‘, z *hue*) a úroveň bílé (*w*‘, z *whiteness*) a černé (*b*‘, z *blackness*). Odstín odpovídá číslu v rozmezí 0 až 360, kdežto bělost a čern jsou reprezentovány desetinným číslem v rozmezí 0 až 360. a 1.

```
\definecolor[Azure][h=75, w=0.2, b=0.7] Text in “Azure”.
```

- **Stupnice šedé:** vychází z komponenty nazvané (*s*‘, z *scale*), která měří množství šedé. Musí to být číslo mezi 0 a 1. Například:

```
\definecolor[light grey][s=0.65]: Text in “light grey”.
```

It is also possible to define a new colour from another colour. For example, the colour in which titles are written in this introduction is defined as

Je také možné definovat novou barvu z jiné barvy. Například barva, ve které jsou v tomto úvodu napsány nadpisy, je definována jako

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

Chapter 3

Struktura dokumentu

Table of Contents: **3.1** Strukturální členění v dokumentech; **3.2** Typy sekcí a jejich hierarchie; **3.3** Syntaxe společná pro příkazy sekcí; **3.4** Formát a konfigurace sekcí a jejich názvů; 3.4.1 Příkazy `\setuphead` a `\setupheads`; 3.4.2 Části názvu sekce; 3.4.3 Kontrola číslování (v číslovaných sekcích); 3.4.4 Barva a styl titulku; 3.4.5 Umístění čísla a textu názvu; 3.4.6 Příkazy nebo akce; 3.4.7 Další konfigurovatelné funkce; 3.4.8 Další možnosti `\setuphead`; **3.5** Definice nových příkazů sekcí; **3.6** Makrostruktura dokumentu;

3.1 Strukturální členění v dokumentech

S výjimkou velmi krátkých textů (jako je například dopis) je dokument obvykle strukturován do bloků nebo textových skupin, které mají zpravidla hierarchické uspořádání. Neexistuje žádný standardní způsob pojmenování těchto bloků: například u románů se strukturální členění obvykle nazývá „kapitoly“, ačkoli některé - ty delší - mají větší bloky obvykle nazývané „části“, které seskupují několik kapitol dohromady. V divadelních dílech se rozlišují „aktů“ a „scén“. Akademické příručky se (někdy) dělí na „části“ a „lekce“, „témata“ nebo „kapitoly“, které zase často mají i vnitřní členění; stejný druh složitěho hierarchického členění se často vyskytuje i v jiných akademických nebo technických dokumentech (například v textech, jako je tento, věnovaný vysvětlení počítačového programu nebo systému. Dokonce i zákony jsou strukturovány do „knihy“ (nejdelší a nejsložitější, jako jsou zákoníky), „nápisy“, „kapitoly“, „oddíly“, „pododdíly“. Vědecké a technické dokumenty mohou také dosahovat až šesti, sedmi nebo někdy dokonce osmiúrovňové hloubky vnoření těchto druhů členění.

Tato kapitola se zaměřuje na analýzu mechanismu, který ConTeXt nabízí pro podporu těchto strukturálních dělení. Budu je označovat souhrnným termínem „sekc“.

Neexistuje jasný termín, který by umožňoval obecně označit všechny tyto druhy strukturálního dělení. Termín „oddíl“, který jsem zvolil, se zaměřuje spíše na strukturální dělení než na cokoli jiného, i když nevýhodou je, že jedno z předem určených strukturálních dělení ConTeXtu se nazývá „oddíl“. Doufám, že to nezpůsobí zmatek, a věřím, že z kontextu bude dostatečně snadné určit, zda mluvíme o sekci jako o obecném a celkovém odkazu na strukturální dělení, nebo o konkrétním dělení, které ConTeXt nazývá sekci.

Každá „oddíl“ (obecně řečeno) znamená:

- Přiměřeně rozsáhlé *strukturní členění dokumentu*, které může zahrnovat další členění nižší úrovně. Z tohoto pohledu „oddíly“ znamenají textové bloky s hierarchickým vztahem mezi nimi. Na dokument jako celek lze z hlediska jeho oddílů nahlížet jako na strom. Dokument *jako takový* je kmenem, každá jeho kapitola je větví, která zase může mít větve, které se mohou také dělit atd.

Jasná struktura je velmi důležitá pro čtení a pochopení dokumentu. Tento úkol je však na autorovi, nikoli na sazeči. A přestože není úkolem ConTeXt aby z nás udělal lepší autory, než jsme, celá řada příkazů sekcí, které obsahuje a kde je hierarchie mezi nimi zcela jasná, by nám mohla pomoci psát lépe strukturované dokumenty.

- A *structure name*, který bychom mohli nazvat jeho „title“. nebo „label“. Toto jméno struktury se vypíše:
 - Vždy (nebo téměř vždy) v místě dokumentu, kde začíná strukturní členění.
 - Někdy také v obsahu, v záhlaví nebo zápatí stránek, na kterých se nachází daná část.

ConTeXt nám umožňuje automatizovat všechny tyto úkoly tak, že stačí pouze jednou uvést, jakými formátovacími prvky má být název strukturní jednotky vytištěn a zda má či nemá být uveden také v obsahu nebo v záhlaví či zápatí. K tomu ConTeXt potřebuje pouze vědět, kde každá strukturní jednotka začíná a končí, jak se jmenuje a na jaké hierarchické úrovni se nachází.

3.2 Typy sekcí a jejich hierarchie

ConTeXt rozlišuje mezi *číslovanými* a *nečíslovanými* sekcemi. První z nich, jak už název napovídá, jsou číslovány automaticky a posílají se do obsahu a někdy také do záhlaví a/nebo zápatí stránek.

ConTeXt má hierarchicky uspořádané předdefinované příkazy sekcí, které se nacházejí v [tabulka 3.1](#).

Pokud jde o předdefinované oddíly, je třeba uvést následující upřesnění:

- V [tabulka 3.1](#) jsou příkazy sekcí zobrazeny v tradiční podobě. Hned ale uvidíme, že je lze použít také jako *environments* (`\startchapter ... \stopchapter`) a že právě tento přístup se doporučuje.
- Tabulka obsahuje pouze prvních 6 úrovní sekcí. Při testech jsem však našel až 12 úrovní: Po `\subsubsubsection` následuje `\subsubsubsubsection` a tak

Level	Numbered sections	Unnumbered sections
1	<code>\part</code>	–
2	<code>\chapter</code>	<code>\title</code>
3	<code>\section</code>	<code>\subject</code>
4	<code>\subsection</code>	<code>\subsubsubject</code>
5	<code>\subsubsection</code>	<code>\subsubsubject</code>
6	<code>\subsubsubsection</code>	<code>\subsubsubsubject</code>
...

Tabulka 3.1 Příkazy sekcí v ConT_EXt

dále až k `\subsubsubsubsubsubsubsubsubsection`, nebo `\subsubsubsubsubsubsubsubsubject`.

Měli bychom však mít na paměti, že výše uvedené (příliš hluboké) nižší úrovně sotva mohou zlepšit porozumění textu! Především je pravděpodobné, že budeme mít rozsáhlé oddíly, které se budou nevyhnutelně zabývat několika záležitostmi, a to čtenáři ztíží *grasp* jejich obsahu. Přílišná hloubka úrovní může také znamenat, že čtenář ztratí celkový smysl textu, a výsledkem bude přílišná roztříštěnost příslušné látky. Mám za to, že obecně stačí čtyři úrovně; velmi výjimečně může být potřeba jít do šesti nebo sedmi úrovní, ale větší hloubka by byla dobrým nápadem jen zřídka.

Z hlediska psaní zdrojového souboru může skutečnost, že vytvoření dalších podúrovní znamená přidání dalšího „sub“ k předchozí úrovni, způsobit, že zdrojový soubor je téměř nečitelný: není legrace snažit se zjistit úroveň příkazu s názvem „subsubsubsubsubsection“, protože musím spočítat všechny „subs“! Takže radím, že pokud opravdu potřebujeme tolik úrovní hloubky, bylo by lepší od páté úrovně (subsubsection) definovat vlastní příkazy sekcí (viz [section 3.5](#)) a dát jim srozumitelnější názvy než předdefinované.

- Nejvyšší úroveň sekce (`\part`) existuje pouze pro číslované názvy a má tu zvláštnost, že název části není vytisknuto. I když se však název netiskne, je zavedena prázdná stránka (na které můžeme předpokládat, že se název vytiskne, jakmile uživatel příkaz překonfiguruje) a číslování *part* je zohledněno při výpočtu číslování kapitol a dalších oddílů.

Důvodem, proč výchozí verze `\part` nic netiskne, je podle wiki ConT_EXtto, že téměř vždy nadpis na této úrovni vyžaduje specifický layout; a i když je to pravda, nezdá se mi to jako dostatečně dobrý důvod, protože v praxi se často předefinovávají i kapitoly a sekce a skutečnost, že části nic netisknou, nutí začínajícího uživatele, aby se *dip* do dokumentace a zjistil, co je špatně.

- Ačkoli první úroveň členění je „part“, je pouze teoretická a abstraktní. V konkrétním dokumentu bude první úroveň členění ta, která odpovídá prvnímu příkazu členění v dokumentu. To znamená, že v dokumentu, který neobsahuje části, ale kapitoly, bude první úroveň kapitola. Pokud však dokument neobsahuje ani kapitoly, ale pouze oddíly, bude hierarchie pro tento dokument začínat oddíly.

3.3 Syntaxe společná pro příkazy sekcí

Všechny příkazy sekcí, včetně všech úrovní vytvořených uživatelem (viz [sekce 3.5](#)), umožňují následující alternativní formy syntaxe (pokud například používáme úroveň „section“):

```
\section [Label] {Title}
\section [Options]
\startsection [Options] [Variables] ... \stopsection
```

Ve třech výše uvedených způsobech jsou argumenty v hranatých závorkách nepovinné a lze je vynechat. Budeme se jim věnovat samostatně, ale nejprve je vhodné objasnit, že ve značce IV je doporučován právě třetí z těchto tří přístupů.

- V první syntaktické podobě, kterou bychom mohli nazvat „classic“, příkaz přijímá dva argumenty, jeden nepovinný v hranatých závorkách a druhý povinný v závorkách. Nepovinný argument slouží k přiřazení příkazu ke značce, která bude použita pro vnitřní odkazy (viz [sekce 5.2](#)). Povinným argumentem mezi kulatými závorkami je název sekce.
- Další dvě formy syntaxe jsou spíše ve stylu ConT_EXt vše, co příkaz potřebuje vědět, je sděleno prostřednictvím hodnot a možností zavedených mezi hranaté závorky.

Připomeňme si, že v [sekcích 1.3.1 a 1.4](#) jsem uvedl, že v ConT_EXt je rozsah příkazu uveden v kulatých závorkách a jeho volby v hranatých závorkách. Ale když se nad tím zamyslíme, název konkrétního sekčního příkazu není rozsahem jeho použití, takže aby byl v souladu s obecnou syntaxí, neměl by být uveden mezi kudrnatými závorkami, ale jako volba. ConT_EXt umožňuje tuto výjimku, protože se jedná o klasický způsob práce v T_EXu, ale poskytuje alternativní formy syntaxe, které jsou více v souladu s jeho celkovým návrhem.

Možnosti jsou typu přiřazení hodnoty (OptionName=Value) a jsou následující:

- **reference**: Označení pro křížové odkazy.
- **title**: Název sekce, který bude vytištěn v těle dokumentu.
- **list**: Název oddílu, který bude vytištěn v obsahu.
- **marking**: Název oddílu, který se má tisknout v záhlaví nebo zápatí stránky.
- **bookmark**: Název oddílu, který má být převeden na záložku *bookmark* v souboru PDF.
- **ownnumber**: V tomto případě bude tato volba obsahovat číslo přidělené danému oddílu.

Volby „list“, „marking“ a „bookmark“ by se samozřejmě měly používat pouze tehdy, pokud chceme použít jiný název, který nahradí hlavní název nastavený pomocí „title“. možnost. To je velmi užitečné například v případě, kdy je nadpis pro záhlaví příliš dlouhý; ačkoli k tomu můžeme použít také příkazy `\nomarking` a `\nolist` (něco velmi podobného). Na druhou stranu musíme mít na paměti, že pokud text nadpisu (volba „title“) obsahuje nějaké čárky, bude třeba jej uzavřít do kudrnatých závorek, a to jak celý text, tak čárku, aby ConTeXt věděl, že čárka je součástí nadpisu. Totéž platí pro volby: „list“, „marking“ a „bookmark“. Proto, abychom nemuseli hlídat, zda jsou v názvu nějaké čárky, je podle mě dobré si zvyknout hodnotu některé z těchto možností vždy uzavírat do kudrnatých závorek.

Následující řádky tedy například vytvoří kapitolu s názvem „A Test Chapter“ spojenou se štítkem „test“ pro křížové odkazy, přičemž záhlaví bude „Chapter test“ namísto „A Test Chapter“.

```
\chapter
[
    title={A Test Chapter},
    reference={test},
    marking={Chapter test}
]
```

Syntaxe `\startSectionType` změní sekci na *environment*. Je to více v souladu se skutečností, že, jak jsem řekl na začátku, na pozadí je každá sekce odlišným blokem textu, ačkoli ConTeXtve výchozím nastavení nepovažuje *environments* generované příkazy sekcí za *groups*. Stejně tak tento postup doporučuje Mark IV; dost možná proto, že tento způsob vytváření sekcí vyžaduje, abychom výslovně uvedli, kde každá sekce začíná a končí, což usnadňuje konzistentní strukturu a pravděpodobně má lepší podporu pro výstupy XML a EPUB. Pro výstup XML je to vlastně nezbytné.

Při použití `\startSectionName` je povoleno použít jednu nebo více proměnných jako argumenty mezi hranatými závorkami. Jejich hodnotu pak můžeme později použít na jiných místech dokumentu pomocí příkazu `\structureuservariable`.

Uživatelské proměnné umožňují velmi pokročilé použití ConTeXt díky tomu, že v závislosti na hodnotě konkrétní proměnné lze rozhodovat o tom, zda fragment zkompileovat, nebo ne, případně jakým způsobem, nebo s jakou šablonou. Tyto nástroje ConTeXtu však přesahují rozsah materiálu, kterým se chci v tomto úvodu zabývat.

3.4 Formát a konfigurace sekcí a jejich názvů

3.4.1 Příkazy `\setuphead` a `\setupheads`

Ve výchozím nastavení přiřazuje ConTeXt každé úrovni řezu určité vlastnosti, které ovlivňují především (ale nejen) formát, v jakém je nadpis zobrazen v hlavním textu dokumentu, ale ne způsob, jakým je nadpis zobrazen v obsahu nebo v záhlaví a zápatí. Tyto vlastnosti můžeme změnit pomocí příkazu `\setuphead`, jehož syntaxe je:

`\setuphead[Sections][Options]`

kde

- **Sections** odkazuje na název jedné nebo více sekcí (oddělených čárkami), které mají být příkazem ovlivněny. Může to být:
 - Kterákoli z předdefinovaných sekcí (část, kapitola, nadpis atd.), v tomto případě na ně můžeme odkazovat buď podle názvu, nebo podle úrovně. Pro odkaz na ně podle jejich úrovně použijeme slovo „*sekce-NumLevel*“, kde *NumLevel* je číslo úrovně příslušné sekce. Takže „*section-1*“ se rovná „*part*“, „*section-2*“ se rovná „*chapter*“ atd.
 - Jakýkoli druh sekce, který jsme sami definovali. V tomto ohledu viz [sekce 3.5](#).
- **Options** jsou možnosti konfigurace. Jsou typu explicitního přiřazení hodnoty (`OptionName=value`). Počet způsobilých voleb je velmi vysoký (přes šedesát), a proto je vysvětlím rozdělením do kategorií podle jejich funkce. Musím však upozornit, že se mi nepodařilo zjistit, k čemu některé z těchto voleb slouží a jak se používají. O těchto možnostech tedy nebudu hovořit.

Dříve jsem uvedl, že `\setuphead` ovlivňuje sekce, které jsou výslovně uvedeny. To však neznamená, že by úprava určité sekce neměla nijak ovlivnit ostatní sekce, pokud nebyly v příkazu výslovně uvedeny. Ve skutečnosti je tomu právě naopak: úprava určité sekce ovlivňuje ostatní sekce *kteřé jsou s ní propojeny*, i když to v příkazu nebylo výslovně uvedeno. vazba mezi jednotlivými sekcemi je dvojího druhu:

- Nečíslované příkazy jsou propojeny s odpovídajícím číslovaným příkazem stejné úrovně, takže změna vzhledu číslovaného příkazu ovlivní nečíslovaný příkaz

stejně úrovně, ale ne naopak: změna nečíslovaného příkazu nemá vliv na číslovaný příkaz. To například znamená, že pokud změníme nějaký aspekt příkazu „chapter“. (úroveň 2), změníme tento aspekt také v „title“; ale změna „title“ neovlivní „chapter“.

- Příkazy jsou hierarchicky propojeny, takže pokud změníme *určité vlastnosti* v určité úrovni, změna ovlivní všechny úrovně, které následují po ní. K tomu dochází pouze u některých funkcí. Například barva: pokud stanovíme, že podsekce se budou zobrazovat červeně, změníme na červenou i podsekce, podsekce atd. Totéž se však neděje u jiných funkcí, jako je například styl písma.

Spolu s `\setuphead` ConTeXt poskytuje příkaz `\setupheads`, který globálně ovlivňuje všechny příkazy sekcí. Na wiki ConTeXt je v souvislosti s tímto příkazem uvedeno, že podle některých lidí nefunguje. Podle mých testů tento příkaz pro některé možnosti funguje, ale pro jiné ne. Zejména nefunguje s volbou „style“, což je zarážející, protože styl nadpisů je s největší pravděpodobností jediná věc, kterou bychom chtěli změnit globálně, aby ovlivnila všechny nadpisy. Podle mých testů však funguje s jinými možnostmi, jako je například „number“ nebo „color“. Takže například `\setupheads[color=blue]` zajistí, že všechny nadpisy v našem dokumentu budou vytisknuty modře.

Protože jsem trochu líný na to, abych se obtěžoval testovat každou možnost, jestli funguje nebo ne s `\setupheads` (nezapomeňte, že jich je více než šedesát), v následujícím textu budu odkazovat pouze na `\setuphead`.

Nakonec: než se budeme zabývat konkrétními možnostmi, měli bychom si všimnout něčeho, co je uvedeno ve wiki ConTeXt i když to pravděpodobně není uvedeno na správném místě: některé možnosti fungují pouze tehdy, pokud používáme syntaxi `\startSectionName`.

Tyto informace jsou obsaženy v souvislosti s `\setupheads`, ale ne v `\setuphead`, kde je vysvětlena většina možností a kde se zdá, že je to nejrozumnější místo, pokud to má být řečeno pouze na jednom místě. Na druhou stranu se informace zmiňuje pouze o volbě „insidesection“, aniž by bylo jasné, zda se tak děje i u jiných voleb.

3.4.2 Části názvu sekce

Než se pustíme do konkrétních možností, které nám umožňují nastavit vzhled nadpisů, je vhodné začít upozorněním, že nadpis oddílu může mít až tři různé části, které ConTeXt umožňuje formátovat společně nebo odděleně. Tyto prvky nadpisu jsou následující:

- **Samotný název**, tedy text, ze kterého se skládá. V zásadě se tento nadpis zobrazuje vždy, s výjimkou sekcí typu „part“, kde se nadpis standardně nezobrazuje. Volbou, která určuje, zda se nadpis zobrazí, nebo ne, je „placehead“,



jejíž hodnoty mohou být „yes“, „no“, „hidden“, „empty“ nebo „section“. Význam prvních dvou je jasný. Nejsem si však jistý výsledkem zbývajících hodnot této možnosti.

Pokud tedy chceme, aby se nadpis zobrazoval v sekcích první úrovně, mělo by být naše nastavení následující:

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

Jak již víme, názvy některých oddílů lze automaticky odesílat do záhlaví a obsahu. Pomocí voleb `list` a `marking` příkazů sekcí můžeme uvést alternativní název, který se má posílat místo něj. Při psaní nadpisu je také možné použít příkazy `\nolist` nebo `\nomarking`, aby byly určité části nadpisu v obsahu nebo v záhlaví nahrazeny elipsami. Například:

```
\chapter{Vlivy \nomarking{19. století} impresionismu \nomarking{v 21.
století}}
```

Do záhlaví napíše „Vlivy ... impresionismu ...“.

- **Číslování.** To platí pouze pro číslované oddíly (part, chapter, section, sub-section...), nikoliv pro nečíslované (title, subject, subsubject). Ve skutečnosti to, zda je určitý oddíl číslovaný nebo ne, závisí na volbách „number“ a „incrementnumber“, jejichž možné hodnoty jsou „yes“ a „no“. V číslovaných sekcích jsou obě nastaveny jako `yes` a v nečíslovaných sekcích jako `no`.

Proč existují dvě možnosti ovládání stejné věci? Protože ve skutečnosti tyto dvě možnosti řídí dvě věci: jedna je, zda je sekce číslovaná, nebo ne (`incrementnumber`), a druhá, zda se číslo zobrazí, nebo ne (`number`). Pokud je pro sekci nastaveno `incrementnumber=yes` a `number=no`, získáme sekci, která je navenek (vizuálně) nečíslovaná, ale uvnitř stále počítaná. To by bylo užitečné pro zařazení takového oddílu do obsahu, protože obvykle by obsahoval pouze číslované oddíly. V tomto ohledu viz sekce ?? v [sekce 4.1.7](#).

- **Štítek** pro název. V zásadě je tento prvek v názvech prázdný. Můžeme mu však přiřadit hodnotu a v takovém případě se před číslem a vlastním názvem vypíše štítek, který jsme této úrovni přiřadili. Například v názvech kapitol můžeme chtít, aby se vytisklo slovo „Chapter“ nebo slovo „Part“ pro části. K tomu nepoužíváme příkaz `\setuphead`, ale příkaz `\setuplabeltext`, který nám umožňuje přiřadit textovou hodnotu štítkům různých úrovní členění. Chceme-li tedy například v našem dokumentu před nadpisy kapitol napsat „Chapter“, měli bychom nastavit:

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

V příkladu jsem za přiřazený název vložil vyhrazený znak „~“, který za slovo vloží nezlomitelnou mezeru. Pokud nám nevadí, že mezi označením a číslem dojde ke zlomu řádku, můžeme jednoduše přidat prázdnou mezeru. Tato mezera (ať už jakéhokoli druhu) je však důležitá; bez ní bude číslo spojeno se štítkem a místo „Chapter 1“ bychom viděli například „Chapter 1“.

3.4.3 Kontrola číslování (v číslovaných sekcích)

Již víme, že předdefinované číslované oddíly (část, kapitola, oddíl...) a to, zda je určitý oddíl číslován či nikoli, závisí na volbách „number“ a „incrementnumber“ nastavených pomocí `\setuphead`.

Ve výchozím nastavení je číslování různých úrovní automatické, pokud jsme volbě „yes“ nepřidělili hodnotu „ownnumber“. Pokud „ownnumber=yes“, musí být uvedeno číslo přiřazené každému příkazu. To se provádí:

- Pokud je příkaz vyvolán pomocí klasické syntaxe, přidáním argumentu s číslem před text nadpisu. Například:
`\kapitola{13}{Název kapitoly}` vytvoří kapitolu, které bylo ručně přiřazeno číslo 13.
- Pokud byl příkaz vyvolán se syntaxí specifickou pro ConTeXt (`\Section Type [Options]` nebo `\startSection Type [Options]`), s volbou „ownnumber“. Například:
`\chapter[title={Chapter title}, ownnumber=13]`, vytvoří kapitolu, které bylo ručně přiřazeno číslo 13.

Když ConTeXt automaticky čísluje, používá vnitřní počítadla, která ukládají čísla různých úrovní; existuje tedy počítadlo pro části, další pro kapitoly, další pro oddíly atd. Pokaždé, když ConTeXt najde příkaz sekce, provede následující akce:

- Zvýší čítač přiřazený úrovni odpovídající tomuto příkazu o 1.
- Vynuluje přidružené čítače na všech úrovních pod úrovní daného příkazu na 0.

To například znamená, že při každém nalezení nové kapitoly se čítač kapitol zvýší o 1 a všechny příkazy sekce, podsekce, podsekce atd. se vrátí na hodnotu 0; čítač částí však není ovlivněn.

Chcete-li změnit číslo, od kterého se má začít počítat, použijte příkaz `\setupheadnumber` takto:

`\setupheadnumber[SectionType][Number from which to count]`

kde *Number from which to count* je číslo, od kterého se počítají oddíly jakéhokoli typu. Pokud je tedy *Number from which to count* rovno nule, bude první sekce 1; pokud je rovno 10, bude první sekce 11.

Tento příkaz nám také umožňuje změnit vzor pro automatický přírůstek, takže můžeme například dosáhnout toho, aby se kapitoly nebo oddíly počítaly po dvou nebo po třech. Takže `\setupheadnumber[section][+5]` zobrazí kapitoly číslované jako 5 z pěti; a `\setupheadnumber[chapter][14, +5]` zobrazí, že první kapitola začíná číslem 15 (14+1), druhá bude 20 (15+5), třetí 25 atd.

Ve výchozím nastavení se číslování oddílů zobrazuje arabskými číslicemi a je zahrnuto číslování všech předchozích úrovní. To znamená: v dokumentu, v němž existují části, kapitoly, oddíly a pododdíly, bude konkrétní pododdíl označovat, které části, kapitole a oddílu odpovídá. Čtvrtý pododdíl druhého oddílu třetí kapitoly první části tedy bude „1.3.2.4“.

Dvě základní možnosti, které určují způsob zobrazení čísel, jsou:

- **převod:** Řídí typ číslování, které bude použito. Umožňuje řadu hodnot v závislosti na požadovaném typu číslování:
 - **Číslování arabskými číslicemi:** Klasické číslování: 1, 2, 3, ... se získá pomocí hodnot `n`, `N` nebo `numbers`.
 - **Číslování římskými číslicemi.** Tři způsoby, jak to udělat:
 - ★ Velká římská čísla: `I`, `R`, `Romannumerals`.
 - ★ Malá římská čísla: `i`, `r`, `Romannumerals`.
 - ★ Římská čísla malými písmeny: `KR`, `RK`.
 - **Číslování pomocí písmen.** Tři způsoby, jak to udělat:
 - ★ Velká písmena: `A`, `Character`
 - ★ Malá písmena: `a`, `character`
 - ★ Malá písmena: `AK`, `KA`
 - **Číslování slovy.** To znamená, že píšeme slovo, které označuje číslo. Tak například z „3“ se stane „Three“. Existují dva způsoby, jak to udělat:
 - ★ Slova začínající velkým písmenem: `Words`.
 - ★ Slova psaná malými písmeny: `word`.
 - **Číslování pomocí symbolů:** Číslování pomocí symbolů používá různé sady symbolů, v nichž je každému symbolu přiřazena číselná hodnota.

Vzhledem k tomu, že sady symbolů používané ConTeXtem mají velmi omezený počet, je vhodné tento typ číslování používat pouze tehdy, pokud maximální počet, kterého lze dosáhnout, není příliš vysoký. ConTeXt poskytuje čtyři různé sady symbolů: **set 0**, **set 1**, **set 2** a **set 3**. Níže jsou uvedeny symboly, které každá z těchto sad používá pro číslování. Všimněte si, že maximální počet, kterého lze dosáhnout, je 9 v **set 0** a **set 1** a 12 v **set 2** a **set 3**:

Set 0:	•	–	★	▷	◦	○	◯	□	✓			
Set 1:	★	★★	★★★	‡	‡‡	‡‡‡	*	**	***			
Set 2:	1	2	3	4	5	6	7	8	9	10	11	12
Set 3:	1	2	3	4	5	6	7	8	9	10	11	12

- **sectionsegments:** Tato volba nám umožňuje určit, zda se má zobrazit číslování předchozích úrovní. Můžeme určit, které předchozí úrovně budou zobrazeny. To se provádí určením počáteční a poslední úrovně, která se má zobrazit. Identifikace úrovně může být provedena jejím číslem (část=1, kapitola=2, sekce=3 atd.) nebo názvem (část, kapitola, sekce atd.). Tak například „**sectionsegments=2:3**“ označuje, že se má zobrazit číslování kapitol a sekcí. Je to úplně stejné, jako když řeknete „**sectionsegments=chapter:section**“. Chceme-li naznačit, že se mají zobrazit všechna čísla nad určitou úrovní, můžeme jako hodnotu „**optionsegments**“ použít příkaz *InitialLevel:** nebo *InitialLevel:**. Například „**sectionsegments=3:***“ znamená, že číslování se zobrazí od úrovně 3 (sekce).

Představte si tedy například, že chceme, aby části byly číslovány římskými číslicemi s velkými písmeny; kapitoly arabskými číslicemi, ale bez uvedení čísla části, ke které patří; oddíly a pododdíly arabskými číslicemi včetně čísel kapitol a oddílů a pododdílů velkými písmeny. Měli bychom psát následující:

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

3.4.4 Barva a styl titulku

K dispozici jsou následující možnosti ovládání stylu a barvy:

- **Styl** se ovládá pomocí voleb „**style**“, „**numberstyle**“ a „**textstyle**“ podle toho, zda chceme ovlivnit celý nadpis, pouze číslování nebo pouze text. Pomocí kterékoli z těchto možností můžeme zahrnout příkazy, které ovlivňují

písmo, konkrétně: konkrétní písmo, styl (roman, sans serif nebo typewriter), alternativu (kurzíva, tučné písmo, šikmé písmo...) a velikost. Pokud chceme uvést pouze jednu vlastnost stylu, můžeme to udělat buď pomocí názvu stylu (například „bold“ pro tučné písmo), nebo uvedením jeho zkratky („bf“), nebo příkazu, který jej generuje (`\bf`, v případě tučného písma). Chceme-li označit několik znaků současně, musíme to udělat pomocí příkazů, které je generují, a zapsat je jeden po druhém. Na druhou stranu mějte na paměti, že pokud označíme pouze jeden rys, ostatní rysy stylu se automaticky vytvoří s výchozími hodnotami dokumentu, a proto se málokdy doporučuje vytvořit pouze jeden rys stylu.

- **Barva** se nastavuje pomocí možností „color“, „numbercolor“ a „textcolor“ podle toho, zda chceme nastavit barvu celého nadpisu, nebo jen barvu číslování či textu. Zde uvedená barva může být jednou z předdefinovaných barev ConTeXtu nebo jinou barvou, kterou jsme sami definovali a které jsme předtím přiřadili jméno. Nemůžeme zde však přímo použít příkaz pro definici barvy.

Kromě těchto šesti možností je k dispozici ještě pět dalších možností, které umožňují vytvořit některé sofistikovanější funkce, s nimiž můžeme dělat prakticky cokoli. Jedná se o tyto možnosti: Začněme vysvětlením prvních tří možností: „command“, „numbercommand“, „textcommand“, „deepnumbercommand“ a „deeptextcommand“:

- **command** označuje příkaz, který bude mít dva argumenty, číslo a název sekce. Může to být běžný ConTeXt příkaz nebo příkaz, který jsme sami definovali.
- **numbercommand** je podobný příkazu „command“, ale tento příkaz přijímá pouze argument s číslem sekce.
- **textcommand** je také podobný „command“, ale přijímá pouze argument s textem názvu.

Tyto tři možnosti nám umožňují dělat prakticky cokoli, co chceme. Pokud například chci, aby byly oddíly zarovnané doprava, uzavřeny v rámečku a mezi číslem a textem byl zalomení řádku, mohu jednoduše vytvořit příkaz, který to udělá, a pak tento příkaz uvést jako hodnotu příkazu „command“. Toho by se dosáhlo pomocí následujících řádků:

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

\setuphead
[section]
[command=\AlignSection]
```

Pokud současně nastavíme volby „command“ a „style“, použije se příkaz na nadpis i s jeho stylem. To například znamená, že pokud jsme nastavili „textstyle=\em“

a „`textcommand=\WORD`“, příkaz `\WORD` (který text, který bere jako argument, píše velkými písmeny), bude na nadpis aplikován jeho styl, tj.: `\WORD{\em Title text}`. Pokud chceme, aby to bylo naopak, tj. že se styl aplikuje na obsah nadpisu po aplikaci příkazu, měli bychom místo možností „`textcommand`“ a „`numbercommand`“ použít možnosti „`deeptextcommand`“ a „`deepnumbercommand`“. To by ve výše uvedeném příkladu vygenerovalo „`{\em\WORD{Title text}}`“.

Ve většině případů by nebyl rozdíl v tom, zda to uděláte tak či onak. V některých případech však může být.

3.4.5 Umístění čísla a textu názvu

Volba „`alternative`“ řídí dvě věci současně: umístění číslování vzhledem k textu nadpisu a umístění samotného nadpisu (včetně čísla a textu) vzhledem ke stránce, na které je zobrazen, a obsahu oddílu. Jedná se o dvě různé věci, ale protože se obě řídí stejnou volbou, jsou řízeny současně.

Umístění nadpisu ve vztahu ke stránce a prvnímu odstavci obsahu sekce je řízeno následujícími možnými hodnotami „`alternative`“:

- **text**: Název oddílu je integrován s prvním odstavcem jeho obsahu. Efekt je podobný tomu, který se vytváří v L^AT_EXu pomocí `\paragraph` a `\subparagraph`.
- **odstavec**: Název oddílu bude samostatným odstavcem.
- **normal**: Název sekce bude umístěn na výchozí místo, které ConT_EXt poskytuje pro daný typ sekce. Obvykle je to „`paragraph`“.
- **middle**: Nadpis se píše jako samostatný, vycentrovaný odstavec. Pokud se jedná o číslovaný příkaz, číslo a text jsou odděleny na různých řádcích a oba jsou vycentrovány.

Podobného efektu jako při použití „`alternative=middle`“ se dosáhne při použití volby „`align`“, která řídí zarovnání nadpisů. Může nabývat hodnot „`left`“, „`middle`“ nebo „`flushright`“. Pokud však nadpis pomocí této volby vycentrujeme, číslo a text se zobrazí na stejném řádku.

- **margintext**: Tato volba způsobí, že se celý nadpis (číslování a text) vytiskne v prostoru vyhrazeném pro okraj.

Umístění čísla ve vztahu k textu nadpisu udávají následující možné hodnoty „`alternative`“:

- **margin/inmargin:** Název je samostatný odstavec. Číslování se zapisuje do prostoru vyhrazeného pro okraj. Nepřišel jsem na to, jaký je rozdíl mezi použitím „margin“ a „inmargin“.
- **reverse:** Nadpis tvoří samostatný odstavec, ale běžné pořadí je obrácené a nejprve je vtištěn text a pak číslo.
- **top/bottom:** U nadpisů, jejichž text zabírá více než jeden řádek, tyto dvě volby určují, zda bude číslování zarovnáno s prvním řádkem nadpisu nebo s posledním řádkem.

3.4.6 Příkazy nebo akce

Je možné uvést jeden nebo více příkazů, které se provedou předtím. (možnost „before“) nebo po (možnost „after“). Tyto volby se hojně používají k vizuálnímu označení nadpisu. „before=\blank“ přidá prázdné místo. Chceme-li přidat ještě více místa, můžeme napsat „before={\blank[3*big]}“. V tomto případě jsme obklopili hodnotu volby kroucenými závorkami, abychom se vyhnuli chybě. Mohli bychom také vizuálně označit vzdálenost mezi předchozím a následujícím textem pomocí „before=\hairline, after=\hairline“, což by nakreslilo vodorovnou čáru před a za nadpisem.



Velmi podobné volbám „before“ a „after“ jsou volby „before“ a „after“. „commandbefore“ a „commandafter“. Podle mých testů usuzuji, že rozdíl spočívá v tom, že první z nich dvě provádějí akce před a po zahájení psaní nadpisu jako zatímco druhé dva se vztahují k příkazům, které budou provedeny před a po zadání *textu nadpisu*.

Pokud chceme před nadpis vložit zlom stránky, musíme použít příkaz „page“, která umožňuje kromě jiných hodnot i „ano“ pro vložení zalomení stránky, „vlevo“ pro vložení tolika zalomení stránky, kolik je potřeba, aby se zajistilo, aby nadpis začínal na sudé stránce, „vpravo“ pro zajištění toho, aby nadpis název začíná na liché stránce, nebo „no“, pokud chceme zakázat zalamování stránek. nucený zlom stránky. Tato volba naopak pro úroveň nižší než „chapter“, bude fungovat pouze v případě, že použijeme „continue=no“, jinak nebude fungovat, pokud je oddíl, pododdíl nebo příkaz na adrese první stránce kapitoly.

Ve výchozím nastavení začínají kapitoly v ConTeXtu na nové stránce. Pokud je že oddíly začínají také na nové stránce, je problém v tom. vzniká problém, co udělat s první sekci kapitoly, která, je třeba na začátku kapitoly: pokud je tato sekce také na začátku kapitoly, je třeba ji zařadit na první stránku. začíná stránkový zlom, skončíme se stránkou, která otevírá kapitolu. obsahuje pouze název kapitoly, což není příliš estetické. Proto můžeme nastavit volbu „continue“, název, musím říci, že mi není příliš jasný: pokud „continue=yes“, bude stránka se nebude vztahovat na oddíly, které jsou na první stránce kapitoly. Pokud „continue=no“, zlom stránky se použije i tak.

Pokud místo příkazů sekcí použijeme prostředí sekcí (`\start ... \stop`), máme k dispozici také volbu „`insidesection`“, pomocí které můžeme označit jeden nebo více příkazů, které budou provedeny jednou. nadpis byl napsán a my jsme již uvnitř sekce. Tato stránka nám například umožní zajistit, aby se ihned po zadání příkazu začátku kapitoly bude automaticky napsán obsah. with („`insidesection=\placecontent`“)

3.4.7 Další konfigurovatelné funkce

Kromě těch, které jsme již viděli, můžeme nakonfigurovat následující funkce další funkce pomocí `\setuphead`:

- **Meziřádky.** Ovládá se pomocí „`interlinespace`“, který přebírá jako svou hodnotu název dříve vytvořeného příkazu `interlinea` pomocí `\defineinterlinespace` a nakonfigurovaného příkazem `\nastavení meziprostoru`.
- **Zarovnání.** Volba „`align`“ ovlivňuje zarovnání odstavce obsahujícího nadpis. Mimo jiné může mít hodnotu následující hodnoty: „`flushleft`“. (vlevo), „`flushright`“. (vpravo), „`middle`“ (centred), „`inner`“ (inner margin) a „`outer`“ (outer margin).
- **Okraj.** Pomocí možnosti „`margin`“ můžeme ručně nastavit. okraj nadpisu.
- **Odsazení prvního odstavce.** Hodnota „`indentnext`“ (která může být "ano", "ne" nebo "auto") řídí zda první řádek prvního odstavce oddílu bude nebo nebude odsazen. bude odsazen. Zda má být odsazen, nebo ne (v dokumentu kde je první řádek odstavců obecně odsazen) je to otázka otázkou vkusu.
- **Šířka.** Ve výchozím nastavení zabírají nadpisy potřebnou šířku pokud tato šířka není větší než šířka řádku, v takovém případě je šířka nadpis zabere více než jeden řádek. Ale s „`width`“ můžeme nadpisu přiřadit konkrétní šířku. Příkaz „`numberwidth`“ a „`textwidth`“ přiřadíme v tomto pořadí šířku číslování nebo šířku textu nadpisu.
- **Oddělení čísla a textu.** Příkazy „`distance`“. „`textdistance`“ nám umožňují řídit vzdálenost oddělující číslo od jeho textu.
- Styl záhlaví a zápatí sekcí. K tomu používáme volbu „`header`“ a „`footer`“.

3.4.8 Další možnosti `\setuphead`

S možnostmi, které jsme již viděli, můžeme vidět, že konfigurace jsou možnosti konfigurace nadpisů sekcí téměř neomezené. Nicméně, `\setuphead` má asi třicet možností, které jsem jsem nezmínil. Většinou proto, že jsem nezjistil, k čemu slouží a jak. se používají, několik proto, že jejich vysvětlení by mě nutilo jít do podrobností. aspekty, kterými se v tomto úvodu nehodlám zabývat.



3.5 Definice nových příkazů sekcí

Vlastní příkazy sekcí můžeme definovat pomocí `\definehead`, jehož syntaxe je:

`\definehead[CommandName][Model][Configuration]`

kde

- **CommandName** představuje název nového příkazu sekce. bude mít.
- **Model** představuje název existujícího příkazu sekce, který bude použit jako model, z něhož bude nový příkaz zpočátku dědit. všechny jeho vlastnosti.

Ve skutečnosti nový příkaz zdědí mnohem více než jen svůj počáteční příkaz vlastnosti od vzoru: stává se jakýmsi přizpůsobeným instancí modelu, ale sdílí s ním například interní příkaz čítač, který řídí číslování.

- **Configuration** is the customised configuration of our new command. Here we can use exactly the same options as in `\setuphead`.

Nový příkaz není nutné konfigurovat v okamžiku jeho zadání. vytvoření. To lze provést později pomocí `\setuphead` a ve skutečnosti v příkladech uvedených v příručkách ConTeXt a na jeho wiki se zdá, že je to normální způsob.

3.6 Makrostruktura dokumentu

Kapitoly, oddíly, pododdíly, nadpisy... strukturují dokument; jsou to kapitoly, oddíly, pododdíly, nadpisy... organizují jej. Ale spolu se strukturou vyplývající z těchto druhů příkazů, v některých tištěných knihách, zejména v těch, které pocházejí z tzv. akademické sféry, existuje jakýsi druh *macro-ordering* knihy. s ohledem nikoli na obsah, ale na funkci, kterou každá z nich plní. z těchto velkých částí v knize plní. Tímto způsobem rozlišujeme mezi:

- Úvodní část dokumentu obsahující titulní stranu, stranu s poděkováním, stranu s věnováním, obsah, případně předmluvu, prezentační stránku atd.
- Hlavní část dokumentu, která obsahuje základní informace. text dokumentu, rozdělený do částí, kapitol, oddílů, pododdílů atd. Tato část je obvykle nejrozsáhlejší a důležitá.
- Doplnkový materiál tvořený dodatky nebo přílohami, které rozvíjejí nebo ilustrují některé otázky, jimiž se zabývá hlavní část, nebo poskytují dodatečnou dokumentaci, kterou nenapsal autor hlavního textu. hlavní části apod.
- Závěrečná část dokumentu, kde můžeme nalézt bibliografii, rejstříky, slovníčky atd.

Ve zdrojovém souboru můžeme každou z těchto částí ohraničit pomocí příkazu prostředí, které vidíme v [tabulka 3.2](#).

Část dokumentu	Příkaz
Počáteční část	<code>\startfrontmatter [Options] ... \stopfrontmatter</code>
Hlavní tělo	<code>\startbodymatter [Options] ... \stopbodymatter</code>
Přílohy	<code>\startappendices [Options] ... \stopappendices</code>
Závěrečná část	<code>\startbackmatter [Options] ... \stopbackmatter</code>

Tabulka 3.2 Prostředí, které odráží makrostrukturu dokumentu

Tato čtyři prostředí umožňují stejné čtyři možnosti: „page“, „before“, „after“ a „number“, a jejich hodnoty a jsou stejné jako v `\setuphead` (viz. sekce ??), i když bychom měli poznamenat, že zde se jedná o „number=no“ zruší číslování všech sekcí. příkazů v prostředí.

Zahrnout některou z těchto velkých sekcí do našeho dokumentu má smysl pouze tehdy, když pokud mezi nimi chceme vytvořit nějaký druh rozlišení. Snad záhlaví nebo číslování stránek ve frontmatteru. Konfigurace každého z těchto bloků se provádí pomocí `\setupsectionblock`, jehož syntaxe je:

`\setupsectionblock[Block name] [Options]`

kde *Název bloku* může být `frontpart`, `bodypart`, `dodatek` nebo `zadní část` a volby mohou být stejné jako právě zmíněné: „page“, „number“, „before“ a „after“. Takže například pro zajištění toho, že v *frontmatter* bude stránky číslovány římskými čísly, v preambuli našeho dokumentu bychom měli napsat:

```
\setupsectionblock
  [frontpart]
  [
    before={\setuppagenumbering[conversion=Romannumerals]}
  ]
```

Výchozí konfigurace ConT_EXtu pro tyto čtyři bloky znamená, že:

- Tyto čtyři bloky začínají novou stránku.
- Číslování sekcí se mění v každém z těchto bloků:
 - V `frontmatter` a `backmatter` se ve výchozím nastavení všechny číslované oddíly nečíslovány.
 - V `bodymatter` mají kapitoly arabské číslování.
 - V `appendices` jsou kapitoly číslovány velkými písmeny. písmeny.

Je také možné vytvářet nové bloky oddílů pomocí příkazů `\definesectionblock`.

Chapter 4

Obsah, rejstříky, seznamy

Table of Contents: **4.1 Obsah;** 4.1.1 Celkový pohled na obsah; 4.1.2 Zcela automatický obsah s názvem; 4.1.3 Automatický obsah bez názvu; 4.1.4 Prvky k začlenění do obsahu: volba `criterion`; 4.1.5 Rozvržení obsahu: `alternativní` možnost; 4.1.6 Formát položek obsahu; 4.1.7 Ruční úpravy obsahu; A Zahrnování nečíslovaných sekcí v obsahu; B Ruční přidávání položek do obsahu; C Vyloučení z obsahu konkrétní sekce patřící k typu sekce; D Text názvu sekce; **4.2 Seznamy, kombinované seznamy a obsah na základě seznamu;** 4.2.1 Seznamy v ConTeXtu; 4.2.2 Seznamy nebo indexy obrázků, tabulek a dalších položek; 4.2.3 Kombinované seznamy; **4.3 Index;** 4.3.1 Generování indexu; A Předchozí definice položek v rejstříku a označení bodů ve zdrojovém souboru, které na ně odkazují; B Generování konečného indexu; 4.3.2 Formátování předmětového rejstříku; 4.3.3 Vytváření dalších indexů;

Obsah a rejstřík jsou globálním aspektem dokumentu. Téměř všechny dokumenty budou mít obsah, zatímco pouze některé dokumenty budou mít rejstřík. Pro mnoho jazyků (ale ne pro angličtinu) spadají obsah i rejstřík pod obecný termín „index“. Pro čtenáře angličtiny bude obsah obvykle na začátku (dokumentu nebo možná v některých případech také na začátku kapitol) a rejstřík bude na konci.

Obojí znamená konkrétní aplikaci mechanismu pro interní odkazy, jehož vysvětlení je zahrnuto v [section 5.2](#).

4.1 Obsah

4.1.1 Celkový pohled na obsah

V předchozí kapitole jsme zkoumali příkazy, které umožňují vytvořit strukturu dokumentu tak, jak byl napsán. Tato sekce se zaměřuje na obsah a rejstřík, které nějakým způsobem *odrážejí* strukturu dokumentu. Obsah je velmi užitečný pro získání představy o dokumentu jako celku (pomáhá kontextualizovat informace) a pro vyhledání přesného místa, kde by se konkrétní pasáž mohla nacházet. Knihy s velmi složitou strukturou, s mnoha oddíly a pododdíly s různou úrovní hloubky, zdá se, vyžadují jiný druh obsahu, protože málo podrobný (možná pouze s prvními dvěma nebo třemi úrovněmi oddílů) hodně pomáhá získat celkovou představu o obsahu dokumentu, ale není příliš užitečný pro lokalizaci konkrétní pasáže; Na rozdíl od velmi podrobného obsahu na druhé straně, kde je snadné minout les pro

stromy a ztratit celkový pohled na dokument. To je důvod, proč někdy knihy s obzvláště složitou strukturou obsahují více než jeden obsah: jeden na začátku není příliš podrobný s hlavními částmi a podrobnější obsah na začátku každé kapitoly a také možná rejstřík na konci.

Všechny tyto mohou být generovány ConT_EXtem automaticky a relativně snadno. Můžeme:

- Vygenerovat úplný nebo částečný obsah v libovolném místě dokumentu.
- Rozhodnout o obsahu obou.
- Konfigurovat jejich vzhled do posledního detailu.
- Zahrnout do obsahu hypertextové odkazy, které nám umožní přejít přímo do příslušné sekce.

Ve skutečnosti je tato poslední možnost standardně zahrnuta ve všech tabulkách obsahu za předpokladu, že byla v dokumentu povolena funkce interaktivity. Viz v tomto ohledu [section 5.3](#).

Vysvětlení toho v referenční příručce ConT_EXtu je podle mého názoru poněkud matoucí, což je podle mně způsobeno tím, že je popsáno příliš mnoho informací najednou. Mechanismus pro vytváření obsahu v ConT_EXtu má mnoho částí; a pro text, který se je snaží vysvětlit všechny najednou, je obtížné být srozumitelným. Zvláště pro čtenáře, který je na scéně nový. Naproti tomu vysvětlení v wiki je prakticky omezeno na příklady: velmi užitečné pro učení *triků*, ale nedostatečné – myslím – pro pochopení mechanismu a toho, jak to funguje. to je důvod, proč strategie, kterou jsem se rozhodl použít k vysvětlení věcí v tomto úvodu, začíná předpokladem něčeho, co není striktně pravdivé (nebo není úplně pravda): že v ConT_EXtu je něco, co se nazývá *obsah*. Začneme tím, že jsou vysvětleny *normální* příkazy pro generování obsahu, a když jsou tyto příkazy a jejich konfigurace dobře známé, myslím, že toto je chvíle pro představení – i když spíše na teoretické než na praktičtější úrovni – informace o těch částech mechanismu, které byly do té doby vynechány. Znalost těchto dodatečných *částí* nám umožňuje vytvářet mnohem více přizpůsobené tabulky obsahu než ty, které můžeme nazývat *normálními* vytvořené pomocí příkazů vysvětlených až do tohoto bodu; ve většině případů to však nebudeme muset dělat.

4.1.2 Zcela automatický obsah s názvem

Základní příkazy pro generování automaticky generovaného obsahu z očíslovaných částí dokumentu (`part`, `chapter`, `section` atd.) jsou `\completecontent` a `\placecontent`. Hlavní rozdíl mezi těmito dvěma příkazy je ten, že první přidává do obsahu *název*; aby tak učinil, vloží bezprostředně před obsah *nečíslovanou kapitolu*, jejíž výchozí název je *Obsah*.

Proto `\completecontent`:

- Vloží do místa, kde je nalezen, novou nečíslovanou kapitolu s názvem „Obsah“.

Připomínáme, že v ConTeXtu je příkaz používáný ke generování nečíslované sekce na stejné úrovni jako kapitoly `\title` (viz [section 3.2](#)). Proto ve skutečnosti `\completecontent` nevkládá *Kapitolu* (`\chapter`), ale *Název* (`\title`). Neřekl jsem to, protože se domnívám, že pro čtenáře může být matoucí používat zde názvy nečíslovaných příkazů sekce, protože výraz *Název* má také širší význam a pro čtenáře je snadné neidentifikovat to s konkrétní úrovní dělení, na kterou odkazujeme.

- Tato *kapitola* (ve skutečnosti `\title`) je formátována přesně stejně jako ostatní nečíslované kapitoly v dokumentu; který ve výchozím nastavení obsahuje konec stránky.
- Obsah se vytiskne hned za názvem.

Zpočátku je vygenerovaný obsah *kompletní*, jak můžeme odvodit z názvu příkazu, který jej generuje (`\completecontent`). Ale na jedné straně můžeme omezit úroveň hloubky obsahu, jak je vysvětleno v [section 4.1.3](#), a na druhé straně, protože tento příkaz je *citlivý* na místo, kde se nachází v zdrojovém souboru (viz, co je dále řečeno o `\placecontent`), pokud `\completecontent` není nalezen na začátku dokumentu, je možné, že vygenerovaný obsah není úplný; a v některých místech zdrojového souboru je dokonce možné, že je příkaz zjevně ignorován. Pokud k tomu dojde, řešením je vyvolat příkaz s volbou „`criterium=all`“. Pokud jde o tuto volbu, viz také [section 4.1.3](#).

Ke změně výchozího názvu přiřazeného k obsahu použijeme příkaz `\setupheadtext`, jehož syntaxe je:

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

kde *Language* je nepovinné a odkazuje na jazykový identifikátor používaný ConTeXtem (viz [section 1.5](#)) a *Element* odkazuje na element, jehož jméno chceme změnit („content“ v případě obsahu) a *Name* je jméno nebo název, který chceme dát našemu obsahu. Například

`\setupheadtext [cs] [content=Obsah]`

zajistí, že obsah generovaný `\completecontent` bude mít název „Obsah“ namísto „Table of Contents“.

Navíc `\completecontent` umožňuje stejné konfigurační možnosti jako `\placecontent`, na jejichž vysvětlení odkazují (další část).

4.1.3 Automatický obsah bez názvu

Obecný příkaz pro vložení obsahu bez názvu, generovaný automaticky z příkazů pro dělení dokumentu, je `\placecontent`, jehož syntaxe je:

`\placecontent[Options]`

Obsah bude v zásadě obsahovat absolutně všechny očíslované sekce, i když jeho hloubku můžeme omezit příkazem `\setupcombinedlist` (o kterém budeme hovořit dále). Takže například:

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

omezí obsah obsahu na kapitoly a oddíly.

Zvláštností tohoto příkazu je, že je citlivý na své umístění ve zdrojovém souboru. To je velmi snadné vysvětlit na několika příkladech, ale mnohem obtížnější, pokud chceme přesně specifikovat, jak příkaz funguje a které nadpisy jsou v jakém případě zahrnuty do obsahu. Začněme tedy příklady:

- `\placecontent` umístěný na začátku dokumentu, před příkazem první sekce (část, kapitola nebo oddíl, podle situace) vygeneruje úplný obsah.

Nejsem si opravdu jistý, že obsah generovaný ve výchozím nastavení je *kompletní*, domnívám se, že obsahuje dostatek úrovní sekcí, aby byl ve většině případů úplný; ale mám podezření, že to nepřekročí osmou úroveň dělení. V každém případě, jak je uvedeno výše, můžeme upravit úroveň dělení, kterou obsah dosáhne pomocí

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

- Naproti tomu stejný příkaz umístěný uvnitř části, kapitoly nebo oddílu bude generovat výhradně obsah obsahu tohoto prvku, nebo jinými slovy kapitoly, oddíly a další nižší úrovně dělení konkrétní části, nebo oddíly (a další úrovně) konkrétní kapitoly, nebo pododdíly konkrétního oddílu.

Pokud jde o technické a podrobné vysvětlení, abychom správně porozuměli výchozímu fungování `\placecontent`, je nezbytné si uvědomit, že různé sekce jsou ve skutečnosti *prostředí* pro ConTeXt Mark IV, které začínají `\startSectionType` a končí `\stopSectionType` a mohou být obsaženy v jiných nižších příkazech sekce. Takže když to vezmeme v úvahu, můžeme říci, že `\placecontent` ve výchozím nastavení generuje obsah, který bude obsahovat pouze:

- Prvky, které patří do *prostředí* (úroveň oddílu), kde je umístěn příkaz. To znamená, že příkaz po umístění do kapitoly nebude obsahovat oddíly nebo pododdíly z jiných kapitol.
- Prvky, které mají úroveň dělení nižší než úroveň odpovídající bodu, kde je umístěn příkaz. To znamená, že pokud je příkaz v kapitole, jsou zahrnuty pouze oddíly, pododdíly a další nižší úrovně; ale pokud je příkaz v oddílu, bude rozdělen tak, aby byl obsah na úrovni pododdílu.

Kromě toho je pro vygenerování obsahu nutné, aby byl `\placecontent` nalezen *před* prvním oddílem kapitoly, ve které se nachází, nebo před prvním pododdílem oddílu, ve které se nachází, atd.

Nejsem si jistý, zda mé vysvětlení výše bylo jasné. Snad na poněkud podrobnějším příkladu než na předchozích příkladech lépe pochopíme, co tím myslím: představme si následující strukturu dokumentu:

- Kapitola 1
 - Oddíl 1.1
 - Oddíl 1.2
 - ★ Pododdíl 1.2.1
 - ★ Pododdíl 1.2.2
 - ★ Pododdíl 1.2.3
 - Oddíl 1.3
 - Oddíl 1.4
- Kapitola 2

Takže: `\placecontent` umístěný před Kapitolou 1 vygeneruje úplný obsah, podobný tomu, který generuje `\completecontent`, ale bez názvu. Ale pokud je příkaz umístěn v Kapitole 1 a před oddílem 1.1, bude obsah obsahovat pouze kapitolu; a pokud je umístěn na začátku oddílu 1.2, obsah bude pouze obsahem tohoto oddílu. Ale pokud je příkaz umístěn například mezi oddíly 1.1 a 1.2, bude ignorován. Bude také ignorován, pokud je umístěn na konec oddílu nebo na konec dokumentu.

To vše se samozřejmě vztahuje pouze na případ, kdy příkaz nezahrnuje možnosti. Zejména volba `criterium` změní toto výchozí chování.

Z možností, které `\placecontent` umožňuje, vysvětlím pouze dvě z nich, ty nejdůležitější pro nastavení obsahu, a navíc jediné, které jsou (částečně) zdokumentovány v referenční příručce ConTeXtu. Volba `criterium`, která ovlivňuje obsah obsahu ve vztahu k místu ve zdrojovém souboru, kde je umístěn příkaz; a možnost `alternative`, která ovlivňuje obecné rozložení obsahu, který se má vygenerovat.

4.1.4 Prvky k začlenění do obsahu: volba `criterium`

Výchozí chování `\placecontent` ve vztahu k pozici příkazu ve zdrojovém souboru bylo vysvětleno výše. Volba `criterium` toto chování mění. Mimo jiné může nabývat následujících hodnot:

- `all`: obsah bude kompletní, bez ohledu na místo ve zdrojovém souboru, kde se příkaz nachází.
- `previous`: obsah bude obsahovat pouze příkazy oddílu (na úrovni, na které se nacházíme), které *předchází* `\placecontent`. Tato možnost je určena pro obsahy, které se zapisují na konec příslušného dokumentu nebo oddílu.

- `part`, `chapter`, `section`, `subsection`...: znamená, že obsah by měl být omezen na uvedenou úroveň dělení.
- `component`: ve vícesouborových projektech (viz [section 2.6](#)) bude vygenerován pouze obsah odpovídající *komponentě*, kde je příkaz `\placecontent` nebo `\completecontent` nalezen.

4.1.5 Rozvržení obsahu: alternativní možnost

Možnost `alternative` řídí celkové rozložení obsahu. Její hlavní hodnoty můžete vidět v [table 4.1](#).

alternative	Obsah položek obsahu	Poznámky
a	Číslo – Název – Strana	Jeden řádek na položku
b	Číslo – Název – Mezery – Strana	Jeden řádek na položku
c	Číslo – Název – Vodící tečky – Strana	Jeden řádek na položku
d	Číslo – Název – Strana	Průběžný obsah
e	Název	Orámován
f	Název	Zarovnán doleva, zarovnán doprava nebo na střed
g	Název	Zarovnán na střed

Tabulka 4.1 Způsoby formátování obsahu

První čtyři alternativní hodnoty poskytují veškeré informace o každé sekci (její číslo, název a číslo stránky, kde začíná), a jsou proto vhodné pro papírové i elektronické dokumenty. Poslední tři alternativy nás informují pouze o názvu, takže jsou vhodné pouze pro elektronické dokumenty, kde není nutné znát číslo stránky, kde sekce začíná, za předpokladu, že obsah na něj obsahuje hypertextový odkaz, což je v ConTeXtu výchozí nastavení.

Dále se domnívám, že pro skutečné pochopení rozdílů mezi různými alternativami je nejlepší, aby si čtenář vygeneroval testovací dokument, kde je může podrobně analyzovat.

4.1.6 Formát položek obsahu

Viděli jsme, že možnost `alternative` v `\placecontent` nebo `\completecontent` nám umožňuje ovládat obecné rozvržení obsahu, tj. jaké informace budou zobrazeny pro každý nadpis a zda budou či nebudou jednotlivé nadpisy oddělovány zalomením řádku. Konečné úpravy každé položky obsahu se provádějí pomocí příkazu `\setuplist`, jehož syntaxe je následující:

`\setuplist[Element][Configuration]`

kde *Element* odkazuje na určitý druh sekce. Může to být `part`, `chapter`, `section` atd. Můžeme také konfigurovat více než jeden prvek současně a oddělit je čárkami.

Configuration má až 54 možností, přičemž mnoho z nich, jako obvykle, není výslovně zdokumentováno; ale to nebrání těm, které jsou zdokumentovány, nebo těm, které nejsou dostatečně jasné, aby umožnily úplnou úpravu obsahu.

Nyní vysvětlím nejdůležitější možnosti, seskupím je podle jejich užitečnosti, ale než se do nich pustíme, pamatujme, že položka obsahu, v závislosti na hodnotě **alternative**, může mít až tři různé složky: Číslo sekce, název sekce a číslo stránky. Možnosti konfigurace nám umožňují konfigurovat různé komponenty globálně nebo samostatně:

- *Zahrnutí (nebo ne) různých komponent*: Pokud jsme zvolili alternativu, která kromě názvu obsahuje číslo sekce a číslo stránky (alternativy ,a‘ ,b‘ ,c‘ nebo ,d‘), možnosti **headnumber=no** nebo **pagenumber=no** znamenají, že pro konkrétní úroveň, kterou konfiguruje, se číslo sekce (**headnumber**) nebo číslo stránky (**pagenumber**) nezobrazuje.
- *Barva a styl*: Již víme, že položka, která generuje konkrétní sekci v obsahu, může mít (v závislosti na alternativě) až tři různé součásti: číslo sekce, nadpis a číslo stránky. Můžeme společně označit styl a barvu pro tři komponenty pomocí možností **style** a **color**, nebo to udělat jednotlivě pro každou komponentu pomocí **numberstyle**, **textstyle** nebo **pagestyle** (pro styl) a **numbercolor**, **textcolor** nebo **pagecolor** pro barvu.

Chcete-li ovládat vzhled každé položky, kromě stylu samotného, můžeme použít nějaký příkaz na celou položku nebo na jeden z jejích různých prvků. K tomu existují možnosti **command**, **numbercommand**, **pagecommand** a **textcommand**. Zde uvedený příkaz může být standardní příkaz ConTeXtu nebo příkaz, který jsme sami vytvořili. Číslo sekce, text názvu a číslo stránky budou předány jako argument volbě **command**, zatímco název oddílu bude předán jako argument **textcommand** a číslo stránky **pagecommand**. Takže například následující věta zajistí, že názvy oddílů budou napsány (falešnými) malými kapitálkami:

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

- *Oddělení ostatních prvků obsahu*: Volby **before** a **after** nám umožňují označit příkazy, které budou provedeny před (**before**) a po (**after**) vysázení položky obsahu. Normálně se tyto příkazy používají k nastavení buď mezery nebo nějakého oddělovacího prvku mezi předchozími a následujícími položkami.
- *Odsazení prvku*: nastavte s volbou **margin**, která nám umožňuje nastavit míru levého odsazení, které budou mít položky úrovně, kterou konfiguruje.
- *Hypertextové odkazy vložené do obsahu*: Ve výchozím nastavení obsahují položky rejstříku hypertextový odkaz na stránku dokumentu, kde daná sekce začíná. Pomocí možnosti **interaction** můžeme tuto funkci zakázat (**interaction=no**) nebo omezit část položky rejstříku, kde bude hypertextový odkaz,

což může být číslo sekce (`interaction=number` nebo `interaction=sectionnumber`), název sekce (`interaction=text` nebo `interaction=title`) nebo číslo stránky (`interaction=page` nebo `interaction=pagename`).

- *Další aspekty:*
 - **width:** určuje vzdálenost mezi číslem a názvem sekce. Může to být rozměr nebo klíčové slovo `fit`, které nastavuje přesnou šířku čísla sekce.
 - **symbol:** umožňuje nahrazení čísla sekce *symbolem*. Jsou podporovány tři možné hodnoty: `one`, `two` a `three`. Hodnota `none` pro tuto možnost odstraní číslo sekce z obsahu.
 - **numberalign:** označuje zarovnání prvků číslování; může být `left`, `right`, `middle`, `flushright`, `flushleft`.

Mezi mnoha možnostmi konfigurace obsahu není žádná, která by nám umožňovala přímo ovlivňovat vzdálenost mezi řádky. Ve výchozím nastavení to bude příkaz, který se vztahuje na dokument jako celek. Často je však vhodnější, aby řádky v obsahu byly o něco *těsnější* než zbytek dokumentu. Abychom toho dosáhli, měli bychom uzavřít příkaz, který generuje obsah (`\placecontent` nebo `\completecontent`) ve skupině, kde jsou nastaveny jiné meziřádkové mezery. Například:

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

4.1.7 Ruční úpravy obsahu

Již jsme vysvětlili dva základní příkazy pro generování tabulek obsahu (`\placecontent` a `\completecontent`) a také jejich možnosti. Pomocí těchto dvou příkazů jsou obsahy automaticky generovány, sestavovány z existujících očíslovaných sekcí v dokumentu nebo v bloku nebo segmentu dokumentu, na který se obsah vztahuje. Nyní vysvětlím určitá *nastavení*, která můžeme provést, aby obsah obsahu nebyl tak *automatický*. Z toho vyplývá:

- Možnost zahrnout také některé nečíslované názvy sekcí do obsahu.
- Možnost ručního odeslání konkrétní položky, která neodpovídá přítomnosti očíslované sekce, do obsahu.
- Možnost vyloučit určitou očíslovanou sekci z obsahu.
- Možnost, že název určité sekce uvedený v obsahu se přesně neshoduje s názvem obsaženým v těle dokumentu.

A. Zahrnování nečíslovaných sekcí v obsahu

Mechanismus, kterým ConT_EXt vytváří obsah, je nastavený tak, že jsou automaticky zahrnuty všechny očíslované sekce, což, jak jsem již řekl (viz [section 3.4.2](#)) závisí na těchto dvou (`number` a `incrementnumber`) možnostech, které můžeme změnit pomocí `\setuphead` pro každý druh sekce. Bylo tam také vysvětleno, že typ sekce, kde `incrementnumber=yes` a `number=no` bude vnitřně, ale ne externě číslovaná sekce.

Pokud tedy chceme, aby určitý typ nečíslované sekce – například `title` – byl zahrnut do obsahu, musíme změnit hodnotu možnosti `incrementnumber` pro tento typ sekce a nastavit ji na `yes` a pak zahrnout tento typ sekce mezi ty, které se mají zobrazit v obsahu, což se provede, jak je vysvětleno výše, pomocí `\setupcombinedlist`:

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

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

Pak můžeme, pokud si to přejeme, naformátovat tuto položku pomocí `\setuplist` přesně stejným způsobem jako kterýkoli z ostatních; například:

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

Poznámka: Právě vysvětlený postup bude zahrnovat všechny výskyty příslušného nečíslovaného typu sekce v našem dokumentu (v našem příkladu sekce typu `title`). Pokud si přejeme zahrnout pouze konkrétní výskyt tohoto typu sekce do obsahu, je vhodnější tak učinit postupem vysvětleným níže.

B. Ruční přidávání položek do obsahu

Z libovolného místa ve zdrojovém souboru můžeme odeslat buď položku (simulující existenci sekce, která ve skutečnosti neexistuje), nebo příkaz do obsahu.

Chcete-li odeslat položku, která simuluje existenci sekce, která ve skutečnosti neexistuje, použijte `\writetolist`, jehož syntaxe je:

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

ve kterém

- První argument označuje úroveň, kterou musí mít tato položka sekce v obsahu: `chapter`, `section`, `subsection` atd.
- Druhý argument, který je volitelný, umožňuje tuto položku konfigurovat určitým způsobem. Pokud je ručně odeslaný vstup vynechán, bude zformátován stejně jako všechny položky úrovně označené prvním argumentem; i když musím podotknout, že v mých testech se mi to nepodařilo zprovoznit.



Jak v oficiálním ConT_EXtovém seznamu příkazů (viz [section 1.6](#)), tak na wiki je nám řečeno, že tento argument umožňuje stejné hodnoty jako `\setuplist`, což je příkaz, který nám umožňuje formátovat různé položky obsahu. Ale trvám na tom, že v mých testech se mi nepodařilo žádným způsobem změnit vzhled položky obsahu odeslané ručně.



- Třetí argument má odrážet číslování, které má prvek odeslaný do obsahu, ale ani to se mi nepodařilo při mých testech uvést do provozu.
- Poslední argument obsahuje text, který má být odeslán do obsahu.

To je užitečné například v případě, že chceme poslat konkrétní nečíslovanou sekci, ale pouze tu do obsahu. V `section ??` je vysvětleno, jak dostat celou kategorii nečíslovaných sekcí k odeslání do obsahu; ale pokud na něj chceme poslat pouze konkrétní výskyt typu sekce, je pohodlnější použít příkaz `\writetolist`. A tak například, pokud chceme, aby část našeho dokumentu obsahující bibliografii nebyla číslovaná, ale přesto byla zahrnuta do obsahu, napíšeme:

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

Podívejte se, jak používáme nečíslovanou verzi `section`, což je `subject`, pro sekci, ale posíláme ji do indexu ručně, jako by to byla očíslovaná sekce (`section`).

Dalším příkazem určeným k ručnímu ovlivnění obsahu je `\writebetweenlist`, který se používá k odeslání nikoli samotné položky, ale *příkazu* do obsahu, z určitého místa v dokumentu. Pokud například chceme zahrnout řádek mezi dvě položky v obsahu, můžeme kdykoli v dokumentu, který se nachází mezi dvěma příslušnými oddíly, napsat následující:

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

C. Vyloučení z obsahu konkrétní sekce patřící k typu sekce

Obsah je vytvořen z *typu sekcí* stanovených, jak již víme, volbou `list` z `\setupcombinedlist`, takže pokud se v obsahu musí objevit určitý *typ sekce*, neexistuje způsob, jak z něj vyloučit konkrétní sekci, kterou z jakýchkoli důvodů nechceme v obsahu.

Normálně, pokud nechceme, aby se tam objevila sekce, co bychom udělali, je použít její *neočíslovaný ekvivalent*, například `title` namísto `chapter`, `subject` místo `section` atd. Tyto sekce se neodesílají do obsahu a ani nejsou číslovány.

Pokud však z nějakého důvodu chceme, aby určitá sekce byla očíslována, ale neobjevila se v obsahu, i když jiné typy tohoto druhu ano, můžeme použít *trik*, který spočívá ve vytvoření nového typu sekce, který je klonem daného úseku. Například:

```
\definehead[MojePodsekce][subsection]
\section{První sekce}
\subsection{První podsekce}
\MojePodsekce{Druhá podsekce}
\subsection{Třetí podsekce}
```

Tímto se zajistí, že při vkládání sekce typu `MojePodsekce` se zvýší počítadlo podsekcí, protože tato sekce je *klon* podsekcí, ale obsah se nezmění, protože ve výchozím nastavení neobsahuje typy `MojePodsekce`.

D. Text názvu sekce

Pokud nechceme, aby byl název konkrétní sekce obsažený v obsahu totožný s názvem zobrazeným v těle dokumentu, máme k dispozici dva postupy:

- Vytvoření sekce nikoli s tradiční syntaxí (`\SectionType{Title}`), ale pomocí `\SectionType [Options]` nebo pomocí `\startSectionType [Options]` a přiřadit text, který chceme napsat v obsahu k volbě `list` (viz `section ??`).
- Při psaní názvu příslušné sekce do těla dokumentu použijte příkaz `\nolist`: tento příkaz způsobí, že text, který bere jako argument, bude v obsahu nahrazen třemi tečkami. Například:

```
\chapter [title={\nolist{Přibližný a-mírně se opakující} úvod do reality
zjevného}]
```

by ConTeXt vysázel jako název kapitoly v těle dokumentu „Přibližný a mírně se opakující úvod do reality zjevného“, ale do obsahu by poslal následující text „... úvod do reality zjevného“.

Pozor: To, na co jsem právě poukázal ohledně příkazu `\nolist`, je uvedeno jak v referenční příručce ConTeXtu, tak na [wiki](#). Mně to však vyhazuje chybu při kompilaci, která mi říká, že příkaz `\nolist` není definován.

4.2 Seznamy, kombinované seznamy a obsah na základě seznamu

Interně pro ConTeXt není obsah nic jiného než *kombinovaný seznam*, který, jak jeho název napovídá, sestává z kombinace jednoduchých seznamů. Základní pojem,

ze kterého ConT_EXt staví obsah, je tedy seznam. Několik seznamů je sloučeno pro vytvoření obsahu. Ve výchozím nastavení ConT_EXt obsahuje předdefinovaný kombinovaný seznam nazvaný „**content**“ a to je to, s čím dosud zkoumané příkazy pracují: `\placecontent` a `\completecontent`.

4.2.1 Seznamy v ConT_EXtu

V ConT_EXtu je *seznam* řada číslovaných prvků, o kterých je třeba si zapamatovat tři věci:

1. Číslo.
2. Jméno nebo název.
3. Stránka, kde je nalezen.

To se děje s očíslovanými sekcemi; ale také s dalšími prvky dokumentu, jako jsou obrázky, tabulky atd. Obecně se jedná o ty prvky, pro které existuje příkaz, jehož název začíná `\place`, který je umístí jako `\placetable`, `\placefigure`, atd.

Ve všech těchto případech ConT_EXt automaticky vygeneruje seznam různých časů, kdy se typ příslušného prvku objevil, jeho číslo, název a stránku. Tak například existuje seznam kapitol, nazvaný `chapter`, pro další ze sekcí nazvaný `section`; ale také další pro tabulky (nazvaný `table`) nebo pro obrázky (nazvaný `figure`). Seznamy generované automaticky ConT_EXtem se vždy nazývají stejně jako položka, kterou ukládají.

Automaticky se vygeneruje i seznam, pokud vytvoříme např. nový typ číslované sekce: při jeho vytváření budeme implicitně vytvářet i seznam, který je ukládá. A pokud pro nečíslovanou sekci ve výchozím nastavení nastavíme možnost `incrementnumber=yes`, čímž se stane očíslovanou sekcí, implicitně také vytvoříme seznam s tímto názvem.

Spolu s implicitními seznamy (automaticky definovanými ConT_EXtem) můžeme vytvářet vlastní seznamy pomocí `\definelist`, jehož syntaxe je

`\definelist [ListName] [Configuration]`

Položky v seznamu jsou přidány:

- V sezonech předdefinovaných ConT_EXtem nebo jím vytvořených jako výsledek vytvoření nového plovoucího objektu (viz [section 4.5](#)), automaticky pokračuje, když je do dokumentu vložena položka ze seznamu, buď oddělovacím příkazem nebo příkazem `\placewhatever` pro jiné typy seznamů, například: `\placefigure`, vloží do dokumentu libovolný obrázek, ale také vloží odpovídající položku do seznamu.

- Ručně v libovolném seznamu pomocí `\writetolist[ListName]`, jak již bylo vysvětleno v subsection B v section 4.1.7. K dispozici je také příkaz `\writebetweenlist`. V té části to bylo také vysvětleno.

Jakmile je seznam vytvořen a všechny jeho položky jsou v něm zahrnuty, tři základní příkazy, které se k němu vztahují, jsou `\setuplist`, `\placelist` a `\completelist`. První nám umožňuje konfigurovat, jak seznam vypadá; poslední dva vloží daný seznam na místo v dokumentu, kde je najde. Rozdíl mezi `\placelist` a `\completelist` je podobný jako rozdíl mezi `\placecontent` a `\completecontent` (viz sekce 4.1.2 a 4.1.3).

Takže například,

`\placelist[section]`

vloží seznam sekcí včetně hypertextového odkazu na ně, pokud je povolena interaktivita dokumentu a pokud v `\setuplist` nemáme nastaveno `interaction=no`. Seznam sekcí není úplně stejný jako obsah založený na sekcích: myšlenka obsahu obvykle zahrnuje i nižší úrovně (podsekce, podpodsekce atd.). Ale seznam sekcí bude obsahovat pouze sekce samotné.

Syntaxe těchto příkazů je:

`\placelist[ListName] [Options]`

`\setuplist[ListName] [Configuration]`

Možnosti `\setuplist` již byly vysvětleny v section 4.1.6 a možnosti pro `\placelist` jsou stejné jako pro `\placecontent` (viz section 4.1.3).

4.2.2 Seznamy nebo indexy obrázků, tabulek a dalších položek

Z toho, co bylo dosud řečeno, je vidět, že protože ConTeXt automaticky vytváří seznam obrázků umístěných v dokumentu pomocí příkazu `\placefigure`, generování seznamu nebo rejstříku obrázků na určitém místě v našem dokumentu je stejně jednoduché jako použití příkazu `\placelist[figure]`. A pokud chceme vygenerovat seznam s názvem (podobný tomu, co získáme pomocí `\completecontent`), můžeme to udělat pomocí `\completelist[figure]`. Podobně můžeme postupovat s dalšími čtyřmi předdefinovanými druhy plovoucích objektů v ConTeXtu: tabulkami („table“), grafikami („graphic“), *intermezzy* („intermezzo“) a chemickými vzorci („chemical“), ačkoli pro konkrétní případy, ConTeXt již obsahuje příkaz, který je generuje bez názvu: (`\placelistoffigures`, `\placelistoftables`, `\placelistofgraphics`, `\placelistofintermezzi` a `\placelistofchemicals`) a další, který

je generuje s názvem: (`\completelistoffigures`, `\completelistoftables`, `\completelistofgraphics`, `\completelistofintermezzi` a `\completelistofchemicals`), podobným způsobem jako `\completecontent`.

Stejně tak pro plovoucí objekty, které jsme sami vytvořili (viz [section 4.5](#)), se automaticky vytvoří `\placelistof<FloatName>` a `\completelistof<FloatName>`.

Pro seznamy, které jsme vytvořili pomocí `\definelist`, můžeme vytvořit index pomocí `\placelist[ListName]` nebo pomocí `\completelist[ListName]`.

4.2.3 Kombinované seznamy

Kombinovaný seznam je, jak jeho název napovídá, seznam, který kombinuje položky z různých dříve definovaných seznamů. Ve výchozím nastavení ConTeXt definuje kombinovaný seznam pro tabulky obsahu, jejichž název je „`content`“, ale můžeme vytvořit další kombinované seznamy pomocí `\definecombinedlist`, jehož syntaxe je:

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

kde

- *Name*: je název, který bude mít nový kombinovaný seznam.
- *Lists*: odkazuje na názvy seznamů, které mají být kombinovány, oddělené čárkami.
- *Options*: Možnosti konfigurace pro seznam. Mohou být uvedeny v době definování seznamu, nebo pravděpodobně nejlépe, když je seznam vyvolán. Hlavní možnosti (které již byly vysvětleny) jsou `criterion` ([subsection 4.1.4](#) v [section 4.1.3](#)) a `alternative` (v [subsection 4.1.5](#) ve stejné sekci).

Vedlejším efektem vytvoření kombinovaného seznamu pomocí `\definecombinedlist` je, že také vytvoří příkaz nazvaný `\placeListName`, který slouží k vyvolání seznamu, to znamená: k jeho zahrnutí do výstupního souboru. Takže například,

```
definecombinedlist[obsah]
```

vytvoří příkaz `\placeobsah`; a

```
definecombinedlist[content]
```

vytvoří příkaz `\placecontent`

Ale počkat, `\placecontent`! Není to příkaz, který se používá k vytvoření *normálního* obsahu? Skutečně: to znamená, že standardní obsah je ve skutečnosti ConTeXtem vytvořen pomocí následujícího příkazu:


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

Jakmile je náš kombinovaný seznam definován, můžeme jej nakonfigurovat (nebo překonfigurovat) pomocí `\setupcombinedlist`, který umožňuje již vysvětlené možnosti `criterium` (viz [subsection 4.1.4](#) v [section 4.1.3](#)) a `alternative` (viz [subsection 4.1.5](#) ve stejné sekci), stejně jako možnost `list` sloužící ke změně zahrnutých seznamů v kombinovaném seznamu.

Oficiální seznam příkazů ConTeXtu (viz [section 1.6](#)) nezmiňuje volbu `list` mezi možnostmi povolenými pro `\setupcombinedlist`, ale používá se na několika příkladech použití tohoto příkazu na wiki (která jej navíc nezmiňuje ani na stránce věnované tomuto příkazu). Také jsem zkontroloval, že tato možnost funguje.

4.3 Index

4.3.1 Generování indexu

Předmětový rejstřík se skládá ze seznamu významných termínů, obvykle umístěném na konci dokumentu, označujícího stránky, kde lze daný předmět nalézt.

Když byly knihy sázeny ručně, bylo generování předmětového rejstříku složitým a také únavným úkolem. Jakákoli změna stránkování by mohla ovlivnit všechny položky v rejstříku. Nebyly proto příliš běžné. Dnes však počítačové mechanismy pro sazbu znamenají, že i když tento úkol bude pravděpodobně i nadále zdoluhavý, již není tak složitý, protože pro počítačový systém není tak obtížné udržovat aktuální seznam dat spojených s položkou rejstříku.

K vytvoření předmětového rejstříku potřebujeme:

1. Určit která slova, termíny nebo koncepty mají být jeho součástí. To je úkol, který může udělat pouze autor.
2. Zkontrolovat, ve kterých bodech dokumentu se každý záznam v budoucím rejstříku objeví. I když, abychom byli přesní, více než *kontrolovat* místa ve zdrojovém souboru, kde se diskutuje o konceptu nebo problému, co děláme, když pracujeme s ConTeXtem, je *označit* tato místa vložením příkazu, který pak poslouží k automatickému vygenerování indexu. Toto je ta zdoluhavá část.
3. Nakonec vygenerujeme a naformátujeme index umístěním na místo v dokumentu, které si zvolíme. Tohle je v ConTeXtu docela jednoduché a vyžaduje pouze jeden příkaz: `\placeindex`.

A. Předchozí definice položek v rejstříku a označení bodů ve zdrojovém souboru, které na ně odkazují

Základní práce je ve druhém kroku. Je pravda, že počítačové systémy to také usnadňují v tom smyslu, že můžeme provést globální textové vyhledávání, abychom našli místa ve zdrojovém souboru, kde se o konkrétním předmětu jedná. Ale také bychom se neměli slepě spoléhat na takové textové vyhledávání: dobrý předmětový rejstřík musí být schopen odhalit každé místo, kde se diskutuje o konkrétním předmětu, i když se tak děje bez použití *standardního* termínu pro jeho odkazování.

Chcete-li *označit* skutečný bod ve zdrojovém souboru a přiřadit jej ke slovu, termínu nebo myšlence, které se objeví v indexu, použijeme příkaz `\index`, jehož syntaxe je následující:

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

kde *Alphabetical* je volitelný argument, který se používá k označení alternativního textu k textu samotné položky rejstříku, aby bylo možné řadit podle abecedy, a *Index entry* je text, který se objeví v rejstříku, spojený s touto značkou. Můžeme také použít funkce formátování, které chceme použít, a pokud se v textu objeví vyhrazené znaky, musí být zapsány obvyklým ConTeXtovým způsobem.

Možnost seřadit položku rejstříku podle abecedy jiným způsobem, než jak je ve skutečnosti napsána, je velmi užitečná. Vzpomeňte si například na tento dokument, pokud chci vygenerovat záznam v rejstříku pro všechny odkazy na příkaz `\TeX`. Například sekvence `\index{\backslash TeX}` vypíše příkaz nikoli podle ‚t‘ v ‚TeX‘, ale mezi symboly, protože výraz zaslaný do indexu začíná znakem obrácené lomítka. To se provede napsáním `\index[tex]{\backslash TeX}`.

Položky indexu budou ty, které chceme. Aby byl předmětový rejstřík skutečně užitečný, musíme se trochu více snažit při zodpovězení otázky: jaké pojmy bude čtenář dokumentu s největší pravděpodobností hledat; takže například může být lepší definovat položku jako „nemoc, Hodgkins“ než ji definovat jako „Hodgkinova nemoc“, protože obsáhlejším termínem je „nemoc“.

Podle konvence jsou položky v předmětovém rejstříku vždy psány malými písmeny, pokud se nejedná o vlastní jména.

Pokud má rejstřík několik úrovní hloubky (jsou povoleny až tři) pro přiřazení konkrétní položky rejstříku ke konkrétní úrovni, použije se znak ‚+‘ následovně:

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

V prvním případě jsme definovali položku druhé úrovně nazvanou *Položka 2*, která bude podpoložkou *Položky 1*. Ve druhém případě jsme definovali položku třetí úrovně nazvanou *Položka 3*, která bude podpoložkou *Položky 2*, což je zase podpoložkou *Položky 1*. Například

Můj `\index{pes}`pes je `\index{pes+chrt}`chrt zvaný Rocket.
Nemá rád `\index{kočka+toulavé}`toulavé kočky.

Některé podrobnosti výše stojí za zmínku:

- Příkaz `\index` je obvykle umístěn *před* slovem, se kterým je spojen, a normálně od něj není oddělen mezerou. Tím se zajistí, že příkaz bude na stejné stránce jako slovo, ke kterému je připojen:
 - Pokud by je oddělovala mezeru, mohla by existovat možnost, že by ConTeXt zvolil právě tuto mezeru pro zalomení řádku, což by také mohlo skončit jako konec stránky a v takovém případě by byl příkaz na jedné stránce a slovo, se kterým je spojeno na další stránce.
 - Pokud by příkaz měl následovat *za* slovem, bylo by možné, aby toto slovo bylo rozděleno na slabiky a mezi dvě jeho slabiky by se vložilo zalomení řádku, což by také znamenalo konec stránky a v takovém případě by příkaz ukazoval na další stránku začínající slovem, na které ukazuje.
- Podívejte se, jak jsou termíny druhé úrovně zavedeny ve druhém a třetím vzhledu příkazu.
- Také zkontrolujte, jak při třetím použití příkazu `\index`, ačkoli slovo, které se v textu objeví, je „kočky“, termín, který bude odeslán do indexu, je „kočka“.
- Konečně: podívejte se, jak byly tři položky pro předmětový rejstřík zapsány na pouhé dva řádky. Už jsem zmínil, že označení přesných míst ve zdrojovém souboru je zdoluhavé. Nyní dodám, že označit jich příliš mnoho je kontraproduktivní. Příliš rozsáhlý rejstřík není v žádném případě výhodnější než stručnější, ve kterém jsou všechny informace relevantní. Proto jsem již dříve řekl, že rozhodnutí, která slova vygenerují záznam v rejstříku, by mělo být výsledkem vědomého rozhodnutí autora.

Pokud chceme, aby byl náš rejstřík skutečně užitečný, termíny, které se používají jako synonyma, musí být v rejstříku seskupeny pod jeden hlavní termín. Ale protože je možné, aby čtenář v rejstříku vyhledával informace podle kteréhokoli z jiných hesel, je běžné, že rejstřík obsahuje položky, které odkazují na jiné položky. Například předmětový rejstřík příručky občanského práva by mohl být něco jako

smluvní invalidita
viz *neplatnost*.

Toho nedosáhneme pomocí příkazu `\index`, ale pomocí `\seeindex`, jehož formát je:

```
\seeindex[Alphabetical] {Entry1} {Entry2}
```

kde *Entry1* je záznam rejstříku, který bude odkazovat na druhý; a *Entry2* je referenční cíl. V našem předchozím příkladu bychom museli napsat:

```
\seeindex{smluvní invalidita}{neplatnost}
```

V `\seeindex` můžeme také použít znak `,+` k označení podúrovni pro kterýkoli z jeho dvou argumentů v hranatých závorkách.

B. Generování konečného indexu

Jakmile označíme všechny položky pro index v našem zdrojovém souboru, skutečné vygenerování indexu se provede pomocí příkazů `\placeindex` nebo `\completeindex`. Tyto dva příkazy prohledají zdrojový soubor pro příkazy `\index` a vygenerují seznam všech položek, které by měl mít index, s přidružením termínu k číslu stránky odpovídajícímu tomu, kde našel příkaz `\index`. Potom abecedně seřadí seznam výrazů, které se objevují v rejstříku, sloučí případy, kdy se stejný výraz vyskytuje více než jednou, a nakonec vloží správně formátovaný výsledek do konečného dokumentu.

Rozdíl mezi `\placeindex` a `\completeindex` je podobný rozdílu mezi `\content` a `\completecontent` (viz [section 4.1.2](#)): `\placeindex` je omezen na generování rejstříku a jeho vkládání, zatímco `\completeindex` nejdříve vloží do finálního dokumentu novou kapitolu, která se standardně nazývá „Index“, do které bude index vysázen.

4.3.2 Formátování předmětového rejstříku

Předmětové rejstříky jsou konkrétní aplikací obecnější struktury, kterou ConTeXt volá „*register*“; proto je index formátován příkazem:

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

Pomocí tohoto příkazu můžeme:

- Určit, jak bude index vypadat s jeho různými prvky. Konkrétně:
 - Záhloví rejstříku, což jsou obvykle písmena abecedy. Ve výchozím nastavení jsou tato písmena malá. Pomocí `alternative=A` je můžeme nastavit jako velká písmena.
 - Samotné položky a jejich číslo stránky. Vzhled závisí na možnostech `textstyle`, `textcolor`, `textcommand` a `deeptextcommand` pro položku

a `pagestyle`, `pagecolor` a `pagecommand` pro číslo stránky. Pomocí `pagenumber=no` můžeme také vygenerovat předmětový rejstřík bez čísel stránek (i když nevím, jestli by to mohlo být užitečné).

- Volba `distance` měří šířku oddělení mezi názvem položky a čísly stránek; ale také měří velikost odsazení pro podpoložky.

Myslím si, že názvy možností `style`, `textstyle`, `pagestyle`, `color`, `textcolor` a `pagecolor` jsou dostatečně jasné, aby nám řekly, co každá z nich dělá. Pro `command`, `pagecommand`, `textcommand` a `deeptextcommand` odkazují na vysvětlení pro podobně pojmenované možnosti v [section 3.4.4](#), týkající se konfigurace příkazů sekcí.

- Chcete-li nastavit obecný vzhled indexu, který mimo jiné zahrnuje příkazy, které se mají provést před (`before`) nebo za (`after`) indexem, počet sloupců, které musí mít (`n`), zda mají být sloupce stejné nebo ne (`balance`), zarovnání položek (`align`) atd.

4.3.3 Vytváření dalších indexů

Vysvětlil jsem předmětový rejstřík, jako by v dokumentu byl možný pouze jeden takový rejstřík; ale pravdou je, že dokumenty mohou mít tolik indexů, kolik chcete. Může existovat například rejstřík osobních jmen, který shromažďuje jména osob zmíněných v dokumentu s uvedením místa, kde jsou citováni. Ty jsou stále jakýmsi indexem. V právním textu bychom také mohli vytvořit speciální rejstřík pro zmínky o občanském zákoníku; nebo v dokumentu, jako je tento, rejstřík marker v něm vysvětlených atd.

K vytvoření dalšího indexu v našem dokumentu použijeme příkaz `\defineregister`, jehož syntaxe je:

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

kde *IndexName* je název, který bude mít nový index, a *Configuration* řídí, jak funguje. Je také možné konfigurovat index později pomocí

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

Jakmile bude vytvořen nově pojmenovaný index *IndexName*, budeme mít k dispozici příkaz `\IndexName` pro označení položek, které tento index bude mít, podobným způsobem, jako jsou položky označeny pomocí `\index`. Příkaz `seeIndexName` nám také umožňuje vytvářet položky, které odkazují na jiné položky.

Například: mohli bychom vytvořit index ConTeXtových příkazů v tomto dokumentu pomocí příkazu:

