Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE

Petr Löffelmann

MP3 editor

Katedra aplikované matematiky

Vedoucí bakalářské práce: Mgr. Martin Mareš, Ph.D.

Studijní program: Informatika

Studijní obor: Správa počítačových systémů

Praha 2011

Rád bych poděkoval Mgr. Martinu Marešovi, Ph.D. za ochotu a cenné rady při zpracovávání této práce. Zvláštní díky patří také Martinu Drábovi a Bc. Aleně Řezankové za obětavou pomoc při řešení mnoha problémů.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne ....

Název práce: MP3 editor

Autor: Petr Löffelmann

Katedra / Ústav: Katedra aplikované matematiky

Abstrakt: Práce pojednává o digitalizovaném zvuku uloženém ve formátu MPEG 1 Layer III (MP3) a za svůj hlavní cíl si klade implementovat editační knihovnu, která bude nad MP3 soubory provádět vybrané úpravy. Text se v počátku zabývá obecným popisem metod používaných při kompresi zvuku do tohoto formátu, dále se věnuje struktuře MP3 souboru a následně rozebírá proces jeho dekomprese. V následujících částech se práce soustředí na popis implementace samotné editační knihovny a kroky, které musí být učiněny pro vykonání editačních operací. Dodatkem k práci je programátorská dokumentace knihovny a CD s její funkční verzí.

Klíčová slova: MP3, editor, komprese

Title: MP3 editor

Author: Petr Löffelmann

Department: Department of Applied Mathematics

Abstract: The thesis focuses on digitized sound in MPEG 1 Layer III (MP3) format and its main goal it to implement an audio library which provides its user with selected editing operations. The text provides an overview of methods used during sound compression, then it describes structure of an MP3 file and finally analyses the decompression process. Further on the library implementation and the steps that need to be taken in order to execute the editing actions while putting the earlier obtained knowledge into practise are explained. In the end the programmer documentation and CD containing working copy of the library is attached.

Keywords: MP3, editor, compression

Obsah

Úvod

Různých audio editorů existuje v počítačovém světě již mnoho a některé se dají považovat za opravdu kvalitní. Většina z nich je ale stavěna na práci s nekomprimovanými audio daty a proto musí před editací všechny nahrávky uložené ztrátovými kodeky dekódovat a následně je opět zakódovat, což vede k dalším ztrátám kvality. Program, či spíše knihovna, kterou si v tomto textu popíšeme, nemá za cíl konkurovat existujícím editorům, protože dělá něco jiného, než naprostá většina z nich. Operuje totiž přímo nad komprimovanými daty, při čemž nedochází k dalším ztrátám kvality nahrávky. Základními operacemi, které zatím knihovna podporuje, je odstranění, zesílení a zeslabení některé části souboru a sloučení více souborů do jednoho.

Ač je MP3 ztrátový formát, při kódovacím procesu se, kvůli dosažení lepších výsledků, využívá jak ztrátové, tak bezztrátové komprese. Právě o bezztrátové kompresi si povíme nejdříve a popíšeme algoritmus Huffmanova kódování, díky kterému jsou MP3 soubory menší až o 20 %. Hned záhy si osvětlíme metody ztrátové komprese, kde hrají prim psychoakustický model spolu s perceptuálním kodekem, které určují, jaké části nahrávky není ucho schopno vnímat, a proto je lze odstranit.

Následně se podíváme na standard, jenž definuje nejen kompresi MP3 souborů, ale také způsoby komprese videa a přenosu komprimovaných dat. Trochu blíže si povíme o některých zajímavých postupech a možnostech, které standard nabízí.

Dále nás čeká popis struktury MP3 souboru, u kterého si vysvětlíme, k čemu slouží jednotlivá datová pole hlavičky a *Side Information* a jak se mění jejich množství a délka v závislosti na ostatních parametrech. Nezapomeneme si říct ani důležité informace z dekódovacího procesu, které zasadí jednotlivé postupy a možnosti nastavení do většího celku.

Nakonec se zaměříme na samotnou knihovnu, u které rozebereme rozhodnutí, která byla v průběhu vývoje učiněna, probereme implementaci jednotlivých funkčních celků a popíšeme si také, jak celou knihovnu použít.