```
\defineregister[macro]
```

to by vytvořilo příkaz `\macro`. To mi umožňuje označit všechny odkazy na příkazy ConTeXtu jako položku rejstříku a poté vygenerovat index pomocí `\placemacro` nebo `\completemacro`.

Vytvoření nového indexu umožňuje příkazu `\IndexName` označit jeho položky a příkazy `\placeIndexName` a `\completeIndexName` pro generování indexu. Ale tyto dva poslední příkazy jsou ve skutečnosti zkratky dvou obecnějších příkazů aplikovaných na dotýčný index. Tedy `\placeIndexName` je ekvivalentní `\placeregister[IndexName]` a `\completeIndexName` je ekvivalentní `\completeregister[IndexName]`.

Chapter 5

Reference a hypertextové odkazy

Table of Contents: 5.1 Referenční typy; 5.2 Interní reference; 5.2.1 Štítek v referenčním cíli; 5.2.2 Příkazy v referenčním bodě původu pro získání dat z cílového bodu; 5.2.3 Základní příkazy pro získávání informací ze štítku; A Získání informací spojených se štítkem pomocí příkazu `\ref`; B Zjištění, kam vede odkaz; 5.2.4 Automatické generování předpon pro zamezení duplicitních štítků; 5.3 Interaktivní elektronické dokumenty; 5.3.1 Zapnutí interaktivity v dokumentech; 5.3.2 Základní konfigurace pro interaktivitu; 5.4 Hypertextové odkazy na externí dokumenty; 5.4.1 Příkazy, které pomáhají při sazbě hypertextových odkazů, ale nevytvářejí je; 5.4.2 Příkazy; 5.5 Vytvoření záložek ve finálním PDF;

5.1 Referenční typy

Vědecké a technické dokumenty obsahují velké množství odkazů:

- Někdy odkazují na jiné dokumenty, které jsou základem pro to, co je řečeno, nebo které jsou v rozporu s tím, co je vysvětlováno, nebo které rozvíjejí či dále rozvádějí myšlenku, o níž se pojednává, atd. V těchto případech se říká, že odkaz je *externí*, a pokud má být dokument akademicky přísný, má odkaz formu *citací* z literatury.
- Běžně se však stává, že dokument v jednom ze svých oddílů odkazuje na jiný oddíl, a v takovém případě se říká, že odkaz je *interní*. Interní odkaz se vyskytuje také tehdy, když je v dokumentu komentován určitý aspekt konkrétního obrázku, tabulky, poznámky nebo prvku podobné povahy, přičemž se na něj odkazuje číslem nebo stránkou, na které se nachází.

Z důvodu přesnosti je třeba, aby interní odkazy směřovaly na přesné a snadno identifikovatelné místo v dokumentu. Proto jsou tyto odkazy vždy odkazem buď na číslované prvky (jako například, když řekneme „viz tabulka 3.2“ nebo „kapitola 7“), nebo na čísla stránek. Neurčité odkazy typu „jak jsme již řekli“ nebo „jak uvidíme dále“ nejsou pravými odkazy a neexistuje žádný zvláštní požadavek, ani žádný zvláštní nástroj pro jejich sazbu. Osobně také odrazují

své studenty doktorského a magisterského studia od jakéhokoli obvyklého používání této praxe.

Interní odkazy se také běžně nazývají „křížové odkazy“, ačkoli v tomto dokumentu budu jednoduše používat termín „odkazy“ obecně a „interní odkazy“, pokud chci být konkrétní.

Abych objasnil terminologii, kterou používám pro odkazy, budu místo v dokumentu, kde je odkaz zaveden, nazývat *původ* a místo, na které ukazuje, *cíl*. Takto viděno bychom řekli, že odkaz je interní, pokud jsou původ a cíl ve stejném dokumentu, a externí, pokud jsou původ a cíl v různých dokumentech.

Z hlediska sazby dokumentu:

- Externí odkazy nepředstavují žádný zvláštní problém, a proto v zásadě nevyžadují žádný nástroj pro jejich zavedení: všechna potřebná data z cílového dokumentu jsou mi k dispozici a mohu je použít v referenci. Pokud je však výchozím dokumentem elektronický dokument a cílový dokument je rovněž dostupný na webu, pak je možné do odkazu zahrnout hypertextový odkaz, který umožňuje přejít přímo na cílový dokument. V těchto případech lze říci, že původní dokument je *interaktivní*.
- Naproti tomu interní odkazy představují výzvu pro sazbu dokumentu, protože každý, kdo má zkušenosti s přípravou středně dlouhých vědeckých a technických dokumentů, ví, že je téměř nevyhnutelné, aby se číslování stránek, oddílů, obrázků, tabulek, vět nebo podobných údajů, které jsou uvedeny v odkazu, v průběhu přípravy dokumentu měnilo, což velmi ztěžuje jeho aktualizaci.

V předpočítačových dobách se autoři vyhýbali vnitřním odkazům, a ty, které byly nevyhnutelné, jako například obsah (který, pokud je doplněn číslem stránky každé části, je příkladem vnitřního odkazu), se psaly na konci.

Dokonce i ty nejomezenější systémy pro sazbu, jako jsou textové procesory, umožňují zahrnout určitý druh vnitřních křížových odkazů, jako je například obsah. To však není nic ve srovnání s komplexním mechanismem správy odkazů obsaženým v ConTeXtu, který může kombinovat mechanismus správy interních odkazů zaměřený na udržování odkazů v aktuálním stavu s použitím hypertextových odkazů, což se samozřejmě netýká pouze externích odkazů.

5.2 Interní reference

K vytvoření vnitřního odkazu jsou zapotřebí dvě věci:

1. Štítek nebo identifikátor v cílovém bodě. ConTeXt při kompilaci přiřadí k tomu štítku konkrétní data. Jaké údaje budou přiřazeny, závisí na druhu štítku; může to být číslo sekce, číslo poznámky, číslo obrázku, číslo přiřazené k určité položce v číslovaném seznamu, název sekce atd.

2. Příkaz v bodě původu, který načte data přiřazená ke štítku spojeným s cílovým bodem a vloží je do bodu původu. Příkaz se liší podle toho, která data ze štítku chceme vložit do bodu původu.

Když přemýšlíme o referenci, děláme to ve smyslu „původ \rightarrow cíl“, takže by se mohlo zdát, že nejdříve by měly být vysvětleny záležitosti týkající se původu a teprve potom ty, které se týkají cíle. Domnívám se však, že je snazší pochopit logiku odkazů, pokud je vysvětlení obrácené.

5.2.1 Štítek v referenčním cíli

V této kapitole se pod pojmem *štítek* rozumí textový řetězec, který bude spojen s cílovým bodem odkazu a interně používán pro získání určitých informací o cílovém bodu odkazu, jako je například číslo stránky, číslo oddílu atd. Ve skutečnosti informace spojené s každým štítkem závisí na postupu jeho vytvoření. ConTeXt tyto štítky nazývá *reference*, ale myslím, že tento druhý termín, protože má mnohem širší význam, je méně jasný.

Štítek přidružený k cílovému odkazu:

- Potřebuje, aby každý potenciální cíl v dokumentu byl jedinečný, aby jej bylo možné bez pochybností identifikovat. Pokud použijeme stejný štítek pro různé cíle, ConTeXt nevyhodí chybu při kompilaci, ale způsobí, že všechny odkazy budou směřovat na první štítek, který najde (ve zdrojovém souboru), což bude mít vedlejší účinek, že některé z našich odkazů mohou být chybné, a co hůř, že si jich nevšimneme. Proto je důležité se při vytváření štítku ujistit, že nový štítek, který přiřazujeme, již nebyl přiřazen dříve.
- Může obsahovat písmena, číslice, interpunkční znaménka, prázdná místa atd. Tam, kde se vyskytují prázdná místa, stále platí obecná pravidla ConTeXtu týkající se těchto druhů znaků (viz. [sekce 2.2.1](#)), takže například „Můj pěkný štítek“ a „Můj pěkný štítek“ jsou považovány za stejné, i když je v obou použit jiný počet prázdných míst.

Protože není nijak omezeno, které znaky mohou být součástí štítku a kolik jich může být, radím používat takové názvy štítků, které jsou jasné a pomohou nám porozumět zdrojovému souboru, až ho třeba budeme číst dlouho poté, co byl původně napsán. Proto příklad, který jsem uvedl dříve („Můj pěkný štítek“), není dobrým příkladem, protože nám neříká nic o cíli, na který štítek ukazuje. Pro tento nadpis by byl například lepší štítek „sec:Cílové štítky“.

Pro přiřazení konkrétního cíle ke štítku existují v zásadě dva postupy:

1. Pomocí argumentu nebo volby příkazu se vytvoří prvek, na který bude ukazovat popis. Z tohoto hlediska všechny příkazy, které vytvářejí nějakou strukturu nebo textový prvek otevřený jako referenční cíl, obsahují volbu „reference“,

která se používá k zahrnutí štítku. Někdy je místo *volby* značkou obsah celého argumentu.

Dobrým příkladem toho, co se snažím říci, jsou příkazy sekce, které, jak víme z (sekce ??), umožňují několik druhů syntaxe. V klasické syntaxi je příkaz zapsán jako:

```
\section[Štítek]{Název}
```

a v syntaxi specifické pro ConT_EXt je příkaz zapsán jako

```
\startsection  
[title=Název, reference=Štítek, ... ]
```

V obou případech příkaz předpokládá zavedení štítku, který bude spojen s daným oddílem (nebo kapitolou, pododdílem atd.).

Řekl jsem, že tato možnost se nachází ve *všech příkazech*, které umožňují vytvořit textový prvek, který je možným cílem odkazu. Jedná se o všechny textové prvky, které lze číslovat, mimo jiné o oddíly, plovoucí objekty všeho druhu (tabulky, obrázky a podobně), poznámky pod čarou nebo poznámky na konci textu, citace, číslované seznamy, popisy, definice atd.

Pokud je štítek zadán přímo jako argument, a nikoli jako volba, které je přiřazena hodnota, je možné pomocí ConT_EXtu přiřadit několik štítků k jednomu cíli. Například:

```
\chapter[štítek1, štítek2, štítek3]{Moje kapitola}
```

Není mi jasné, jakou výhodu by mohlo mít několik různých označení pro jeden cíl, a mám podezření, že to lze udělat ne proto, že by to přinášelo výhody, ale kvůli nějakému *internímu* požadavku ConT_EXtu platnému pro určité druhy argumentů.

2. Pomocí příkazů `\pagereference`, `\reference` nebo `\textreference`, jejichž syntaxe je:

```
\pagereference[štítek]  
\reference[štítek]{Text}  
\textreference[štítek]{Text}
```

- Štítek vytvořený pomocí `\pagereference` umožňuje získat číslo stránky.
- Štítky vytvořené pomocí `\reference` a `\textreference` umožňují získat číslo stránky a text, který je s nimi spojen a který je uveden jako argument.

Jak v `\reference`, tak v `\textreference` text, který je spojen se značkou, zmizí jako takový z konečného dokumentu v místě, kde je umístěn příkaz (cíl reference), ale může být načten a znovu se objevit v místě původu reference.

Již dříve jsem uvedl, že každý štítek je spojen s určitými informacemi týkajícími se cílového bodu. Jaké informace to jsou, závisí na typu štítku, o který se jedná:

- Všechny štítky *pamatují* (v tom smyslu, že umožňují načtení) číslo stránky příkazu, který je vytvořil. U štítků připojených k oddílům, které mohou mít několik stránek, bude toto číslo odpovídat číslu stránky, na které daný oddíl začíná.
- Štítky vložené příkazem, který vytváří číslovaný textový prvek (oddíl, poznámku, tabulku, obrázek atd.) si *pamatují* číslo přiřazené k tomuto prvku (číslo oddílu, číslo poznámky atd.)
- Pokud má tento prvek *název*, jako je tomu například u sekcí, ale také u tabulek, pokud byly vloženy pomocí příkazu `\placetable`, budou si tento název pamatovat.
- Štítky vytvořené pomocí `\pagereference` si pouze *pamatují* číslo stránky.
- Příkazy vytvořené pomocí `\reference` nebo `\textreference` si *pamatují* také text, který je s nimi spojen a který tyto příkazy přijímají jako argument.

Ve skutečnosti si nejsem jistý skutečným rozdílem mezi příkazy `\reference` a `\textreference`. Domnívám se, že je možné, že návrh těchto tří příkazů, které umožňují vytváření štítků, se snaží běžet paralelně se třemi příkazy, které umožňují získávání informací ze štítků (což uvidíme za chvíli); ale pravdou je, že podle mých testů se `\reference` a `\textreference` zdají být nadbytečnými příkazy.

5.2.2 Příkazy v referenčním bodě původu pro získání dat z cílového bodu

Příkazy, které vysvětlím dále, načítají informace ze štítků a navíc, pokud je náš dokument interaktivní, vytvářejí spojení s referenčním cílem. Důležité na těchto příkazech jsou však informace, které se ze štítku načítají. Chceme-li pouze vygenerovat spojení, aniž bychom ze štítku načítali jakékoli informace, musíme použít příkaz `\goto`, který je vysvětlen v [sekci 5.4.2](#).

5.2.3 Základní příkazy pro získávání informací ze štítku

Vzhledem k tomu, že každý štítek spojený s cílovým bodem může obsahovat různé informace, je logické, že ConTeXt obsahuje tři různé příkazy pro získání těchto informací: podle toho, které informace z referenčního cílového bodu chceme získat, použijeme jeden nebo druhý z těchto příkazů:

- Příkaz `\at` umožňuje získat číslo stránky štítku.
- U štítků, které si kromě čísla stránky pamatují i číslo prvku (číslo oddílu, číslo poznámky, číslo položky, číslo tabulky atd.), umožňuje příkaz `\in` toto číslo získat.
- Konečně u štítků, které si pamatují text spojený se štítkem (název sekce, název obrázku vložený pomocí `\placefigure` atd.), umožňuje příkaz `\about` tento text načíst.

Tři příkazy `\at` `\in` `\about` mají stejnou syntaxi:

```
\at{Text}[štítek]
\in{Text}[štítek]
\about{Text}[štítek]
```

- Štítek je štítek, ze kterého chceme získat informace.
- Text je text napsaný těsně před informací, kterou chceme příkazem získat. Mezi text a údaje štítku, které příkaz načítá, se vloží neoddělitelná mezera, a pokud je funkce interaktivity zapnuta tak, že příkaz kromě načtení informace vygeneruje odkaz, který nám umožní přejít na cílové místo, bude text uvedený jako argument součástí odkazu (bude to text, na který lze kliknout).

V následujícím příkladu vidíme, jak `\in` získá číslo sekce a `\at` číslo stránky.

`V~\in{sekcí}[sec:target labels], která začíná na \at{page}[sec:target labels], jsou vysvětleny vlastnosti štítků používaných pro interní odkazy.`

V [sekci 5.2.1](#), která začíná na [page 129](#) jsou vysvětleny vlastnosti štítků používaných pro interní odkazy.

Všimněte si, že ConT_EXt automaticky vytvořil hypertextové odkazy (viz. [sekce 5.3](#)) a že text, který `\in` a `\at` přebírají jako argument, je součástí odkazu. Kdybychom to však napsali jinak, výsledek by byl:

`V~části \in{}[sec:targetlabels], která začíná na str. \at{}[sec:target labels], jsou vysvětleny vlastnosti štítků používaných pro interní odkazy.`

V části [5.2.1](#), která začíná na stránce [129](#), jsou vysvětleny vlastnosti štítků používaných pro interní odkazy.

Text zůstává stejný, ale slova *section* a *page*, která odkazu předcházejí, nejsou v odkazu obsažena, protože již nejsou součástí příkazu.

Pokud ConT_EXt není schopen najít popisek, na který ukazují příkazy `\at`, `\in` nebo `\about`, nedojde k chybě při kompilaci, ale tam, kde by se informace získané těmito příkazy měly objevit v konečném dokumentu, se objeví „??“.

Existují dva důvody, proč ConT_EXt nemůže najít štítek:

1. Při psaní jsme udělali chybu.
2. Zpracováváme pouze část dokumentu a značka ukazuje na dosud nezpracovanou část (viz. [sections 2.5.1 a 2.6](#)).

V prvním případě bude třeba chybu opravit. Proto je dobré, když dokončíme kompilaci celého dokumentu (a druhý případ již není možný), vyhledat v PDF všechny výskyty citace ?? a zkontrolovat, zda v dokumentu nejsou *rozbité* odkazy.

A. Získání informací spojených se štítkem pomocí příkazu `\ref`

Každý z `\at`, `\in` a `\about` načte některé prvky štítku. K dispozici je další příkaz, který nám umožňuje zachránit některý prvek štítku, který je uveden. Jedná se o příkaz `\ref`, jehož syntaxe je:

`\ref[Element k~načtení] [štítek]`

kde první argument může být:

- **text**: vrací text přiřazený štítku.
- **název**: vrací název přiřazený štítku.
- **number**: vrací číslo spojené se štítkem. Například v sekcích číslo sekce.
- **page**: vrací číslo stránky.
- **realpage**: vrací skutečné číslo stránky.
- **default**: vrací to, co ConT_EXt považuje za *přírozený* prvek štítku. Obecně se to shoduje s tím, co vrací **číslo**.

Ve skutečnosti je `\ref` mnohem přesnější než `\at`, `\in` nebo `\about`, a tak například rozlišuje mezi číslem stránky a skutečným číslem stránky. Číslo stránky se nemusí shodovat se skutečným číslem, pokud například číslování stránek dokumentu začalo na 1500 (protože tento dokument navazuje na předchozí) nebo pokud byly stránky preambule číslovány římskými číslicemi a při pohledu na ně bylo číslování znovu zahájeno. Podobně `\ref` rozlišuje *text* a *název* spojený s odkazem, což například `\about` nedělá.

Pokud je `\ref` použit k získání informací ze štítku, který takové informace neobsahuje (např. název štítku spojený s poznámkou pod čarou), vrátí příkaz prázdný řetězec.

B. Zjištění, kam vede odkaz

ConTeXt má také dva příkazy, které jsou citlivé na *adresu odkazu*. Pomocí „adresy odkazu“ chci určit, zda se cíl odkazu ve zdrojovém souboru nachází před nebo za původem. Příklad: píšeme náš dokument a chceme odkázat na sekci, která by se mohla nacházet ještě před nebo za sekci, kterou píšeme v konečném obsahu. Jen jsme se ještě nerozhodli. V této situaci by bylo užitečné mít příkaz, který zapíše jeden nebo druhý v závislosti na tom, zda cíl nakonec přijde před nebo za původ v konečném dokumentu. Pro takové potřeby ConTeXt nabízí příkaz `\somewhere`, jehož syntaxe je:

```
\somewhere{Text, pokud má být před}{Text, pokud má být za}[štítek].
```

Například v následujícím textu:

```
Adresu hypertextového odkazu lze také zjistit pomocí příkazu \type{\somewhere}.  
Tímto způsobem můžeme také najít kapitoly nebo jiné textové prvky \somewhere  
{před}{po} [sec:references] a diskutovat o jejich popisu na jiném místě  
\somewhere{před}{po} [sec:interaktivita].
```

Adresu hypertextového odkazu lze také zjistit pomocí příkazu `\somewhere`. Tímto způsobem můžeme najít kapitoly nebo jiné textové prvky `sek:odkazy` a diskutovat o jejich popisu na jiném místě `sek:interaktivita`.

Pro tento příklad jsem v této kapitole použil dva skutečné štítky ve zdrojovém souboru.

Dalším příkazem, který je schopen zjistit, zda štítek, na který ukazuje, je před nebo za, je `\atpage`, jehož syntaxe je:

```
\atpage[štítek]
```

Tento příkaz je docela podobný předchozímu, ale místo toho, aby nám umožnil napsat text sám, v závislosti na tom, zda je popisek před nebo za, `\atpage` vloží výchozí text pro každý z obou případů a, pokud je dokument interaktivní, vloží také hypertextový odkaz.

Text, který `\atpage` vloží, je text spojený se štítky „`precedingpage`“ v případě *štítek*, který přebírá jako argument, je *před* příkazem, a „`hereafter`“ v opačném případě.

Když jsem dospěl k tomuto bodu, zradilo mě předchozí rozhodnutí: v této kapitole jsem se rozhodl nazývat to, co ConTeXt nazývá „odkaz“, „štítek“. Zdálo se mi to jasnější. Ale některé textové fragmenty generované příkazy ConTeXt jako například `\atpage`, se také nazývají „štítky“ (tentokrát v jiném smyslu). (Viz [section 1.5.3](#)). Doufám, že to čtenáře nezmate. Myslím, že kontext nám umožňuje správně rozlišit, který z různých významů *štítek* mám v každém případě na mysli.

Proto můžeme text vložený pomocí `\atpage` měnit stejným způsobem jako text jakéhokoli jiného štítku:

```
\setuplabeltext[cs][precedingpage=Nový text]  
\setuplabeltext[cs][hereafter=Nový text]
```

V tomto bodě se domnívám, že je v „ConT_EXt Standalone“ (distribuce, kterou používám) malá chyba. Při zkoumání názvů předdefinovaných štítků v ConT_EXt které lze změnit pomocí `\setuplabeltext`, jsou dva páry štítků, které jsou kandidáty na použití pomocí `\atpage`:

- „předchozí stránka“ a „následující stránka“.
- „hencefore“ a „hereafter“.

Mohli bychom předpokládat, že `\atpage` použije buď první, nebo druhou dvojici. Ve skutečnosti však pro položky předcházející použije „precedingpage“ a pro ty následující použije „hereafter“, což je podle mého názoru nekonzistentní.

5.2.4 Automatické generování předpon pro zamezení duplicitních štítků

V rozsáhlém dokumentu není vždy snadné vyhnout se duplicitě značek. Proto je vhodné zavést určitý řád do způsobu, jakým vybíráme štítky, které použijeme. Jedním z postupů, který pomáhá, je používání předpon pro štítky, které se liší podle typu štítku. Například „sec:“ pro oddíly, „fig:“ pro obrázky, „tbl:“ pro tabulky atd.

S ohledem na tuto skutečnost obsahuje ConT_EXt soubor nástrojů, které umožňují:

- ConT_EXtu samotnému automaticky generovat popisky pro všechny přípustné prvky.
- Každému ručně generovanému štítku převzít předponu, ať už tu, kterou jsme si sami předem určili, nebo automaticky generovanou ConT_EXtem.

Podrobné vysvětlení tohoto mechanismu je zdlouhavé, a přestože jde nepochybně o užitečné nástroje, nemyslím si, že jsou nezbytné. A protože je nelze vysvětlit několika slovy, raději je nevysvětluji a odkazuji na to, co je o nich uvedeno v referenční příručce ConT_EXt nebo ve [wiki](#) o této problematice.

5.3 Interaktivní elektronické dokumenty

Interaktivní mohou být pouze elektronické dokumenty, ale ne všechny elektronické dokumenty. *Elektronický* dokument je takový, který je uložen v počítačovém souboru a lze jej otevřít a číst přímo na obrazovce. Na druhé straně elektronický dokument, který je vybaven nástroji, které umožňují uživateli s ním pracovat, je interaktivní, to znamená, že s ním můžeme dělat více než jen číst. O interaktivitu se jedná například tehdy, když dokument obsahuje tlačítka, která provádějí nějakou akci, nebo odkazy, jejichž prostřednictvím můžeme přejít na jiné místo v dokumentu nebo na externí dokument; nebo když jsou v dokumentu oblasti,

kam může uživatel psát, nebo jsou v něm videa či zvukové klipy, které lze přehrát, atd.

Všechny dokumenty generované ConT_EXtem jsou elektronické (protože ConT_EXt generuje PDF, které je z definice elektronickým dokumentem), ale ne vždy jsou interaktivní. Aby byly interaktivní, je nutné to výslovně uvést, jak je uvedeno v následující části.

Mějte však na paměti, že ačkoli ConT_EXt generuje interaktivní PDF, k tomu, abychom tuto interaktivitu ocenili, potřebujeme čtečku PDF, která ji umí, protože ne všechny čtečky PDF umožňují používat hypertextové odkazy, tlačítka a podobné prvky, které jsou pro interaktivní dokumenty vhodné.

5.3.1 Zapnutí interaktivity v dokumentech

ConT_EXt standardně nepoužívá interaktivní funkce, pokud to není výslovně uvedeno, což se obvykle děje v preambuli dokumentu. Příkaz, který tuto utilitu zapíná, je:

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

Za normálních okolností by se tento příkaz použil pouze jednou, a to v preambuli dokumentu, když chceme vygenerovat interaktivní dokument. Ve skutečnosti jej však můžeme používat jak často chceme, a to změnou stavu interaktivity dokumentu. Příkaz „state=start“ interaktivitu zapíná, zatímco „state=stop“ ji vypíná, takže můžeme interaktivitu vypnout v některých kapitolách nebo částech našeho dokumentu, kde to chceme.

Nenapadá mě žádný důvod, proč bychom chtěli mít v interaktivních dokumentech neinteraktivní části. Ale na filozofii ConT_EXtu je důležité, aby něco bylo technicky možné, i když to pravděpodobně nepoužijeme, takže nabízí postup, jak to udělat. Právě tato filosofie dává ConT_EXtu tolik možností a zabraňuje tomu, aby tak jednoduchý úvod, jako je tento, byl *stručný*.

Jakmile je interakce navázána:

- Některé příkazy ConT_EXtu již obsahují hypertextové odkazy. Tedy:
 - Příkazy pro vytváření obsahů, které budou v zásadě, pokud není výslovně uvedeno jinak, interaktivní, tj. kliknutím na položku v obsahu se přejde na stránku, kde začíná daná část.
 - Příkazy pro interní odkazy, které jsme viděli v první části této kapitoly, kdy kliknutím na ně automaticky přejdete na cíl odkazu.

- Poznámky pod čarou a závěrečné poznámky, kde se po kliknutí na kotvu poznámky v hlavním textu dostaneme na stránku, kde je poznámka napsána, a po kliknutí na značku poznámky v textu poznámky se dostaneme na místo v hlavním textu, kde bylo volání provedeno.
- atd.
- Je povolena možnost používat další příkazy určené speciálně pro interaktivní dokumenty, jako jsou například prezentace. Ty využívají řadu nástrojů spojených s interaktivitou, jako jsou tlačítka, nabídky, překryvy obrázků, vložený zvuk nebo video atd. Vysvětlení toho všeho by bylo příliš dlouhé a kromě toho jsou prezentace poněkud zvláštním druhem dokumentu. Proto na následujících řádcích popíšu jednu funkci spojenou s interaktivitou: hypertextové odkazy.

5.3.2 Základní konfigurace pro interaktivitu

`\setupinteraction`, kromě povolení nebo zakázání interakce, umožňuje nastavit některé záležitosti s ní spojené; především, ale nejen, barvu a styl odkazů. K tomu slouží následující volby příkazu:

- `color`: řídí *normální* barvu odkazů.
- `contrastcolor`: určuje barvu odkazů, jejichž cíl je na stejné stránce jako původ. Doporučuji, aby tento volitelný prvek byl nastaven na stejný obsah jako předchozí.
- `style`: řídí styl odkazu.
- `název`, `podnázev`, `autor`, `datum`, `klíčové slovo`: Hodnoty přiřazené těmto volbám budou převedeny do metadat PDF generovaného ConTeXtem.
- `kliknutí`: Tato volba určuje, zda se má odkaz při kliknutí na něj zobrazit.

5.4 Hypertextové odkazy na externí dokumenty

Budu rozlišovat mezi příkazy, které nevytvářejí odkaz, ale pomáhají zadat adresu URL odkazu, a příkazy, které vytvářejí hypertextový odkaz. Podívejme se na ně odděleně:

5.4.1 Příkazy, které pomáhají při sazbě hypertextových odkazů, ale nevytvářejí je

Adresy URL bývají velmi dlouhé a obsahují znaky všech typů, dokonce i znaky, které jsou v ConTeXtu vyhrazené a nelze je použít přímo. Kromě toho, pokud

je nutné URL zobrazit v dokumentu, je velmi obtížné odstavec napsat, protože URL může přesáhnout délku řádku a nikdy neobsahuje prázdná místa, která lze použít k vložení zalomení řádku. V adrese URL navíc není rozumné vkládat do slov spojovník, aby se vložil zlom řádku, protože čtenář by jen stěží poznal, zda spojovník skutečně tvoří součást adresy URL.

Proto ConTeXt poskytuje dva nástroje pro sazbu URL. První z nich je určena především pro adresy URL, které budou použity interně, ale nebudou zobrazeny v dokumentu. Druhý je určen pro adresy URL, které musí být zapsány v textu dokumentu. Podívejme se na něodděleně:

`\useURL`

Tento příkaz nám umožňuje zapsat adresu URL do preambule dokumentu a přiřadit ji ke jménu, takže když ji chceme použít v našem dokumentu, můžeme ji vyvolat pomocí jména, které je s ní spojeno. Je to užitečné zejména u adres URL, které budou v dokumentu použity několikrát.

Příkaz umožňuje dvě použití:

1. `\useURL[Přidružený název] [URL]`
2. `\useURL[Přidružený název] [URL] [] [Text odkazu]`

- V první verzi je adresa URL jednoduše spojena se jménem, pod kterým bude vyvolána v našem dokumentu. Ale pak, abychom mohli URL použít, budeme muset při jeho vyvolání nějak uvést, který klikací text se v dokumentu zobrazí.
- V druhé verzi obsahuje poslední argument klikací text. Třetí argument existuje pro případ, že chceme rozdělit adresu URL na dvě části, takže první část obsahuje přístupovou adresu a druhá část název konkrétního dokumentu nebo stránky, kterou chceme otevřít. Příklad: adresa dokumentu, který vysvětluje, co je to ConTeXt
<http://www.pragma-ade.com/general/manuals/what-is-context.pdf>. Tuto adresu můžeme zapsat celou do druhého argumentu a třetí část ponechat prázdnou:

```
\useURL [WhatIsCTX]
[http://www.pragma-ade.com/general/manuals/what-is-context.pdf]
[]
[Co je \ConTeXt?]
```

ale můžeme ji také rozdělit na dva argumenty:

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

[Co je \ConTeXt?]

V obou případech budeme mít tuto adresu spojenou se slovem „WhatIs-CTX“, takže pro vložení odkazu na tuto adresu použijeme příkaz, který používáme pro vytvoření odkazu; místo samotné adresy URL můžeme jednoduše napsat „WhatIsCTX“.

Chceme-li v kterémkoli místě textu reprodukovat adresu URL, kterou jsme spojili se jménem pomocí `\useURL`, můžeme použít `\url[Přidružené jméno]`, který vloží adresu URL spojenou s tímto jménem do dokumentu. Tento příkaz však, přestože zapíše adresu URL, nevytvoří žádný odkaz.

Formát, ve kterém se zobrazují adresy URL zapsané pomocí `\url`, není ten, který se obecně nastavuje pomocí `\setupinteraction`, ale ten, který se specificky nastavuje pro tento příkaz pomocí `\setupurl`, který umožňuje nastavit styl (volba `style`) a barvu (volba `colour`).

`\hyphenatedurl`

Tento příkaz je určen pro adresy URL, které budou zapsány v textu našeho dokumentu, a má ConTeXt zahrnout do adresy URL zalomení řádků, pokud je to nutné pro správnou sazbu odstavce. Jeho formát je:

`\hyphenatedurl{URLadresa}`

Navzdory názvu příkazu `\hyphenatedurl` se název adresy URL nepřevádí na pomlčku. Bere však v úvahu, že některé znaky běžné v adresách URL jsou vhodným místem pro vložení zalomení řádku před nebo za ně. Můžeme přidat požadované znaky do seznamu znaků, u kterých je povolen zlom řádku. K tomu máme k dispozici tři příkazy:

`\sethyphenatedurlnormal{Znaky}`
`\sethyphenatedurlbefore{Znaky}`
`\sethyphenatedurlafter{Znaky}`

Tyto příkazy přidávají znaky, které berou jako argumenty, do seznamu znaků podporujících zalomení řádku před seznam znaků podporujících pouze zalomení řádku a za seznam znaků umožňujících pouze zpětné zalomení řádku.

`\hyphenatedurl` lze použít vždy, když je třeba zapsat adresu URL, která se ve výsledném dokumentu objeví v nezměněné podobě. Lze jej dokonce použít jako poslední argument příkazu `\useURL` ve verzi tohoto příkazu, kde poslední argument vybírá text, na který lze kliknout a který se zobrazí v konečném dokumentu. Například:

`\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}]
```

V argumentu `\hyphenatedurl` lze použít všechny vyhrazené znaky kromě tří, které musí být nahrazeny příkazy:

- % musí být nahrazena `\letterpercent`
- # musí být nahrazen `\letterhash`
- \ musí být nahrazena položkou `\letterescape` nebo `\letterbackslash`.

Pokaždé, když `\hyphenatedurl` vloží zalomení řádku, provede příkaz `\hyphenatedurlseparator`, který ve výchozím nastavení nic neudělá. Pokud jej však nadefinujeme, vloží se do adresy URL reprezentativní znak podobně jako u normálních slov, kdy se vloží pomlčka, která označuje, že slovo pokračuje na dalším řádku. Například:

```
\def\hyphenatedurlseparator{\curvearrowright}
```

zobrazí následující obzvláště dlouhou webovou adresu:

```
https://support.microsoft.com/?scid=http://support.microsoft.com:80~  
/support/kb/articles/Q208/4/27.ASP&NoWebContent=1.
```

5.4.2 Příkazy

Pro vytvoření odkazů na předdefinované adresy URL pomocí `\useURL` můžeme použít příkaz `\from`, který se omezuje na vytvoření odkazu, ale nezapisuje žádný text, na který by bylo možné kliknout. Jako text odkazu se použije výchozí text v `\useURL`. Jeho syntaxe je:

```
\from[Název]
```

kde *Název* je název dříve přiřazený k adrese URL pomocí `\useURL`.

K vytvoření odkazů a jejich přiřazení ke klikacímu textu, který nebyl předem definován, slouží příkaz `\goto`, který se používá jak k vytváření interních, tak externích odkazů. Jeho syntaxe je:

```
\goto{Klikatelný text}[Cíl]
```

kde *Klikatelný text* je text, který se má zobrazit ve výsledném dokumentu a kde se kliknutím myši vytvoří skok, a *Cíl* může být:

- Štítek z našeho dokumentu. V tomto případě `\goto` vygeneruje skok podobným způsobem jako například již zkoumané příkazy `\in` nebo `\at`. Na rozdíl od těchto příkazů však nebudou získány žádné informace spojené se štítkem.

- Samotná adresa URL. V tomto případě musí být výslovně uvedeno, že se jedná o URL, a to zapsáním příkazu následujícím způsobem:

```
\goto{Klikatelný text}[url(URL)]
```

kde URL zase může být název, který byl dříve přiřazen k URL pomocí `\useURL`, nebo samotné URL, v tomto případě musíme při zápisu URL zajistit, aby byly vyhrazené znaky ConTeXtu správně zapsány v ConTeXtu. Zápis adresy URL podle pravidel ConTeXtu neovlivní funkčnost odkazu.

5.5 Vytvoření záložek ve finálním PDF

Soubory PDF mohou mít interní seznam záložek obsahu, který umožňuje čtenáři zobrazit obsah dokumentu ve zvláštním okně programu pro prohlížení PDF a pohybovat se v něm pouhým kliknutím na jednotlivé oddíly a pododdíly.

Ve výchozím nastavení ConTeXt neposkytuje výstupnímu PDF seznam záložek obsahu, ačkoli je to tak jednoduché, když do něj vložíte příkaz `\placebookmarks`, jehož syntaxe je:

```
\placebookmarks[Seznam sekcí]
```

kde *Seznam sekcí* je čárkou oddělený seznam úrovní sekcí, které se mají objevit v seznamu obsahu.

U tohoto příkazu mějte na paměti následující poznámky:

- Podle mých testů `\placebookmarks` nefunguje, pokud je v preambuli dokumentu. Ale v těle dokumentu (mezi `\starttext` a `\stoptext` nebo mezi `\startproduct` a `\stopproduct`) nezáleží na tom, kam jej umístíte: seznam záložek bude zahrnovat i oddíly nebo pododdíly před příkazem. Domnívám se však, že pro správné pochopení zdrojového souboru je nejrozumnější umístit příkaz na začátek.
- Typy oddílů definované uživatelem (pomocí `\definehead`) nejsou vždy umístěny na správném místě v seznamu záložek. Je vhodnější je vyloučit.
- Pokud název oddílu v kterémkoli oddílu obsahuje poznámku pod čarou nebo poznámku pod čarou, považuje se text poznámky pod čarou za součást záložky.
- Jako argument můžeme místo seznamu sekcí jednoduše uvést symbolické slovo „all“, které, jak jeho název napovídá, bude obsahovat všechny sekce; podle mých testů však toto slovo kromě toho, co jsou jistě sekce, obsahuje i texty umístěné tamtéž pomocí některých nesekcčních příkazů, takže výsledný seznam je poněkud nepředvídatelný.

Ne všechny programy pro prohlížení souborů PDF umožňují zobrazovat záložky a mnohé programy, které tuto funkci mají, ji nemají ve výchozím nastavení aktivovanou. Proto se pro kontrolu výsledku této funkce musíme ujistit, že náš program pro čtení PDF tuto funkci podporuje a má ji aktivovanou. Myslím, že si vzpomínám, že například Acrobat ve výchozím nastavení záložky nezobrazuje, přestože na jeho panelu nástrojů je tlačítko pro jejich zobrazení.

III

Vybrané oblasti úpravy dokumentu

Chapter 1

Znaky, slova, text a horizontální prostor

Table of Contents: 1.1 Získání znaků, které nejsou běžně dostupné z klávesnice; 1.1.1 Diakritika a speciální písmena; 1.1.2 Traditional ligatures; 1.1.3 Řecká písmena; 1.1.4 Různé symboly; 1.1.5 Definování znaků ; 1.1.6 Použití předdefinovaných sad symbolů; 1.2 Speciální formáty znaků; 1.2.1 Velká písmena, malá písmena a falešná malá písmena; 1.2.2 Text s horním nebo dolním indexem; 1.2.3 Doslovný text; 1.3 Mezery mezi znaky a slovy; 1.3.1 Automatické nastavení horizontálního prostoru; 1.3.2 Změna mezery mezi znaky ve slově; 1.3.3 Příkazy pro přidání vodorovné mezery mezi slovy; 1.4 Složená slova; 1.5 Jazyk textu; 1.5.1 Nastavení a změna jazyka; 1.5.2 Konfigurace jazyka; 1.5.3 Štítky spojené s jednotlivými jazyky ; 1.5.4 Některé příkazy související s jazykem; A Date-related commands; B Příkaz `\přeložit`; C Příkazy `\quote` a `\quotation`;

Základním prvkem všech textových dokumentů je znak: znaky jsou seskupeny do slov, která pak tvoří řádky, z nichž se skládají odstavce, z nichž se skládají stránky.

Tato kapitola, počínaje „*znakem*“, vysvětluje některé nástroje ConT_EXtu týkající se znaků, slov a textu.

1.1 Získání znaků, které nejsou běžně dostupné z klávesnice

V textovém souboru kódovaném jako UTF-8 (viz [sekce 2.1](#)) můžeme použít libovolný znak nebo symbol, a to jak z živých jazyků, tak z mnoha již zaniklých. Protože jsou však možnosti klávesnice omezené, většinu znaků a symbolů povolených v UTF-8 obvykle nelze získat přímo z klávesnice. To se týká zejména mnoha diakritických znamének, tj. znaků umístěných nad (nebo pod) určitými písmeny, které jim dávají zvláštní hodnotu; ale také mnoha dalších znaků, jako jsou matematické symboly, tradiční ligatury atd. Mnoho těchto znaků můžeme získat v ConT_EXtu pomocí příkazů.

1.1.1 Diakritika a speciální písmena

Téměř všechny západní jazyky mají diakritiku (s významnou výjimkou angličtiny) a klávesnice obecně umí generovat diakritiku odpovídající regionálním jazykům. Španělská klávesnice tedy dokáže vygenerovat veškerou diakritiku potřebnou pro španělštinu (v podstatě přízvuky a diaerézu) a také některá diakritická znaménka používaná v jiných jazycích, jako je katalánština (grave accents a cedillas) nebo francouzština (cedillas, grave a circumflex accents); ne však například některá znaménka používaná v portugalštině, jako je tilda na některých samohláskách ve slovech jako „navegação“.

T_EX byl navržen ve Spojených státech, kde klávesnice většinou neumožňují používat diakritiku; Donald Knuth mu proto dal sadu příkazů, které umožňují získat téměř všechny známé diakritické znaky (alespoň v jazycích používajících latinku). Pokud používáme španělskou klávesnici, nemá příliš smysl používat tyto příkazy k získání diakritiky, kterou lze získat přímo z klávesnice. Přesto je důležité vědět, že tyto příkazy existují a jaké jsou, protože španělská (nebo italská, nebo francouzská...) klávesnice nám neumožňuje generovat všechny možné diakritické znaky.

V tabulce ?? nalezneme příkazy a zkratky, které nám umožňují tuto diakritiku získat. Ve všech případech není důležité, zda použijeme příkaz nebo zkratku. V tabulce jsem jako příklad použil písmeno ‚u‘, ale tyto příkazy fungují s jakoukoli samohláskou (většina z nich¹) a také s některými souhláskami a polosamohláskami.

- Protože většina zkrácených příkazů jsou *řídící symboly* (viz sekce ??), může být písmeno, na které má diakritika dopadnout, napsáno bezprostředně za

¹ Z příkazů nalezených v tabulce ?? tilda nefunguje s písmenem ‚e‘, a nevím proč.

Jméno	Znak	Zkratka	Příkaz
Acute accent	ú	\'u	\uacute
Grave accent	ù	\`u	\ugrave
Circumflex accent	û	\^u	\ucircumflex
Dieresis or umlaut	ü	\"u	\udiaeresis, \uumlaut
Tilde	ũ	\~u	\utilde
Macron	ū	\=u	\umacron
Breve	ǔ	\u u	\ubreve

Tabulka 1.1 Přízvuky a další diakritická znaménka

příkazem nebo od něj odděleno. Tak například: pro získání portugalského ‚ã‘ můžeme napsat znaky `\=a` nebo `_a`.¹ Ale v případě breve (`\u`), kdy se jedná o *řídící slovo*, je prázdná mezera povinná.

- V případě dlouhé verze příkazu bude písmeno, na které diakritika připadá, prvním písmenem názvu příkazu. Takže například `\emacron` umístí makron nad malé písmeno ‚e‘ (ē), `\Emacron` udělá totéž nad velké písmeno ‚E‘. (Ē), zatímco `\Amacron` udělá totéž nad velkým písmenem ‚A‘. (Ā).

Zatímco příkazy v [tabulce 1.1](#) pracují se samohláskami a některými souhláskami, existují další příkazy pro generování některých diakritických znamének a speciálních písmen, které pracují pouze s jedním nebo několika písmeny. Jsou uvedeny v [tabulce 1.2](#).

Jméno	Znak	Zkratka	Příkaz
Skandinávské O	ø, Ø	\o, \O	
Skandinávské A	å, Å	\aa, \AA, {\r a}, {\r A}	\aring, \Aring
Polské L	ł, Ł	\l, \L	
Německé Eszett	ß	\ss, \SS	
‚i‘ and ‚j‘ bez tečky	ı, Ĳ	\i, \j	
Maďarské umlaut	ű, Ű	\H u, \H U	
Cedilla	ç, Ç	\c c, \c C	\ccedilla, \Ccedilla

Tabulka 1.2 More diacritics and special letters

Rád bych upozornil, že některé příkazy ve výše uvedené tabulce generují znaky z jiných znaků, zatímco jiné příkazy fungují pouze v případě, že font, který používáme, je pro daný znak výslovně určen. Takže pokud jde o německý Eszett (ß), v tabulce jsou uvedeny dva příkazy, ale pouze jeden znak, protože písmo, které zde pro tento text používám, poskytuje pouze velkou verzi německého Eszett (což je něco zcela běžného).

¹ Pamatujte, že v tomto dokumentu reprezentujeme prázdné mezery, pokud je důležité, aby byly vidět, pomocí `_`.

To je pravděpodobně důvod, proč nemohu dostat skandinávské A jako velké písmeno, i když „`\r A`“ a `\Aring` fungují správně.

Maďarská umlout funguje také s písmenem ,o‘ a cedilla s písmeny ,k‘, ,l‘, ,n‘, ,r‘, ,s‘ a ,t‘, v malém nebo velkém písmu. Příkazy, které se mají použít, jsou `\kcedilla`, `\lcedilla`, `\ncedilla` ... v tomto pořadí.

1.1.2 Traditional ligatures

Ligatura vzniká spojením dvou nebo více grafémů, které se obvykle píší samostatně. Toto spojení dvou znaků často začínalo jako druh zkratky v ručně psaných textech, až nakonec dosáhlo určité typografické nezávislosti. Některé z nich byly dokonce zařazeny mezi znaky, které jsou obvykle definovány v typografickém písmu, jako například ampersand, ,&‘, který vznikl jako zkrácení latinské kopuly (spojky) „et“, nebo německý Eszett (ß), který, jak jeho název napovídá, vznikl jako kombinace ,s‘ a ,z‘. V některých vzorech písma můžeme i dnes vystopovat původ těchto dvou znaků; nebo je možná vidím, protože vím, že tam jsou. Zejména u písma Pagella pro ,&‘ a u písma Bookman pro ,ß‘.

Jako cvičení doporučuji (po přečtení [Kapitoly 2](#), kde je vysvětleno, jak na to) zkusit reprezentovat tyto znaky těmito písmi v dostatečně velké velikosti (například 30 pt), aby bylo možné zjistit jejich složky.

Dalšími tradičními ligaturami, které se nestaly tak populárními, ale dodnes se občas používají, jsou latinské koncovky „oe“ a „ae“, které se občas psaly jako ,œ‘ nebo ,æ‘ na znamení, že v latině tvoří diftong. Těchto ligatur lze v ConTeXtu dosáhnout pomocí příkazů uvedených v [tabulce 1.3](#).

Ligatury	Zkratky	Příkaz
æ, Æ	<code>\ae</code> , <code>\AE</code>	<code>\aeligature</code> , <code>\AEligature</code>
œ, Œ	<code>\oe</code> , <code>\OE</code>	<code>\oeligature</code> , <code>\OEligature</code>

Tabulka 1.3 Tradiční ligatury

Ligatura, která bývala ve španělštině (kastilštině) tradiční a která se dnes v písmech obvykle nevyskytuje, je ,Ð‘: zkrácení ,D‘ a ,E‘. Pokud je mi známo, neexistuje v ConTeXtu žádný příkaz, který by nám umožnil toto použít,¹ ale můžeme si ho vytvořit, jak je vysvětleno v sekci ??.

Vedle předchozích ligatur, které jsem nazval *tradiční*, protože pocházejí z rukopisu, se po vynálezu knihtisku vyvinuly určité ligatury tištěného textu, které budu nazývat „typografické ligatury“, které ConTeXt považuje za pomůcky písma a které program spravuje automaticky, ačkoli můžeme ovlivnit, jak se s těmito pomůckami

¹ V L^AT_EXu naopak můžeme použít příkaz `\DH` implementovaný balíčkem „fontenc“.

písmena zachází (včetně ligatur) pomocí `\definefontfeature`. (v tomto úvodu není vysvětleno).

1.1.3 Řecká písmena

V matematických a fyzikálních vzorcích se běžně používají řecké znaky. Proto ConTeXt zahrnoval možnost generování celé řecké abecedy, velkých i malých písmen. Zde je příkaz postaven na anglickém názvu daného řeckého písmene. Pokud je první znak napsán malými písmeny, získáme malé řecké písmeno, a pokud je napsán velkými písmeny, získáme velké řecké písmeno. Například příkaz `\mu` vygeneruje malou verzi tohoto písmene (μ), zatímco `\Mu` vygeneruje velkou verzi (M). V [tabulce 1.4](#) vidíme, který příkaz generuje jednotlivá písmena řecké abecedy, malá i velká.

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

Tabulka 1.4 Řecká abeceda

Všimněte si, že u malých písmen některých znaků (epsilon, kappa, theta, pi, rho, sigma a phi) existují dvě možné varianty.

1.1.4 Různé symboly

Spolu se znaky, které jsme právě viděli, nabízí T_EX (a tedy i ConT_EXt) příkazy pro generování libovolného počtu symbolů. Takových příkazů je mnoho. Rozšířený, i když neúplný seznam jsem uvedl v Příloze ??.

1.1.5 Definování znaků

Pokud potřebujeme použít znaky, které nejsou dostupné z klávesnice, můžeme vždy najít webovou stránku s těmito znaky a zkopírovat je do zdrojového souboru. Pokud používáme kódování UTF-8 (jak je doporučeno), bude to fungovat téměř vždy. Ale také na wiki ConTeXtu je stránka s hromadou symbolů, které lze jednoduše zkopírovat a vložit do našeho dokumentu. Chcete-li je získat, klikněte [na tento odkaz](#).

Pokud však potřebujeme některý z daných znaků použít vícekrát, není kopírování a vkládání nejefektivnějším způsobem. Bylo by vhodnější definovat znak tak, aby byl spojen s příkazem, který jej pokaždé vygeneruje. K tomu slouží příkaz `\definecharacter`, jehož syntaxe je:

```
\definecharacter Jméno Znak
```

kde

- **Jméno** je jméno přiřazené novému znaku. Nemělo by to být jméno existujícího příkazu, protože by se tím tento příkaz přepsal.

je znak generovaný při každém spuštění `\Name`. Tento znak můžeme označit třemi způsoby:

- Jednoduchým zapsáním nebo vložením do zdrojového souboru (pokud jsme jej zkopírovali z jiného elektronického dokumentu nebo webové stránky).
- Uvedením čísla přiřazeného danému znaku v písmu, které právě používáme. Chceme-li zobrazit znaky obsažené v písmu a čísla s nimi spojená, můžeme použít `\showfont[Font name] command`.
- Vytvoření nového znaku pomocí některého z příkazů pro tvorbu složených znaků, které uvidíme bezprostředně poté.

Jako příklad prvního použití se na chvíli vraťme k oddílům věnovaným ligaturám (1.1.2). Tam jsem hovořil o tradiční ligatuře ve španělštině, kterou dnes v písmech obvykle nenajdeme: „Đ“. Tento znak bychom mohli přiřadit například k příkazu `\decontract`, takže se bude generovat vždy, když napíšeme `\decontract`. Uděláme to pomocí:

```
\definecharacter decontract Đ
```

Chceme-li vytvořit nový znak, který není v našem písmu a nelze jej získat z klávesnice, jako je tomu v případě příkladu, který jsem právě uvedl, musíme nejprve najít nějaký text, kde se tento znak nachází, zkopírovat jej a být schopni jej vložit do naší definice. Ve skutečném příkladu, který jsem právě uvedl, jsem původně zkopíroval „Đ“ z Wikipedie.

ConT_EXt obsahuje také některé příkazy, které umožňují vytvářet složené znaky a které lze použít v kombinaci s `\definecharacter`. Složenými znaky myslím znaky, které mají také diakritiku. Příkazy jsou následující:

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

Například: jak již víme, ConT_EXt má ve výchozím nastavení pouze příkazy pro psaní určitých písmen s cedulkou (c, k, l, n, r, s y t), která jsou obvykle součástí písem. Pokud bychom chtěli použít ‚b‘, mohli bychom použít příkaz `\buildtextcedilla` takto:

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

Tento příkaz vytvoří nový příkaz `\bcedilla`, který vygeneruje ‚b‘ s cedulkou: ‚b‘. Tyto příkazy doslova ‚vybudují‘ nový znak, který bude vygenerován, i když ho naše písmo nemá. Tyto příkazy slouží k překrytí jednoho znaku jiným a následnému pojmenování tohoto překrytí.

Při testech se mi nepodařilo zprovoznit `\buildmathaccent` ani `\buildtextognek`. Proto se o nich od této chvíle již nebudu zmiňovat.

`\buildtextaccent` přijme jako argumenty dva znaky a jeden z nich překryje druhým, přičemž jeden z nich mírně zvýší. Ačkoli se nazývá „buildtextaccent“, není nutné, aby některý ze znaků přijatých jako argumenty byl přízvukem; překrytí však poskytne lepší výsledky, pokud tomu tak je, protože v tomto případě se překrytím přízvuku na znak sníží pravděpodobnost, že se přízvuk přepíše. Na druhou stranu překrývání dvou znaků, které mají za normálních podmínek stejnou základní linii, je ovlivněno tím, že příkaz jeden ze znaků mírně vyzdvihne nad druhý. Proto nemůžeme tento příkaz použít například k získání výše zmíněné zkratky ‚Ð‘, protože pokud napíšeme např.

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

v našem zdrojovém souboru, mírné vyvýšení nad základní linii ‚D‘, které tento příkaz vytváří, znamená, že efekt (‚Ð‘), který vytváří, není příliš dobrý. Pokud to však výška znaků umožňuje, mohli bychom vytvořit kombinaci. Například,

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

by definoval znak ‚_‘, který by byl spojen s příkazem `\unusual`.

Ostatní příkazy pro sestavení mají jediný argument - znak, ke kterému diakritika vygenerovaná jednotlivými příkazy přidá. Níže uvedu příklad každého z nich, sestaveného na základě písmene ‚z‘:

- `\buildtextbottomcomma` přidá čárku pod znak, který bere jako argument (`,z'`).
- `\buildtextbottomdot` přidá tečku pod znak, který přebírá jako argument (`,z'`).
- `\buildtextcedilla` přidá cedilku pod znak, který přebírá jako argument (`,z'`).
- `\buildtextgrave` přidá nad znak, který přebírá jako argument, velké přízvučné znaménko (`ẑ`).
- `\buildtextmacron` přidá malý proužek pod znak, který bere jako argument (`z̄`).

Na první pohled se zdá, že příkaz `\buildtextgrave` je zbytečný, protože máme příkaz `\buildtextaccent`; pokud však zkontrolujete přízvuk grave vytvořený prvním z těchto dvou příkazů, vypadá to o něco lépe. Následující příklad ukazuje výsledek obou příkazů při dostatečné velikosti písma, aby bylo možné ocenit rozdíl:

$\grave{z} - \grave{\grave{z}}$

1.1.6 Použití předdefinovaných sad symbolů

„ConTeXt Standalone“ obsahuje spolu se samotným ConTeXtem řadu předdefinovaných sad symbolů, které můžeme v dokumentech používat. Tyto sady se nazývají „cc“, „cow“, „fontawesome“, „jmn“, „mvs“ a „nav“. Každá z těchto množin obsahuje také některé podmnožiny:

- **cc** zahrnuje „cc“.
- **cow** zahrnuje „cownormal“ a „cowcontour“.
- **fontawesome** zahrnuje „fontawesome“.
- **jmn** zahrnuje „navigation 1“, „navigation 2“, „navigation 3“ a „navigation 4“.
- **mvs** zahrnuje „astronomic“, „zodiac“, „europe“, „martinvogel 1“, „martinvogel 2“ and „martinvogel 3“.
- **nav** zahrnuje „navigation 1“, „navigation 2“ a „navigation 3“.

Wiki také uvádí sadu nazvanou **was**, která zahrnuje „wasy general“, „wasy music“, „wasy astronomy“, „wasy astrology“, „wasy geometry“, „wasy physics“ a „wasy apl“. Ale ve své distribuci jsem je nenašel a mé testy, kterými jsem se k nim snažil dostat, selhaly.

Pro zobrazení konkrétních symbolů obsažených v každé z těchto sad se používá následující syntaxe:

```
\usesymbols[Set]
\showsymbolset[Subset]
```

Například: chceme-li zobrazit symboly obsažené v „mvs/zodiac“, pak do zdrojového souboru musíme zapsat:

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

a dostaneme následující výsledek:

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

Všimněte si, že u každého symbolu je uveden nejen jeho název, ale i symbol. Příkaz `\symbol` nám umožňuje použít kterýkoli ze symbolů. Jeho syntaxe je:

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

kde podmnožina je jedna z podmnožin přidružených k některé z množin, které jsme dříve načetli pomocí `\usesymbols`. Pokud bychom například chtěli použít astrologický symbol spojený s Vodnářem (nalezený v mvs/zodiac), museli bychom napsat

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

čímž získáme „Vodnář“, která se pro všechny účely považuje za „znak“, a proto je ovlivněna velikostí písma, která je aktivní při tisku. Můžeme také použít `\definecharacter` pro přiřazení daného symbolu k příkazu. Například

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

vytvoří nový příkaz `\Aries`, který vygeneruje znak „Aries“.

Tyto symboly bychom mohli použít také například v prostředí seznamu. Například:

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

vytvoří

```
věc
věc
  věc
  věc
věc
```

1.2 Speciální formáty znaků

Přesněji řečeno jsou to příkazy *formátu*, které ovlivňují použité písmo, jeho velikost, styl nebo variantu. Tyto příkazy jsou vysvětleny v kapitole [\[sec:fontscol\]](#). Avšak při širším pohledu můžeme za formátovací příkazy považovat i příkazy, které nějakým způsobem mění znaky, které berou jako argument (a mění tak jejich vzhled). Na některé z těchto příkazů se podíváme v této části. Jiné, jako je podtržený nebo řádkovaný text s řádky nad nebo pod textem (např. tam, kde chceme poskytnout prostor pro odpověď na otázku), uvidíme v [sekci 3.5](#).

1.2.1 Velká písmena, malá písmena a falešná malá písmena

Samotná písmena mohou být velká nebo malá. Pro ConT_EXt jsou velká a malá písmena odlišné znaky, takže v zásadě bude psát písmena tak, jak je najde napsaná. Existuje však skupina příkazů, které nám umožňují zajistit, aby text, který berou jako argument, byl vždy napsán velkými nebo malými písmeny:

- `\word{text}`: převede text převzatý jako argument na malá písmena.
- `\Word{text}`: převede první písmeno textu převzatého jako argument na velké písmeno.

- `\Words{text}`: převede první písmeno každého ze slov přijatých jako argument na velká písmena; ostatní písmena jsou malá.
- `\WORD{text}` nebo `\WORDS{text}`: zapíše text převzatý jako argument velkými písmeny.

Velmi podobné těmto příkazům jsou `\cap` a `\Cap`: ty také píší text, který přebírají, s velkým písmenem. ale pak na něj použijí faktor škálování, který se rovná faktoru použitému v příkazu přípona „x“ v příkazech pro změnu písma (viz. sekce ??), takže ve většině písem budou velká písmena mít stejnou výšku jako malá písmena, což nám dává jakýsi *falešný* efekt malých písmen. Ve srovnání s pravými malými písmeny (viz [sekcí 2.5.2](#)) mají následující výhody:

1. `\cap` a `\Cap` budou fungovat s jakýmkoli písmem, na rozdíl od skutečných malých kapitálek, které fungují pouze s písmy a styly, které je výslovně obsahují.
2. Právě malé kapitálky jsou naopak variantou písma, která, jako taková je nekompatibilní s jakoukoli jinou variantou, jako je tučné písmo, kurzíva nebo šikmé písmo. Nicméně `\cap` a `\Cap` jsou plně kompatibilní s jakoukoliv variantou písma.

Rozdíl mezi `\cap` a `\Cap` spočívá v tom, že zatímco `\Cap` aplikuje škálovací faktor na všechna písmena slov, která tvoří jeho argument, `\Cap` nepoužije žádné škálování na první písmeno každého slova tak dosáhneme podobného efektu, jako když použijeme skutečné kapiálky v textu malými písmeny. Pokud se text, který se bere jako argument v „kapitálkách“ skládá z několika slov, velikost velkého písmene v příkazu prvního písmene každého slova bude zachována.

V následujícím příkladu tedy

```
OSN, jejíž \Cap{prezident} má svou  
kancelář v~sídle \cap{oSN}...
```

```
OSN, jejíž PREZIDENT má svou kancelář v sídle  
OSN...
```

musíme si především všimnout rozdílu ve velikosti mezi prvním „OSN“ (velkými písmeny) a podruhé (malými kapitálkami „OSN“). V příkladu jsem napsal `\cap{oSN}`. podruhé, abychom viděli, že nezáleží na tom, jestli argument napíšeme podruhé. který `\cap` přijímá, velkými nebo malými písmeny: příkaz převede všechny na velká písmena a pak použije škálovací koeficient; na rozdíl od příkazu `\Cap`, který první písmeno neškáluje.

Tyto příkazy mohou být také *vnořené*, v takovém případě by se škálování použilo ještě jednou, což by vedlo k dalšímu zmenšení, jako v následujícím příkladu, kde je slovo „kapitál“ v prvním řádku opět zmenšeno:

```
\cap{Lidé, kteří nashromáždili svůj
\cap{kapitál} na úkor ostatních
jsou nejčastěji
{\bf dekapitováni} v~revolučních
dobách}.
```

```
LIDÉ, KTERÍ NASHROMÁŽDILI SVŮJ KAPITÁL NA ÚKOR
OSTATNÍCH JSOU NEJČASTĚJI DEKAPITOVÁNÍ V RE-
VOLUČNÍCH DOBÁCH.
```

Příkaz `\nocap` aplikovaný na text, na který je aplikován `\cap`, zruší efekt `\cap` v textu, který je jeho argumentem. Například:

```
\cap{Když mi byl Jeden, právě jsem začínal, když mi byl Jeden, právě jsem začínal, když
když mi byly dva roky, byl jsem \nocap{nearly}nový (A. A. Milne)}}.
nový (A. A. Milne)}}.
```

Pomocí `\setupcapitals` můžeme nastavit, jak bude `\cap` fungovat, a můžeme také definovat různé verze příkazu, z nichž každá bude mít vlastní název a specifickou konfiguraci. To můžeme provést pomocí `\definecapitals`.

Oba příkazy fungují podobně:

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

Parametr „Name“ v `\setupcapitals` je nepovinný. Pokud není použit, konfigurace ovlivní samotný příkaz `\cap`. Pokud je použit, je třeba, aby název, který jsme předtím přiřadili v `\definecapitals`, byl přiřazen nějaké aktuální konfiguraci.

V obou příkazech, konfigurace umožňuje tři možnosti: „title“, „sc“ a „style“, přičemž první a druhý umožňuje jako hodnoty „yes“ a „no“. Pomocí „title“ určujeme, zda se psaní velkých písmen bude týkat i nadpisů (což se ve výchozím nastavení děje), a pomocí „sc“ určujeme, zda má být příkaz psán skutečně malými písmeny („yes“), nebo falešnými malými písmeny („no“). Ve výchozím nastavení se používají falešné malé kapitálky, což má tu výhodu, že příkaz funguje, i když používáte písmo, které nemá implementovány malé kapitálky. Třetí hodnota „style“ nám umožňuje určit příkaz stylu, který má být použit na text ovlivněný příkazem `\cap`.

1.2.2 Text s horním nebo dolním indexem

Již víme (viz [sekce 1.1](#)), že v matematickém režimu rezervované znaky „_“ a „^“ převedou znak nebo skupinu, která bezprostředně následuje, na horní nebo dolní index. Pro dosažení tohoto efektu mimo matematický režim obsahuje ConTeXt následující příkazy:

- `\high{Text}`: zapíše text, který bere jako argument, jako horní index.
- `\low{Text}`: zapíše text, který bere jako argument, jako dolní index.
- `\cmdlohi{Podpis}{Superscript}`: zapíše oba argumenty, jeden nad druhým: dole první argument a nahoře druhý, což přináší zajímavý efekt:

```
\lohi{dole}{hore} | hore
                   | dole
```

1.2.3 Doslovný text

Latinský výraz *verbatim* (z *verbum* = *word* + přípona *atim*), který by se dal přeložit jako „doslovně“ nebo „slovo za slovo“, se používá v programech pro zpracování textu, jako je ConTeXt pro označení fragmentů textu, které by se neměly vůbec zpracovávat, ale měly by se vysypat tak, jak jsou napsány, do výsledného souboru. ConTeXt k tomu používá příkaz `\typ` určený pro krátké texty, které nezabírají více než jeden řádek, a prostředí `typing` určené pro texty delší než jeden řádek. Tyto příkazy se hojně používají v počítačových knihách pro zobrazení fragmentů kódu a ConTeXt tyto texty formátuje monospace písmem, jako by to dělal psací stroj nebo počítačový terminál. V obou případech je text odeslán do konečného dokumentu bez *zpracování*, což znamená, že mohou používat vyhrazené znaky nebo speciální znaky, které budou ve výsledném souboru přepsány *tak, jak jsou*. Stejně tak, pokud argument `\type` nebo obsah `\starttyping` obsahuje příkaz, bude tento příkaz *zapsán* do konečného dokumentu, ale nebude proveden.

Příkaz `\type` má kromě toho následující zvláštnost: jeho argument *může* být obsažen v kudrnatých závorkách (jak je v ConTeXtobvyklé), ale k ohraničení (obklopení) argumentu lze použít jakýkoli jiný znak.

Když ConTeXt čte příkaz `\typ`, předpokládá, že znak, který není prázdnou mezerou bezprostředně za názvem příkazu, bude sloužit jako oddělovač jeho argumentu; proto se domnívá, že obsah argumentu začíná dalším znakem a končí znakem před dalším výskytem *oddělovače*.

K lepšímu pochopení nám pomůže několik příkladů:

```
\type 1Tweedledum a-Tweedledee1
\type |Tweedledum a-Tweedledee|
\type zTweedledum a-Tweedledeez
\type (Tweedledum a-Tweedledee(
```

Všimněte si, že v prvním příkladu je prvním znakem za názvem příkazu `,1'`, ve druhém `,|'` a ve třetím `,z'`; takže: v každém z těchto případů bude ConTeXt považovat za argument `\type` vše mezi tímto znakem a dalším výskytem téhož znaku. Totéž platí pro poslední příklad, který je také velmi poučný, protože v zásadě bychom mohli předpokládat, že pokud je úvodním oddělovačem argumentu `,('`, měl by být i závěrečným oddělovačem `,)'`, ale není tomu tak,

protože `(` a `)` jsou různé znaky a `\type`, jak jsem řekl, hledá závěrečný oddělovač, který je stejný jako znak použitý na začátku argumentu.

Existují pouze dva případy, kdy `\type` umožňuje, aby úvodní a závěrečný oddělovač byly různé znaky:

- Pokud je úvodním oddělovačem znak `{`, myslí si, že závěrečným oddělovačem bude `}`.
- Pokud je úvodním oddělovačem znak `<<`, myslí si, že uzavíracím oddělovačem bude `>>`. Tento případ je jedinečný také tím, že jako oddělovače jsou použity dva po sobě jdoucí znaky.

Nicméně: skutečnost, že `\type` umožňuje libovolný oddělovač, neznamená, že bychom měli používat oddělovače „weird“. Z hlediska *čitelnosti* a *srozumitelnosti* zdrojového souboru je nejlepší ohraničit argument `\type` kulatými závorkami, kde je to možné, jak je to běžné v ConTeXtu; a když to možné není, protože v argumentu `\type` jsou kulaté závorky, použít symbol: nejlépe takový, který není vyhrazeným znakem ConTeXtu. Například: `\type *Jedná se o uzavírací kudrnatou závorku: ‘}’*`.

Oba `\type` i `\starttyping` lze konfigurovat pomocí `\setuptype` a `\setuptyping`. Můžeme také vytvořit jejich přizpůsobenou verzi pomocí `\definetype` a `\definetyping`. Pokud jde o vlastní konfigurační možnosti těchto příkazů, odkazují na „`setup-cs.pdf`“ (v adresáři `tex/texmf-context/doc/context/documents/general/qrcs`).

Dva velmi podobné příkazy jako `\type` jsou:

- `\typ`: funguje podobně jako `\type`, ale nezakáže spojovník.
- `\tex`: příkaz určený pro psaní textů o T_EXu nebo ConT_EXtu: přidává zpětnou mezeru před text, který bere jako argument. Jinak se tento příkaz liší od `\type` tím, že zpracovává některé vyhrazené znaky, které najde v textu, který bere jako argument. Zejména s kudrnatými závorkami uvnitř `\tex` bude zacházet stejným způsobem, jakým s nimi obvykle zachází v ConT_EXt.

1.3 Mezery mezi znaky a slovy

1.3.1 Automatické nastavení horizontálního prostoru

Mezera mezi jednotlivými znaky a slovy (v T_EXu nazývaná *horizontální mezera*) je obvykle nastavena automaticky pomocí ConT_EXtu:

- Mezera mezi znaky, které tvoří slovo, je definována samotným písmem, které s výjimkou písem s pevnou šířkou obvykle používá větší či menší množství bílé plochy v závislosti na oddělovaných znacích, a tak například mezera mezi `,A‘` a `,V‘` (`,AV‘`) je obvykle menší než mezera mezi `,A‘` a `,X‘` (`,AX‘`). Kromě těchto

možných odchylek, které závisí na kombinaci příslušných písmen a jsou předem definovány písmem, je však mezera mezi znaky, které tvoří slovo, obecně pevnou a neměnnou mírou.

- Naproti tomu mezera mezi slovy na stejném řádku může být pružnější.
 - V případě slov v řádku, jejichž šířka musí být stejná jako šířka ostatních řádků v odstavci, je změna mezer mezi slovy jedním z mechanismů, které ConTeXt používá k dosažení řádků stejné šířky, jak je podrobněji vysvětleno v [sekci 2.3](#). V těchto případech ConTeXt vytvoří přesně stejnou vodorovnou mezera mezi všemi slovy v řádku (s výjimkou níže uvedených pravidel), přičemž zajistí, aby mezera mezi slovy v různých řádcích odstavce byla co nejpodobnější.
 - Kromě nutnosti zvětšit nebo zmenšit mezery mezi slovy, aby bylo možné řádky ospravedlnit, však ConTeXt v závislosti na aktivním jazyce bere v úvahu určitá typografická pravidla, podle nichž typografická tradice spojená s daným jazykem přidává na některých místech bílé místo navíc, jako je tomu například v některých částech anglické typografické tradice, která přidává bílé místo navíc za tečku.

Tyto dodatečné bílé mezery fungují v angličtině a možná i v některých dalších jazycích (i když je také pravda, že v mnoha případech se dnes vydavatelé v angličtině rozhodují, že za tečkou nebudou dělat mezera navíc), ale ne ve španělštině, kde je typografická tradice jiná. Proto můžeme tuto funkci dočasně povolit pomocí `\setupspacing[broad]` a zakázat pomocí `\setupspacing[packed]`. Můžeme také změnit výchozí konfiguraci pro španělštinu (a pro jakýkoli jiný jazyk včetně angličtiny), jak je vysvětleno v [sekci 1.5.2](#).

1.3.2 Změna mezery mezi znaky ve slově

Změna výchozí mezery pro znaky, které tvoří slovo, je z typografického hlediska považována za velmi špatný postup, s výjimkou nadpisů a titulků. ConTeXt však poskytuje příkaz pro změnu této mezery mezi znaky ve slově:¹ `\stretched`, jehož syntaxe je následující:

```
\stretched[Configuration]{Text}
```

kde *Konfigurace* umožňuje některou z následujících možností:

¹ Je velmi typické pro filozofii ConTeXtu, protože obsahuje příkaz pro něco, co sama dokumentace ConTeXtu nedoporučuje. Přestože se usiluje o typografickou dokonalost, cílem je také dát autorovi absolutní kontrolu nad vzhledem jeho dokumentu: zda je lepší nebo horší, je zkrátka na jeho odpovědnosti.

- **factor**: celé nebo desetinné číslo, které vyjadřuje vzdálenost, kterou je třeba získat. Nemělo by to být příliš vysoké číslo. Faktor 0,05 je již viditelný pouhým okem.
- **width**: udává celkovou šířku, kterou musí mít text předkládaný příkazu tak, aby příkaz sám vypočítal potřebné rozestupy pro rozložení znaků v tomto prostoru.

Podle mých testů, pokud je šířka nastavená pomocí možnosti `width` menší než šířka potřebná k zobrazení textu s *factor* rovným 0,25, možnost *width* a tento faktor jsou ignorovány. Hádám, že je to proto, že `\stretched` nám umožňuje pouze zvětšit mezeru mezi znaky ve slově, nikoliv ji zmenšit. Nerozumím však tomu, proč se jako minimální míra pro volbu `width` používá šířka potřebná k zobrazení textu s faktorem 0,25, a ne *natural width* textu (s faktorem 0).

- **style**: příkaz nebo příkazy stylu, které se použijí na text převzatý jako argument.
- **color**: barva, ve které bude text převzatý jako argument napsán.

V následujícím příkladu si tedy můžeme názorně ukázat, jak by příkaz fungoval při použití na stejnou větu, ale s různou šířkou:

```
\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

Na tomto příkladu je vidět, že rozložení horizontálního prostoru mezi jednotlivými znaky není rovnoměrné. Písmena ‚x‘ a ‚t‘ v „text“ a ‚e‘ a ‚b‘ v „test“ jsou vždy mnohem blíže u sebe než ostatní znaky. Nepodařilo se mi zjistit, proč k tomu dochází.

Při použití bez argumentů použije příkaz celou šířku řádku. Na druhou stranu v textu, který je argumentem tohoto příkazu, je příkaz `\\` nadefinován a místo zalomení řádku vloží vodorovnou mezeru. Například:

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

t e s t

t e x t

Výchozí konfiguraci příkazu můžeme přizpůsobit pomocí příkazu `\setupstretched`.



Neexistuje žádný příkaz `\definestretched`, který by nám umožnil nastavit přizpůsobené konfigurace spojené s názvem příkazu, nicméně v oficiálním seznamu příkazů (viz sekci ??) je uvedeno, že `\setupstretched` pochází z `\setupcharacterkerning` a že existuje `\definecharacterkerning`. Při mých testech se mi však nepodařilo nastavit žádnou přizpůsobenou konfiguraci pro `\stretched` pomocí poslední jmenovaného příkazu, i když musím přiznat, že jsem tomu také nevěnoval mnoho času.

1.3.3 Příkazy pro přidání vodorovné mezery mezi slova

Již víme, že pro zvětšení mezery mezi slovy je zbytečné. přidat dvě nebo více po sobě jdoucích mezer, protože ConTeXt absorbuje všechny po sobě jdoucí mezery, jak je vysvětleno v [sekcí 2.2.1](#). Pokud chceme zvětšit mezeru mezi slovy, musíme přejít k jednomu z následujících kroků příkazů, které nám to umožňují:

- `\,` vloží do dokumentu velmi malé prázdné místo (tzv. tenkou mezeru). Používá se například k oddělení tisíců v souboru čísel (např. 1 000 000) nebo k oddělení jednoduché uvozovky od dvojité uvozovky. Např: „`1\,473\,451`“ vytvoří „1 473 451“.
- `\space` nebo „`\`“ (zpětné lomítko následované mezerou, kterou jsem vzhledem k tomu, že se jedná o neviditelný znak, reprezentoval jako „`\`“) zavádí další mezeru.
- `\enskip`, `\quad` a `\qquad` vloží do dokumentu prázdné místo o velikosti půl *em*, 1 *em* nebo 2 *em*. Nezapomeňte, že *em* je míra závislá na velikosti písma a odpovídá šířce ‚m‘, která se obvykle shoduje s velikostí písma v bodech. Při použití písma o velikosti 12 bodů nám tedy `\enskip` poskytne mezeru 6 bodů, `\quad` 12 bodů a `\qquad` 24 bodů.

Vedle těchto příkazů, které nám poskytují prázdný prostor v přesných rozměrech, zavádějí příkazy `\hskip` a `\hfill` vodorovný prostor různých rozměrů:

`\hskip` nám umožňuje přesně určit, kolik prázdného místa chceme přidat. Tedy:

<code>Toto je \hskip 1cm 1 centimetr\\</code>	Toto je	1 centimetr
<code>Toto jsou \hskip 2cm 2 centimetri\\</code>	Toto jsou	2 centimetri
<code>Toto je \hskip 2.5cm 2.5 centimetru\\</code>	Toto je	2.5 centimetru

Uvedená mezera může být záporná, což způsobí překrytí jednoho textu druhým. Tedy:

<code>Je to spíše fraška než</code>	Je to spíše fraška
<code>\hskip -1cm komedie</code>	komedie

`\hfill` zase zavede tolik bílého místa, kolik je potřeba, aby zabíralo celý řádek, což nám umožňuje vytvářet zajímavé efekty, jako je text zarovnaný doprava, vycentrovaný text nebo text na obou stranách řádku, jak ukazuje následující příklad:

`\hfill Na pravé straně\\`
`Na obou \hfill stranách`

Na obou

Na pravé straně
 stranách

1.4 Složená slova

V této části mám na mysli slova, která jsou formálně chápána jako jedno slovo, a ne slova, která jsou jednoduše spojena. Není vždy snadné toto rozlišení pochopit: „rainbow“ se zjevně skládá ze dvou slov („rain + bow“), ale žádný anglický mluvčí člověk by o těchto spojených výrazech nepřemýšlel jinak než jako o jednom slově. Na druhé straně máme slova, která se někdy kombinují pomocí spojovníku nebo zpětného lomítka. Obě slova mají odlišný význam a použití, ale jsou spojena (a v některých případech se mohou stát jedním slovem, ale zatím ne!). Tak například můžeme najít slova jako „French–Canadian“ nebo „(inter)communication“ (i když můžeme také najít „intercommunication“ a zjistit, že mluvčí veřejnost nakonec přijala obě slova jako jediné slovo. Tak se jazyk vyvíjí).

Složená slova představují pro ConTeXt určité problémy, které souvisejí především s jejich možným spojováním na konci řádku. Pokud je spojovacím prvkem pomlčka, pak z typografického hlediska není problém s pomlčkou na konci řádku v tomto místě, ale museli bychom se vyhnout druhé pomlčce v druhé části slova, protože by nám zůstaly dvě po sobě jdoucí pomlčky, což by mohlo způsobit problémy s porozuměním.

Příkaz „|“ slouží k tomu, aby ConTeXt sdělil, že dvě slova tvoří složené slovo. Tento příkaz výjimečně nezačíná zpětným lomítkem a umožňuje dvě různá použití:

- Můžeme použít dvě po sobě jdoucí svislé čárky (fajfky) a napsat například „Spanish| |Argentine“.
- Dvě svislé čárky mohou mít spojovací /oddělovací prvek mezi dvěma slovy, jako například „spojovací|/|oddělovací“.

V obou případech ConTeXt pozná, že se jedná o složené slovo, a použije příslušná pravidla pro spojování tohoto typu slov. Rozdíl mezi použitím dvou po sobě jdoucích svislých čárek (fajfek) nebo orámováním oddělovače slov jimi je v tom, že v prvním případě ConTeXt použije oddělovač, který je předdefinován jako `\setuphyphenmark`, nebo jinými slovy pomlčku, která je výchozí („--“). Pokud tedy napíšeme „obrázek| |rámeček“, ConTeXt vygeneruje „Obrázek–rámeček“.

Pomocí `\setuphyphenmark` můžeme změnit výchozí oddělovač (v případě, že potřebujeme dvě roury). Přípustné hodnoty pro tento příkaz jsou „--“, „---“, „-“, „(,)“, „=“, „/“. Mějte však na paměti, že hodnota „=“ se stane pomlčkou (stejně jako „---“).

Obvyklé použití „|“ je s pomlčkami, protože se obvykle používá mezi složenými slovy. Příležitostně však může být oddělovačem závorka, pokud chceme například „(inter)space“, nebo to může být lomítko vpřed, jako v „vstup/výstup“. Pokud chceme, aby v těchto případech platila běžná pravidla pro spojování složených slov, můžeme napsat „(inter|)|prostor“ nebo „vstup|/|výstup“. Jak jsem již řekl, „|=|“ je považováno za zkratku „|---|“ a vkládá pomlčku jako oddělovač (—).

1.5 Jazyk textu

Znaky tvoří slova, která obvykle patří do některého jazyka. Pro ConT_EXt je důležité znát jazyk, ve kterém píšeme, protože na něm závisí řada důležitých věcí. Především:

- Spojování slov.
- Výstupní formát určitých slov.
- Určité záležitosti týkající se sazby spojené s tradicí sazby daného jazyka.

1.5.1 Nastavení a změna jazyka

ConT_EXt předpokládá, že jazykem bude angličtina. To lze změnit dvěma postupy:

- Pomocí příkazu `\mainlanguage`, který se používá v preambuli pro změnu hlavního jazyka dokumentu.
- Pomocí příkazu `\language`, který slouží ke změně aktivního jazyka na libovolném místě dokumentu.

Oba příkazy očekávají argument tvořený libovolným identifikátorem jazyka (nebo kódem). K identifikaci jazyka se používá buď dvoupísmenný mezinárodní kód jazyka uvedený v normě ISO 639-1, který je stejný jako kód používaný například na webu, nebo anglický název daného jazyka, případně někdy nějaká zkratka názvu v angličtině.

V [tabulce 1.5](#) nalezneme kompletní seznam jazyků podporovaných ConT_EXtem spolu s ISO kódem pro každý z těchto jazyků a případně i kódem pro některé výslovně uvedené jazykové varianty.¹

¹ [Tabulka 1.5](#) obsahuje shrnutí seznamu získaného pomocí následujících příkazů:

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

Pokud budete číst tento dokument dlouho po jeho napsání (2020), je možné, že ConT_EXt bude obsahovat další jazyky, takže by bylo dobré použít tyto příkazy pro zobrazení aktualizovaného seznamu jazyků

Jazyk	Kód ISO	Jazykové varianty
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	

Tabulka 1.5 Podpora jazyků v ConTeXtu

Chceme-li například nastavit španělštinu (kastilštinu) jako hlavní jazyk dokumentu, můžeme použít některou z následujících tří možností:

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

Pro aktivaci určitého jazyka *uvnitř* dokumentu můžeme použít buď příkaz `\language[Language code]`, nebo konkrétní příkaz pro aktivaci daného jazyka. Tak například `\en` aktivuje angličtinu, `\fr` aktivuje francouzštinu, `\es` aktivuje španělštinu, `\ca` aktivuje katalánštinu. Jakmile je aktuální jazyk aktivován, zůstane jím, dokud výslovně nepřepneme na jiný jazyk nebo dokud skupinu, ve které byl jazyk aktivován, nezavřeme. Jazyky tedy fungují stejně jako příkazy pro změnu písma.

Všimněte si však, že jazyk nastavený příkazem `\language` nebo některou z jeho zkratk (`\en`, `\fr`, `\de` atd.) nemá vliv na jazyk, ve kterém se tisknou popisky (viz sekce 1.5.3).

Ačkoli může být pracné označit jazyk všech slov a výrazů, které v dokumentu používáme a které nepatří do hlavního jazyka dokumentu, je důležité tak učinit, pokud chceme získat správně napsaný výsledný dokument, zejména v odborné práci. Neměli bychom označovat celý text, ale pouze tu část, která nepatří do hlavního jazyka. Někdy je možné označování jazyka automatizovat pomocí makra. Například pro tento dokument, v němž jsou průběžně citovány příkazy ConTeXt jejichž původním jazykem je angličtina, jsem navrhl makro, které kromě toho, že příkaz запиše v příslušném formátu a barvě, označí jej jako anglické slovo. Ve své odborné práci, kde potřebuji citovat velké množství francouzské a italské bibliografie, jsem do své bibliografické databáze začlenil pole pro zjišťování jazyka díla, abych mohl automatizovat uvádění jazyka v citacích a seznamech bibliografických odkazů.

Pokud v jednom dokumentu používáme dva jazyky, které používají různé abecedy (například angličtinu a řečtinu nebo angličtinu a ruštinu), existuje trik, díky kterému nebudeme muset označovat jazyk výrazů sestavených pomocí alternativní abecedy: upravte nastavení hlavního jazyka (viz další část) tak, aby se načetly i výchozí vzory spojovníku pro jazyk, který používá jinou abecedu. Chceme-li například používat angličtinu a starořečtinu, následující příkaz nás ušetří nutnosti označovat jazyk textů v řečtině:

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

Funguje to jen proto, že angličtina a řečtina používají jinou abecedu, takže v obou jazycích nemůže dojít k rozporu ve vzorcích spojovníku, a proto můžeme načíst oba jazyky současně. Ale ve dvou jazycích, které používají stejnou abecedu, povede současné načtení vzorů spojovníku nutně k nevhodnému spojování.

1.5.2 Konfigurace jazyka

ConTeXt spojuje fungování určitých nástrojů s konkrétním jazykem, který je v daném okamžiku aktivní. Výchozí přiřazení lze změnit pomocí `\setuplanguage`, jehož syntaxe je:

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

kde *Jazyk* je kód jazyka, který chceme konfigurovat, a *Konfigurace* obsahuje konkrétní konfiguraci, kterou chceme pro daný jazyk nastavit (nebo změnit). Konkrétně je povoleno až 32 různých konfiguračních možností, ale já se budu zabývat pouze těmi, které se zdají být vhodné pro úvodní text, jako je tento:

- **date:** umožňuje nastavit výchozí formát data. Viz dále na [page 165](#).
- **lefthyphenmin**, **rightthyphenmin:** minimální počet znaků, které musí být vlevo nebo vpravo, aby bylo podporováno spojování slov. Například `\setuplanguage[cs][lefthyphenmin=4]` neprovede pomlčku u žádného slova, pokud jsou nalevo od případné pomlčky méně než 4 znaky.

- **spacing**: možné hodnoty pro tuto volbu jsou „broad“ nebo „packed“. V prvním případě (široký) budou použita pravidla pro odstupňování slov v angličtině, což znamená, že za tečkou a dalším znakem bude přidáno určité množství prázdného místa navíc. Na druhé straně „spacing=packed“ zabrání použití těchto pravidel. Pro češtinu je výchozí nastavení broad.
- **leftquote, rightquote**: označují znaky (resp. příkazy), které `\quote` použije vlevo a vpravo od textu, který je jeho argumentem (pro tento příkaz viz strana 167).
- **leftquotation, rightquotation**: označuje znaky (nebo příkazy), které `\quotation` použije vlevo a vpravo od textu, který je jeho argumentem (pro tento příkaz viz strana 167).

1.5.3 Štítky spojené s jednotlivými jazyky

Mnoho příkazů ConTeXtu automaticky generuje určité texty (nebo *štítky*), jako například příkaz `\placetable`, který pod vloženou tabulku zapíše štítek „Table xx“, nebo `\placefigure`, který vloží štítek „Figure xx“.

Tyto štítky jsou citlivé na jazyk nastavený pomocí `\mainlanguage`. (ale ne, pokud jsou nastaveny pomocí `\language`) a můžeme je změnit pomocí příkazu

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

kde *Key* je výraz, podle kterého ConTeXt zná popisek, a *Label* je text, který má ConTeXt vygenerovat. Takže například,

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

uvidíte, že pokud je hlavním jazykem španělština, obrázky vložené pomocí `\placefigure` se nenazývají „Figure x“, ale „Imagen x“. Všimněte si, že za samotným textem na štítku musí být ponechána prázdná mezera, aby se zajistilo, že štítek nebude připojen k dalšímu znaku. V příkladu jsem použil vyhrazený znak „~“; mohl jsem také napsat „[figure=Imagen{ }]“ a uzavřít prázdnou mezeru mezi kudrnaté závorky, abych zajistil, že se jí ConTeXt nezbaví.

Jaké štítky můžeme předefinovat pomocí `\setuplabeltext`? Dokumentace ConTeXt není v tomto bodě tak úplná, jak bychom mohli doufat. Referenční příručka 2013 (která o tomto příkazu vysvětluje nejvíce) uvádí „chapter“, „table“, „figure“, „appendix“... a přidává „další srovnatelné textové prvky“. Můžeme předpokládat, že názvy budou anglickými názvy daného prvku.



Jednou z výhod *svobodného softwaru* je, že zdrojové soubory jsou dostupné uživateli, takže se do nich můžeme podívat. Udělal jsem to a *slíděním* ve zdrojových souborech ConTeXtu jsem objevil soubor „lang-txt. lua“, dostupný v `tex/texmf-context/tex/context/base/mkiv`, který podle mého názoru obsahuje předdefinované štítky a jejich různé překlady; takže pokud

kdykoli ConT_EXt vygeneruje nadefinovaný text, který chceme změnit, abychom viděli název štítku, který je s tímto textem spojen, můžeme otevřít dotyčný soubor a najít ten, který chceme změnit. Tímto způsobem můžeme zjistit, jaké jméno štítku je s ním spojeno.

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

Některé štítky, které lze nadefinovat pomocí `\setuplabeltext`, jsou ve výchozím nastavení prázdné; například „chapter“ nebo „section“. Je to proto, že ConT_EXt ve výchozím nastavení nepřidává štítky k příkazům pro vytváření sekcí. Chceme-li tuto výchozí operaci změnit, stačí tyto štítky nadefinovat v preambuli našeho dokumentu, a tak například `\setuplabeltext[chapter=Chapter~]` uvidí, že kapitolám předchází slovo „Chapter“.

Nakonec je důležité zdůraznit, že ačkoli obecně v ConT_EXtu příkazy, které umožňují jako argument několik čárkou oddělených možností, může poslední možnost končit čárkou a nic špatného se nestane. V `\setuplabeltext` by to při kompilaci vyvolalo chybu.

1.5.4 Některé příkazy související s jazykem

A. Date-related commands

ConT_EXt má tři příkazy související s datem, které v době svého spuštění vytvářejí výstup v aktivním jazyce. Jedná se o tyto příkazy:

- `\currentdate`: spuštěn bez argumentů v dokumentu, jehož hlavním jazykem je angličtina, vrací systémové datum ve formátu „Day Month Year“. Například: „11 September 2020“. Můžeme mu však také říci, aby použil jiný formát (jak by se stalo v USA a některých dalších částech anglicky mluvícího světa, které se řídí svým systémem uvádění měsíce před dnem, odtud nechvalně známé datum 11. září), nebo aby zahrnul název dne v týdnu (`weekday`), nebo aby zahrnul pouze některé prvky data (`day`, `month`, `year`).

Chcete-li označit jiný formát data, „dd“ nebo „day“ představují dny, „mm“ měsíce (ve formátu čísel), „month“ měsíce v abecedním formátu malými písmeny a „MONTH“ velkými písmeny. Pokud jde o rok, „yy“ zapíše pouze poslední číslice, zatímco „year“ nebo „y“ zapíše všechny čtyři. Pokud chceme mezi složkami data nějaký oddělovací prvek, musíme jej zapsat výslovně. Například

`\currentdate[weekday, dd, month]`

při spuštění 9. září 2020 napíše „středa 9. září“.

- `\date`: tento příkaz, spuštěný bez jakéhokoli argumentu, vytvoří přesně stejný výstup jako `\currentdate`, tedy aktuální datum ve standardním formátu. Jako argument však lze zadat konkrétní datum. K tomu slouží dva argumenty: prvním argumentem můžeme uvést den („d“), měsíc („m“) a rok („y“) odpovídající datu, které chceme reprezentovat, zatímco druhým argumentem (nepovinným) můžeme uvést formát data, které má být reprezentováno. Chceme-li například zjistit, který den v týdnu se setkali John Lennon a Paul McCartney, což je událost, která se podle Wikipedie odehrála 6. července 1957, můžeme napsat

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

a tak bychom zjistili, že se taková historická událost stala v sobotu.

- `\month` přijímá jako argument číslo a vrací název měsíce odpovídající tomuto číslu.

B. Příkaz `\přeložit`

Příkaz `přeložit` podporuje řadu frází spojených s určitým jazykem, takže do výsledného dokumentu bude vložena ta či ona fráze v závislosti na jazyce, který je v daném okamžiku aktivní. V následujícím příkladu je příkaz `translate` použit k přiřazení čtyř frází ke španělštině a angličtině, které jsou uloženy ve vyrovnávací paměti (pokud jde o prostředí `buffer`, viz sekce ??):

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

takže pokud vložíme vyrovnávací paměť `buffer` na místo v dokumentu, kde je aktivována španělština, budou přehrávány španělské fráze, ale pokud je na místě v dokumentu, kam je vyrovnávací paměť vložena, aktivována angličtina, budou vloženy anglické fráze. Tedy:

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

vygeneruje

Su carta de fecha

Su referencia

Nuestra referencia

Fecha

while

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

vygeneruje

Your letter dated

Your reference

Our reference

Date

C. Příkazy `\quote` a `\quotation`

Jedna z nejčastějších typografických chyb v textových dokumentech vzniká, když jsou uvozovky (jednoduché nebo dvojité) otevřeny, ale nejsou výslovně uzavřeny. Aby k tomu nedocházelo, ConTeXt nabízí příkazy `\quote` a `\quotation`, které citují text, který je jejich argumentem; `\quote` použije jednoduché uvozovky a `\quotation` použije dvojité uvozovky.

Tyto příkazy jsou jazykově citlivé v tom smyslu, že pro otevírání a zavírání uvozovek používají výchozí znaky nebo sadu příkazů pro daný jazyk (viz. [sekce 1.5.2](#)); takže například pokud chceme použít španělštinu jako výchozí styl pro dvojité uvozovky - guillemety nebo chevrons (úhlové závorky)) typické pro španělštinu, italštinu a francouzštinu, napíšeme:

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

Tyto příkazy však nezvládají vnořené uvozovky; ačkoli můžeme vytvořit nástroj, který to udělá, s využitím toho, že `\quote` a `\quotation` jsou skutečné aplikace toho, co ConTeXt nazývá *delimitedtext*, a že je možné definovat další aplikace pomocí `\definedelimitedtext`. Tedy následující příklad:

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

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

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

vytvoří tři příkazy, které umožní až tři různé úrovně citování. První úroveň s posttranními uvozovkami, druhá s dvojitými uvozovkami a třetí s jednoduchými uvozovkami.

Samozřejmě pokud používáme jako hlavní jazyk angličtinu, pak se automaticky použijí výchozí jednoduché a dvojité uvozovky (kudrnaté, nikoli rovné, jak najdete v tomto dokumentu!).

Chapter 2

Odstavce, řádky a svislá mezera

Table of Contents: **2.1 Odstavce a jejich charakteristika;** 2.1.1 Automatické odsazování prvních řádků odstavců; 2.1.2 Speciální odsazení odstavce; **2.2 Svislá mezera mezi odstavci;** 2.2.1 `\setupwhitespace`; 2.2.2 Odstavce, mezi kterými není žádná svislá mezera navíc; 2.2.3 Přidání dalšího svislého prostoru v určitém místě dokumentu; 2.2.4 `\setupblank` a `\defineblank`; 2.2.5 Další postupy pro dosažení většího vertikálního prostoru; **2.3 Jak ConTeXt vytváří řádky tvořící odstavce;** 2.3.1 Použití vyhrazeného znaku ‘~’; 2.3.2 Slovní spojení; 2.3.3 Úroveň tolerance pro zalomení řádků; 2.3.4 Vynucení zalomení řádku v určitém bodě; **2.4 Meziřádkový prostor;** **2.5 Ostatní záležitosti týkající se řádkování;** 2.5.1 Převod zalomení řádků ve zdrojovém souboru na zalomení řádků ve výsledném dokumentu; 2.5.2 Číslování řádků; **2.6 Horizontální a vertikální zarovnání;** 2.6.1 Horizontální zarovnání; 2.6.2 Vertikální zarovnání;

Celkový vzhled dokumentu je dán především velikostí a rozvržením stránek, které jsme si ukázali v kapitole [Chapter 1](#), zvoleným písmem, o němž pojednává kapitola [Chapter 2](#), a dalšími záležitostmi, jako jsou mezery mezi řádky, zarovnání odstavců a rozestupy mezi nimi atd. Tato kapitola se zaměřuje na tyto další záležitosti.

2.1 Odstavce a jejich charakteristika

Odstavec je pro ConTeXt základní jednotkou textu. Existují dva postupy pro začátek odstavce:

1. Vložení jednoho nebo více po sobě jdoucích prázdných řádků do zdrojového souboru.
2. Příkazy `\par` nebo `\endgraf`.

Obvykle se používá první z těchto postupů, protože je jednodušší a vytváří zdrojové soubory, které jsou lépe čitelné a srozumitelné. Vkládání odřádkování pomocí explicitního příkazu se obvykle provádí pouze uvnitř makra (viz sekce ??) nebo v buňce tabulky (viz sekce ??).

V dobře typizovaném dokumentu je z typografického hlediska důležité, aby se odstavce od sebe vizuálně odlišovaly. Toho se obvykle dosahuje dvěma postupy: mírným odsazením prvního řádku každého odstavce nebo mírným zvětšením mezery mezi odstavci, někdy i kombinací obou postupů, i když na některých místech se to nedoporučuje, protože je to považováno za nadbytečné.

S tím úplně nesouhlasím. Prosté odsazení prvního řádku ne vždy dostatečně vizuálně zvýrazní oddělení odstavců; zvětšení mezer, které není doprovázeno odsazením, však působí problémy v případě odstavce, který začíná na začátku stránky, a my si proto nemusíme být jisti, zda se jedná o nový odstavec, nebo o pokračování z předchozí stránky. Kombinace obou postupů pochybnosti odstraňuje.

Podívejme se nejprve, jak se pomocí ConTeXtu provádí odsazování řádků a odstavců.

2.1.1 Automatické odsazování prvních řádků odstavců

Automatické vkládání malé odrážky na první řádek odstavců je ve výchozím nastavení vypnuto. Můžeme ji povolit, opět zakázat a po jejím povolení určit rozsah odsazení pomocí příkazu `\setupin-denting`, který umožňuje určit, zda má být odsazení povoleno, nebo ne:

- **always**: všechny odstavce budou odsazeny bez ohledu na to, zda jsou odsazeny.
- **yes**: povolí *normální* odsazení odstavce. Některé odstavce, kterým předchází dodatečná svislá mezera, například první odstavec sekce nebo odstavce následující za určitými prostředími, nebudou odsazeny.
- **no**, **not**, **never**, **none**: vypne automatické odsazování prvního řádku v odstavcích.

V případech, kdy jsme povolili automatické odsazování, můžeme stejným příkazem také určit, jak velké odsazení má být. K tomu můžeme výslovně použít rozměr (například 1,5 cm) nebo symbolická slova “small”, “medium” a “big”, která označují, že chceme malé, střední nebo velké odsazení.

V některých typech písma (mimo jiné ve španělštině) bylo výchozí odsazení dva čtverčíky. V typografii byl čtverčík (původně *quad*, *quadrat*) kovový *distanční prvek* používaný při sazbě na stroji. Termín se později ujal jako obecné označení pro dvě běžné velikosti mezer v typografii, bez ohledu na použitou formu sazby. Čtverčík je mezera, která je široká jedno em; stejně široká jako výška písma (Wikipedie). U dvanáctibodového písma by tedy čtverčík měl šířku 12 bodů a výšku 12 bodů. ConTeXt má dva čtverčíkové příkazy: `\quad`, který generuje jednu mezeru výše uvedeného druhu, a `\qqquad`, který generuje dvojnásobek tohoto množství, ale na základě použitého písma. Odsazení dvou čtveřic s písmenem o velikosti 11 bodů bude měřit 22 bodů a s písmenem o velikosti 12 bodů 24 bodů.

Pokud je odsazování povoleno a nechceme, aby byl určitý odstavec odsazen, musíme použít příkaz `\noindentation`.

Obecně ve svých dokumentech zapínám automatické odsazování pomocí `\setupindenting[yes, big]`. V tomto dokumentu jsem to však neudělal, protože kdybych odsazování povolil, velký počet krátkých vět a příkladů by vedl k vizuálně nepřehlednému vzhledu stránek.

2.1.2 Speciální odsazení odstavce

Jedním z grafických postupů pro zvýraznění odstavce je odsazení pravé nebo levé (nebo obou) strany odstavce. To se používá například pro blokové citace.

ConTeXt má prostředí, které nám umožňuje změnit odsazení odstavce a zvýraznit text v odstavci. Jedná se o prostředí “**narrower**”:

```
\startnarrower[Options] ... \stopnarrower
```

kde *Options* může být:

- **left**: odsazení levého okraje.
- **Num*left**: odsazení levého okraje vynásobením *normal* odsazení číslem *Num*. (například **2*left**).
- **right**: odsazení pravého okraje.
- **Num*right**: odsazení pravého okraje vynásobením *normal* odsazení číslem *Num*. (například **2*right**).
- **middle**: odsazení obou okrajů. Toto je výchozí nastavení.
- **Num*middle**: odsazení obou stran, přičemž se odsazení *normal* vynásobí číslem *Num*.

Při vysvětlování možností jsem se zmínil o *odsazení normal*; to se týká velikosti levého a pravého odsazení, které “**narrower**” používá ve výchozím nastavení. Toto *množství* lze nakonfigurovat pomocí nástroje `\setupnarrower`, který umožňuje následující konfigurační možnosti:

- **left**: velikost odsazení, které se použije na levý okraj.
- **right**: velikost odsazení pravého okraje.
- **middle**: velikost odsazení, které se použije na oba okraje.
- **before**: příkaz, který se spustí před vstupem do prostředí.
- **after**: příkaz, který se spustí po opuštění prostředí.

Pokud chceme v dokumentu používat různé konfigurace užšího prostředí, můžeme každé z nich přiřadit jiný název pomocí `\definennarrower[Name][Configuration]`.

kde *Name* je název spojený s touto konfigurací a kde *Configuration* umožňuje stejné hodnoty jako `\setupnarrower`.

2.2 Svislá mezera mezi odstavci

2.2.1 `\setupwhitespace`

Jak již víme z (sekce 2.2.2), ConTeXtu nezáleží na tom, kolik po sobě jdoucích prázdných řádků je ve zdrojovém souboru: jeden nebo více prázdných řádků vloží do konečného dokumentu jeden zlom odstavce. Chcete-li zvětšit mezeru mezi odstavci, nepomůže přidání dalšího prázdného řádku do zdrojového souboru. Místo toho je tato funkce řízena příkazem `\setupwhitespace`, který umožňuje následující hodnoty:

- **none**: znamená, že mezi odstavci nebude žádná další svislá mezera.
- **small**, **medium**, **big**: vloží malou, střední nebo velkou vertikální mezeru. Skutečná velikost mezery vložené těmito hodnotami závisí na velikosti písma.
- **line**, **halfline**, **quarterline**: změří dodatečné prázdné místo v poměru k výšce řádků a vloží řádek navíc, půlřádek nebo čtvrtřádek navíc.
- **DIMENSION**: určuje skutečný rozměr mezery mezi odstavci. Například `\setupwhitespace[5pt]`.

Obecně platí, že není vhodné nastavovat přesný rozměr jako hodnotu pro `\setupwhitespace`. Je vhodnější používat symbolické hodnoty `small`, `medium`, `big`, `halfline` nebo `quarterline`. Je tomu tak ze dvou důvodů:

- Symbolické hodnoty jsou pružné rozměry (viz sekce 1.8.2), což znamená, že mají *normální* rozměry, ale je povoleno určité zmenšení nebo zvětšení této hodnoty, což pomáhá ConTeXtu při sazbě stránek tak, aby zlomy odstavců byly esteticky podobné. Pevná míra odstupu mezi odstavci však ztěžuje dosažení dobrého stránkování dokumentu.
- Symbolické hodnoty `small`, `medium`, `big`, atd. se vypočítávají na základě velikosti písma, takže pokud se v určitých částech změní, změní se i velikost svislých mezer mezi odstavci a konečný výsledek bude vždy harmonický. Naopak pevně danou hodnotu svislého odstupu změny velikosti písma neovlivní, což se obvykle projeví v dokumentu se špatně rozloženým bílým místem (z estetického hlediska) a neodpovídajícím pravidlům typografické úpravy.

Pokud byla nastavena hodnota pro vertikální řádkování odstavců, jsou k dispozici dva další příkazy: `\nowhitespace`, který odstraňuje jakoukoli dodatečnou mezeru mezi jednotlivými odstavci, a `\whitespace`, který dělá opak. Tyto příkazy jsou však zapotřebí jen zřídka, protože ConTeXt samotný vertikální odstup mezi odstavci zvládá docela dobře, zejména pokud byl jeden z předdefinovaných rozměrů vložen jako hodnota, vypočtená z aktuální velikosti aktivního písma a odstupu.



Význam `\nowhitespace` je zřejmý. Ne však nutně samotný `\whitespace`, protože jaký smysl má nařizovat svislé řádkování pro konkrétní odstavce, když svislé řádkování již bylo obecně stanoveno pro všechny odstavce? Při psaní pokročilých maker však může být `\whitespace` užitečný v kontextu cyklu, který má učinit rozhodnutí na základě hodnoty určité podmínky. To je víceméně pokročilé programování a nebudu se jím zde zabývat.

2.2.2 Odstavce, mezi kterými není žádná svislá mezera navíc

Pokud chceme, aby určité části našeho dokumentu měly odstavce, které nejsou odděleny další svislou mezerou, můžeme samozřejmě upravit obecnou konfiguraci `\setupwhitespace`, ale to je svým způsobem v rozporu s filozofií ConTeXtu, podle které by obecné konfigurační příkazy měly být umístěny výhradně v preambuli zdrojového souboru, aby bylo dosaženo konzistentního a snadno změnitelného obecného vzhledu dokumentů. Proto prostředí “packed”, jehož obecná syntaxe je následující

```
\startpacked[Space] ... \stoppacked
```

kde *Space* je nepovinný argument udávající, jaká vertikální mezera je požadována mezi odstavci v prostředí. Pokud není uveden, nebude použita žádná svislá mezera navíc.

2.2.3 Přidání dalšího svislého prostoru v určitém místě dokumentu

Pokud v určitém místě dokumentu nestačí normální svislé mezery mezi odstavci, můžeme použít příkaz `\blank`. Při použití bez argumentů vloží `\blank` stejnou vertikální mezeru, jaká byla nastavena příkazem `\setupwhitespace`. Můžeme však uvést buď konkrétní rozměr mezi hranatými závorkami, nebo jednu ze symbolických hodnot vypočtených z velikosti písma: *small*, *medium* nebo *big*. Tyto rozměry můžeme také vynásobit celým číslem a tak dále, například `\blank[3*medium]` vloží ekvivalent tří středních řádkových zlomů. Můžeme také spojit dvě velikosti dohromady, například `\blank[2*big, medium]` vloží dva velké a střední zlomy.

Protože příkaz `\blank` je určen ke zvětšení svislé mezery mezi odstavci, nemá žádný účinek, pokud je mezi dva odstavce, jejichž mezera má být zvětšena, vložen stránkový zlom; a pokud vložíme dva nebo více příkazů `\blank` za sebou, použije se pouze jeden z nich (ten s největší mezerou, která má být vložena). Žádný účinek nemá ani příkaz `\blank` umístěný za zlomem stránky. V těchto případech však můžeme vložení svislé mezery vynutit pomocí symbolického slova “force” jako volby příkazu. Chceme-li tedy například, aby se názvy kapitol v našem dokumentu objevovaly dále na stránce, takže celková délka stránky bude menší než délka

ostatních stránek (což je poměrně častá typografická praxe), musíme v konfiguraci příkazu `\chapter` napsat např:

```
\setuphead
[chapter]
[
  page=yes,
  before={\blank[4cm, force]},
  after={\blank[3*medium]}
]
```

Tato sekvence příkazů zajistí, že kapitoly budou vždy začínat na nové stránce a že se popis kapitoly posune o čtyři centimetry dolů. Bez použití volby “force” to nebude fungovat.

2.2.4 `\setupblank` a `\defineblank`

Dříve jsem uvedl, že `\blank`, použitý bez argumentů, je ekvivalentní `\blank[big]`. To však můžeme změnit pomocí `\setupblank`, například nastavením `\setupblank[0,5cm]` nebo `\setupblank[medium]`. Při použití bez argumentů nastaví `\setupblank` hodnotu na velikost aktuálního písma.

Stejně jako u `\setupwhitespace` je bílé místo vložené pomocí `\blank`, pokud je jeho hodnota jednou z předdefinovaných symbolických hodnot, pružným rozměrem, který umožňuje určité přizpůsobení. Tuto hodnotu můžeme změnit pomocí “fixed” s možností pozdějšího obnovení výchozí hodnoty pomocí (“flexible”). Tak například pro text ve dvou sloupcích se doporučuje nastavit `\setupblank[fixed, line]` a při návratu k jednomu sloupci `\setupblank[flexible, default]`.

Pomocí `\defineblank` můžeme přiřadit určitou konfiguraci k názvu. Obecný formát tohoto příkazu je:

```
\defineblank[Name] [Configuration]
```

Jakmile je naše konfigurace bílého místa definována, můžeme ji použít pomocí `\blank[ConfigurationName]`.

2.2.5 Další postupy pro dosažení většího vertikálního prostoru

V \TeX u je příkazem, který vkládá vertikální mezeru navíc, `\vskip`. Tento příkaz, stejně jako téměř všechny příkazy \TeX u, funguje také v $\text{Con}\text{\TeX}$ tu, ale jeho použití se důrazně nedoporučuje, protože narušuje vnitřní fungování některých maker $\text{Con}\text{\TeX}$ tu. Místo něj se doporučuje použít `\godown`, jehož syntaxe je:

```
\godown[Dimension]
```

kde *Dimension* musí být číslo bez nebo s desetinnými místy, následované měrnou jednotkou. Například `\godown[5cm]` posune stránku o 5 centimetrů dolů; pokud je však změna stránky menší než toto množství, `\godown` se posune pouze na další stránku. Podobně `\godown` nebude mít žádný účinek na začátku stránky, ačkoli ho můžeme *obelstít* například tím, že napíšeme “`_ \godown[3cm]`”¹, který nejprve vloží prázdné místo, které bude znamenat, že již nejsme na začátku stránky, a poté se posune o tři centimetry dolů.

Jak víme, `\blank` umožňuje jako argument zadat i přesný rozměr. Z hlediska uživatele je tedy zápis `\blank[3cm]` nebo `\godown[3cm]` prakticky stejný. Existují však mezi nimi některé jemné rozdíly. Tak například dva po sobě jdoucí příkazy `\blank` nelze kumulovat, a pokud se tak stane, použije se pouze ten, který ukládá větší vzdálenost. Naproti tomu dva nebo více příkazů `\godown` se mohou dokonale kumulovat.

Dalším poměrně užitečným příkazem T_EXu, jehož použití vConT_EXtu nečiní žádné problémy, je `\vfill`. Tento příkaz vloží flexibilní vertikální prázdné místo až na konec stránky. Je to, jako by příkaz *zatlačil* dolů to, co je napsáno za ním. To umožňuje zajímavé efekty, například jak umístit určitý odstavec na dno stránky tím, že jej jednoduše předřadíte příkazu `\vfill`. Efekt `\vfill` je nyní obtížné ocenit, pokud jeho použití není kombinováno s vynucenými stránkovými zlomy, protože nemá smysl tlačit odstavec nebo řádek textu dolů, pokud odstavec, jak roste, roste nahoru.

Chceme-li například zajistit, aby byl řádek umístěn na konci stránky, měli bychom napsat:

```
\vfill
Řádek na spodní straně
\page[yes]
```

Stejně jako všechny ostatní příkazy, které vkládají svislou mezeru, nemá `\vfill` na začátku stránky žádný účinek. Můžeme jej však *obelstít* tím, že před něj vložíme vynucenou mezeru. Tak například:

```
\page[yes]
\vfill
Prostřední čára
\vfill
\page[yes]
```

vertikálně vycentruje větu “prostřední řádek” na stránce.

¹ Připomeňme si, že v tomto dokumentu používáme znak ‘`_`’ k vyjádření prázdného místa, pokud je pro nás důležité, abychom ho viděli.

2.3 Jak ConT_EXt vytváří řádky tvořící odstavec

Jedním z hlavních úkolů systému pro sazbu je převzít dlouhý řetězec slov a rozdělit jej na jednotlivé řádky vhodné velikosti. Například každý odstavec v tomto textu byl rozdělen na řádky o šířce 15 cm, ale autor se nemusel starat o takové detaily, protože ConT_EXt vybírá body zlomu po zvážení každého odstavce jako celku, takže poslední slova odstavce mohou skutečně ovlivnit rozdělení prvního řádku. Výsledkem je, že mezera mezi slovy v celém odstavci je co nejjednodušší.

Právě v tomto ohledu si můžeme nejlépe všimnout odlišného způsobu práce textových procesorů a lepší kvality dosažené v systémech, jako je ConT_EXt. Textový procesor totiž po dosažení konce řádku a přechodu na další upraví bílé místo v právě ukončeném řádku, aby umožnil pravé zarovnání. To dělá s každým řádkem a na konci bude mít každý řádek v odstavci jiný mezislovní odstup. To může způsobit velmi špatný efekt (např. ‘rivers’ bílé místo procházející textem). ConT_EXt naproti tomu zpracovává odstavec jako celek a pro každý řádek vypočítá, kolik bodů zlomu je přípustných, a velikost mezislovního odstupe, který by vznikl v důsledku zlomu řádku. Protože bod zlomu na řádku ovlivňuje potenciální body zlomu na dalších řádcích, může být celkový počet možností velmi vysoký; to však pro ConT_EXt nepředstavuje problém. Konečné rozhodnutí učiní na základě celého odstavce a zajistí, aby mezera mezi slovy na každém řádku byla *co nejjednodušší*, což vede k mnohem lepšímu sazbě odstavců; vizuálně jsou kompaktnější.

Za tímto účelem ConT_EXt testuje různé alternativy a každé z nich přiřadí hodnotu *badness* na základě jejích parametrů. Ty byly stanoveny po důkladném studiu umění typografie. Po prozkoumání všech možností nakonec ConT_EXt vybere nejméně vhodnou možnost (tu s nejmenší hodnotou špatnosti). Obecně to funguje docela dobře, ale nevyhnutelně se vyskytnou případy, kdy jsou vybrány body zlomu řádků, které nejsou nejlepší nebo které se nám nezdají být nejlepší. Proto někdy budeme chtít program říci, že některá místa nejsou dobrými body zlomu. V jiných případech pak budeme chtít vynutit zlom v určitém bodě.

2.3.1 Použití vyhrazeného znaku ‘~’

Hlavními kandidáty na body zlomu jsou samozřejmě bílé mezery mezi slovy. K označení toho, že určitá mezera by nikdy neměla být nahrazena zlomem řádku, používáme, jak již víme, vyhrazený znak ‘~’, který T_EX nazývá *tie*, spojující dvě slova dohromady.

Obecně se doporučuje používat tuto nezlomitelnou mezeru v následujících případech:

- Mezi částmi, které tvoří zkratku. Například U~S.
- Mezi zkratkami a termínem, na který se vztahují. Například Dr~Anne Ruben nebo p.~45.

- Mezi čísly a termínem, který je s nimi spojen. Například `Elizabeth~II, 45~volumes`.
- Mezi číslicemi a symboly, které jim předcházejí nebo za nimi následují, pokud se nejedná o horní indexy. Například `73~km, $~53`; avšak `35'`.
- V procentech vyjádřeno slovy. Například `dvacet~pro~cent`.
- Ve skupinách čísel oddělených bílým místem. Například `5~357~891`. Ačkoli v těchto případech je vhodnější použít tzv. *tenká mezera*, kterého se v ConTeXtu dosahuje příkazem `\,`, a tedy zapsat `5\,357\,891`.
- Aby zkratka nebyla jedinou položkou na daném řádku. Například:

`Existují odvětví, jako je zábava, komunikační média, obchod atd.`

K těmto případům KNUTH (otec T_EXu) přidává následující doporučení:

- Po zkratce, která není na konci věty.
- V odkazu na části dokumentu, jako jsou kapitoly, přílohy, obrázky atd. Například `Chapter~12`.
- Mezi křestním jménem a iniciálou druhého jména osoby nebo mezi iniciálou křestního jména a příjmením. Například `Donald~E. Knuth, A.~Einstein`.
- Mezi matematickými symboly ve spojení s názvy. Například `dimension~d, width~w`.
- Mezi symboly v sérii. Například `{1,~2, \dots,~n}`.
- Když se číslo striktně váže s předložkou. Například `od 0 do~1`.
- Když jsou matematické symboly vyjádřeny slovy. Například `rovná se~a~n`.
- V seznamech v rámci odstavce. Například: `(1)~zelená, (2)~červená, (3)~modrá`.

Mnoho případů? Typografická dokonalost má bezpochyby svou cenu v podobě dodatečného úsilí. Je jasné, že pokud nechceme, nemusíme tato pravidla uplatňovat, ale není na škodu je znát. Kromě toho - a tady mluvím ze své zkušenosti - jakmile si na jejich uplatňování (nebo na kterékoli z nich) zvykneme, stane se to automatickým. Je to podobné, jako když při psaní slov dáváme přízvuk (jak to musíme dělat ve španělštině): těm z nás, kteří to dělají, pokud jsme zvyklí psát je automaticky, netrvá napsání slova s přízvukem o nic déle než u slova bez přízvuku.

2.3.2 Slovní spojení

S výjimkou jazyků, které se skládají převážně z jednoslabičných slov, je poměrně obtížné dosáhnout optimálního výsledku, pokud jsou body přerušení řádku pouze v mezislovním prostoru. Proto ConTeXt analyzuje také možnost vložení řádkového zlomu mezi dvě slabiky slova; a k tomu je nezbytné, aby znal jazyk, ve kterém je text napsán, protože pravidla pro spojování jsou pro každý jazyk jiná. Proto je důležitý příkaz `\mainlanguage` v preambuli dokumentu.

Může se stát, že ConT_EXt nedokáže slovo vhodně spojit. Někdy to může být způsobeno tím, že jeho vlastní pravidla pro dělení slov stojí v cestě tomuto úkolu (ConT_EXt například nikdy nerozdělí slovo na dvě části, pokud tyto části nemají minimální počet písmen); nebo proto, že slovo je nejednoznačné. Koneckonců, co by ConT_EXt mohl udělat se slovem “unionised”? Toto slovo by se mohlo objevit ve větě jako “the unionised workforce”, ale mohlo by se také objevit v chemickém textu jako “an unionised particle”. (tj. un-ionised). A co kdyby se ConT_EXt musel vypořádat se slovem “manslaughter” jako posledním slovem na stránce před zlomem stránky. Může toto slovo rozdělit jako man-slaughter (správně), ale může ho také rozdělit jako mans-laughter (nejednoznačně).

Ať už je důvod jakýkoli, pokud nejsme spokojeni s tím, jak bylo slovo rozděleno, nebo je to nesprávné, můžeme to změnit tak, že výslovně označíme potenciální místa, kde lze slovo rozdělit pomocí kontrolního symbolu `\-`. Tak například, pokud by nám “unionised” dělalo problémy, mohli bychom jej ve zdrojovém souboru zapsat jako “union\ -ised”; nebo pokud bychom měli problém s “manslaughter”, mohli bychom zapsat “man\ -slaughter”.

Pokud je problémové slovo v našem dokumentu použito vícekrát, pak je vhodné ukázat, jak by mělo být v naší preambuli rozděleno pomocí příkazu `\hyphenation`: tento příkaz, který je určen k zařazení do preambule zdrojového souboru, přijímá jako argument jedno nebo více slov (oddělených čárkami) a určuje body, ve kterých mohou být rozdělena pomlčkou. Například:

```
\hyphenation{union-ised, man-slaughter}
```

Pokud slovo, které je předmětem tohoto příkazu, neobsahuje spojovník, bude to mít za následek, že slovo nebude nikdy spojeno. Stejného efektu lze dosáhnout použitím příkazu `\hbox`, který vytvoří kolem slova nedělitelný vodorovný rámeček, nebo příkazem `\unhyphenated`, který zabrání spojování slova nebo slov, která bere jako argumenty. Zatímco `\hyphenation` působí globálně, `\hbox` a `\unhyphenated` působí lokálně, což znamená, že příkaz `\hyphenation` ovlivňuje všechny výskyty slov obsažených v jeho argumentu v dokumentu; na rozdíl od `\hbox` nebo `\unhyphenated`, které působí pouze v místě zdrojového souboru, kde se s nimi setkáte.

Interně je fungování spojovníku řízeno proměnnými T_EX `\pretolerance` a `\tolerance`. První z nich řídí přípustnost dělení provedeného pouze na bílém místě. Ve výchozím nastavení je to 100, ale pokud ji změníme například na 10 000, pak bude ConT_EXt vždy považovat za přípustné, aby došlo k rozdělení řádku, které neznamena rozdělení slov podle slabik, což znamená, že *de facto* odstraňujeme dělení na základě slabik. Zatímco kdybychom například nastavili hodnotu `\pretolerance` na -1, nutili bychom ConT_EXt pokaždé použít spojovník na konci řádku.

Pro `\pretolerance` můžeme přímo nastavit libovolnou hodnotu tak, že ji jednoduše přiřadíme v našem dokumentu. Například:

```
\pretolerance=10000
```

ale můžeme také manipulovat s touto hodnotou pomocí hodnot “lesshyphenation” a “morehyphenation” v `\setupalign`. V tomto ohledu viz [sekce 2.6.1](#).

2.3.3 Úroveň tolerance pro zalomení řádků

Při hledání možných bodů zlomu řádku je ConTeXt obvykle přísný, což znamená, že raději nechá slovo přesáhnout pravý okraj, protože ho nedokázal spojit, a raději nevkládá zlom řádku před slovo, pokud by to vedlo k příliš velkému zvětšení mezislovního prostoru na daném řádku. Toto výchozí chování obvykle poskytuje optimální výsledky a jen výjimečně některé řádky poněkud vystupují na pravé straně. Myšlenka je taková, že autor (nebo sazeč) tyto výjimečné případy po dokončení dokumentu přezkoumá a učiní příslušné rozhodnutí, kterým může být `\break`. příkaz před slovem, které přesahuje, nebo může také znamenat jiné znění odstavce tak, že se toto slovo přesune na jiné místo.

V některých případech však může být nízká tolerance ConTeXtu problémem. V těchto případech mu můžeme říci, aby byl tolerantnější k bílým místům v řádcích. K tomu slouží příkaz `\setuptolerance`, který nám umožňuje změnit úroveň tolerance při výpočtu zlomů řádků, které ConTeXt nazývá “horizontální tolerance”. (protože ovlivňuje horizontální prostor) a “vertikální tolerance” při výpočtu zlomů stránek. O tom budeme hovořit v [sekci 2.6.2](#).

Vodorovná tolerance (která ovlivňuje zalomení řádků) je ve výchozím nastavení nastavena na hodnotu “**verystrict**”. Tuto hodnotu můžeme změnit nastavením některé z následujících hodnot: “**strict**”, “**tolerant**”, “**verytolerant**” nebo “**stretch**”. Takže například,

```
\setuptolerance[horizontal, verytolerant]
```

téměř znemožní, aby řádek přesáhl pravý okraj, i když to znamená vytvoření velmi velké a nevzhledné mezery mezi slovy na řádku.

2.3.4 Vynucení zalomení řádku v určitém bodě

Pro vynucení zalomení řádku v určitém bodě použijeme příkazy `\break`, `\crlf` nebo `\.`. První z nich, `\break`, způsobí v místě, kde je umístěn, zalomení řádku, což s největší pravděpodobností způsobí, že řádek, na kterém je příkaz umístěn, bude esteticky deformován a mezi slovy na tomto řádku bude obrovské množství bílého místa. Jak je vidět na následujícím příkladu, ve kterém je `\break` příkaz na třetím řádku (ve zdrojovém fragmentu vlevo) vede k druhému docela ošklivému řádku (ve formátovaném textu vpravo).

```
On the corner of the old quarter I saw
him \emph{swagger} along like
the\break tough guys do when they walk,
hands always in their overcoat pockets,
so no one can know which of them carries
the dagger.
```

```
On the corner of the old quarter I saw him swag-
ger along like the
tough guys do when they walk, hands always
in their overcoat pockets, so no one can know
which of them carries the dagger.
```

To avoid this effect, we can use the `\` or `\crlf` commands that also insert a forced line break, but they fill in the original line with enough blank space to align it to the left:

```
On the corner of the old quarter I saw
him \emph{swagger} along like
the\ tough guys do when they walk,
hands always in their overcoat pockets,
so no one can know which of them carries
the dagger.
```

```
On the corner of the old quarter I saw him swag-
ger along like the
tough guys do when they walk, hands always
in their overcoat pockets, so no one can know
which of them carries the dagger.
```

Pokud vím, na *normálních* řádcích nejsou rozdíly mezi `\` a `\crlf`; ale v názvu sekce rozdíl je:

- `\` generuje zalomení řádku v těle dokumentu, ale ne při přenosu názvu oddílu do obsahu.
- `\crlf` generuje zalomení řádku, které se použije jak v těle dokumentu, tak při přenosu názvu oddílu do obsahu.

Přerušení řádku by nemělo být zaměňováno s přerušením odstavce. Zlom řádku jednoduše ukončuje aktuální řádek a začíná další řádek, ale zůstáváme ve stejném odstavci, takže vzdálenost mezi původním řádkem a novým řádkem bude určena normálním odstupem v rámci odstavce. Proto existují pouze tři scénáře, kdy lze doporučit vynucení zlomu řádku:

- Ve výjimečných případech, kdy ConT_EXt nenalezne vhodný zlom řádku, takže řádek vyčnívá vpravo. V těchto případech (které se vyskytují velmi zřídka, hlavně když řádek obsahuje nedělitelné *boxy* nebo *doslovný* text [viz [sekce 1.2.3](#)]) může být užitečné vynutit zlom řádku pomocí `\break`. těsně před slovem, které vyčnívá do pravého okraje.
- V odstavcích, které jsou vlastně tvořeny jednotlivými řádky, z nichž každý obsahuje informace nezávislé na předchozích řádcích, např. v záhlaví dopisu, kde první řádek může obsahovat jméno odesílatele, druhý jméno příjemce a třetí datum; nebo v textu o autorství díla, kde jeden řádek obsahuje jméno autora, druhý jeho funkci nebo akademickou pozici a třetí řádek třeba datum

atd. V těchto případech je třeba zalomení řádku vynutit pomocí příkazů `\\` nebo `\crlf`. Je také běžné, že tyto druhy odstavců jsou zarovnány doprava.

- Při psaní básní nebo podobných druhů textů oddělit jeden verš od druhého. I když v tomto druhém případě je vhodnější použít prostředí `lines` vysvětlené v [sekci 2.5.1](#).

2.4 Meziřádkový prostor

Meziřádkový prostor je vzdálenost mezi řádky, které tvoří paragraf. ConTeXt ji vypočítá automaticky na základě aktuálně použitého písma a především na základě základní velikosti nastavené pomocí `\setupbodyfont` nebo `\switchtobodyfont`.

Prostor mezi řádky můžeme ovlivnit pomocí příkazu `\setupinterlinespace`, který umožňuje tři různé druhy syntaxe:

- `\setupinterlinespace [..Meziřádkový prostor..]`, kde *Meziřádkový prostor* je přesná hodnota nebo symbolické slovo, které přiřazuje předem definovaný meziprostor:
 - Pokud se jedná o přesnou hodnotu, může to být rozměr (například 15pt) nebo jednoduché, celé či desetinné číslo (například 1,2). V tomto druhém případě je číslo interpretováno jako “počet řádků” na základě výchozího meziřádkového prostoru ConTeXtu.
 - Pokud se jedná o symbolické slovo, může to být “small”, “medium” nebo “big”, z nichž každý použije malou, střední nebo velkou meziřádkovou mezeru, vždy na základě výchozí meziřádkové mezery ConTeXt
- `\setupinterlinespace [...]=...]`. V tomto režimu se mezisvazkový prostor nastavuje explicitní změnou založených měr, pomocí kterých ConTeXt počítá příslušné mezisvazkové mezery. V tomto režimu se mezery nastavují explicitní změnou měr, na jejichž základě ConTeXt vypočítává příslušné mezery. Již dříve jsem uvedl, že řádkování se vypočítává na základě konkrétního písma a jeho velikosti, ale to bylo jen pro zjednodušení: písmo a jeho velikost ve skutečnosti slouží k tomu, aby se stanovily určité míry, na jejichž základě se vypočítává meziřádkový prostor. Pomocí tohoto přístupu `\setupinterlinespace` se tyto míry upraví, a tím se změní i meziřádkový prostor. Skutečné míry a hodnoty, s nimiž lze tímto postupem manipulovat (jejichž význam nebudu vysvětlovat, protože přesahuje rámec jednoduchého úvodu), jsou tyto: `line`, `height`, `depth`, `minheight`, `mindepth`, `distance`, `top`, `bottom`, `stretch` a `shrink`.
- `\setupinterlinespace [Název]`. V tomto režimu nastavíme nebo nakonfiguruje specifický a přizpůsobený typ řádkování, který byl dříve definován pomocí `\defineinterlinespace`.

Pomocí

```
\defineinterlinespace[Name] [Configuration]
```

můžeme určitou konfiguraci meziřádkového prostoru přiřadit konkrétnímu názvu, který pak můžeme jednoduše spustit v určitém bodě našeho dokumentu pomocí `\setupinterlinespace[Name]`. Pro návrat k normálnímu meziřádkovému prostoru bychom pak museli napsat `\setupinterlinespace[reset]`.

2.5 Ostatní záležitosti týkající se řádkování

2.5.1 Převod zalomení řádků ve zdrojovém souboru na zalomení řádků ve výsledném dokumentu

Jak již víme (viz [sekce 2.2.2](#)), ConT_EXt ve výchozím nastavení ignoruje zlomy řádků ve zdrojovém souboru, které považuje za prosté prázdné mezery, pokud se nevyskytují dva nebo více po sobě jdoucích zlomů řádků, v takovém případě se vloží zlom odstavce. Mohou však nastat situace, kdy máme zájem respektovat řádkové zlomy v původním zdrojovém souboru tak, jak tam byly vloženy, například při psaní poezie. K tomu nám ConT_EXt nabízí prostředí “`lines`”, jehož formát je:

```
\startlines[Options] ... \stoplines
```

kde mohou být mimo jiné tyto možnosti:

- **space**: Pokud je tato volba nastavena s hodnotou “on”, bude prostředí kromě respektování zlomů řádků ve zdrojovém souboru respektovat také prázdné mezery ve zdrojovém souboru a dočasně ignorovat pravidlo absorpce.
- **before**: Text nebo příkaz, který se má spustit před vstupem do prostředí.
- **after**: Text nebo příkaz, který se má spustit po ukončení prostředí.
- **inbetween**: Text nebo příkaz, který se spustí při vstupu do prostředí.
- **indenting**: Hodnota udávající, zda mají být odstavce v prostředí odsazeny (viz [sekce 2.1.1](#)).
- **align**: Zarovnání řádků v prostředí (viz [sekce 2.6](#)).

- **style**: Příkaz stylu, který se použije v prostředí.
- **color**: Barva, která se použije v prostředí.

Tak například,

<pre>\startlines One-one was a race horse. Two-two was one too. One-one won one race. Two-two won one too. \stoplines</pre>	<pre>One-one was a race horse. Two-two was one too. One-one won one race. Two-two won one too.</pre>
-------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------

Můžeme také upravit výchozí způsob práce s prostředím pomocí `\setuplines` a stejně jako u mnoha jiných příkazů ConTeXtu je možné přiřadit konkrétní konfiguraci tohoto prostředí jméno. To provedeme příkazem `\definelines`, jehož syntaxe je:

```
\definelines[Name] [Configuration]
```

kde jako konfiguraci můžeme zahrnout stejné možnosti, které byly vysvětleny obecně pro prostředí. Jakmile jsme si definovali vlastní řádkové prostředí, měli bychom pro jeho vložení napsat:

```
\startlines[Name] ... \stoplines
```

2.5.2 Číslování řádků

V některých typech textů je běžné zavést určitý druh číslování řádků, například v textech o počítačovém programování, kde je poměrně běžné, že fragmenty kódu nabízené jako příklady jsou číslovány, nebo v básních, kritických edicích atd. Pro všechny tyto situace nabízí ConTeXt prostředí `linenumbering`, jehož formát je

```
\startlinenumbering[Options] ... \stoplinenumbering
```

K dispozici jsou tyto možnosti:

- **continue**: V případech, kdy existuje více než jedna část dokumentu, která vyžaduje číslování řádků, tato volba zajistí, že číslování začne znovu pro každou část (“`continue=no`”, výchozí hodnota). Na druhou stranu, pokud má číslování řádků pokračovat od místa, kde skončila předchozí část, zvolíme “`continue=yes`”.
- **start**: Uvádí číslo prvního řádku v případech, kdy nechceme, aby to bylo ‘1’, nebo aby odpovídalo předchozímu výčtu.
- **step**: Pomocí této volby můžeme určit, že se číslo bude tisknout pouze v určitých intervalech. Například u básní je běžné, že se číslo objevuje pouze v násobcích 5 (verše 5, 10, 15...).

Všechny tyto možnosti lze obecně pro všechna prostředí *linenumbering* v našem dokumentu označit pomocí `\setuplinenumbering`. Tento příkaz nám také umožňuje konfigurovat další aspekty číslování řádků:

- **conversion:** Typ číslování řádků. Může to být kterýkoli z těch, které jsou vysvětleny na [page 99](#) ohledně číslování kapitol a oddílů.
- **style:** Příkaz (nebo příkazy) určující styl číslování řádků (písmo, velikost, varianta...).
- **color:** Barva, kterou se číslo řádku vytiskne.
- **location:** Kde bude umístěno číslo řádku. Může to být kterákoli z následujících položek: text, begin, end, default, left, right, inner, outer, inleft, inright, margin, inmargin.
- **distance:** Vzdálenost mezi číslem řádku a samotným řádkem.
- **align:** Zarovnání čísel. Může být: vnitřní, vnější, flushleft, flushright, left, right, middle nebo auto.
- **command:** Příkaz, kterému bude číslo řádku předáno jako parametr před tiskem.
- **width:** Šířka vyhrazená pro tisk čísla řádku.
- **left, right, margin:**

Můžeme také vytvořit různé přizpůsobené konfigurace číslování řádků pomocí `\defineline numbering` tak, aby konfigurace byla spojena s názvem:

```
\defineline numbering[Name] [Configuration]
```

Jakmile je konkrétní konfigurace definována a přiřazena k názvu, můžeme ji použít pomocí