1 Než začneme

Dříve než se začneme zabývat detaily o MP3 či samotné implementaci, podívejme se na několik obecnějších problémů, jejichž porozuměním si ulehčíme čtení dalšího textu.

1.1 Několik slov o kompresi dat

Komprese dat je postup, který se snaží zmenšit objem dat, jenž je nutný k uložení informace. Používá se např. při archivaci nebo třeba při přenosu přes síť s omezenou datovou propustností. Pomocí zvolených kompresních algoritmů jsou z dat odstraněny redundantní informace. Kompresi dat můžeme rozdělit do dvou základních kategorií.

Bezztrátová komprese bývá typicky méně účinná, ale její velkou výhodou je, že komprimovaný soubor lze opačným postupem rekonstruovat do původní podoby. Její účinnost bývá až 50 %, přičemž existuje matematický důkaz toho, že data lze bezztrátově zkomprimovat pouze na určitou mez.

Druhým typem je komprese ztrátová. Při ní dochází ke ztrátě některých informací a rekonstruovaná data nejsou nikdy shodná s originálem. Používá se hlavně při kompresi zvuku a obrazu, kde jsou ztráty akceptovatelné, zanedbatelné či dokonce nepostřehnutelné. MP3 je klasickým zástupcem ztrátové komprese, na které se největší mírou podílí psychoakustický model a perceptuální kódování, protože určují, která data jsou lidským uchem slyšitelná a která ne, přičemž neslyšitelná data jsou záhy z nahrávky odstraněna.

Je tedy zřejmé, že v mnoha oblastech (jako na například komprese programů či textových dokumentů) je bezztrátová komprese nutností, protože není přípustné, abychom po zkomprimování a následné dekompresi získali jen část původních dat. I v MP3 souborech je však bezztrátové komprese využito. Při vytváření MP3 souboru jsou data nejdříve zkomprimována za pomoci psychoakustických modelů, které dokážou určit neslyšitelná data, a následně je jejich objem ještě zmenšen o 15 až 20 % pomocí bezztrátové komprese.

Bezztrátová kompresní metoda použitá u MP3 souborů se nazývá Huffmanovo kódování (1), které se obzvlášť hodí, když chceme kódovat každý znak samostatnou posloupností bitů, a patří do rodiny tzv. entropických kodeků. Entropické kodeky přiřazují každému symbolu na vstupu bezprefixové kódové slovo, jehož délka se snižuje s počtem výskytů (co je to bezprefixové slovo, si povíme hned v příští kapitole). Tímto poměrně jednoduchým trikem je docíleno toho, že symboly, které se ve vstupním textu vyskytují často, zabírají v zakódovaném textu méně místa a proto je celý text kratší.

1.2 Huffmanovo kódování

Představme si, že máme libovolný řetězec složený ze znaků „a“ a „b“, např. „abbbaabaaabaaa“. Pokud takový řetězec uložíme pomocí běžně používaného kódování ISO-8859-2, které potřebuje 8 bitů na znak, dostáváme jednoduchým výpočtem, že k uložení daného řetězce budeme potřebovat 112 bitů. Pokud chceme takový řetězec uložit pomocí minimálního množství bitů, stačí nám jeden bit pro každý znak. Písmeno „a“ budeme značit nulou a písmeno „b“ jedničkou, takže budeme pro uložení potřebovat pouhých 14 bitů.

To bylo až příliš jednoduché. Co když budeme chtít zakódovat řetězec složený z více písmen, např. „abcdaabaaabaaa“? Teď už sice nemůžeme použít jeden bit na znak, ale vidíme, že nám budou stačit dva.