```
\startlinenumbering[Name] ... \stoplinenumbering
```

2.6 Horizontální a vertikální zarovnání

Příkaz, který obecně ovládá zarovnání textu, je `\setupalign`. Tento příkaz se používá k ovládání horizontálního i vertikálního zarovnání.

2.6.1 Horizontální zarovnání

Pokud *přesná* šířka řádku textu nezabírá celou možnou šířku, vzniká problém, jak naložit s bílými znaky.¹ V zásadě můžeme v tomto ohledu udělat tři věci:

1. Nahromadíme ji na jedné ze dvou stran čáry: pokud ji nahromadíme na levé straně, čára bude vypadat *trochu posunutá* doprava, zatímco pokud ji nahromadíme na pravé straně, čára zůstane na levé straně. V prvním případě hovoříme o *zarovnání doprava* a v druhém o *zarovnání doleva*. Ve výchozím nastavení používá ConTeXt zarovnání vlevo na poslední řádek odstavců.

Pokud je několik po sobě jdoucích řádků zarovnáno vlevo, je pravá strana nepravdělná; pokud je však zarovnání vpravo, vypadá nerovnoměrně levá strana. Pro pojmenování možností, které zarovnávají jednu nebo druhou stranu, ConTeXt nenastavuje stranu, na které jsou zarovnány, ale stranu, na které jsou nerovnoměrné. Proto volba `flushright` vede k zarovnání vlevo a `flushleft` k zarovnání vpravo. Jako zkratky `flushright` a `flushleft` podporuje `\setupalign` také hodnoty `right` a `left`. Ale **pozor**: zde je význam slov zavádějící. Přestože `left` znamená “left” a `right` znamená “right”, `\setupalign[left]` zarovnává vpravo a `\setupalign[right]` zarovnává vlevo. Pokud by čtenáře zajímalo, proč byla tato poznámka učiněna, stálo by za to citovat z ConTeXt wiki: “ConTeXt používá volby `flushleft` a `flushright`. Zarovnání vpravo a vlevo je ve všech příkazech, které akceptují volbu zarovnání, obrácené oproti obvyklým směrům, ve smyslu “ragged left” a “ragged right”. Bohužel, když Hans poprvé psal tuto část ConTeXtu, měl na mysli zarovnání ‘ragged right’ a ‘ragged left’, nikoli ‘flush left’ a ‘flush right’. A teď, když už to takhle nějakou dobu funguje, není možné to změnit, protože změna by porušila zpětnou kompatibilitu se všemi existujícími dokumenty, které to používají.”.

V dokumentech připravených pro oboustranný tisk jsou kromě pravého a levého okraje také vnitřní a vnější okraje. Hodnoty `flushinner`. (nebo jednoduše `inner`) a `flushouter`. (nebo jednoduše `outer`) v těchto případech určují odpovídající zarovnání.

2. Rozložte ji na oba okraje. Výsledkem bude vycentrování čáry. Volba `\setupalign`, která toto provádí, je `middle`.
3. Rozdělte je mezi všechna slova tvořící řádek, v případě potřeby zvětšením mezislovní mezery tak, aby byl řádek přesně stejně široký jako prostor, který je pro něj k dispozici. V těchto případech hovoříme o *justified lines*. To je také výchozí hodnota ConTeXtu, proto v `\setupalign` není žádná zvláštní volba

¹ Pod pojmem *přesná* šířka mám na mysli šířku řádku *předtím*, než ConTeXt upraví velikost mezislovní mezery, aby umožnil justifikaci.

pro její stanovení. Pokud jsme však zarovnání zarovnané ve výchozím nastavení změnili, můžeme jej obnovit pomocí `\setupalign[reset]`.

Hodnoty pro `\setupalign`, které jsme právě viděli (`right`, `flushright`, `left`, `flushleft`, `inner`, `flushinner`, `outer`, `flushouter` a `middle`), lze kombinovat s `broad`, což vede k poněkud hrubšímu zarovnání.

Další dvě možné hodnoty `\setupalign`, které ovlivňují horizontální zarovnání, souvisejí s dělením slov na konci řádku, protože to, zda se tak stane, závisí na tom, zda je míra přesného řádku větší nebo menší; to zase ovlivňuje zbývající bílé místo.

Za tímto účelem `\setupalign` umožňuje hodnotu `morehyphenation`, která způsobí, že `TeX` bude pracovat obtížněji při hledání bodů zlomu na základě spojovníku, a `lesshyphenation`, která má opačný účinek. Při použití `\setupalign[horizontal, morehyphenation]` se zmenší zbývající bílé místo v řádcích, a proto bude zarovnání méně patrné. Naopak při použití `\setupalign[horizontal, lesshyphenation]` zůstane více bílého místa a zarovnání bude viditelnější.

`\setupalign` je určen k zařazení do preambule a ovlivnění celého dokumentu nebo k zařazení na určité místo a ovlivnění všeho od tohoto místa až do konce. Pokud chceme změnit zarovnání pouze jednoho nebo několika řádků, můžeme použít:

- Prostředí “`alignment`”, určené k ovlivnění několika řádků. Jeho obecný formát je:

```
\startalignment[Options] ... \stopalignment
```

kde *Options* jsou libovolné z možností přípustných pro `\setupalign`.

- Příkazy `\leftaligned`, `\midaligned` nebo `\rightaligned` způsobují zarovnání vlevo, na střed nebo vpravo; pokud chceme, aby poslední slovo v odstavci (ale pouze toto, nikoli zbytek řádku) bylo zarovnáno vpravo, můžeme použít `\wordright`. Všechny tyto příkazy vyžadují, aby se text, který má být ovlivněn, nacházel mezi kudrnatými závorkami.

Na druhou stranu si všimněte, že pokud slova “`right`” a “`left`” v příkazu `\setupalign` způsobí opačné zarovnání, než naznačuje jejich název, nestane se tak v případě příkazů `\leftaligned` a `\rightaligned`, které způsobí přesně takové zarovnání, jaké naznačuje jejich název: `left` vlevo a `right` vpravo.

2.6.2 Vertikální zarovnání

Jestliže horizontální zarovnání přichází v úvahu, když šířka řádku nezabírá celý prostor, který má k dispozici, vertikální zarovnání ovlivňuje výšku celé stránky: jestliže přesná výška textu na stránce nezabírá celou výšku, co uděláme se zbývajícím bílým prostorem? Můžeme ho nahromadit nahoře (“`height`”), což znamená, že text na stránce bude posunut dolů; můžeme ho nahromadit dole (“`bottom`”) nebo ho rozdělit mezi odstavce (“`line`”). Výchozí hodnota pro svislé zarovnání je “`bottom`”.

Vertikální úroveň tolerance

Stejným způsobem, jakým můžeme měnit úroveň tolerance ConTeXtu, pokud jde o velikost přípustné horizontální mezery v řádku (horizontální tolerance) pomocí `\setuptolerance`, můžeme měnit i jeho vertikální toleranci, tj. toleranci pro mezery mezi odstavci větší, než jakou ConTeXtve výchozím nastavení považuje za přiměřenou pro dobře nastavenou stránku. Možné hodnoty pro vertikální toleranci jsou stejné jako pro horizontální toleranci: `verystrict`, `strict`, `tolerant` a `verytolerant`. Výchozí hodnota je `\setuptolerance [vertical, strict]`.

Kontrola vdov a siroteků

Jedním z aspektů, který nepřímo ovlivňuje vertikální vyrovnaní, je kontrola vdov a siroteků. Oba jevy znamenají, že zlom stránky způsobí, že jeden řádek odstavce je izolován na jiné stránce než zbytek odstavce. To se nepovažuje za typograficky vhodné. Pokud je řádek, který je oddělen od zbytku odstavce, první na stránce, hovoříme o *ovdovělém řádku*; pokud je řádek oddělený od svého odstavce poslední na stránce, hovoříme o *osiřelém řádku*.

Ve výchozím nastavení ConTeXt neimplementuje kontrolu, která by zajistila, že se tyto řádky nebudou vyskytovat. To však můžeme změnit změnou některých vnitřních proměnných ConTeXtu: `\widowpenalty` kontroluje widowedlines a `\clubpenalty` kontroluje orphanedlines. Následující prohlášení v preambuli našeho dokumentu tedy zajistí, že tato kontrola bude provedena:

```
\widowpenalty=10000
\clubpenalty=10000
```

Provedení této kontroly znamená, že ConTeXt se vyhne vložení zlomu stránky, který odděluje první nebo poslední řádek odstavce od stránky, na které se nachází zbytek. Toto zamezení bude více či méně důsledné v závislosti na hodnotě, kterou přiřadíme proměnným. Při hodnotě 10,000, jakou jsem použil v příkladu, bude kontrola absolutní; při hodnotě například 150 nebude kontrola tak přísná a občas se mohou vyskytnout některé rozšířené nebo osiřelé řádky, kdy je alternativa z typografického hlediska horší.

Chapter 3

Speciální konstrukce a odstavce

Table of Contents: **3.1 Poznámky pod čarou a závěrečné poznámky;**
3.1.1 Typy poznámek v ConT_EXt a příkazy s nimi spojené; 3.1.2 Podrobný pohled na poznámky pod čarou a poznámky na konci; 3.1.3 Místní poznámky; 3.1.4 Vytváření a používání přizpůsobených typů poznámek; 3.1.5 Konfigurace poznámek; 3.1.6 Dočasné vyloučení poznámek při kompilaci; **3.2 Odstavce s více sloupci;**
3.2.1 Prostředí `\startcolumns`; 3.2.2 Paralelní odstavec; **3.3 Strukturované seznamy;** 3.3.1 Výběr druhu seznamu a oddělovače mezi položkami; 3.3.2 Uspořádané seznamy; A Souřadné seznamy; 3.3.3 Vložení položek do seznamu; 3.3.4 Základní konfigurace seznamu; 3.3.5 Dodatečná konfigurace seznamu; 3.3.6 Jednoduché seznamy s příkazem `\items`; 3.3.7 Předurčení chování seznamu a vytvoření vlastních typů seznamů; **3.4 Popisy a výčty;** 3.4.1 Popisy; 3.4.2 Výčty; **3.5 Čáry a rámce;**
3.5.1 Jednoduché čáry; 3.5.2 Čáry spojené s textem; 3.5.3 Orámovaná slova nebo texty; **3.6 Další zajímavá prostředí a stavby;**

3.1 Poznámky pod čarou a závěrečné poznámky

Poznámky jsou “druhotné textové prvky, které slouží k různým účelům, jako je objasnění nebo rozšíření hlavního textu, uvedení bibliografických odkazů na zdroje včetně citací, odkazů na jiné dokumenty nebo vyjádření smyslu textu”. [*Libro de Estilo de la Lengua española* (Příručka španělského jazyka), s. 195]. Jsou důležité zejména v textech akademické povahy. Mohou být umístěny na různých místech stránky nebo dokumentu. Dnes jsou nejrozšířenější ty, které se nacházejí v patě stránky (nazývají se proto poznámky pod čarou); někdy jsou také umístěny na některém z okrajů (poznámky na okraji), na konci každé kapitoly či oddílu nebo na konci dokumentu (poznámky na konci). Ve zvláště složitých dokumentech mohou existovat také různé řady poznámek: poznámky autora, poznámky překladatele, aktualizace atd. Zejména v kritických edicích může být poznámkový aparát poměrně složitý a jen několik málo systémů pro sazbu je schopno jej podporovat. ConT_EXt je jedním z nich. K dispozici je řada příkazů pro vytváření a konfiguraci různých typů poznámek.

Pro vysvětlení je užitečné začít tím, že poukážeme na různé prvky, které mohou být součástí poznámky:

- *Značka* nebo *poznámka Kotva*: Značka umístěná v těle textu, která označuje, že je k němu připojena poznámka. Ne všechny typy poznámek mají přiřazenu *kotvu*, ale pokud je s nimi spojena, objevuje se tato *kotva* na dvou místech: v místě hlavního textu, na který poznámka odkazuje, a na začátku samotného textu poznámky. Přítomnost stejné referenční značky na obou místech umožňuje, aby poznámka byla spojena s hlavním textem.
- *Poznámka ID nebo identifikátor*: Písmeno, číslo nebo symbol, který identifikuje poznámku a odlišuje ji od ostatních poznámek. Některé poznámky, například poznámky na okraji, mohou ID postrádat. Pokud tomu tak není, ID se obvykle shoduje s *kotvou*.

Pokud budeme uvažovat výhradně o poznámkách pod čarou, neuvidíme žádný rozdíl mezi tím, co jsem právě nazval *odkazová značka*, a *id*. U ostatních druhů poznámek rozdíl jasně vidíme: Například řádkové poznámky mají *id*, ale nemají referenční značku.

- *Text* nebo *Obsah* poznámky, která se vždy nachází na jiném místě stránky nebo dokumentu než příkaz, který poznámku vytváří a označuje její obsah.
- *Štítek* spojená s poznámkou: Štítek nebo název spojený s poznámkou, který se nezobrazuje ve výsledném dokumentu, ale umožňuje na něj odkazovat a vyhledávat jeho ID na jiném místě dokumentu.

3.1.1 Typy poznámek v ConTeXt a příkazy s nimi spojené

V ConTeXtu máme různé typy poznámek. Prozatím je pouze vyjmenuji, popíši je obecně a uvedu informace o příkazech, které je vytvářejí. Později se budu věnovat prvním dvěma:

- **Poznámky pod čarou**: Nepochybně nejoblíbenější, a to do té míry, že je běžné, že se všechny typy poznámek označují obecně jako *poznámky pod čarou*. Poznámky pod čarou zavádějí značku s identifikátorem *id* poznámky v místě dokumentu, kde se příkaz nachází, a vkládají vlastní text poznámky na konec stránky, kde se značka objevuje. Vytvářejí se příkazem `\footnote`.
- **Koncové poznámky**: Tyto poznámky, které se vytvářejí příkazem `\koncová poznámka`, se vkládají na místo v dokumentu, kde se nachází značka s ID poznámky; obsah poznámky se však vkládá na jiné místo v dokumentu a vložení se provádí jiným příkazem (`\placenotes`).
- **Poznámky k okraji**: Jak již název napovídá, píše se na okraj textu a nemají žádné ID ani automaticky generovanou značku či kotvu v těle dokumentu.

Dva hlavní příkazy (nikoli však jedině), které je vytvářejí, jsou `\inmargin` a `\margintext`.

- **Poznámky k řádku:** Typ poznámky typický pro prostředí, kde jsou řádky číslovány, jako například v případě `\startlinenumbering ... \stoplinenumbering` (viz [section 2.5.2](#)). Poznámka, která je obvykle napsána dole, odkazuje na konkrétní číslo řádku. Generují se příkazem `\linenote`, který se konfiguruje pomocí `\setuplinenote`. Tento příkaz nevytiskne v těle textu žádnou značku, ale v samotné poznámce vypíše číslo řádku, na který se poznámka vztahuje (používá se jako *ID*).

Nyní se budu věnovat výhradně prvním dvěma typům poznámek:

- Poznámky k okrajům jsou zpracovány jinde ([section 1.7](#)).
- Poznámky k řádkům mají velmi specializované použití (zejména v kritických edicích) a domnívám se, že v úvodním dokumentu, jako je tento, stačí, aby čtenář věděl, že existují.

Zájemcům však doporučuji video (ve španělštině) doplněné textem (rovněž ve španělštině) o kritických edicích v ConT_EXt, jehož autorem je Pablo Rodríguez. Je k dispozici na adrese [tento odkaz](#). Je také docela užitečný pro pochopení několika obecných nastavení poznámek obecně.

3.1.2 Podrobný pohled na poznámky pod čarou a poznámky na konci

Syntaxe příkazů poznámek pod čarou a poznámek na konci a mechanismy jejich konfigurace a přizpůsobení jsou dosti podobné, protože ve skutečnosti jsou oba typy poznámek zvláštními instancemi obecnější konstrukce (poznámky), jejíž další instance lze nastavit příkazem `\definenote` (viz [section 3.1.4](#)).

Syntaxe příkazu, který vytvoří každý z těchto typů poznámek, je následující:

```
\footnote[Label]{Text}  
\footnote[Label]{Text}
```

kde

- *Label* je nepovinný argument, který poznámce přiřadí štítek, který nám umožní odkazovat na ni jinde v dokumentu.
- *Text* je obsah poznámky. Může být libovolně dlouhý a může obsahovat speciální odstavce a nastavení, i když je třeba poznamenat, že pokud jde o poznámky pod čarou, správné rozložení stránky je v dokumentech s hojnými a příliš dlouhými poznámkami poměrně obtížné.

V zásadě lze v textu poznámky použít jakýkoli příkaz, který lze použít v hlavním textu. Podařilo se mi však ověřit, že některé konstrukce a znaky, které v hlavním textu nepředstavují žádný problém, generují chybu při kompilaci, pokud se vyskytují v textu poznámky. Tyto případy jsem zjistil při testování, ale nijak jsem je neuspořádal.

Pokud byl argument *Label* použit k nastavení štítku pro poznámku, příkaz `\note` nám umožní získat ID dané poznámky. Tento příkaz vypíše ID poznámky spojené se štítkem, který přebírá jako argument, v dokumentu. Tedy např:

<pre>Humpty Dumpty\footnote[humpty]{Pravděpodobně nejznámější postava anglické říkanky} seděla na zdi, Humpty Dumpty\note[humpty] měl velký pád.\\ Všichni královi koně a všichni královi muži nemohli dát Humptyho\note[humpty] zase dohromady</pre>	<pre>Humpty Dumpty¹ seděl na zdi, Humpty Dumpty¹ měl velký pád. Všichni královští koně a všichni královští muži Nemohli dát Humptyho¹ znovu dohromady.</pre> <hr style="width: 20%; margin-left: 0;"/> <p>¹ pravděpodobně nejznámější anglická postavička z říkanky</p>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Hlavní rozdíl mezi `\footnote` a `\endnote` je v místě, kde se poznámka objevuje:

`\footnote` Text poznámky se zpravidla vytiskne v dolní části stránky, na které se příkaz nachází, takže značka poznámky a její text (nebo začátek textu, pokud má být rozložen na dvě stránky) se objeví na stejné stránce. Za tímto účelem ConTeXt provede nezbytné úpravy při sazbě stránky tak, že vypočítá prostor potřebný pro umístění poznámky ve spodní části stránky.

V některých prostředích však `\footnote` vloží text poznámky nikoli na konec stránky, ale pod prostředí. To je například případ tabulek nebo prostředí `columns`. Chceme-li, aby se v těchto případech poznámky uvnitř prostředí nacházely ve spodní části stránky, měli bychom místo příkazu `\footnote` použít příkaz `\footnote` v kombinaci s výše uvedeným příkazem `\footnote`. První z nich, který rovněž podporuje popisek jako nepovinný argument, vytiskne pouze text poznámky, ale nikoli značku. Protože však `\note` vytiskne pouze značku bez textu, kombinace obou příkazů nám umožní umístit poznámku na požadované místo. Můžeme tedy například napsat `\note[MyLabel]` v rámci tabulky nebo vícesloupcového prostředí, a jakmile se z tohoto prostředí dostaneme, můžeme napsat `\footnotetext[MyLabel]{Text poznámky}`.

Dalším příkladem použití `\footnotetext` v kombinaci s `\note` jsou poznámky uvnitř jiných poznámek. Například:

```
Tato%
\footnote{nebo tato\note[noteB], chcete-li.}%
\footnotetext[noteB]
{nebo možná i tato\note[noteC].}
\footnotetext[noteC]{může být něco úplně jiného.}
je věta s vnořenými poznámkami.
```

Tato¹ je věta s vnořenými poznámkami.

¹nebo tato² chcete-li.

³nebo možná i tato

⁵mohlo by to být něco úplně jiného.

\endnote vytiskne pouze kotvu poznámky v místě zdrojového souboru, kde se nachází. Vlastní obsah poznámky se vloží na jiné místo v dokumentu pomocí jiného příkazu (**\placenotes**[**endnote**]), který v místě, kde se nachází, vloží obsah všech poznámek na konci dokumentu (nebo dané kapitoly či oddílu).

3.1.3 Místní poznámky

Prostředí **\startlocalfootnotes** znamená, že poznámky pod čarou v něm obsažené jsou považovány za *lokální* poznámky, což znamená, že jejich číslování bude resetováno a že obsah poznámek nebude automaticky vložen spolu s ostatními poznámkami, ale pouze v tom místě dokumentu, kde se nachází příkaz **\place-localfootnotes**, což může, ale nemusí být uvnitř prostředí.

3.1.4 Vytváření a používání přizpůsobených typů poznámek

Speciální typy poznámek můžeme vytvořit pomocí příkazu **\definernote**. To může být užitečné ve složitých dokumentech, kde jsou poznámky od různých autorů, nebo pro různé účely, abychom graficky odlišili jednotlivé typy poznámek v našem dokumentu pomocí jiného formátu a jiného číslování.

Syntaxe **\definernote** je následující:

```
\definernote [Name] [Model] [Configuration]
```

kde

- *Name* je název, který přiřadíme našemu novému typu poznámky.
- *Model* je model poznámky, který bude použit na začátku. Může to být **footnote** nebo **endnote**; v prvním případě bude náš model poznámky fungovat jako poznámky pod čarou a v druhém případě jako poznámky na konci, ačkoli pro jejich vložení do dokumentu bychom nepoužili **\placenotes**[**endnote**], ale **\placenotes**[*Name*]. (název, který jsme těmto druhům poznámek přiřadili).

Teoreticky je tento argument volitelný, i když při mých testech některé poznámky vytvořené bez něj nebyly vidět a neměl jsem trpělivost zjišťovat, co bylo příčinou.

- *Konfigurace* je nepovinný druhý argument, který nám umožňuje odlišit náš nový typ poznámek od jeho vzoru: buď nastavením jiného formátu, nebo jiného typu číslování, nebo obojího.

Podle oficiálního seznamu příkazů ConT_EXt (viz section ??) jsou nastavení, která lze zadat při vytváření nového typu poznámky, založena na nastaveních, která lze zadat později pomocí `\setupnote`. Jak však brzy uvidíme, ve skutečnosti existují dva možné příkazy pro nastavení poznámek: `\setupnote` a `\setupnotation`. Proto se domnívám, že je vhodnější tento argument při vytváření typu poznámky vynechat a naše nové poznámky pak nastavit pomocí příslušných příkazů. Alespoň se to lépe vysvětluje.

Například následující položka vytvoří nový typ poznámky s názvem “Modrá poznámka”, který bude podobný poznámkám pod čarou, ale jeho obsah bude vytištěn tučně a modře:

```
\definernote [BlueNote] [footnote]
\setupnotation
[BlueNote]
[color=blue, style=bf]
```

Jakmile vytvoříme nový typ poznámky, např. *BlueNote*, bude k dispozici příkaz umožňující jeho použití. V našem příkladu to bude `\BlueNote`, jehož syntaxe bude podobná jako `\footnote`:

```
\BlueNote[Label]{Text}
```

3.1.5 Konfigurace poznámek

Konfigurace poznámek (poznámky pod čarou nebo poznámky na konci, poznámky vytvořené pomocí `\definernote` a také řádkové poznámky vytvořené pomocí `\li-`

`nenote`) se provádí pomocí dvou příkazů: `\setupnote` a `\setupnotation`¹. Syntaxe obou je podobná:

```
\setupnote[NoteType][Configuration]
\setupnotation[NoteType][Configuration]
```

kde *NoteType* odkazuje na druh poznámky, kterou konfigurujeme (`footnote`, `endnote` nebo název nějakého typu poznámky, který jsme sami vytvořili), a *konfiguration* obsahuje konkrétní konfigurační možnosti příkazu.

Problémem je, že z názvů těchto dvou příkazů není příliš jasné, jaký je mezi nimi rozdíl a co který z nich konfiguruje; příliš nepomáhá ani skutečnost, že mnoho možností těchto příkazů není zdokumentováno. Po dlouhém zkoušení jsem nebyl schopen dospět k žádnému závěru, který by mi umožnil pochopit, proč se některé věci konfigurují pomocí jednoho, zatímco jiné pomocí druhého,² Snad s výjimkou toho, že kvůli volbám, které jsem provedl, aby to fungovalo, `\setupnotation` vždy ovlivňuje text poznámky nebo ID, které je vytištěno s textem poznámky, zatímco

¹ `\setupnote` má 35 přímých konfiguračních možností a 45 dalších možností zděděných z `\setupframed`; `\setupnotation` má 45 přímých konfiguračních možností a dalších 23 zděděných z `\setupcounter`. Protože tyto volby nejsou zdokumentovány, a přestože u mnoha z nich můžeme jejich užitečnost zjistit z jejich názvu, musíme si ověřit, zda je naše intuice pravdivá, či nikoli; a také s ohledem na to, že mnohé z těchto voleb umožňují řadu hodnot a všechny je třeba otestovat... Uvidíte, že pro napsání tohoto výkladu jsem musel provést poměrně velké množství testů; a přestože provedení jednoho testu je rychlé, provedení mnoha testů je pomalé a nudné. Proto doufám, že mi čtenář promine, když vám řeknu, že kromě dvou obecných konfiguračních příkazů pro poznámky, které uvádím v hlavním textu a na které se v následujícím výkladu zaměřím, vynechám ve výkladu další čtyři potenciální možnosti konfigurace:

- `\setupnotes` a `\setupnotations`: Jinými slovy, stejný název, ale v množném čísle. Na wiki se píše, že jednotné a množné číslo příkazu jsou synonyma, a já tomu věřím.
- `\setupfootnotes` a `\setupendnotes`: Předpokládáme, že se jedná o specifické aplikace pro poznámky pod čarou a poznámky na konci. Možná by však bylo jednodušší vysvětlit konfiguraci poznámek na základě těchto příkazů, protože se mi nepodařilo zprovoznit první možnost (`numberconversion`), kterou jsem zkoušel pomocí `\setupfootnotes`, ačkoli vím, že ostatní možnosti těchto příkazů fungují... Byl jsem příliš líný na to, abych k mnoha testům, které jsem už musel udělat, abych mohl napsat to, co následuje, přidal testy potřebné k zahrnutí těchto dvou příkazů do vysvětlení.

Jsem však toho názoru (na základě několika náhodných testů, které jsem provedl), že vše, co funguje v těchto dvou příkazech, ale jejichž vysvětlení vynechávám, funguje i v příkazech, u kterých vysvětlení uvádím.

² Je zde stránka [ConTeXt wiki](#) kterou jsem objevil (není věnována přímo poznámkám), která naznačuje změnu v `\setupnotation` ovládání textových poznámek, dle jejich vložení, a `\setupnote` prostředí poznámek dle toho, kam budou vloženy (?) To však neodpovídá skutečnosti, že například šířka textu poznámky (která souvisí s jejím *insertion*) je řízena volbou `width` v `\setupnote` a nikoli stejnojmennou volbou `\setupnotation`. To, co je zde řízeno, je šířka mezery mezi značkou a textem poznámky.

`\setupnote` má některé volby, které ovlivňují značku pro poznámku vloženou do hlavního textu.

Nyní se pokusím uspořádat to, co jsem zjistil po provedení několika testů s různými možnostmi obou příkazů. Většinu voleb obou příkazů ponechám stranou, protože nejsou zdokumentovány a nemohl jsem vyvodit žádné závěry o tom, k čemu slouží a za jakých podmínek by se měly používat:

- **ID použité pro značku:**

Poznámky jsou vždy označeny číslem. Zde můžeme nastavit následující:

- *První číslo:* řídí se podle `start` v `\setupnotation`. Jeho hodnota musí být celé číslo a používá se k zahájení počítání not.
- *Systém číslování,* který závisí na volbě `numberconversion` v `\setupnotation`. Jeho hodnoty mohou být:
 - ★ *Arabské číslice:* `n`, `N` nebo číslo.
 - ★ *Římské číslice:* `I`, `R`, římské číslice, `i`, `r`, římské číslice. První tři jsou velká písmena římských číslic a poslední tři malá písmena.
 - ★ *Číslování písmeny:* `A`, `Znak`, `Znaky`, `a`, `znak`, `znaky` v závislosti na tom, zda chceme, aby písmena byla velká (první tři možnosti) nebo malá (ostatní).
 - ★ *Číslování slovy.* Jinými slovy, napíšeme slovo, které označuje číslo, a tak se například z ‘3’ stane ‘3’. Jsou možné dva způsoby. Volba `Words` zapisuje slova velkými písmeny a `words` malými písmeny.
 - ★ *Číslování pomocí symbolů:* v závislosti na zvolené možnosti můžeme použít čtyři různé sady symbolů: `set 0`, `set 1`, `set 2` o `set 3`. Na stránce page ?? je příklad symbolů použitých v každé z těchto možností.
- *Událost, která určuje opětovné spuštění číslování not:* To závisí na volbě `way` v `\setupnotation`. Pokud je hodnota `bytext`, budou všechny poznámky v dokumentu číslovány postupně bez obnovení číslování. Při hodnotě `byapter`, `bysection`, `bysubsection` atd. se počítadlo poznámek vynuluje při každé změně kapitoly, sekce nebo podsekce, zatímco při hodnotě `byblock` se číslování vynuluje při každé změně bloků v makrostruktuře dokumentu (viz [section 3.6](#)). Hodnota `bypage` způsobí, že se počítadlo poznámek restartuje při každé změně stránky.

- **Nastavení značky poznámky:**

- Zda se má zobrazit, nebo ne: `number` možnost v `\setupnotation`.
- Umístění značky ve vztahu k textu poznámky: `alternative` v `\setupnotation`: může nabývat některé z následujících hodnot: `left`, `inleft`,

`leftmargin`, `right`, `inright`, `rightmargin`, `inmargin`, `margin`, `innermargin`, `outermargin`, `serried`, `hanging`, `top`, `command`.

- Formát značky v samotné poznámce: `numbercommand` v `\setupnotation`.
- Formát značky v těle textu: Formát textu: `textcommand` v `\setupnote`.

Volby `numbercommand` a `textcommand` se musí skládat z příkazu, který přebírá obsah značky jako argument. Může se jednat o příkaz, který si definujete sami. Zjistil jsem však, že fungují i jednoduché formátovací příkazy (`\bf`, `\it` atd.), ačkoli se nejedná o příkazy, které by musely přijímat argument.

- Vzdálenost mezi značkou a textem (v samotné poznámce): Možnosti `distance` a `width` v položce `\setupnotation`. Nepodařilo se mi zjistit, jaký je rozdíl (pokud vůbec nějaký je) mezi použitím jedné nebo druhé možnosti.
- Existence nebo neexistence hypertextového odkazu umožňujícího přechod mezi značkou v hlavním textu a značkou v poznámce: Možnost `interaction` v `\setupnote`. Při hodnotě `yes` bude odkaz existovat, při hodnotě `no` nebude.

• Konfigurace samotného textu poznámky.

Můžeme ovlivnit následující aspekty:

- umístění položky: závisí na volbě `location` v `\setupnote`.

V zásadě již víme, že poznámky pod čarou se umísťují na konec stránky (`location=page`) a poznámky na konci stránky v místě, kde se nachází `\placenotes[endnote]`. (`location=text`), můžeme však tuto funkci upravit a nastavit poznámky pod čarou například jako `location=text`, což způsobí, že poznámky pod čarou budou fungovat podobně jako poznámky na konci, takže se objeví v tom místě dokumentu, kde se nachází příkaz `\placenotes[footnote]` nebo příkaz specifický pro poznámky pod čarou `\placefootnotes`. Tímto postupem bychom mohli například vytisknout poznámky pod odstavcem, ve kterém se nacházejí.

- Oddělení odstavců mezi poznámkami: ve výchozím nastavení je každá poznámka vytištěna ve vlastním odstavci, ale můžeme nechat všechny poznámky na stejné stránce vytisknout v jednom odstavci nastavením volby `paragraph` v `\setupnote` na “yes”.
- Styl, ve kterém bude zapsán samotný text poznámky: volba `style` v `\setupnotation`.
- Velikost písmen: volba `bodyfont` v `\setupnote`.

Tato možnost je určena pouze pro případ, kdy chceme ručně nastavit velikost písma pro poznámky pod čarou. Téměř nikdy není dobré to dělat, protože ve výchozím nastavení

ConT_EXt nastaví velikost písma v poznámkách pod čarou tak, aby bylo menší než hlavní text, ale s velikostí *kteřá je úměrná velikosti písma v hlavním textu*.

- Levý okraj textu poznámky: volba `margin` v `\setupnotation`.
 - Maximální šířka: možnost `width` v `\setupnote`.
 - Počet sloupců: volba `n` v `\setupnote` určuje, zda bude text poznámky ve dvou nebo více sloupcích. Hodnota ‘n’ musí být celé číslo.
- **Mezera mezi poznámkami nebo mezi poznámkami a textem:** zde máme následující možnosti:
 - `rule` v `\setupnote` určuje, zda mezi oblastí poznámky a oblastí stránky s hlavním textem bude nebo nebude umístěna čára (pravidlo). Jeho možné hodnoty jsou `yes`, `on`, `no` a `off`. První dvě hodnoty pravidlo zapínají a poslední ho vypínají.
 - `before`, in `\setupnotation`: příkaz nebo příkazy, které se spustí před vložením textu poznámky. Slouží k vložení dalších mezer, dělících čar mezi poznámky atd.
 - `after`, in `\setupnotation`: příkaz nebo příkazy, které se spustí po vložení textu poznámky.

3.1.6 Dočasné vyloučení poznámek při kompilaci

Příkazy `\notesenabledfalse` a `\notesenabledtrue` říkají ConT_EXt aby povolil, resp. zakázal kompilaci poznámek. Tato funkce může být užitečná, pokud si přejeme získat verzi bez poznámek v případě, že dokument obsahuje četné a rozsáhlé poznámky. Z mé osobní zkušenosti například v případě, že opravuji doktorskou práci, dávám přednost tomu, abych si ji přečetl poprvé najednou, bez poznámek, a poté provedl druhé čtení se zpracovanými poznámkami.

3.2 Odstavce s více sloupci

Vytipování textu ve více než jednom sloupci je možnost, kterou lze zavést:

- a. Jako obecná vlastnost rozvržení stránky.
- b. Jako vlastnost některých konstrukcí, například strukturovaných seznamů nebo poznámek pod čarou či poznámek na konci textu.
- c. Jako funkce aplikovaná na určité odstavce v dokumentu.

V každém z těchto případů bude většina příkazů a prostředí fungovat bez problémů, i když pracujeme s více než jedním sloupcem. Existují však určitá omezení; především ve vztahu k plovoucím objektům obecně (viz [section 4.1](#)) a zejména k tabulkám ([section 4.3](#)), i když nejsou plovoucí.

Pokud jde o počet povolených sloupců, ConTeXt nemá žádné teoretické omezení. Existují však fyzikální limity, které je třeba vzít v úvahu:

- Šířka papíru: neomezený počet sloupců vyžaduje neomezenou šířku papíru (pokud má být dokument vytištěn) nebo obrazovky (pokud se jedná o dokument určený k zobrazení na obrazovce). V praxi, s ohledem na *normální* šířku papírů, které se prodávají a používají pro tvorbu knih, a obrazovek počítačových zařízení, je obtížné, aby se text složený z více než čtyř nebo pěti sloupců dobře vešel.
- Velikost paměti počítače: referenční příručka ConTeXt uvádí, že v *normálních* systémech (ani zvlášť výkonných, ani zvlášť omezených ve zdrojích) lze zpracovat 20 až 40 sloupců.

V této části se zaměřím na použití vícesloupcového mechanismu ve speciálních odstavcích nebo fragmentech, protože

- Více sloupců jako možnost rozvržení stránky již bylo diskutováno (v [subsection B](#) sekce [section 1.3.4](#)).
- Možnosti, které nabízejí některé konstrukce, jako jsou strukturované seznamy nebo poznámky pod čarou, sazba textu ve více než jednom sloupci, jsou diskutovány ve vztahu k dané konstrukci nebo prostředí.

3.2.1 Prostředí `\startcolumns`

Obvyklý postup pro vkládání fragmentů složených z několika sloupců do dokumentu je použití prostředí `columns`, jehož formát je:

```
\startcolumns[Configuration] ... \stopcolumns
```

kde *Konfigurace* nám umožňuje ovládat mnoho aspektů prostředí. Požadovanou konfiguraci můžeme uvést při každém volání prostředí, nebo přizpůsobit výchozí fungování prostředí pro všechna volání prostředí, čehož dosáhneme pomocí příkazu

```
\setupcolumns[Configuration]
```

V obou případech jsou možnosti konfigurace stejné. Nejdůležitější z nich, seřazené podle funkce, jsou následující:

- **Možnosti, které určují počet sloupců a mezeru mezi nimi:**

- **n**: řídí počet sloupců. Pokud je tento parametr vynechán, budou vygenerovány dva sloupce.
 - **nleft**, **nright**: tyto volby se používají při oboustranném rozvržení dokumentu (viz [subsection A of section 1.3.4](#)) pro určení počtu sloupců na levé (sudé), resp. pravé (liché) stránce.
 - **distance**: mezera mezi sloupci.
 - **separator**: určuje, co bude oddělovat sloupce. Může to být **mezera** (výchozí hodnota) nebo **pravidlo**, v takovém případě bude mezi sloupci vytvořena čára (pravidlo). V případě, že je mezi sloupci vytvořeno pravidlo, lze toto pravidlo zase nakonfigurovat pomocí následujících dvou možností:
 - ★ **rulecolor**: barva řádku.
 - ★ **rulethickness**: tloušťka čáry.
 - **maxwidth**: maximální šířka sloupců + mezera mezi nimi.
- **Možnosti, které řídí rozložení textu ve sloupcích:**
 - **balance**: ve výchozím nastavení ConTeXt *balances* rozdělí text mezi sloupce tak, aby měly víceméně stejné množství textu. Tuto volbu však můžeme nastavit pomocí příkazu “no” text nezačne ve sloupci, dokud nebude předchozí sloupec zaplněn.
 - **direction**: určuje, jakým směrem se text rozdělí mezi sloupce. Ve výchozím nastavení je dodrženo přirozené pořadí čtení (zleva doprava), ale při zadání této volby s hodnotou **reverse** je výsledkem pořadí zprava doleva.
 - **Možnosti ovlivňující sazbu textu v prostředí:**
 - **tolerance**: text napsaný ve více než jednom sloupci znamená, že šířka řádku v rámci sloupce je menší, a jak bylo vysvětleno při popisu mechanismu, který ConTeXt používá pro konstrukci řádků ([section 2.3](#)), to ztěžuje nalezení optimálních bodů pro vkládání zlomů řádků. Tato volba nám umožňuje dočasně změnit vodorovnou toleranci v prostředí (viz [section 2.3.3](#)), aby se usnadnila sazba textu.
 - **align**: řídí horizontální zarovnání řádků v prostředí. Může nabývat některé z následujících hodnot: **right**, **flushright**, **left**, **flushleft**, **inner**, **flushinner**, **outer**, **flushouter**, **middle** nebo **broad**. Význam těchto voleb viz [section 2.6.1](#).
 - **color**: určuje název barvy, ve které bude text v prostředí napsán.

3.2.2 Paralelní odstavce

Specifickou verzí vícesloupcové kompozice jsou paralelní odstavce. V tomto typu konstrukce je text rozdělen do dvou sloupců (obvykle, i když někdy i více než dvou), ale není mu dovoleno mezi nimi volně plynout, místo toho je zachována přísná kontrola nad tím, co se v jednotlivých sloupcích objeví. To je velmi užitečné například při tvorbě dokumentů, které kontrastují dvě verze textu, jako je nová a stará verze nedávno novelizovaného zákona, nebo ve dvojjazyčných vydáních; nebo také při psaní glosářů pro specifické definice textu, kde se text, který má být definován, objevuje vlevo a definice vpravo apod.

Normálně bychom pro zpracování těchto typů odstavců použili mechanismus tabulek, ale je to proto, že většina textových procesorů není tak výkonná jako ConT_EXt který má příkazy `\defineparagraphs` a `\setupparagraphs`, které vytvářejí tento typ odstavců pomocí mechanismu sloupců, který je sice omezený, ale flexibilnější než mechanismus tabulek.

Pokud vím, tyto paragrafy nemají žádný zvláštní název. Nazval jsem je “parallel paragraphs”, protože mi to připadá jako popisnější termín než ten, který pro ně používá ConT_EXt: “*paragraphs*”.

Základním příkazem je zde `\defineparagraphs`, jehož syntaxe je:

`\definičníodstavce[Název][Configuration]`

kde *Název* je název, který této konstrukci dáme, a *Konfigurace* jsou vlastnosti, které bude mít, a které lze také později nastavit příkazem

`\setupparagraphs[Name][Column][Configuration]`

kde *Název* je název zadaný při vytváření, *Sloupec* je nepovinný argument umožňující konfigurovat konkrétní sloupec a *Konfigurace* nám umožňuje určit, jak bude fungovat v praxi.

Například:

```
\defineparagraphs
[MurciaFacts]
[n=3, before={\blank},after={\blank}]
```

```
\setupparagraphs
[MurciaFacts][1]
[width=.1\textwidth, style=bold]
```

```
\setupparagraphs
[MurciaFacts][2]
[width=.4\textwidth]
```


Výše uvedený fragment by vytvořil tříslopcové prostředí s názvem MurciaFacts a poté by nastavil, aby první sloupec zabíral 10 procent šířky řádku a byl napsán tučně, a druhý sloupec by zabíral 40 procent šířky řádku. Protože třetí sloupec není nastaven, bude mít zbývající šířku, tj. 50 %.

Po vytvoření prostředí můžeme na jeho základě napsat stručnou historii Murcie:

```
\startMurciaFacts
825
\MurciaFacts
Město Murcia založeno.
\MurciaFacts Původ města Murcia je nejistý, ale existují důkazy, že jej pod
názvem Madina (nebo Medina) Mursiya nechal založit v roce 825 emír al-Andalus
Abderramán II., pravděpodobně na místě mnohem staršího osídlení. \stopMurciaFacts
```

825 Město Murcia založeno.

Původ města Murcia je nejistý, ale existují důkazy, že jej pod názvem Madina (nebo Medina) Mursiya nechal založit v roce 825 emír al-Andalus Abderramán II., a to pravděpodobně na místě mnohem starší osady.

Pokud bychom chtěli pokračovat ve vyprávění příběhu o Murcii, byla by pro další událost nutná nová instance prostředí (`\startMurciaFacts`), protože tímto mechanismem nelze zahrnout několik řádků.

Na základě právě uvedeného příkladu bych rád zdůraznil následující podrobnosti:

- Jakmile je prostředí vytvořeno například pomocí `\defineparagraphs[MaryPoppins]`, stane se z něj normální prostředí, které začíná `\startMaryPoppins` a končí `\stopMaryPoppins`.
- Vytvoří se také příkaz `\MaryPoppins`, který se v prostředí používá k určení okamžiku změny sloupce.

Pokud jde o možnosti konfigurace paralelních odstavců (`\setupparagraphs`), chápu, že v této fázi úvodu a s ohledem na právě uvedený příklad je čtenář již připraven zjistit účel jednotlivých možností, takže níže uvedu pouze název a typ možností a případně jejich možné hodnoty. Nezapomeňte však, že `\setupparagraphs [Name] [Configuration]` nastavuje konfigurace, které ovlivňují celé prostředí, zatímco `\setupparagraphs [Name] [NumColumn] [Configuration]` aplikuje konfigurace výhradně na uvedený sloupec.

- | | | |
|----------------------------|--------------------------------------------------------|-------------------------------------|
| • n : Číslo | • align : Odvozeno od <code>\setuptalign</code> | • rulecolor : barva pravidla |
| • before : Příkaz | • inner : Příkaz | • style : Styl Příkaz |
| • after : Příkaz | • rule : zapnuto vypnuto | • color : Barva |
| • width : Rozměr | • rulethickness : Rozměr | |
| • distance : Rozměr | | |

Výše uvedený seznam možností není úplný; vyloučil jsem z něj ty, které bych zde normálně nevysvětloval. Využil jsem také toho, že se nacházíme v části věnované sloupcům, abych zobrazil seznam možností v trojicích sloupců, i když jsem to neudělal pomocí žádného z příkazů vysvětlených v této části, ale pomocí volby `columns` v prostředí `itemize`, kterému je věnována následující část.

3.3 Strukturované seznamy

Pokud jsou informace prezentovány uspořádaně, jsou pro čtenáře snáze uchopitelné. Pokud je však uspořádání vnímatelné i vizuálně, zvýrazní to pro čtenáře skutečnost, že zde máme strukturovaný text. Proto existují určité *konstrukce* nebo *mechanismy*, které se snaží zdůraznit vizuální uspořádání textu, a tím přispívají k jeho strukturovanosti. Z nástrojů, které ConTeXt k tomuto účelu autorovi zpřístupňuje, je nejdůležitějším nástrojem, který je předmětem této části, prostředí `itemize`, které slouží k vytváření toho, co bychom mohli nazvat *strukturované seznamy*.

Seznamy se skládají z posloupnosti *textových prvků* (které budu nazývat *elementy*), přičemž každému z nich předchází znak, který jej pomáhá zvýraznit a odlišit od ostatních a který budu nazývat “separator”. Oddělovačem může být číslo, písmeno nebo symbol. Obvykle (ale ne vždy) jsou *položky* odstavce a seznam je formátován tak, aby byla zajištěna *viditelnost* oddělovače pro každý prvek; obvykle se použije závěsná odrážka, která jej zvýrazní ¹. V případě vnořených seznamů se odsazení každého z nich postupně zvětšuje. Jazyk HTML obvykle nazývá seznamy, kde je oddělovačem číslo nebo znak, který se postupně zvětšuje, *uspořádané seznamy*, což znamená, že každý prvek seznamu bude mít jiný oddělovač, který nám umožní odkazovat na každý prvek podle jeho čísla nebo identifikátoru; a dává název *neuspořádané seznamy*, kde je pro každý prvek seznamu použit stejný znak nebo symbol.

ConTeXt automaticky řídí číslování nebo abecední řazení oddělovače v číslovaných seznamech, stejně jako odsazení, které musí mít vnořené seznamy; a v případě vnořených neuspořádaných seznamů se také stará o výběr jiného znaku nebo symbolu, který umožňuje na první pohled rozlišit úroveň *položky* v seznamu podle symbolu, který mu předchází.

V referenční příručce se uvádí, že maximální úroveň vnoření v seznamech je 4, ale předpokládám, že to platilo v roce 2013, kdy byla příručka napsána. Podle mých testů se zdá, že vnořování seznamů *uspořádaných* není nijak omezeno (v mých testech jsem dosáhl až 15 úrovní vnoření). Zatímco pro neuspořádané seznamy se zdá, že také neexistuje omezení v tom smyslu, že bez ohledu na to, kolik vnoření zahrneme, nebude vygenerována žádná chyba; ale pro neuspořádané seznamy ConTeXt použije výchozí symboly pouze pro prvních devět úrovní vnoření.

¹ V typografii se odrážka, která se vztahuje na všechny řádky odstavce kromě prvního, nazývá *závěsná odrážka*, což usnadňuje nalezení prvního slova nebo znaku odstavce.

V každém případě je třeba zdůraznit, že nadměrné používání vnořování v seznamech může mít opačný účinek, než jaký zamýšlíme, a to ten, že se čtenář cítí ztracen a není schopen najít jednotlivé položky v celkové struktuře seznamu. Z tohoto důvodu se osobně domnívám, že ačkoli jsou seznamy mocným nástrojem pro strukturování textu, téměř nikdy není dobré překračovat třetí úroveň vnoření; a i třetí úroveň bychom měli používat jen v určitých případech, kdy to dokážeme zdůvodnit.

Obecným nástrojem pro psaní seznamů v ConT_EXt je prostředí `\itemize`, jehož syntaxe je následující:

```
\startitemize[Options][Configuration] ... \stopitemize
```

kde oba argumenty jsou nepovinné. První z nich umožňuje používat symbolické názvy jako obsah, kterému ConT_EXt přiřadil přesný význam; druhý argument, který se používá jen zřídka, umožňuje přiřadit určité hodnoty určitým proměnným, které ovlivňují fungování prostředí.

3.3.1 Výběr druhu seznamu a oddělovače mezi položkami

3.3.2 Uspořádané seznamy

Ve výchozím nastavení je seznam vytvořený pomocí `itemize` neuspořádaný seznam, ve kterém se automaticky vybere oddělovač v závislosti na úrovni vnoření:

- | | |
|-------------------------------|-------------------------------|
| • Pro první úroveň vnoření. | ○ Pro šestou úroveň vnoření. |
| – Pro druhou úroveň vnoření. | ○ Pro sedmou úroveň vnoření. |
| ★ Pro třetí úroveň vnoření. | □ Pro osmou úroveň vnoření. |
| ▷ Pro čtvrtou úroveň vnoření. | ✓ Pro devátou úroveň vnoření. |
| ◦ Pro pátou úroveň vnoření. | |

Můžeme však výslovně uvést, že chceme použít symbol spojený s určitou úrovní, a to jednoduše předáním čísla úrovně jako argumentu. Příkaz `\startitemize[4]` tedy vygeneruje neuspořádaný seznam, ve kterém bude jako oddělovač použit znak ▷ bez ohledu na úroveň vnoření seznamu.

Předem určený symbol pro každou úroveň můžeme také upravit pomocí `\definesymbol`:

```
\definesymbol[Level]{Symbol spojený s úrovní}
```

Například

```
\definesymbol[1][\diamond]
```

způsobí, že první úroveň neuspořádaných seznamů bude používat symbol ◇. Stejným příkazem můžeme přiřadit některé symboly vyšším úrovním vnoření než devět. Tedy např.

`\definesymbol[10][\copyright]`

přiřadí mezinárodní symbol *copyright*: © na úroveň vnoření 10.

A. Souřadné seznamy

Pro vytvoření uspořádaného seznamu musíme zadat `itemize` druh uspořádání, které chceme. Může to být:

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 , ...				

Rozdíl mezi **n** a **m** spočívá v písmu použitém pro znázornění čísla: **n** používá písmo, které je v daném okamžiku povoleno, zatímco **m** používá jiné, elegantnější, téměř kaligrafické písmo.

Neznám název písma, které **m** používá, a proto jsem ve výše uvedeném seznamu nemohl přesně znázornit typ čísel, která tato volba generuje. Doporučuji čtenářům, aby si to sami vyzkoušeli.

3.3.3 Vložení položek do seznamu

Položky seznamu vytvořeného příkazem `\startitemize` se zpravidla zadávají příkazem `\item`, který má také verzi ve formě prostředí, která je vhodnější pro styl Mark IV: `\startitem ... \stopitem`. Tedy následující příklad:

<pre>\startitemize[a, packed] \startitem První prvek \stopitem \startitem Druhý prvek \stopitem \startitem Třetí prvek \stopitem \stopitemize</pre>	<pre>a. První prvek b. Druhý prvek c. Třetí prvek</pre>
-----------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------

vede k naprosto stejnému výsledku jako

<pre>\startitemize[a, packed] \item První prvek \item Druhý prvek \item Třetí prvek \stopitemize</pre>	<pre>a. První prvek b. Druhý prvek c. Třetí prvek</pre>
--------------------------------------------------------------------------------------------------------	---------------------------------------------------------

`\item` nebo `\startitem` je příkaz *obecně* pro vložení položky do seznamu. Spolu s ním existují následující doplňkové příkazy pro případy, kdy chceme dosáhnout speciálního výsledku:

`\head` Tento příkaz by měl být použit místo `\item`, pokud se chceme vyhnout vložení zlomu stránky za danou položku.

Běžnou konstrukcí je zařazení vnořeného seznamu nebo textového bloku bezprostředně pod prvek seznamu, takže prvek seznamu v jistém smyslu funguje jako *titulek* vnořeného seznamu nebo textového bloku. V těchto případech by nebylo vhodné mezi tímto prvkem a následujícími odstavci udělat stránkový zlom. Použijeme-li pro vložení těchto prvků `\head` místo `\item`, ConTeXt se *bude snažit* (pokud je to možné) neoddělovat takový prvek od následujícího bloku.

\sym Příkaz `\sym{Text}` zadá položku, ve které je text použitý jako argument `\sym` použit jako *oddělovač*, nikoli jako číslo nebo symbol. Tento seznam je například sestaven z položek zadaných pomocí `\sym` místo `\item`.

\sub Tento příkaz, který by se měl používat pouze v uspořádaných seznamech (kde je každá položka uvozena číslem nebo písmenem v abecedním pořadí), způsobí, že položka, která je jím zadána, si ponechá číslo předchozí položky, a aby bylo zřejmé, že se číslo opakuje a že se nejedná o chybu, je vlevo vytištěn znak '+'. To může být užitečné, pokud se odvoláváme na předchozí seznam, u kterého navrhujeme úpravy, ale kde by mělo být v zájmu přehlednosti zachováno číslování původního seznamu.

\mar Tento příkaz zachová číslování položek, ale přidá písmeno nebo znak na okraj (který je mu předán jako argument mezi kudrnaté závorky). Nejsem si úplně jistý, nakolik je užitečný.

Existují dva další příkazy pro zadávání položek, jejichž kombinace přináší velmi zajímavé efekty, a pokud to mohu říci, myslím, že je lepší je vysvětlit na příkladu. `\ran` (zkratka *rozsah*) a `\its`, zkratka *položky*. První z nich přebírá argument (mezi kudrnatými závorkami) a druhá opakuje symbol použitý jako oddělovač v seznamu x-krát (standardně 4krát, ale můžeme to změnit pomocí volby `items`). Následující příklad ukazuje, jak mohou tyto dva příkazy spolupracovat při vytváření seznamu, který napodobuje dotazník:

Po přečtení následujícího úvodu odpovězte na následující otázky:

```
\startitemize[5, packed][width=8em, distance=2em, items=5]
\ran{Ne \hss Ano}
\its Nikdy nebudu používat \ConTeXt, je příliš složitý.
\its Budu ho používat jen pro psaní velkých knih.
\its Budu ho používat vždy.
\its Líbí se mi tak moc, že své další dítě pojmenuji \quotation{Hans}, jako poctu
Hansi Hagenovi.
\stopitemize
```

Po přečtení tohoto úvodu odpovězte na následující otázky:

Ne	Ano	
○ ○ ○ ○ ○		Nikdy nebudu používat ConTeXt, je to příliš složitě.
○ ○ ○ ○ ○		Budu ho používat jen pro psaní velkých knih.

- ◦ ◦ ◦ ◦ Vždycky ho budu používat.
- ◦ ◦ ◦ ◦ Mám ho tak rád, že své další dítě pojmenuji “Hans”, jako poctu Hansi Hagenovi.

3.3.4 Základní konfigurace seznamu

Připomínáme, že “`itemize`” umožňuje použít dva argumenty. Již jsme viděli, že první argument nám umožňuje vybrat typ seznamu, který chceme. Můžeme jej však použít i k určení dalších vlastností seznamu; k tomu slouží následující volby pro “`itemize`” v jeho prvním argumentu:

- **columns**: tato volba určuje, že seznam se skládá ze dvou nebo více sloupců. Za volbou sloupce je třeba zapsat požadovaný počet sloupců jako slova oddělená čárkou: dva, tři, čtyři, pět, šest, sedm, osm nebo devět. **columns**, za kterým nenásleduje žádné číslo, generuje dva sloupce.
- **intro**: tato volba se snaží neoddělovat seznam zalomením řádku od odstavce, který mu předchází.
- **continue**: v uspořádaných seznamech (číselných nebo abecedních) tato volba způsobí, že seznam bude pokračovat v číslování od posledního číslovaného seznamu. Pokud je použita volba **continue**, není nutné uvádět, jaký typ seznamu chceme, protože se předpokládá, že bude stejný jako poslední číslovaný seznam.
- **packed**: je jednou z nejpoužívanějších možností. Její použití způsobí, že vertikální prostor mezi jednotlivými položkami v seznamu se co nejvíce zmenší.
- **nowhite**: vytváří podobný efekt jako **packed**, ale drastičtější: zmenšuje nejen svislou mezeru mezi položkami, ale také svislou mezeru mezi seznamem a okolním textem.
- **broad**: zvětší vodorovnou mezeru mezi oddělovačem položek a textem položky. Mezeru lze zvětšit vynásobením čísla číslem **široký** jako například `\startitemize[2* široký]`.
- **serried**: odstraní vodorovnou mezeru mezi oddělovačem položek a textem.
- **intext**: odstraní závěsnou odrážku.
- **text**: odstraní závěsnou zarážku a zmenší vertikální mezeru mezi položkami.
- **repeat**: ve vnořených seznamech způsobí, že číslování podřízené úrovně *repeat* je na stejné úrovni jako předchozí úroveň. Takto bychom měli na první úrovni:

1, 2, 3, 4; na druhé úrovni: 1.1, 1.2, 1.3 atd. Volba musí být uvedena pro vnitřní seznam, nikoli pro vnější seznam.

- **margin, inmargin:** ve výchozím nastavení je oddělovač seznamu vytištěn vlevo, ale uvnitř textové oblasti (**atmargin**). Volby **margin** a **inmargin** přesunou oddělovač na okraj.

3.3.5 Dodatečná konfigurace seznamu

Druhý argument, rovněž nepovinný, v **\startitemize** umožňuje podrobnější a důkladnější konfiguraci seznamů.

- **before, after:** příkazy, které se spustí před spuštěním, resp. po uzavření prostředí itemize.
- **inbetween:** příkaz, který se spustí mezi dvěma **items**.
- **beforehead, afterhead:** příkaz, který se spustí před nebo za položkou zadanou příkazem **\head**.
- **left, right:** znak, který se má vytisknout vlevo nebo vpravo od oddělovače. Chceme-li například získat abecední seznamy, v nichž jsou písmena obklopena závorkami, museli bychom napsat:

```
\startitemize[a][left=(, right=)]
```

- **stopper:** označuje znak, který se zapíše za oddělovač. Funguje pouze v uspořádaných seznámech.
- **width, maxwidth:** šířka prostoru vyhrazeného pro oddělovač, a tedy pro závěsnou odrážku.
- **factor:** reprezentativní číslo oddělovacího faktoru mezi oddělovačem a textem.
- **distance:** míra vzdálenosti mezi oddělovačem a textem.
- **leftmargin, rightmargin, margin:** margin, který se přidá k levému (**leftmargin**) nebo pravému (**rightmargin**) okraji položek.
- **start:** číslo, od kterého bude číslování položek začínat.
- **symalign, itemalign, align:** zarovnání položek. Umožňuje stejné hodnoty jako **\setupalign**. **symalign** řídí zarovnání oddělovače; **itemalign** textu položky a **align** zarovnání obou.
- **indenting:** odsazení prvního řádku v odstavcíchv prostředí. Viz [section 2.1.1](#)

- `indentnext`: určuje, zda má být odstavec za prostředím odsazen, nebo ne. Hodnoty jsou *ano*, *ne* a *auto*.
- `items`: v položkách zadaných pomocí `\its` udává počet opakování oddělovače.
- `style`, `color`; `headstyle`, `headcolor`; `marstyle`, `marcolor`; `symstyle`, `symcolor`: tyto volby řídí styl a barvu položek, které jsou vloženy do prostředí pomocí příkazů `\item`, `\head`, `\mar` nebo `\sym`.

3.3.6 Jednoduché seznamy s příkazem `\items`

Alternativou prostředí `itemize` pro velmi jednoduché nečíslované seznamy, kde položky nejsou příliš velké, je příkaz `\položky`, jehož syntaxe je:

```
\items[Configuration]{Položka 1, Položka 2, ..., položka n}
```

Jednotlivé položky seznamu jsou od sebe odděleny čárkami. Například:

```
Grafické soubory mohou mít  
mimo jiné tyto přípony:
```

```
\items{png, jpg, tiff, bmp}
```

Grafické soubory mohou mít mimo jiné tyto přípony:

- png
- jpg
- tiff
- bmp

Konfigurační možnosti podporované tímto příkazem jsou podmnožinou možností `itemize`, s výjimkou dvou specifických možností pro tento příkaz:

- `symbol`: tato volba určuje typ seznamu, který bude vygenerován. Podporuje stejné hodnoty jako `itemize` pro výběr určitého typu seznamu.
- `n`: tato volba určuje, od kterého čísla položky bude oddělovač.

3.3.7 Předurčení chování seznamu a vytvoření vlastních typů seznamů

V předchozích částech jsme si ukázali, jak určit, jaký typ seznamu chceme a jaké vlastnosti by měl mít. Dělat to však při každém volání seznamu je neefektivní a obvykle to vede k vytvoření nesouvislého dokumentu, v němž má každý seznam svůj vlastní vzhled, aniž by však jednotlivé vzhledy splňovaly nějaká kritéria.

Preferovaný výsledek pro tuto oblast:

- Předurčuje *normální* chování `itemize` a `\items` v preambuli dokumentu.

- Vytváření vlastních seznamů na míru. Například: abecedně číslovaný seznam, který chceme nazvat *ListAlpha*, seznam číslovaný římskými číslicemi (*ListRoman*) atd.

Prvního dosáhneme pomocí příkazů `\setupitemize` a `\setupitems`. Druhé vyžaduje použití buď příkazu `\defineitemgroup`, nebo `\defineitems`. První vytvoří prostředí seznamu podobné příkazu `itemize` a druhý příkaz podobný příkazu `items`.

3.4 Popisy a výčty

Popisy a výčty jsou dvě konstrukce, které umožňují konzistentní sazbu odstavců nebo skupin odstavců, které rozvíjejí, popisují nebo definují frázi nebo slovo.

3.4.1 Popisy

U popisů rozlišujeme mezi *title* a jeho vysvětlením nebo rozvedením. Nový popis můžeme vytvořit pomocí:

```
\definedescription[Name] [Configuration]
```

kde *Name* je název, pod kterým bude tato nová konstrukce známá, a *Configuration* určuje, jak bude naše nová konstrukce vypadat. Po předchozím příkazu budeme mít nový příkaz a prostředí s názvem, který jsme si zvolili. Tedy:

```
\definedescription[Concept]
```

vytvoří příkazy:

```
\Concept{Název}
\startConcept {Title} ... \stopConcept
```

Příkaz použijeme pro případ, kdy vysvětlující text názvu tvoří pouze jeden odstavec, a prostředí pro názvy, jejichž popis zabírá více než jeden odstavec. Při použití příkazu se za vysvětlující text titulu bude považovat odstavec, který následuje bezprostředně za ním. Při použití prostředí však bude veškerý obsah formátován s příslušným odsazením, aby bylo zřejmé, jak souvisí s nadpisem.

Například:

```
\definedescription
[Concept] [alternative=left, width=1cm, headstyle=bold]

\Concept{Kontextualizovat}
```

Umístěte něco do určitého kontextu nebo napište text pomocí systému pro sazbu `\ConTeXt`. Schopnost správně zařadit text do kontextu v jakékoli situaci je považována za známku inteligence a zdravého rozumu.

Tím se získá následující výsledek:

Kontextualizovat Umístěte něco do určitého kontextu nebo napište text pomocí systému pro sazbu ConTeXt. Schopnost správně zařadit text do kontextu v jakékoli situaci je považována za známku inteligence a zdravého rozumu.

Stejně jako v případě ConTeXt lze vzhled, který bude mít naše nová konstrukce, určit při jejím vytváření pomocí argumentu *Konfigurace* nebo později pomocí `\setupdescription`:

```
\setupdescription[Name] [Configuration]
```

kde *Název* je název našeho nového popisu a *Konfigurace* určuje, jak bude vypadat. Mezi různými možnými konfiguracemi vyzdvihnu např:

- **alternative**: Tato možnost zásadně ovlivňuje vzhled konstrukce. Určuje umístění nadpisu ve vztahu k jeho popisu. Její možné hodnoty jsou `left`, `right`, `inmargin`, `inleft`, `inright`, `margin`, `leftmargin`, `rightmargin`, `innermargin`, `outermargin`, `serried`, `hanging`, jejich názvy jsou dostatečně jasné, abyste si udělali představu o výsledku, i když v případě pochybností je nejlepší provést test, abyste zjistili, jak to vypadá.
- **width**: určuje šířku rámečku, do kterého bude napsán nadpis. V závislosti na hodnotě **alternative** bude tato vzdálenost také součástí odsazení, s nímž bude napsán vysvětlující text.
- **distance**: určuje vzdálenost mezi nadpisem a vysvětlením.
- **headstyle**, **headcolor**, **headcommand**: ovlivňuje vzhled samotného nadpisu: Styl (**headstyle**) a barva (**headcolor**). Pomocí **headcommand** můžeme uvést příkaz, kterému bude text nadpisu předán jako argument. Například: `headcommand=\WORD` zajistí, že text nadpisu bude psán velkými písmeny.
- **style**, **color**: řídí vzhled popisného textu nadpisu.

3.4.2 Výčty

Výčty jsou číslované textové prvky strukturované na několika úrovních. Každý prvek začíná názvem, který se ve výchozím nastavení skládá z názvu struktury a jejího čísla, ačkoli název můžeme změnit pomocí volby **text**. Jsou docela podobné popisům, i když se v tom liší:

- Všechny prvky výčtu mají stejný název.
- Proto se od sebe liší svým číslováním.

Tato struktura může být velmi užitečná například při psaní vzorců, úloh nebo cvičení v učebnici, kdy je třeba zajistit jejich správné číslování a jednotné formátování.

Vytvoříme výčet s

```
\defineenumeration[Name] [Configuration]
```

kde *Název* je název nové konstrukce a *Konfigurace* určuje, jak bude vypadat.

V následujícím příkladu:

```
\defineenumeration
[Exercise]
[alternative=top, before=\blank, after=\blank, between=\blank]
```

Vytvořili jsme novou strukturu s názvem *Cvičení* a jak už to u výčtů bývá, budeme mít k dispozici následující nové příkazy:

```
\Exercise
\startExercise
```

Příkaz se používá pouze pro jeden odstavec *cvičení* a prostředí pro více odstavců *cvičení*. Protože však výčty mohou být až čtyřúrovňové, budou vytvořeny i následující příkazy a prostředí:

```
\subExercise
\startsubExercise
\stopsubExercise
\subsubExercise
\startsubsubExercise
\stopsubsubExercise
\subsubsubExercise
\startsubsubsubExercise
\stopsubsubsubExercise
```

A k ovládání číslování slouží následující doplňkové příkazy:

- `\setEnumerationName`: nastaví aktuální hodnotu číslování.
- `\resetEnumerationname`: nastaví čítač výčtu na nulu.
- `\nextEnumerationName`: zvýší čítač výčtu o jedna.

Vzhled výčtů lze určit v okamžiku jejich vytvoření nebo později pomocí `\setupenumeration`, jehož formát je:

```
\setupenumeration[Name] [Configuration].
```

Pro každý výčet můžeme nakonfigurovat každou jeho úroveň zvlášť. Tak například `\setupenumeration [subExercise] [Configuration]` ovlivní druhou úroveň výčtu s názvem “Exercise”.

Možnosti a hodnoty konfigurovatelné pomocí `\setupenumeration` jsou podobné jako v `\setupdescription`.

3.5 Čáry a rámce

V referenční příručce ConT_EXt se píše, že T_EX má obrovské možnosti správy textu, ale je velmi slabý ve správě grafických informací. Dovolím si nesouhlasit: je pravda, že pro práci s řádky a rámečky nejsou možnosti ConT_EXtu (vlastně T_EXu) tak ohromující jako při sazbě textu. Ale pokračovat v tvrzení, že systém je v tomto ohledu slabý, je podle mě poněkud přitažené za vlasy. Nevím o žádné funkci s řádky a rámečky, kterou by jiné systémy pro sazbu dokumentů uměly a kterou by ConT_EXt nedokázal vygenerovat. A pokud ConT_EXt zkombinujeme s MetaPostem nebo s TiKZ (ConT_EXt má pro to rozšiřující modul), pak jsou možnosti omezeny pouze naší představivostí.

V následujících částech se však omezím na vysvětlení, jak vytvářet jednoduché vodorovné a svislé čáry a rámečky kolem slov, vět nebo odstavců.

3.5.1 Jednoduché čáry

Nejjednodušší způsob kreslení vodorovné čáry je pomocí příkazu `\hairline`, který vytvoří vodorovnou čáru zabírající celou šířku normálního textového řádku.

Na řádku, kde se nachází řádek generovaný příkazem `\hairline`, nesmí být žádný text. Abychom mohli vygenerovat řádek, který může koexistovat s textem na stejném řádku, potřebujeme příkaz `\thinrule`. Tento druhý příkaz použije celou šířku řádku. Proto bude mít v izolovaném odstavci stejný účinek jako `\hairline`, zatímco v opačném případě `\thinrule` vytvoří stejnou horizontální expanzi jako `\hfill` (viz [section 1.3.3](#)), ale místo aby vyplnil horizontální prostor bílým místem (jako `\hfill`), vyplní jej řádkem.

```
Na levé straně\thinrule\\
\thinrule Na pravé straně\\
Na obou\thinrule stranách\\
\thinrule vystředěno\thinrule
```

Na levé straně_____		_____Na pravé straně
_____		_____
Na obou_____		_____stranách
_____		_____
_____vystředěno_____		_____

Pomocí příkazu `\thinrules` můžeme vygenerovat několik řádků. Například `\thinrules[n=2]` vygeneruje dva po sobě jdoucí řádky, každý o šířce normálního řádku. Řádky vygenerované příkazem `\thinrules` lze také konfigurovat, a to

buď ve vlastním volání příkazu s uvedením konfigurace jako jednoho z jeho argumentů, nebo obecně příkazem `\setupthinrules`. Konfigurace zahrnuje tloušťku čáry (`rulethickness`), její barvu (`color`), barvu pozadí (`background`), mezeru mezi řádky (`interlinespace`) atd.

Řadu možností ponechám bez vysvětlení. Čtenář si je může prohlédnout v `setup-cs.pdf` (viz sekci `[sec:qrc-setup-cs]`). Některé možnosti se od jiných liší pouze nuancemi (tj. není mezi nimi téměř žádný rozdíl) a myslím, že je rychlejší, když se čtenář pokusí *vidět* rozdíl, než abych se ho snažil sdělit slovy. Například: Tloušťka čáry, kterou jsem právě řekl, závisí na volbě `rulethickness`. Ovlivňují ji však také volby `height` a `depth`.

Menší čáry lze generovat pomocí příkazů `\hl` a `\vl`. První příkaz generuje vodorovnou čáru a druhý svislou čáru. Oba příkazy přijímají jako parametr číslo, které nám umožňuje vypočítat délku čáry. V příkazu `\hl` číslo měří délku v *ems* (v příkazu není třeba uvádět jednotku měření) a v příkazu `\vl` se argument vztahuje k aktuální výšce čáry.

`\hl[3]` tedy generuje vodorovnou čáru o délce 3 *ems* a `\vl[3]` generuje svislou čáru o výšce odpovídající třem čarám. Nezapomeňte, že indikátor měření čáry musí být vložen mezi hranaté závorky, nikoli mezi závorky kudrnaté. V obou příkazech je argument nepovinný. Pokud není zadán, předpokládá se hodnota 1.

`\fillinline` je další příkaz pro vytvoření vodorovných čar přesné délky. Podporuje více konfigurací, ve kterých můžeme kromě některých dalších funkcí uvést (nebo předem určit pomocí `\setupfillinlines`) šířku (volba `width`).

Zvláštností tohoto příkazu je, že text, který je napsán vpravo, se umístí na levou stranu řádku a oddělí jej od řádku potřebným bílým místem, aby zabíral celý řádek. Například:

```
\fillinline[width=6cm] Název
```

vygeneruje

Name



Domnívám se, že tato podivná operace je způsobena tím, že toto makro bylo navrženo pro psaní formulářů, kde je za textem vodorovná čára, na kterou je třeba něco napsat. Ve skutečnosti samotný název příkazu `fillinline` znamená vyplnit řádek.

Kromě šířky čáry můžeme nastavit i její okraj (`margin`), vzdálenost (`distance`), tloušťku (`rulethickness`) a barvu (`color`).

Téměř totožný s `\fillinline` je `\fillinrules`, i když tento příkaz umožňuje vložit více než jeden řádek (volba “*n*”).

```
\fillinrules[Configuration] {Text} {Text}
```

kde tři argumenty jsou nepovinné.

3.5.2 Čáry spojené s textem

Ačkoli některé z příkazů, které jsme právě viděli, mohou generovat řádky, které koexistují s textem na stejném řádku, tyto příkazy se ve skutečnosti zaměřují na rozložení řádku. Pro psaní řádků spojených s určitým textem má ConT_EXt příkazy:

- `\hr`, která generuje text mezi řádky.
- `\hrule`, která generuje čáry pod textem (podtržení), nad textem (překrytí) nebo přes text (přeškrtnutí).

Pro generování textu mezi řádky je obvyklý příkaz `\textrule`. Tento příkaz nakreslí čáru přes celou šířku stránky a text, který bere jako parametr, zapíše na levou stranu (ale ne na okraj). Například:

```
\textrule{text příkladu}
```

— Příklad textu —



Předpokládá se, že `\textrule` umožňuje volitelný první argument se třemi možnými hodnotami: `top`, `middle` a `bottom`. Po několika testech se mi však nepodařilo zjistit, jaký účinek tyto volby mají.

Podobné prostředí jako `\textrule` je `\starttextrule`, které kromě vložení řádku s textem na začátek prostředí vloží vodorovnou čáru na jeho konec. Formát tohoto příkazu je:

```
\starttextrule[Configuration]{Text na řádku} ... \stoptextrule
```

`\textrule` i `\starttextrule` lze konfigurovat pomocí `\setuptextrule`.

Pro kreslení čar pod, nad nebo přes text se používají následující příkazy:

```
\underbar{Text}
\underbars{Text}
\overbar{Text}
\overbars{Text}
\overstrike{Text}
\overstrikes{Text}
```

Jak vidíme, pro každý typ řádku (pod, nad nebo přes text) existují dva příkazy. Verze příkazu v jednotném čísle nakreslí jedinou čáru pod, přes nebo skrz celý text, který je brán jako argument, zatímco verze příkazu v množném čísle nakreslí čáru pouze přes slova, ale nikoli přes bílé místo.

Tyto příkazy nejsou vzájemně kompatibilní, to znamená, že dva z nich nelze použít na stejný text. Pokud se o to pokusíme, bude mít vždy přednost poslední z nich. Na druhou stranu lze `\underbar` vnořit a podtrhnout to, co již bylo podtrženo.

Referenční příručka upozorňuje, že `\underbar` vypíná spojování slov v textu, která jsou jeho argumentem. Není mi jasné, zda se toto tvrzení vztahuje pouze na `\underbar` nebo na šest příkazů, které zkoumáme.

3.5.3 Orámovaná slova nebo texty

K obklopení textu rámečkem nebo mřížkou používáme:

- Příkazy `\framed` nebo `\inframed`, pokud je text relativně krátký a nezabírá více než jeden řádek.
- Prostředí `\startframedtext` pro delší texty.

Rozdíl mezi `\zarámovaný` a `\nezarámovaný` spočívá v bodě, ze kterého je rámeček vykreslen. V `\frame` se rámeček kreslí směrem nahoru od ideální čáry, zvané základní čára, na které písmena spočívají, ale některá písmena procházejí směrem dolů. V `\inframed` se rámeček kreslí rovněž směrem nahoru, a to od nejnižšího možného bodu na čáře. Například:

Tady jsou `\framed{dva}` dobré
`\inframed{rámečky}`.

Zde jsou dva dobré rámečky.

Oba, ohraničený i záramovaný text, lze přizpůsobit pomocí `\setupframed` a `\startframedtext` se přizpůsobuje pomocí `\setupframedtext`. Možnosti přizpůsobení obou příkazů jsou dosti podobné. Umožňují nám určit rozměry rámečku (`height`, `width`, `depth`), tvar rohů (`framecorner`), které mohou být `rectangular` nebo kulaté (`round`), barvu rámečku (`framecolor`), tloušťku čáry (`framethickness`), zarovnání obsahu (`align`), barvu textu (`foregroundcolor`), barvu pozadí (`background` a `backgroundcolor`) atd.

Pro `\startframedtext` existuje také zdánlivě zvláštní vlastnost: `frame=off`, která způsobí, že se rámeček nevykreslí (ačkoli je stále přítomen, ale neviditelný). Tato vlastnost existuje proto, že vzhledem k tomu, že rámeček kolem odstavce je nedělitelný, je běžné, že celý odstavec je uzavřen v prostředí `framedtext` s vypnutou možností vykreslování rámečku, aby se zajistilo, že v odstavci nebudou vloženy žádné zlomy stránek.

Můžeme také vytvořit vlastní verzi těchto příkazů pomocí `\defineframed` a `\defineframedtext`.

3.6 Další zajímavá prostředí a stavby

V ConT_EXt je ještě mnoho prostředí, o kterých jsem se nezmínil, nebo jen velmi okrajově. Jako příklad uvedu:

- **buffer** *Buffery* jsou fragmenty textu uložené v paměti pro pozdější opětovné použití. *buffer* je definován někde v dokumentu pomocí `\startbuffer[Buffer-Name] ... \stopbuffer` a lze jej libovolně často načítat na jiném místě dokumentu pomocí `\getbuffer[BufferName]`.
- **chemical** Toto prostředí nám umožňuje umístit do něj chemické vzorce. Jestliže T_EX vyniká, kromě mnoha jiných věcí, svou schopností správně psát texty s matematickými vzorci, ConT_EXt se od počátku snažil rozšířit tuto schopnost na chemické vzorce a má toto prostředí, kde jsou povoleny příkazy a struktury pro psaní chemických vzorců.
- **kombinace** Toto prostředí nám umožňuje kombinovat několik plovoucích prvků na jedné stránce. Je užitečné zejména pro zarovnání různých propojených externích obrázků v našem dokumentu.
- **formula** Jedná se o prostředí určené k psaní matematických vzorců.
- **hiding** Text uložený v tomto prostředí nebude zkompileován, a proto se neobjeví ve výsledném dokumentu. To je užitečné pro dočasné zakázání kompilace určitých fragmentů ve zdrojovém souboru. Toho samého dosáhnete označením jednoho nebo více řádků jako komentář. Pokud je však fragment, který chceme zakázat, relativně dlouhý, je efektivnější než označit desítky či stovky řádků zdrojového souboru jako komentář vložit na začátek fragmentu příkaz `\starthiding` a na jeho konec `\stophiding`.
- **legend** V matematickém kontextu T_EX používá jiná pravidla, takže nelze psát normální text. Někdy je však vzorec doprovázen popisem prvků, které jsou v něm použity. K tomuto účelu slouží prostředí `\startlegend`, které nám umožňuje umístit normální text do matematického kontextu.
- **linecorrection** Obvykle ConT_EXt správně spravuje svislou mezeru mezi řádky, ale občas se může stát, že řádek bude obsahovat něco, co ho zkreslí. To se stává hlavně u řádků, které mají fragmenty orámované pomocí `\framed`. V takových případech toto prostředí upraví řádkování tak, aby odstavec vypadal správně.
- **mode** Toto prostředí je určeno k zahrnutí fragmentů do zdrojového souboru, které budou zkompileovány pouze v případě, že je aktivní příslušný režim. Použití *modes* není předmětem tohoto úvodu, ale je to velmi zajímavý nástroj, pokud chceme mít možnost generovat několik verzí s různými formáty z jednoho zdrojového souboru. Doplňujícím prostředím k tomuto prostředí je `\startnotmode`.

- **opposite** Toto prostředí se používá pro sazbu textů, pokud spolu obsah levé a pravé stránky souvisí.
- **quotation** Velmi podobné prostředí jako **narrower**, určené pro vkládání středně dlouhých doslovných citací. Prostředí zajišťuje, že text uvnitř je citován a že okraje jsou zvětšeny tak, aby odstavec s citací na stránce vizuálně vynikl. Je však třeba poznamenat, že podle obvyklého stylu blokových uvozovek v angličtině by neměly být žádné úvodní a závěrečné uvozovky – což činí tento příkaz nebo prostředí méně užitečným.
- **standardmakeup** Toto prostředí je určeno ke generování stránek s názvem dokumentu, což je poměrně běžné u akademických dokumentů určité délky, jako jsou doktorské práce, magisterské práce apod.

Chcete-li se seznámit s některým z těchto prostředí (nebo s dalšími, která jsem nezmínil), doporučuji následující kroky:

1. Informace o prostředí najdete v referenční příručce ConT_EXt. V této příručce nejsou uvedena všechna prostředí, ale o každé položce z výše uvedeného seznamu se v ní něco píše.
2. Napište testovací dokument, ve kterém je použito prostředí.
3. Vyhledejte v oficiálním seznamu příkazů ConT_EXtu (viz section ??) konfigurační volby pro dané prostředí a pak je vyzkoušejte, abyste zjistili, co přesně dělají.

Chapter 4

Obrázky, tabulky a další plovoucí objekty

Table of Contents: 4.1 Co jsou plovoucí objekty a co dělají?; 4.2 Externí obrázky; 4.2.1 Přímé vkládání obrázků; 4.2.2 Vložení obrázku pomocí `\place-figure`; 4.2.3 Vkládání obrázků integrovaných do textového bloku; 4.2.4 Vložená konfigurace a transformace obrázků; A Vložení možnosti příkazu, které způsobí určitou transformaci obrázku; B Specifické příkazy pro transformaci obrázku; 4.3 Tabulky; 4.3.1 Obecné představy o tabulkách a jejich umístění v dokumentu; 4.3.2 Jednoduché tabulky s prostředím `tabulate`; 4.4 Aspekty společné pro obrázky, tabulky a další plovoucí objekty; 4.4.1 Možnosti vložení plovoucího objektu; 4.4.2 Konfigurace názvů plovoucích objektů; 4.4.3 Kombinované vložení dvou nebo více objektů; 4.4.4 Obecná konfigurace plovoucích objektů; 4.5 Definování dalších plovoucích objektů;

Tato kapitola je především o plovoucích předmětech (plovoucích). Ale v návaznosti na tento koncept jej využívá k vysvětlení dvou typů objektů, které nemusí být nutně plovoucí, ačkoli jsou často konfigurovány, jako by byly: externí obrázky a tabulky. Při pohledu na obsah této kapitoly by si někdo mohl myslet, že je to všechno velmi neuspořádané: začíná to povídáním o plovoucích objektech, pak pokračuje povídáním o obrázcích a tabulkách a končí se znovu povídáním o plovoucích objektech. Důvody této nepořádnosti jsou *pedagogické*: obrázky a tabulky lze vysvětlit, aniž bychom příliš trvali na tom, že jsou normálně plovoucí; a přesto, když je začneme zkoumat, hodně nám pomůže zjištění, že, překvapení, už víme o dvou plovoucích objektech.

4.1 Co jsou plovoucí objekty a co dělají?

Pokud by dokument obsahoval pouze *normální* text, bylo by stránkování relativně snadné: znalost maximální výšky textové oblasti stránky stačí k měření výšky různých odstavců, abyste věděli, kam vložit konce stránek. Problém je v tom, že v mnoha dokumentech jsou objekty, fragmenty nebo nedělitelné bloky textu, jako je obrázek, tabulka, vzorec, orámovaný odstavec atd.

Někdy mohou tyto objekty zabírat velkou část stránky, což zase představuje problém, že pokud je musíte vložit na určité místo v dokumentu, nemusí se vejít na

aktuální stránku a musí být náhle přerušeny, takže velký prázdný prostor ve spodní části, takže dotyčný objekt a text, který za ním následuje, se přesune na další stránku. Pravidla dobré sazby však naznačují, že kromě poslední stránky kapitoly by mělo být na každé stránce stejné množství textu.

Je proto vhodné vyhnout se výskytu velkých prázdných vertikálních mezer; a *plovoucí* objekty jsou hlavním mechanismem, jak toho dosáhnout. “plovoucí objekt” je takový, který nemusí být umístěn v přesném bodě dokumentu, ale může se kolem něj *pohybovat* nebo *vznášet*. Cílem je umožnit ConTeXtu rozhodnout o nejlepším místě, z hlediska stránkování, k umístění takových objektů a dokonce jim povolíji přesunout se na jinou stránku; ale vždy se snažte nevzdálit se příliš daleko od bodu zahrnutí do zdrojového souboru.

Proto neexistují žádné objekty, které by musely být plovoucí. Ale jsou předměty, které občas potřebují být plovoucí. Rozhodnutí je každopádně na autorovi nebo osobě odpovědné za sazbu, jde-li o dvě různé osoby.

Možnost změny přesného umístění nedělitelného předmětu nepochybně velmi usnadňuje sazbu pěkně vyvážených stránek; ale problém, který s tím souvisí, je ten, že jelikož v době psaní originálu nevíme, kde přesně takový objekt skončí, je těžké na něj odkazovat. Pokud jsme tedy například do svého dokumentu právě vložil příkaz, který vkládá obrázek a v dalším odstavci jej chci popsat a napsat o něm něco jako: “Jak můžete vidět z předchozího obrázku”, když postava *plave* mohl by být klidně umístěn *za* co jsem právě napsal a výsledkem je nekonzistence: čtenář hledá obrázek *před* textem, který na něj odkazuje a nemůže jej najít, protože plovoucí obrázek skončil *za* tímto odkazem.

Toto je opraveno *číslováním* plovoucích objektů (po jejich rozdělení do kategorií), takže namísto odkazování na obrázek jako “předchozí obrázek” nebo “další obrázek” budeme odkazovat na něj jako “obrázek 1.3”, protože můžeme použít vnitřní referenční mechanismus ConTeXtu, abychom zajistili, že číslo obrázku bude vždy aktuální (viz [section ??](#)). Číslování těchto druhů objektů na druhou stranu usnadňuje vytvoření jejich rejstříku (index tabulek, grafů, obrázků, rovnic atd.). Jak to udělat, viz ([section 4.2](#)).

Mechanismus pro práci s plovoucími objekty v ConTeXtu je poměrně sofistikovaný a občas tak abstraktní, že nemusí být vhodný pro začátečníky. Proto v této kapitole začnu vysvětlením pomocí dvou konkrétních případů: obrázků a tabulek. Pak se to pokusím poněkud zobecnit, abychom pochopili, jak rozšířit mechanismus na další druhy objektů.

4.2 Externí obrázky

Jak čtenář v této fázi ví (protože to bylo vysvětleno v [section ??](#)), ConTeXt je dokonale integrován s MetaPostem a může generovat obrázky a grafiku, které jsou

naprogramované ve velké míře stejným způsobem, jako se programují transformace textu. Existuje také rozšiřující modul pro ConT_EXt¹, který mu umožňuje pracovat s TiKZ.² Ale takovými obrázky se v tomto úvodu nezabýváme (to by si pravděpodobně vynutilo zdvojnásobení jeho délky). Mám zde na mysli použití externích obrázků, které jsou umístěny v souboru na našem pevném disku nebo jsou staženy přímo z internetu pomocí ConT_EXtu.

4.2.1 Přímé vkládání obrázků

K přímému vložení obrázku (ne jako plovoucího objektu) použijeme příkaz `\externalfigure`, jehož syntaxe je

`\externalfigure[Název] [Konfigurace]`

kde

- *Jméno* může být buď název souboru obsahujícího obrázek, nebo webová adresa obrázku nalezeného na internetu, nebo symbolický název, který jsme dříve přiřadili k obrázku pomocí příkazu `\useexternalfigure`, jehož formát je podobný formátu `\externalfigure`, i když vyžaduje první argument se symbolickým názvem, který bude spojen s daným obrázkem.
- *Konfigurace* je volitelný argument, který nám umožňuje aplikovat určité transformace na obrázek předtím, než je vložen do našeho dokumentu. Tento argument blíže prozkoumáme v [section 4.2.4](#).

Povolené formáty obrázků jsou pdf, mps, jpg, png, jp2, jbig, jbig2, jb2, svg, eps, gif nebo tif. ConT_EXt umí přímo spravovat osm z nich, zatímco zbytek (svg, eps, gif nebo tif) je třeba před otevřením převést pomocí externího nástroje, který se mění podle formátu, a proto musí být nainstalován do systému tak, aby ConT_EXt mohl manipulovat s těmito druhy souborů.

Mezi formáty podporované `\externalfigure` jsou také některé video formáty. Konkrétně: QuickTime (přípona .mov), Flash Video (přípona .flv) a MPeg 4 (přípona .mp4). Většina přehrávačů PDF však neví, jak zacházet se soubory PDF s vloženým videem. Nemohu k tomu moc říct, protože jsem žádné testy nedělal.

Příponu souboru není třeba uvádět: ConT_EXt vyhledá soubor se zadaným názvem a jednou z přípon známých formátů obrázků. Je-li více kandidátů, použije se

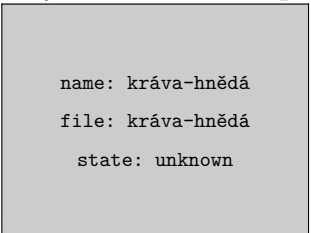
¹ ConT_EXt rozšiřující moduly, které mu poskytují další nástroje, ale nejsou zahrnuty v tomto úvodu.

² Toto je grafický programovací jazyk určený pro práci se systémy založenými na T_EXu. Je to “rekurzivní zkratka” z německé věty “TiKZ ist keinen Zeichenprogramm”, což v překladu znamená: “TiKZ není kreslicí program”. Rekurzivní zkratky jsou jakýmsi žertem programátorů. Když pomineme MetaPost (který neumím používat), domnívám se, že TiKZ je skvělý systém pro programování grafiky.

nejprve formát PDF, pokud existuje a v případě jeho absence formát MPS (grafika generovaná MetaPostem). Pokud tyto dva neexistují, postupuje se v následujícím pořadí: jpeg, png, jpeg 2000, jbig a jbig2.

Pokud skutečný formát obrázku neodpovídá příponě souboru, ve kterém je uložen, ConT_EXt jej nemůže otevřít, pokud neoznačíme aktuální formát obrázku pomocí volby `method`.

Pokud obrázek není umístěn samostatně mimo odstavec, ale je integrován do textového odstavce a jeho výška je větší než řádkování, řádek se upraví tak, aby se předešlo překrývání obrázku s předchozími řádky, jako příklad, který doprovází



```
name: kráva-hnědá
file: kráva-hnědá
state: unknown
```

tento řádek

ConT_EXt standardně vyhledává obrázky v pracovním adresáři, ve svém nadřazeném adresáři a v nadřazeném adresáři tohoto adresáře. Umístění adresáře obsahujícího obrázky, se kterými budeme pracovat, můžeme označit pomocí volby `directory` příkazu `\setupexternalfigures`, která přidá tento adresář do vyhledávací cesty. Pokud chceme, aby vyhledávání probíhalo pouze v adresáři s obrázky, musíme nastavit i volbu `location`. Takže například, aby náš dokument hledal všechny obrázky, které potřebujeme v adresáři “img”, měli bychom napsat:

```
\setupexternalfigures
[directory=img, location=global]
```

Ve volbě `directory` v `\setupexternalfigures` můžeme zahrnout více než jeden adresář a oddělit je čárkami; ale v tomto případě musíme adresáře uzavřít do složených závorek. Například “`directory={img, ~/imágenes}`”.

V adresáři vždy používáme znak ‘/’ jako oddělovač mezi adresáři; včetně systému Microsoft Windows, jehož operační systém používá jako oddělovač adresářů ‘\’.

`\externalfigure` je také schopen používat obrázky hostované na internetu. Takže například následující úryvek vloží logo CervanTeX přímo z internetu do dokumentu. Toto je T_EX španělsky mluvící uživatelská skupina: ¹

¹ Internetové adresy jsou velmi dlouhé a pro zobrazení příkladu se dvěma sloupci není k dispozici mnoho místa. Aby tedy pořadí v levém sloupci správně sedělo, vložil jsem do webové adresy zalomení řádku. Pokud někdo chce příklad zkopírovat a vložit, nebude to fungovat, pokud se tento konec řádku neodstraní.