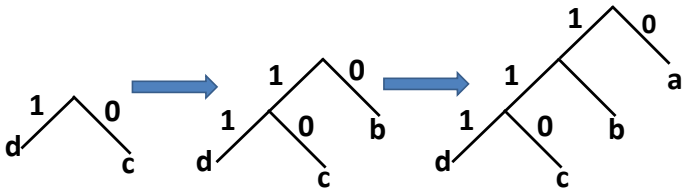
Kód	Znak
00	a
01	b
10	c
11	d

Kód	Znak
0	a
10	b
110	c
111	d

Tabulka 1: Naivní kódování vs. Huffmanovo kódování

Pokud použijeme naivně dva bity, budeme potřebovat na uložení $14 * 2 = 28$ bitů, což ale není optimální. Při použití Huffmanova kódování potřebujeme pouze 21 bitů (devět pro „a“, šest pro „b“, tři pro „c“ a tři pro „d“ (viz Tabulka 1)), což je oproti naivnímu kódování úspora 25 %. Zakódujeme-li text pomocí Huffmanova kódování, dostáváme „010110111001000010000“. Chceme-li tento řetězec dekódovat, načteme jeho první znak, vidíme, že je to nula a víme, že nulou je kódováno pouze písmeno „a“. Načteme tedy další znak, což je jednička. Z této informace můžeme zatím usoudit pouze to, že se nejedná o znak „a“ a musíme proto přečíst další znak. Načetli jsme nulu a víme, že kód 10 znamená „b“. Tímto postupem snadno rozkódujeme celý řetězec. Teď už rozumíme slovu bezprefixový: žádný kód není prefixem (předponou) jiného.

Kde se ale vzala ona magická tabulka kódů? Ještě před kódováním je potřeba projít celý text a zjistit relativní četnosti jednotlivých znaků. Podle získaných četností je pak odspoda postaven binární strom tak, že vždy vybereme dva znaky s nejmenší relativní četností a přidáme je do stromu, načež jejich obě relativní četnosti sečteme a chováme se k nim nadále jako k novému znaku s vyšší četností. Tento postup opakujeme, dokud není celý strom postaven, jak ukazuje Obrázek 1. Kompletní algoritmus stavby i dekódování stromu podrobně popisuje Ryan Bender v (2).



Obrázek 1: Stavba Huffmanova stromu

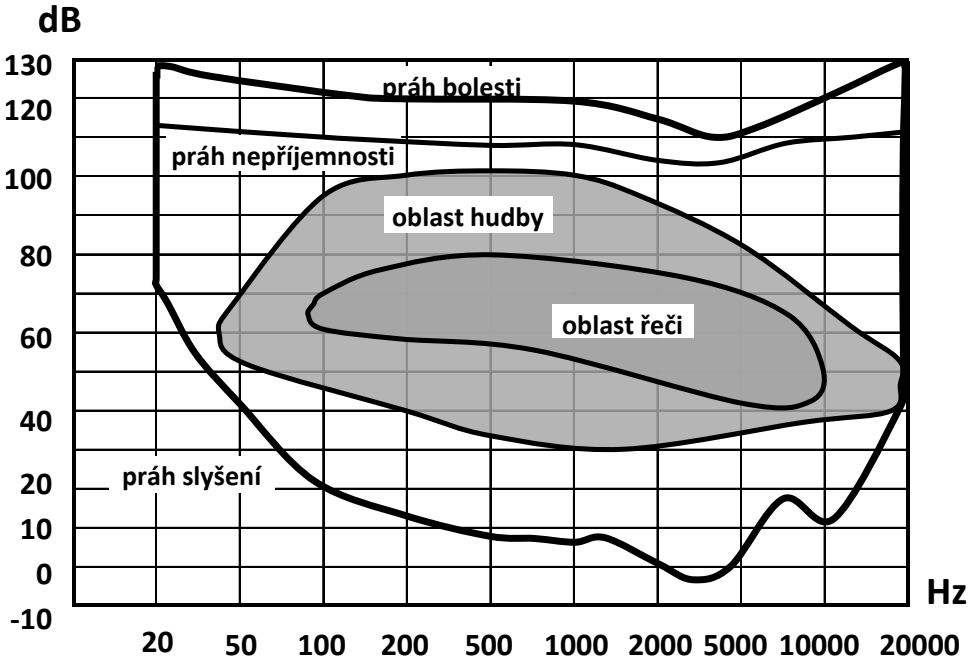
Každý kód popisuje cestu od kořene k danému znaku uvnitř stromu. Tudíž při dekódování řetězce stačí pro získání zakódovaného znaku pouze procházet stromem.

U MP3 jsou tabulky pravděpodobností relativních četností předpočítány a známy jak kodéru, tak dekodéru. Je jich k dispozici celkem 32 a jsou optimalizovány pro různé části audio signálu.

1.3 Psychoakustika a perceptuální kódování

Psychoakustika je vědní disciplína, která se snaží porozumět tomu, jak ucho a mozek spolupracují ve chvíli, kdy ucho zachytí nějaký zvuk. (3)

Člověk je neustále vystaven spoustě různých vlnění. To sestává z obrovského množství různých frekvencí, z nichž jen nepatrnou část jsme schopni zachytit našimi smysly. Lidské ucho je schopno vnímat frekvence v hrubém rozmezí mezi 20 Hz a 24 kHz a toto pásmo se s věkem zužuje. V rozmezí 2 kHz až 4 kHz je ucho nejcitlivější a slyší i málo hlasité zvuky. Jak se frekvence posouvají více k okrajům spektra, jejich hlasitost musí být vyšší, abychom je byli schopni zachytit, jak to ukazuje Obrázek 2.



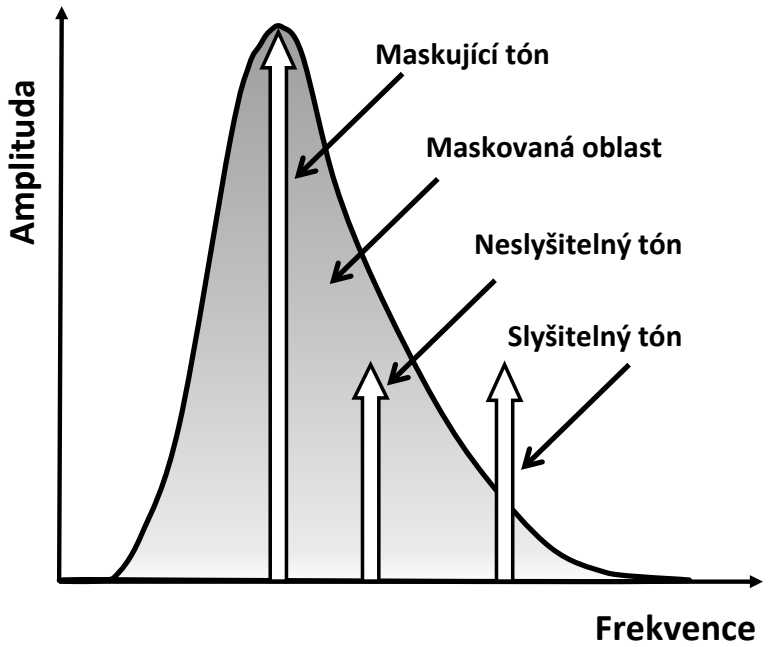
Obrázek 2: Schopnosti ucha při různých frekvencích (4)

Z toho důvodu lidé často nastavují ekvalizéry na svých přehrávačích symetricky, se zvýrazněnými okraji, čímž se hudba zdá příjemnější, protože se skoro vyrovnají úrovně jednotlivých frekvencí v celém slyšitelném spektru.

Vzhledem k tomu, že náš mozek není schopen zpracovat všechny podněty, které jsou k dispozici našim smyslům, nechá se tak považovat za jakýsi filtr informací, které jsme schopni vnímat. Tradiční CD přehrávače se snaží reprodukovat hudbu tak, jak byla zaznamenána. Tedy i u hudby reprodukované z CD nosiče náš mozek filtruje příchozí zvuky. Logickým postupem je filtrovat audio signál ještě před uložením a ušetřit tak spoustu místa za data, která stejně neslyšíme. Přesně tuto činnost zastávají perceptuální kodeky.

Jev, při kterém dochází k tomu, že některé zvuky slyšíme a jiné ne, nazýváme maskování. Existují dva typy maskování, kterých perceptuální kodeky využívají: simultánní a dočasné.

Experimentálně bylo zjištěno, že lidské ucho rozpoznává 24 frekvenčních pásem. Frekvence uvnitř těchto tzv. kritických pásem jsou uchem vzájemně hůře rozlišitelné a snáze dochází k maskování. Předpokládejme, že v audio signálu se vyskytuje jeden výrazný tón. Tento tón vytváří ve svém okolí v rámci jednoho kritického pásma maskovanou oblast. Všechny další tóny v maskované oblasti nebudou slyšet, pokud je jejich amplituda menší, než amplituda maskované oblasti v dané frekvenci. Tento jev (viz Obrázek 3) patří do frekvenční domény a nazývá se simultánní maskování.