```
\externalfigure
[http://www.cervantex.es/files/
cervantex/cervanTeXcolor-small.jpg]
```



Když je dokument obsahující vzdálený soubor poprvé zkompileován, je stažen ze serveru a uložen do adresáře mezipaměti LuaTeXu. Tento soubor v mezipaměti se používá při následných kompilacích. Normálně se vzdálený obrázek stáhne znovu, pokud je obrázek v mezipaměti starší než 1 den. Chcete-li změnit tento práh, podívejte se na [ConTeXt wiki](#).

Pokud ConTeXt nenajde obrázek, který by měl být vložen, nevygeneruje se žádná chyba, ale místo obrázku bude vložen textový blok s informacemi o obrázku, který tam má být. Velikost tohoto bloku bude velikost obrázku (pokud ji ConTeXt zná) nebo jinak standardní velikost. Příklad toho je v [section 4.4.3](#).

4.2.2 Vložení obrázku pomocí `\placefigure`

Obrázky lze vkládat přímo. Ale je lepší to udělat pomocí `\placefigure`. Tento příkaz způsobí v ConTeXtu:

- že ví, že se vkládá obrázek, který musí být začleněn do seznamu obrázků v dokumentu, který pak lze použít, pokud si přejeme, k vytvoření indexu obrázků.
- přiřazení čísla obrázku, a tím usnadnění interní odkazy na něj.
- přidání názvu k obrázku a vytvoření textového bloku mezi obrázkem a jeho názvem, což znamená, že je nelze oddělit.
- automatické nastavení bílého prostoru (horizontálního a vertikálního) potřebného pro správné zobrazení obrázku.
- že umístí obrázek na vyznačené místo a text kolem něj v případě potřeby obtéká.
- že převede obrázek na plovoucí objekt, pokud je to možné, s přihlédnutím k jeho velikosti a specifikacím umístění.¹

¹ Toto je můj závěr, vzhledem k tomu, že mezi možnostmi umístění jsou takové, jako je `force` nebo `split`, které odporují skutečné představě o plovoucím objektu.

Syntaxe tohoto příkazu je následující:

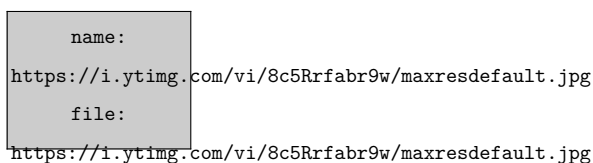
`\placefigure`[Možnosti][Štítek]{Název}{Obrázek}

Různé argumenty mají následující význam:

- *Options* je sada indikací, které obecně odkazují na umístění obrázku. Protože jsou tyto možnosti v tomto a dalších příkazech stejné, vysvětlím je společně později (v [section 4.4.1](#)). Prozatím budu pro příklady používat volbu **here**. Říká ConTeXtu že pokud je to možné, má umístit obrázek přesně do bodu v dokumentu, kde se nachází příkaz, který jej vkládá.
- *Label* je textový řetězec, který interně odkazuje na tento objekt, abychom na něj mohli odkazovat (viz [section ??](#)).
- *Title* je text titulku, který má být přidán k obrázku.
- *Image* je příkaz, který vloží obrázek.

Například

```
\placefigure
[here]
[fig:texknuth]
{\TeX\ logo and photo of {\sc Knuth}}
{\externalfigure[https://i.ytimg.com/vi/8c5Rrfabr9w/maxresdefault.jpg]}
```



Obrázek 4.1. TeX logo
and photo of KNUTH

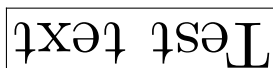
Jak můžeme vidět na příkladu, vložením obrázku (které bylo mimochodem provedeno přímo z obrázku hostovaného na internetu) došlo k určitým změnám ohledně toho, co se stane při přímém použití příkazu `\externalfigure`. Přidá se svislý prostor, obrázek se vycentruje a přidá se název. To jsou *externí* změny zřejmé na první pohled. Z vnitřního hlediska má příkaz také další neméně důležité efekty:

- Nejprve byl obrázek vložen do “seznamu obrázků”, který ConTeXt interně udržuje pro objekty vložené do dokumentu. To zase znamená, že se obrázek objeví v indexu obrázků, který lze vygenerovat pomocí `\placelist[figure]`

(viz [section 4.2](#)), ačkoli existují dva specifické příkazy pro generování indexu obrázků, které jsou `\placelistoffigures` nebo `\completelistoffigures`.

- Za druhé, obrázek byl propojen se štítkem, který byl přidán jako druhý argument do příkazu `\placefigure`, což znamená, že od této chvíle na něj můžeme pomocí tohoto štítku vytvářet interní odkazy (viz [section ??](#)).
- Konečně se obrázek stal plovoucím, což znamená, že pokud by se pro potřeby sazby (stránkování) potřeboval přesunout, ConTeXt by změnil své umístění.

Ve skutečnosti `\placefigure`, navzdory svému názvu, neslouží pouze k vkládání obrázků. Můžeme s ním vložit cokoliv, včetně textu. Avšak s textem nebo jinými položkami vloženými do dokumentu pomocí `\placefigure` bude zacházeno *jako by to byl obrázek*, i když tomu tak není; budou přidány do seznamu obrázků interně spravovaných pomocí ConTeXtu a můžeme použít transformace podobné těm, které používáme pro obrázky, jako je změna měřítka nebo rotace, orámování atd. Tedy následující příklad:



Obrázek 4.2 Using `\placefigure` for text transformations

čehož je dosaženo následovně:

```
\placefigure
[here, force]
[fig:testtext]
{Using \backslash placefigure for text transformations}
{\rotate[rotation=180]{\framed{\tfd Test text}}}
```

4.2.3 Vkládání obrázků integrovaných do textového bloku

S výjimkou velmi malých obrázků, které lze integrovat do řádku bez přílišného narušení mezer mezi odstavci, se obrázky obvykle vkládají do odstavce, který obsahuje pouze je (nebo jinými slovy, obrázek lze považovat za odstavec v jeho vlastní právo). Pokud je obrázek vložen pomocí `\placefigure` a jeho velikost to dovoluje, v závislosti na tom, co jsme uvedli ohledně jeho umístění (viz [section ??](#)), ConTeXt povolí text z předchozího a následujícího odstavce obtékají obraz. Pokud však chceme zajistit, že určitý obrázek nebude oddělen od určitého textu, můžeme použít prostředí `figuretext`, jehož syntaxe je následující:

```
\startfiguretext
[Options]
[Label]
{Title}
```



```
{Image}

... Text

\stopfiguretext
```

Argumenty prostředí jsou úplně stejné jako pro `\placefigure` a mají stejný význam. Ale zde již nejsou možnosti pro umístění plovoucího objektu, ale indikace týkající se integrace obrázku do odstavce; takže například “`left`” zde znamená, že obrázek bude umístěn vlevo, zatímco text poteče vpravo, zatímco “`left, bottom`” bude znamenat, že obrázek musí být umístěn vlevo dole v textu s ním spojeného.

Text napsaný v prostředí je to, co bude obtékat obraz.

4.2.4 Vložená konfigurace a transformace obrázků

A. Vložení možnosti příkazu, které způsobí určitou transformaci obrázku

Poslední argument v příkazu `\externalfigure` nám umožňuje provést určité úpravy vloženého obrázku. Můžeme provést tyto úpravy:

- Obecně pro všechny obrázky, které se mají vložit do dokumentu; nebo pro vložení všech obrázků z určitého bodu. V tomto případě provedeme úpravu příkazem `\setupexternalfigures`.
- Pro konkrétní obrázek, který chceme do dokumentu vložit několikrát. V tomto případě se úprava provádí v posledním argumentu příkazu `\useexternalfigure`, který spojuje externí postavu se symbolickým názvem.
- Přesně ve chvíli, kdy vkládáme konkrétní obrázek. V tomto případě se úprava provádí v samotném příkazu `\externalfigure`.

Změny v obrazu, kterých lze dosáhnout touto cestou, jsou následující:

Změna velikosti obrázku. Můžeme udělat toto:

- *Přiřazením přesné šířky nebo výšky* se něco udělá s možnostmi `width` a `height`; pokud je upravena pouze jedna ze dvou hodnot, druhá se automaticky přizpůsobí tak, aby byl zachován poměr.

Můžeme přiřadit přesnou výšku nebo šířku nebo je uvést jako procento výšky stránky nebo šířky řádku. Například:

```
width=.4\textwidth
```


zajistí, že obrázek bude mít šířku rovnou 40% šířky čáry.

- *Změna měřítka obrázku:* Volba `xscale` změní měřítko obrázku vodorovně; `yscale` to udělá vertikálně a `scale` to udělá horizontálně a vertikálně. Tyto tři možnosti očekávají číslo reprezentující faktor měřítka vynásobené 1000. To znamená: `scale=1000` ponechá obrázek v původní velikosti, zatímco `scale=500` jej zmenší na polovinu, a `scale=2000` zdvojnásobí svou velikost.

Podmíněné měřítko, které se použije pouze v případě, že obrázek překročí určité rozměry, se získá pomocí voleb `maxwidth` a `maxheight`, které převezmou rozměr. Například `maxwidth=.2\textwidth` změní měřítko obrázku pouze v případě, že se ukáže, že je větší než 20% šířky čáry.

Otáčení obrázku. K otočení obrázku používáme volbu `orientace`, která přebírá číslo reprezentující počet stupňů natočení, které budou použity. Otáčení se provádí proti směru hodinových ručiček.

Z wiki vyplývá, že rotace, kterých lze dosáhnout touto volbou, musí být násobky 90 (90, 180 nebo 270), ale abychom dosáhli jiné rotace, museli bychom použít příkaz `\rotate`. Neměl jsem však žádný problém použít otočení o 45 stupňů na obrázek pouze s `orientací=45`, aniž bych musel použít příkaz `\rotate`.

Zarámování obrázku. Můžeme také obklopit obrázek rámečkem pomocí možnosti `frame=on` a nakonfigurovat jeho barvu (`framecolor`), vzdálenost mezi rámečkem a obrázkem (`frameoffset`), tloušťku čáry, která kreslí rám (`rulethickness`) nebo tvaru jeho rohů (`framecorner`), které mohou být zaoblené (`round`) nebo obdélníkové.

Další konfigurovatelné aspekty obrázků. Kromě již viděných aspektů, které znamenají transformaci obrázku, který se má vložit, můžeme pomocí `\setupexternalfigures` nakonfigurovat další aspekty, například kde hledat obrázek (volba `directory`), zda by se měl obrázek hledat pouze v uvedeném adresáři (`location=global`) nebo zda by měl obsahovat také pracovní adresář a jeho nadřazené adresáře (`location=local`) a zda obrázek bude nebo nebude interaktivní (`interakce`) atd.

B. Specifické příkazy pro transformaci obrázku

V ConTeXtu jsou tři příkazy, které vytvářejí určitou transformaci v obrázku a lze je použít v kombinaci s `\externalfigure`. Tady jsou:

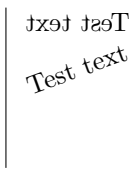
- *Zrcadlový obraz:* dosaženo pomocí příkazu `\mirror`.
- *Clipping:* toho lze dosáhnout pomocí příkazu `\clip`, když šířka (`width`), výška (`height`), horizontální odsazení (jsou uvedeny rozměry `hoffset`) a vertikální odsazení (`voffset`). Například:

```
\clip
[width=2cm, height=1cm, hoffset=3mm, voffset=5mm]
{\externalfigure[logo.pdf]}
```

- *Rotace*. Třetí příkaz schopný aplikovat transformace na obrázek je příkaz `\rotate`. Může být použit ve spojení s `\externalfigure`, ale normálně by to nebylo nutné vzhledem k tomu, že druhý má, jak jsme viděli, volbu `orientace`, která poskytuje stejný výsledek.

Typické použití těchto příkazů je s obrázky, ale ve skutečnosti působí na *boxy*. To je důvod, proč je můžeme použít na jakýkoli text jednoduše tak, že jej uzavřeme do rámečku (což příkaz provede automaticky), což způsobí zvláštní efekty, jako jsou následující:

```
\mirror[Test text]\\
\rotate[rotation=20] {Test text}
```



4.3 Tabulky

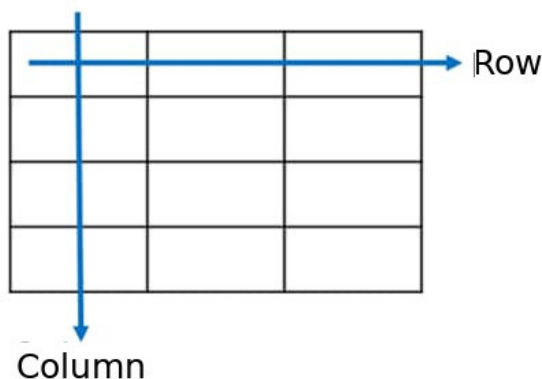
4.3.1 Obecné představy o tabulkách a jejich umístění v dokumentu

Tabulky jsou strukturované objekty, které obsahují text, vzorce nebo dokonce obrázky uspořádané do série *buněk*, které nám umožňují graficky vidět vztah mezi obsahem každé buňky. K tomu jsou informace uspořádány do řádků a sloupců: všechna data (nebo položky) ve stejném řádku mají mezi sebou určitý vztah, stejně jako všechna data (nebo položky) ve stejném sloupci. Buňka je průsečík řádku se sloupcem, jak je znázorněno na figure ??.

Tabulky jsou ideální pro zobrazení textu nebo dat, které spolu souvisejí, protože každá je uzavřena ve své vlastní buňce, i když se její obsah nebo obsah zbývajících buněk mění, relativní pozice jedné vůči ostatním se nezmění.

Ze všech úkonů spojených se sazbou textu je tvorba tabulek jedinou, podle mého názoru, snadněji proveditelnou v grafickém programu (typ textového procesoru) než v ConT_EXtu. Protože je jednodušší *nakreslit* tabulku (což je to, co děláte v programu pro zpracování textu), než *to popsát*, což je to, co děláme, když pracujeme s ConT_EXtem. Každá změna řádku a sloupce vyžaduje přítomnost příkazu, což znamená, že implementace tabulky trvá mnohem déle, místo toho, abychom jednoduše řekli, kolik řádků a sloupců chceme.

**Table of 4 rows
and 3 columns**



Obrázek 4.3 Image of a simple table

ConT_EXt má tři různé mechanismy pro vytváření tabulek; prostředí `tabulate`, které se doporučuje pro jednoduché tabulky a které je nejvíce přímo inspirováno tabulkami T_EXu; tzv. *natural tables*, inspirované HTML tabulkami, vhodné pro tabulky se speciálními požadavky na design, kde například nemají všechny řádky stejný počet sloupců; a takzvané *extrémní tabulky*, jasně založené na XML a doporučené pro střední nebo dlouhé tabulky, které zabírají více než jednu stránku. Ze tří vysvětlím pouze první. Přírozené tabulky jsou poměrně dobře vysvětleny v “ConT_EXt Mark IV exkurze” a pro *extrémní tabulky* je o nich dokument v dokumentaci “ConT_EXt Standalone”.

Něco podobného, se děje s obrázky, které se vyskytují v tabulkách: můžeme jednoduše napsat potřebné příkazy v určitém bodě dokumentu pro vygenerování tabulky a ta bude vložena přesně v tomto bodě, nebo můžeme použít `\příkaz placetable` pro vložení tabulky. To má některé výhody:

- ConT_EXt očíslovuje tabulku a přidá ji do seznamu tabulek umožňujících vnitřní odkazy na tabulku (prostřednictvím jejího číslování) a zahrnout ji do případného rejstříku tabulek.
- Získáme flexibilitu v umístění tabulek v dokumentu, čímž si usnadníme úlohu stránkování.

Formát `\placetable` je podobný tomu, co jsme viděli `\placefigure` (viz [section 4.2.2](#)):

```
\placetable[Options][Label] {Title} {table}
```

Odkazují na sekce ?? a [4.4.2](#) ohledně možností týkajících se umístění tabulky a konfigurace nadpisu. Mezi možnostmi je však jedna, která se zdá být určena výhradně pro tabulky. Toto je volba “`split`”, která, když je nastavena, opravňuje ConTeXt rozšířit tabulku na dvě nebo více stránek, v takovém případě tabulka již nemůže být plovoucím objektem.

Obecně můžeme konfiguraci pro tabulky nastavit příkazem `\setuptables`. Stejně jako u obrázků je také možné generovat index tabulek pomocí `\placelistoftables` nebo `\completelistoftables`. V tomto ohledu viz [section 4.2.2](#).

4.3.2 Jednoduché tabulky s prostředím `tabulate`

Nejjednodušší tabulky jsou ty, kterých se dosáhne pomocí `tabulate` prostředí, jehož formát je:

```
\starttabulate[Rozvržení sloupců tabulky]
... % Obsah tabulky
...
...
\stoptabulate
```

Kde argument v hranatých závorkách popisuje (v kódu) počet sloupců, které tabulka bude mít a (někdy nepřímou) udává jejich šířku. Říkám, že argument popisuje design v kódu, protože na první pohled působí velmi tajemně: skládá se z posloupnosti znaků, z nichž každý má zvláštní význam. Vysvětlím to postupně a po krocích, protože si myslím, že takto je to srozumitelnější.

Toto je typický případ, kdy obrovské množství aspektů, které můžeme konfigurovat, znamená, že k jejich popisu potřebujeme hodně textu. Zdá se, že je to ďábelsky obtížné. Ve skutečnosti pro většinu tabulek, které jsou postaveny v praxi, stačí body 1 a 2. Zbytek jsou další možnosti, o kterých je užitečné vědět, že existují, ale nejsou nezbytné pro sazbu tabulky.

1. **Oddělovač sloupců:** znak “|” se používá k oddělování sloupců tabulky. Takže například “[|1T|rB|]” bude popisovat tabulku se dvěma sloupci, z nichž jeden by měl vlastnosti spojené s indikátory “1” a “T” (který uvidíme hned za ním) a druhý sloupec bude mít vlastnosti spojené s “r” a “B”. Jednoduchá třísloupcová tabulka zarovnaná doleva by například byla popsána jako “[|1|1|1|]”.
2. **Určení základní povahy buněk ve sloupci:** První věc, kterou je třeba určit, když sestavujeme naši tabulku, je, zda chceme, aby byl obsah každé buňky zapsán na jeden řádek, nebo zda naopak, pokud je text libovolného

sloupce příliš dlouhý, chceme, aby jej naše tabulka rozložila na dva nebo více řádků. V prostředí `tabulate` se o této otázce nerozhoduje buňka po buňce, ale je považována za charakteristiku sloupců.

- a. *Jednořádkové buňky*: Pokud má být obsah buněk ve sloupci, bez ohledu na jejich délku, zapsán na jeden řádek, musíme určit zarovnání textu ve sloupci, který lze ponechat (“l”, od *left*), vpravo (“r”, od *right*) nebo na střed (“c”, z *center*).

V zásadě budou tyto sloupce tak široké, aby se vešly do nejširší buňky. Ale můžeme omezit šířku sloupce pomocí specifikátoru “w(Width)”. Například “[lrw(2cm)|c|c|]” bude popisovat tabulku se dvěma sloupci, z nichž první je zarovnán doprava a má přesnou šířku 2 centimetry, a další dva uprostřed a bez omezení šířky.

Je třeba poznamenat, že omezení šířky v jednořádkových sloupcích může způsobit, že text v jednom sloupci překryje text v dalším sloupci. Takže moje rada je, že když potřebujeme sloupce s pevnou šířkou, vždy používejte víceřádkové sloupce buněk.

- b. *Buňky, které mohou v případě potřeby zabírat více než jeden řádek*: specifikátor “p” generuje sloupce, ve kterých text v každé buňce zabere tolik řádků, kolik je potřeba. Pokud jednoduše zadáme “p”, šířka sloupce bude plná dostupná šířka. Ale je také možné uvést “p(Width)”, v takovém případě bude šířka přesně specifikovaná. Tedy následující příklady:

```
\starttabulate[|l|r|p|]
\starttabulate[|l|p(4cm)|]
\starttabulate[|r|p(.6\textwidth)|]
\starttabulate[|p|p|p|]
```

První příklad vytvoří tabulku se třemi sloupci, prvním a druhým z jednoho řádku, zarovnanými doleva a doprava, a třetím, který zabere zbývající šířku a výšku potřebnou k uložení veškerého jejího obsahu. Ve druhém příkladu bude druhý sloupec měřit přesně čtyři centimetry na šířku bez ohledu na jeho obsah; ale pokud se do toho prostoru nevejde, zabere více než jeden řádek. Třetí příklad vypočítá šířku druhého sloupce v poměru k maximální šířce čáry a v posledním příkladu budou tři sloupce, jejichž šířka bude mít stejnou šířku.

Všimněte si, že ve skutečnosti, pokud je buňka čtyřúhelník, specifikátor “p” autorizuje proměnnou výšku buněk ve sloupci v závislosti na délce textu.

3. **Přidání označení k popisu sloupce, o stylu a variantě písma, které se má použít**: jakmile se rozhodne o základní povaze sloupce (šířka a výška buněk, automatická nebo pevná), může ještě přidat do popisu obsahu sloupce znak reprezentující *formát*, ve kterém musí být zapsán. Tyto znaky mohou být

“B” pro tučné písmo, “I” pro kurzívu, “S” pro šikmé písmo, “R” pro písmo v římském stylu nebo “T” pro styl typewriter nápis.

4. Další aspekty, které lze specifikovat v popisu sloupců tabulky

:

- *Sloupce s matematickými vzorci*: specifikátory “m” a “M” umožňují matematický režim ve sloupci, aniž by bylo nutné jej uvádět v každé z jeho buněk. Buňky v tomto sloupci nebudou schopny pojmout normální text.

Přestože T_EX, předchůdce ConT_EXtu, vznikl pro sazbu jakéhokoli druhu matematiky, až dosud jsem o psaní matematiky téměř nic neřekl. V matematickém režimu (který nebudu vysvětlovat) ConT_EXt mění naše normální pravidla a dokonce používá jiné fonty. Matematický režim má dvě varianty: jeden bychom mohli nazvat *lineární* v tom, že vzorec je umístěn v řádku obsahujícím normální text (indikátor “m”) a *kompletní matematický režim*, který zobrazuje vzorce v prostředí, kde není normální text. Hlavním rozdílem mezi těmito dvěma režimy v tabulce je v podstatě velikost, ve které bude vzorec zapsán, a horizontální a vertikální prostor, který jej obklopuje.

- *Přidat další horizontální bílé místo kolem obsahu buněk ve sloupci*: pomocí indikátorů “in”, “jn” a “kn” můžeme přidat další bílé místo vlevo obsahu sloupce (“in”), vpravo (“jn”) nebo na obě strany (“kn”). Ve všech třech případech představuje “n” číslo, kterým se vynásobí prázdné místo, které by normálně zůstalo bez jednoho z těchto specifikátorů (standardně je průměr *em*). Takže například “|j2r|” bude indikovat, že stojíme před sloupcem, který bude zarovnán doprava a ve kterém chceme prázdné místo o šířce 1 *em*.
- *Přidání textu před nebo za obsah každé buňky ve sloupci*. Specifikátory `b{Text}` a `a{Text}` způsobí, že text mezi složenými závorkami bude zapsán před (“b”, z *before*) nebo za (“a”, od *after*) obsah buňky.
- *Použití příkazu format na celý sloupec*. Indikátory “B”, “I”, “S”, “R” “T”, které jsme zmínili dříve, nepokrývají všechny možnosti formátu: např. není tam žádný indikátor pro malá písmena nebo pro *sans serif*, nebo který ovlivňuje velikost písma. Pomocí indikátoru “h\command” můžeme zadat příkaz formátu, který se automaticky použije na všechny buňky ve sloupci. Například “|lf\sc|” vysází obsah sloupce velkými písmeny.
- *Použití libovolného příkazu na všechny buňky ve sloupci*. Nakonec indikátor “h\příkaz”

použije zadaný příkaz na všechny buňky ve sloupci.

V [table 4.1](#) jsou uvedeny některé příklady řetězců specifikace formátu tabulky.

Specifikátor formátu	Význam
l	Vygeneruje sloupec, jehož šířka je automaticky zarovnána doleva.
rB	Vygeneruje sloupec, jehož šířka je automaticky zarovnána doprava a je uvedena tučně.
cIm	Vygeneruje sloupec povolený pro matematický obsah. Na střed a kurzívou.
j4cb{---}	Tento sloupec bude mít obsah vystředěný, bude začínat em pomlčkou (—) a přidá 2 <i>ems</i> mezery vpravo.
l p(.7\šířka textu)	generuje dva sloupce: první je zarovnán doleva a má automatickou šířku. Druhá zabírá 70% celkové šířky čáry.

Tabulka 4.1 Některé příklady, jak určit formát sloupců v *tabulate*

Jakmile je tabulka navržena, je třeba zadat její obsah. Abych vysvětlil, jak to udělat, začnu popisem, jak by měla být vyplněna tabulka, kde máme linky oddělující řádky a sloupce:

- Vždy začínáme nakreslením vodorovné čáry. V tabulce se to provede příkazem `\HL` (z *Horizontal Line*).
- Poté napíšeme první řádek: na začátku každé buňky musíme označit, že začíná nová buňka a že je třeba nakreslit svislou čáru. To se provádí příkazem `\VL` (z *Vertical Line*). Začneme tedy tímto příkazem a zapíšeme obsah každé buňky. Pokaždé, když měníme buňky, opakujeme příkaz `\VL`.
- Na konci řádku výslovně označíme, že bude spuštěn nový řádek příkazem `\NR` (z *Next Row*). Poté zopakujeme `\HL` a nakreslíme novou vodorovnou čáru.
- A tak jeden po druhém zapíšeme všechny řádky tabulky. Když skončíme, přidáme jako další příkaz `\NR` a další `\HL` pro uzavření mřížky spodní vodorovnou čarou.

Pokud nechceme kreslit mřížku tabulky, odstraníme příkazy `\HL` a nahradíme příkazy `\VL` příkazy `\NC` (z *Nový sloupec*).

Není to nijak zvlášť obtížné, když to pochopíme, i když se podíváme na zdrojový kód tabulky, je těžké získat představu, jak bude vypadat. V [table 4.2](#) vidíme příkazy, které mohou (a musí) být použity v tabulce. Jsou některé, které jsem nevysvětlil, ale myslím, že popis, který jsem uvedl, stačí.

Příkaz	Význam
<code>\HL</code>	Vloží vodorovnou čáru
<code>\NC</code>	Začne nový sloupec
<code>\NR</code>	Začne nový řádek
<code>\VL</code>	Vloží svislou čáru ohraničující sloupec (používá se místo <code>\NC</code>)
<code>\NN</code>	Začne sloupec v matematickém režimu (používá se místo <code>\NC</code>)
<code>\TB</code>	Přidá další vertikální mezeru mezi dva řádky
<code>\NB</code>	Označuje, že další řádek začíná nedělitelný blok, ve kterém nemůže být konec stránky

Tabulka 4.2 Příkazy k použití v tabulce

A nyní jako příklad přepíšu kód, kterým byl napsán [table 4.2](#).

```
\placetable
  [tady]
  [tbl:tablecommands]
  {Příkazy k~použití v~tabulce}
{\startttable[|l|p(.6\textwidth)]}
\HL
\NC {\bf Příkaz}
\NC {\bf Význam}
\NR
\HL
\NC \tex{HL}
\NC Vloží vodorovnou čáru
\NR
\NC \tex{NC}
\NC Začne nový sloupec
\NR
\NC \tex{NR}
\NC Začne nový řádek
\NR
\NC \tex{VL}
\NC Vloží svislou čáru ohraničující sloupec (používá se místo \tex{NC})
\NR
\NC \tex{NN}
\NC Začne sloupec v~matematickém režimu (používá se místo \tex{NC})
\NR
\NC \tex{TB}
\NC Přidá další vertikální mezeru mezi dva řádky
\NR
\NC \tex{NB}
\NC Označuje, že další řádek začíná nedělitelný blok, ve kterém nemůže být konec
stránky
\NR
\HL
```


`\stoptabulate}`

Čtenář si všimne, že jsem obecně použil jeden (nebo dva) řádky textu pro každou buňku. Ve skutečném zdrojovém souboru bych pro každou buňku použil pouze řádek textu; v příkladu jsem rozdělil řádky, které jsou příliš dlouhé. Použití jednoho řádku na buňku mi usnadňuje psaní tabulky, protože to, co dělám, je psát obsah každé buňky bez příkazů k oddělení řádků nebo sloupců. Když je vše napsáno, vyberu text z tabulky a požádám svůj textový editor, aby na začátek každého řádku vložil “`\NC`”. Poté každé dva řádky (protože tabulka má dva sloupce) vložím řádek, který přidá příkaz `\NR`, protože každé dva sloupce začíná nový řádek. Nakonec ručně vložím příkazy `\HL` do bodů, kde chci, aby se objevila vodorovná čára. Skoro déle mi trvá to popsat, než to udělat!

Ale také se podívejte, jak v rámci tabulky můžeme používat běžné příkazy ConTeXtu. Zejména v této tabulce neustále používáme `\tex`, což je vysvětleno v [section 1.2.3](#).

4.4 Aspekty společné pro obrázky, tabulky a další plovoucí objekty

Již víme, že obrázky a tabulky nemusí být plovoucí objekty, ale jsou dobrými kandidáty, aby tomu tak bylo, i když je třeba je vložit do dokumentu pomocí příkazů `\placefigure` nebo `\placetable`. Kromě těchto dvou příkazů se stejnou strukturou máme v ConTeXtu příkaz `\placechemical` (pro vložení chemických vzorců), příkaz `\placegraphic` (pro vložení grafiky) a příkaz `\placeintermezzo` pro vložení struktury, kterou ConTeXt nazývá *Intermezzo* a která tuším odkazuje na fragmenty orámovaného textu. Všechny tyto příkazy jsou zase konkrétními aplikacemi obecnějšího příkazu, kterým je `\placefloat`, jehož syntaxe je následující:

`\placefloat[Název] [Možnosti] [Štítek] {Název} {Obsah}`

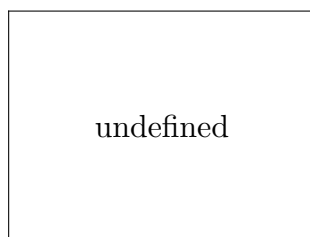
Všimněte si, že `\placefloat` je identický s `\placefigure` a `\placetable` kromě prvního argumentu, který v `\placefloat` přebírá jméno plovoucího objektu. Je to proto, že *každý typ plovoucího objektu lze do dokumentu vložit dvěma různými příkazy*: `\placefloat[TypeName]` nebo `\placeTypeName`. Jinými slovy: `\placefloat[figure]` a `\placefigure` jsou přesně stejné příkazy, stejně jako `\placefloat[table]` je stejný příkaz jako `\placetable`.

Proto budu od nynějška mluvit o `\placefloat`, ale mějte na paměti, že vše, co řeknu, bude platit také pro `\placefigure` nebo `\placetable`, což jsou specifické aplikace uvedeného příkazu.

Argumenty `\placefloat` jsou:

- *Jméno* odkazuje na dotyčný plovoucí objekt. Může to být nějaký předem určený plovoucí objekt (*obrázek*, *tabulka*, *chemikálie*, *intermezzo*) nebo plovoucí objekt vytvořený námi pomocí `\definefloat` (viz [section 4.5](#)).
- *Možnosti* Série symbolických slov, která ConTeXtu říkají, jak má vložit objekt. Velká většina z nich odkazuje na to *kam* vložit. To uvidíme v další části.
- *Štítek* Označení pro budoucí interní odkazy na tento objekt.
- *Název* Text nadpisu, který se má přidat k objektu. Ohledně jeho konfigurace viz [section 4.4.2](#).
- *Obsah* To samozřejmě závisí na typu objektu. Pro obrázky je to obvykle příkaz `\externalimage`; pro tabulky, příkazy, které vytvoří tabulku; pro *intermezzi*, fragment textu v rámečku; atd.

První tři argumenty, které jsou uvedeny v hranatých závorkách, jsou volitelné. Poslední dva (které jsou uvedeny mezi složenými závorkami) jsou povinné, i když mohou být prázdné. Takže například: `\placefloat{}{}` vloží:



Obrázek 4.4

v dokumentu.



Poznámka: Vidíme, že ConTeXt uvažoval, že objekt, který má být vložen, byl obrázek, protože byl očíslován jako obrázek a zahrnut v seznamu obrázků. To mě nutí předpokládat, že obrázky jsou výchozí plovoucí objekty.

4.4.1 Možnosti vložení plovoucího objektu

Argument *Options* v `\placefigure`, `\placetable` a `\placefloat` řídí různé aspekty týkající se vkládání těchto typů objektů. Hlavně místo na stránce, kam bude objekt vložen. Zde je podporováno několik hodnot, každá jiné povahy:

- Některá místa vložení jsou stanovena ve vztahu k prvkům stránky (*top*, *bottom*, *inleft*, *inright*, *inmargin*, *margin*, *leftmargin*, *rightmargin*, *leftedge*, *rightedge*, *innermargin*, *inneredge*, vnější hrana, vnitřní, vnější). Musí se samozřejmě jednat o objekt, který se vejde do

oblasti, kde má být umístěn, a pro tento prvek musí být v rozvržení stránky vyhrazeno místo. Ohledně toho viz sekce 1.2 a 1.3.

- Další možná místa vložení se více vztahují k textu obklopujícímu objekt a jsou indikací, kam by měl být objekt umístěn, aby kolem něj text obtékal. V zásadě hodnoty `left` a `right`.
- Volba `here` je interpretována jako doporučení ponechat objekt na místě ve zdrojovém souboru, kde se nachází. Toto *doporučení* nebude respektováno, pokud požadavky na stránkování neumožňují. Tato indikace je posílena, pokud přidáme volbu `force`, což znamená přesně toto: vynutit vložení objektu do tohoto bodu. Všimněte si, že po vynucení vložení v určitém bodě již objekt nebude plovoucí.
- Další možné volby se týkají stránky, na kterou má být objekt vložen: “`stránka`” jej vloží na novou stránku; “`opposite`” jej vloží na stránku naproti aktuální; “`leftpage`” na sudé stránce; “`rightpage`” na liché stránce.

Existují některé možnosti, které nesouvisejí s umístěním objektu. Mezi nimi:

- `none`: Tato volba potlačí název.
- `split`: Tato volba umožňuje objektu rozšířit se na více než jednu stránku. Musí to být samozřejmě předmět, který je od přírody dělitelný, například tabulka. Když je tato možnost použita a objekt je rozdělen, nelze již říci, že je plovoucí.

4.4.2 Konfigurace názvů plovoucích objektů

Pokud nepoužijeme volbu “`none`” v `\placefloat`, ve výchozím nastavení jsou plovoucí objekty spojeny s názvem, který se skládá ze tří prvků:

- Jméno daného typu objektu. Toto jméno je přesně stejné jako jméno typu objektu; takže pokud například definujeme nový plovoucí objekt s názvem “`sekvence`” a vložíme “`sekvenci`” jako plovoucí objekt, bude nadpis “`sekvence 1`”. Jednoduše zadejte název objektu velkým písmenem.

Navzdory tomu, co bylo právě řečeno, pokud hlavním jazykem dokumentu není angličtina, bude přeložen anglický název pro předdefinované objekty, jako jsou například objekty “`figure`” nebo “`table`”; Takže například objekt “`figure`” v dokumentech ve španělštině se nazývá “`Figura`”, zatímco objekt “`table`” se nazývá “`Tabla`”. Tyto španělské názvy pro předdefinované objekty lze změnit pomocí `\setuplabeltext`, jak je vysvětleno v [section 1.5.3](#).

- Jeho číslo. Ve výchozím nastavení jsou objekty číslovány podle kapitol, takže první tabulka v kapitole 3 bude tabulka “`3.1`”.
- Jeho obsah. Zavedeno jako argument `\placefloat`.

Pomocí `\setupcaptions` nebo `\setupcaption[Object]` můžeme změnit systém číslování a vzhled samotného titulku. První příkaz ovlivní všechny názvy všech objektů a druhý ovlivní pouze název určitého typu objektu:

- Pokud jde o systém číslování, je řízen volbami `number`, `way`, `prefixsegments` a `numberconversion`:
 - číslo může přijmout `ano`, `ne` nebo `none` hodnoty a řídí, zda bude číslo nebo ne.
 - `way` udává, zda bude číslování v celém dokumentu sekvenční (`way=bytext`), nebo zda bude znovu začínat na začátku každé kapitoly (`way=bychapter`) nebo oddílu (`way=bysection`). V případě restartu je vhodné koordinovat hodnotu této volby s volbou `prefixsegments`.
 - `prefixsegments` udává, zda číslo bude mít *prefix* a co to bude. `prefixsegments=chapter` tedy způsobí, že počet objektů vždy začíná číslem kapitoly, zatímco `prefixsegments=section` bude před číslem objektu s číslem sekce.
 - `numberconversion` řídí druh číslování. Hodnoty pro tuto možnost mohou být: Arabská čísla (“čísla”), malá písmena (“a”, “znaky”), velká písmena (“A”, “Characters”), malá písmena “KA”), velká římská čísla (“I”, “R”, “Romannumerals”), malá písmena (“i”, “r”, “romannumerals” nebo malá písmena (“KR”)).
- Vzhled samotného titulku je řízen mnoha možnostmi. Uvedu je, ale pro podrobné vysvětlení významu každého z nich odkazuji na [section 3.4.4](#), kde je vysvětleno ovládání vzhledu příkazů dělení, protože možnosti jsou z velké části stejné. Jedná se o tyto možnosti:
 - Chcete-li ovládat formát všech prvků titulku, `styl`, `barva`, `příkaz`.
 - Chcete-li ovládat pouze formát názvu pro daný druh objektu: `headstyle`, `headcolor`, `headcommand`, `headseparator`.
 - Chcete-li ovládat pouze formát číslování: `číselný příkaz`.
 - Chcete-li ovládat pouze formát samotného titulku: `textový příkaz`.
- Můžeme také ovládat další aspekty, jako je vzdálenost mezi různými prvky, které tvoří nadpis, šířka nadpisu, jeho umístění vzhledem k objektu atd. Zde odkazuji na informace v [ConTeXt wiki](#) týkající se možností, které lze konfigurovat pomocí tohoto příkazu.

4.4.3 Kombinované vložení dvou nebo více objektů

Pro vložení dvou nebo více různých objektů do dokumentu tak, aby je ConTeXt držel pohromadě a zacházel s nimi jako s jedním objektem, máme prostředí `\startcombination`, jehož syntaxe je:

```
\startcombination[Ordering] ... \stopcombination
```

kde *Ordering* udává, jak by měly být objekty seřazeny: pokud je všechny potřeba seřadit vodorovně, *Ordering* udává pouze počet objektů, které mají být kombinovány. Pokud ale chceme objekty spojit do dvou nebo více řádků, budeme muset uvést číslo objektu na řádek, za ním počet řádků a obě čísla oddělit znakem `*`. Například:

```
\startcombination[3*2]
  {\externalfigure[test1]}
  {\externalfigure[test2]}
  {\externalfigure[test3]}
  {\externalfigure[test4]}
  {\externalfigure[test5]}
  {\externalfigure[test6]}
\stopcombination
```

což vytvoří následující zarovnání obrázků.



V předchozím příkladu obrázky, které jsem zkombinoval, ve skutečnosti neexistují, a proto místo obrázků ConTeXt vygeneroval textová pole s informacemi o nich.

Podívejte se na druhou stranu, jak je každý prvek, který má být kombinován v `\startcombination`, uzavřen ve složených závorkách.

Ve skutečnosti nám `\startcombination` nejen umožňuje spojovat a zarovnávat obrázky, ale jakýkoli druh *boxu*, jako jsou texty v prostředí `\startframedtext`,

tabulky atd. Ke konfiguraci kombinace můžeme použít příkaz `\setupcombination` a můžeme také vytvořit předkonfigurované kombinace pomocí `\definecombination`.

4.4.4 Obecná konfigurace plovoucích objektů

Již jsme viděli, že pomocí `\placefloat` můžeme ovládat umístění vkládaného plovoucího objektu a některé další detaily. Je také možné nakonfigurovat:

- Globální charakteristiky určitého typu plovoucího objektu. To se provádí pomocí `\setupfloat[Název typu plovoucího objektu]`.
- Globální charakteristiky všech plovoucích objektů v našem dokumentu. To se provádí pomocí `\setupfloats`.

Mějte na paměti, že stejně jako `\placefloat[figure]` je ekvivalentní `\placefigure`, `\setupfloat[figure]` je ekvivalentní `\setupfigures` a `\setupfloat[table]` je ekvivalentní `\setuptables`.

Co se týče konfigurovatelných možností, odkazuji na oficiální seznam příkazů ConTeXt (section ??).

4.5 Definování dalších plovoucích objektů

Příkaz `\definefloat` nám umožňuje definovat vlastní plovoucí objekty. Jeho syntaxe je:

```
\definefloat[Jednotné jméno] [Množné číslo] [Konfigurace]
```

Kde argument *Konfigurace* je volitelný argument, který nám umožňuje uvést konfiguraci tohoto nového objektu již v době jeho vytvoření. Můžeme to udělat i později pomocí `\setupfloat[Jména v~jednotném čísle]`.

Vzhledem k tomu, že tímto oddílem končíme náš úvod, využijí toho, abych se dostal trochu hlouběji do zdánlivé *džungle* příkazů ConTeXtu které, jakmile je pochopíme, není tolik *džunglí*, ale ve skutečnosti je docela racionální.

Začněme tím, že se sami sebe zeptáme, k čemu vlastně je plovoucí objekt ConTeXtu, přičemž odpověď zní, že je to objekt s následujícími vlastnostmi:

- má určitou volnost s ohledem na své umístění na stránce;
- což znamená, že je k němu přidružen *seznam*, který mu umožňuje číslovat tyto druhy objektů a případně generovat jejich index;

- má název;
- když objekt se skutečně chová jako plovoucí, musí se s ním zacházet jako s nedělitelnou jednotkou, což znamená (v terminologii \TeX u) uzavřen v tzv. *boxu*.

Jinými slovy, plovoucí objekt se ve skutečnosti skládá ze tří prvků: samotného objektu, seznamu s ním spojeného a názvu. K ovládání samotného objektu potřebujeme pouze jeden příkaz pro nastavení jeho umístění a další pro vložení objektu do dokumentu; pro nastavení aspektů seznamu postačují obecné příkazy pro řízení seznamu a pro nastavení aspektů názvu obecné příkazy pro řízení názvu.

A zde přichází na řadu genialita $\text{Con}\text{\TeX}$ tu jednoduchý příkaz pro ovládání plovoucích objektů (\setupfloats) a jednoduchý příkaz pro vkládání plovoucích objektů: \placefloat , mohl být navržen: ale co $\text{Con}\text{\TeX}$ t dělá:

1. Navrhnete příkaz k propojení názvu s konkrétní konfigurací plovoucího objektu. Toto je příkaz \definefloat , který ve skutečnosti nepropojuje jedno jméno, ale dvě jména, jedno v jednotném čísle a jedno v množném čísle.
2. Vytvořte společně s globálním konfiguračním příkazem plovoucích objektů příkaz, který nám umožňuje konfigurovat pouze určitý typ objektu: $\text{\setupfloat[Object]}$.
3. Přidejte do příkazu umístění plovoucího objektu (\placefloat), argument, který nám umožňuje rozlišovat mezi jedním nebo druhým typem: ($\text{\placefloat[Object]}$).
4. Vytvoří příkazy, včetně názvu objektu, pro všechny akce plovoucího objektu. Některé z těchto příkazů (které jsou ve skutečnosti klony jiných obecnějších příkazů) budou používat název objektu v jednotném čísle a jiné jej budou používat v množném čísle.

Když tedy vytvoříme nový plovoucí objekt a řekneme $\text{Con}\text{\TeX}$ tu jaké je jeho jméno v jednotném a množném čísle, $\text{Con}\text{\TeX}$ t:

- Vyhradí místo v paměti pro uložení specifické konfigurace daného typu objektu.
- Vytvoří nový seznam s jednotným jménem tohoto typu objektu, protože plovoucí objekty jsou spojeny se seznamem.
- Vytvoří nový druh “title” propojený s tímto novým typem objektu, aby byla zachována přizpůsobená konfigurace těchto titulků.
- A nakonec vytvoří skupinu nových příkazů specifických pro tento nový typ objektu, jehož jméno je ve skutečnosti synonymem pro obecnější příkaz.

V [table 4.3](#) můžeme vidět příkazy, které se automaticky vytvoří, když definujeme nový plovoucí objekt, a také obecnější příkazy, jejichž synonyma jsou:

příkaz	Synonymum	příklad
<code>\úplný seznam<PluralName></code>	<code>\completelist[PluralName]</code>	<code>\úplný seznam čísel</code>
<code>\<JednotnéJméno></code>	<code>\placefloat[Jednotné jméno]</code>	<code>\obrázku</code>
<code>\seznam<Množné jméno></code>	<code>\[PluralName]</code>	<code>\seznam obrázků</code>
<code>\nastavení???<JednotnéJméno></code>	<code>\setupfloat[SingulárníNázev]</code>	<code>\nastavení</code>

Tabulka 4.3 Příkazy, které se automaticky vytvářejí při vytvoření nového plovoucího objektu

Ve skutečnosti jsou vytvořeny některé další příkazy, které jsou synonymem k předchozím, a protože jsem je nezahrnul do vysvětlení kapitoly, vynechal jsem je z [table 4.3](#): `\start<Jméno-Singulární>`, `\start<JménoSingulární>text` a `\startplace<JménoSingulární>`.

Příkaz používaný pro obrázky jsem použil jako příklad příkazů vytvořených při definování nového plovoucího objektu; a udělal jsem to, protože obrázky, jako tabulky a zbytek floatů předdefinovaných pomocí ConT_EXtu, jsou skutečnými případy `\definefloat`:

```
\definefloat[chemical][chemicals]
\definefloat[figure][figures]
\definefloat[table][tables]
\definefloat[intermezzo][intermezzi]
\definefloat[graphic][graphics]
```

Konečně vidíme, že ve skutečnosti ConT_EXt žádným způsobem nekontroluje žádný druh materiálu obsaženého v každém konkrétním plovoucím objektu; předpokládá, že jde o práci autora. To je důvod, proč můžeme také vkládat text pomocí příkazů `\placefigure` nebo `\placetable`. Avšak text, který je vložen pomocí `\placefigure`, je zahrnut v seznamu obrázků, a pokud je vložen pomocí `\placetable` v seznamu tabulek.

Přílohy

————— Czech version —————

TO BE DONE! —————

————— Czech version —————

TO BE DONE! —————

————— Czech version —————

TO BE DONE! —————