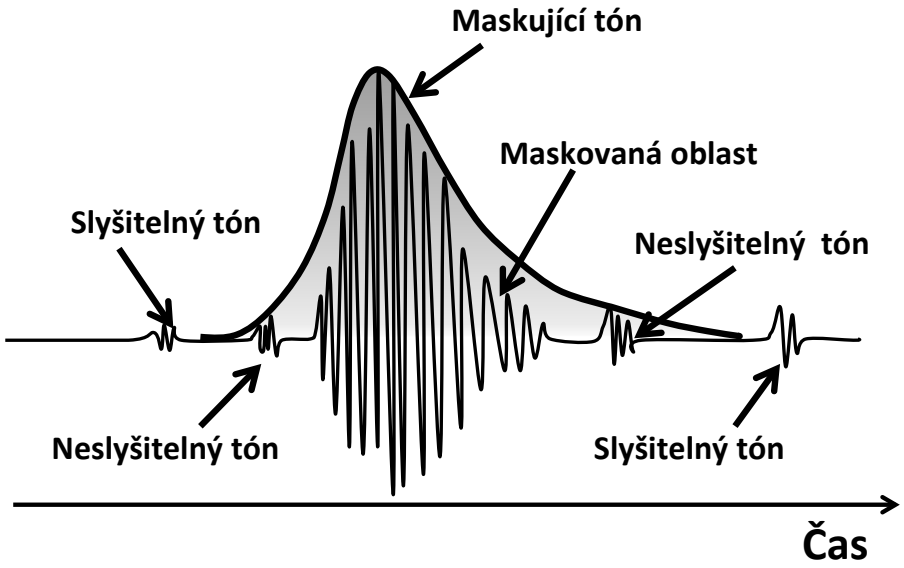


Obrázek 3: Simultánní maskování (5)

Audio signál se vyvíjí v čase. Pokud se ale bavíme o frekvenční doméně, příliš nám na čase nezáleží a spíše nás zajímají úrovně signálu. Ve frekvenční do-

méně probíhá množství transformací, při kterých dochází např. k rozkladu signálu na jednotlivé frekvence nebo fázovým posunům.

Dočasné maskování je na druhou stranu jevem časové domény. U časové domény nás naopak zajímá vývoj signálu jako celku v čase a jednotlivé frekvenční rozložení nehraje takovou roli.



Obrázek 4: Dočasné maskování (6)

Hlasitý tón maskuje slabší tón už chvíli před tím a ještě chvíli po tom, co zní, jak zobrazuje Obrázek 4. Prvnímu jevu se říká pre-maskování druhému pak post-maskování. Pre-maskování trvá obvykle okolo 50 ms, post-maskování pak většinou 50-300 ms a závisí na intenzitě a době trvání maskovacího tónu.

Psychoakustický model v MP3 kodéru využívá k analýze spektra určení dominantních komponent Rychlou Fourierovu transformaci (FFT) (7). Podle dominantních komponent jsou spočítány rozsahy maskovaných oblastí pro každé kritické pásmo a díky znalosti těchto rozsahů lze udržet chyby při kvantizaci v neslyšitelné oblasti.

2 Co je to MPEG-1 Layer III

2.1 Standard MPEG-1

International Organization for Standardization (ISO) je mezinárodní společenství, které vytváří standardy, aby bylo jednodušší směňovat zboží nebo služby. V rámci ISO byla vytvořena Moving Picture Experts Group (MPEG), která dostala za úkol vyvinout standard pro kompresi, dekompresi a vůbec reprezentaci videa, audia či jejich kombinaci. Tento standard měl za úkol býti dostatečně obecný, aby bylo možné rozkódovat data vytvořená libovolným kodérem. Tento předpoklad se může zdát implicitním, nicméně tvůrci ponechali vývojářům mnoho prostoru pro vlastní implementaci některých důležitých částí a umožnili tak případné vylepšování kvality kodéru. U spousty částí říkají pouze, co mají dělat a nikoliv už jak to mají dělat. Samozřejmým byl pak požadavek na zachování maximální zvukové i obrazové kvality, z něhož částečně vyplývá ono ponechání volnosti pro vlastní (lepší) implementaci kodéru.

Vývoj začal v roce 1988 a byl dokončen v roce 1992. Standard dostal oficiální označení ISO/IEC-11172 a skládá se ze systémové, video a audio části (8). V této práci se budeme věnovat výhradně audio části a proto je teď na místě se alespoň rámcově zmínit o tom, co obsahují ostatní dvě části:

Systémová část popisuje způsob přenosu signálu, přičemž standard MPEG-1 umožňuje přenášet data rychlostí 1–2 Mb/s po relativně spolehlivém přenosovém médiu (standard nabízí poměrně malou míru korekcí a ochran, a proto i malé chyby přenosu již přinášejí zaznamenatelné defekty).

Video část využívá (stejně jako audio) perceptuální kódování, což znamená, že redukuje nebo zcela zahazuje informace o částech spektra, které lidské oko není schopno rozlišit a tím velmi výrazně šetří data, která jsou potřeba pro reprezentaci videa. Nejběžnějším rozlišením MPEG-1 videa bylo 320x240 pixelů, které bylo považováno za rozumný kompromis mezi kvalitou, velikostí a nároky na tehdejší hardware. Z dnešního pohledu je také poměrně zajímavá druhá verze video standardu MPEG-2, dokončená v roce 1994, která se u nás v dnešní době masivně prosazuje v digitálním vysílání televize.

Audio část definuje tři vrstvy (anglicky layer) složitosti kodéru, se kterou jde ruku v ruce komprese, které takový kodér dosahuje. Tabulka 2 ukazuje potřebný datový tok pro všechny tři vrstvy a porovnává je s bezztrátovým uložením na CD.

Kódování	Poměr	Potřebný datový tok
CD kvalita	1:1	1,4 Mb/s
Vrstva I	4:1	384 kb/s
Vrstva II	8:1	192 kb/s
Vrstva III (MP3)	12:1	128 kb/s

Tabulka 2: Datový tok nutný pro přenos stereo signálu v CD kvalitě

Je zde vidět, že třetí vrstva dokáže, bez výrazného zkreslení zkomprimovat originální audio dvanáctkrát, což ji činí nejefektivnější a tedy i tou nejvhodnější vrstvou ze všech tří. MPEG-1 Layer III standardu se běžně přezdívá MP3 a občas jsou za něj zaměňovány i jiné vrstvy.

Ještě je potřeba zmínit, že hlavními vývojáři algoritmů v rámci audio části standardu MPEG-1 není samotná MPEG, ale Frauenhofer Institute spolu s německou Univerzitou v Erlangenu, kteří je vyvíjeli od roku 1987, a jejich práce byla pouze začleněna do standardu, což je běžný způsob vzniku standardu.

2.2 Datový tok

Volbou datového doku má uživatel možnost nastavit úroveň a tedy i kvalitu komprese. Informuje jí kodér, kolik bitů může použít pro každou sekundu nahrávky. Standard pro třetí vrstvu definuje datové toky od 8 kb/s do 320 kb/s, kde výchozí hodnota bývá nastavena na 128 kb/s. Větší datový tok umožňuje přesnější reprezentaci audio křivky. Je dobré si také uvědomit, že daný datový tok je u stereo nahrávky rozdělen mezi oba kanály v poměru podle toho, kolik který zrovna potřebuje.

Standard specifikuje dva různé typy datového toku a to konstantní (anglicky Constant Bitrate – CBR) a proměnný (anglicky Variable Bitrate – VBR). Při použití konstantního datového toku (většinou je to výchozí nastavení) jsou všechny části nahrávky kódovány se stejným množstvím bitů za vteřinu. To není ovšem zcela ideální, protože ve většině písní se mění intenzita zvuku (např. tiché intro na piano versus závěrečné finále s kompletní dechovou sekcí). Je tedy logické, že komplexnější část audio souboru bude vyžadovat větší datový tok, což ovšem CBR není schopno poskytnout a tak budou i tyto úseky kódovány stejným množstvím bitů. Variabilní datový tok elegantně řeší tento problém. Podle dynamiky písně lze skoro libovolně přidělovat množství bitů, které bude použito pro zakódování daného úseku. Rozsah použitých bitů lze u každého dobrého kodéru předem nastavit. VBR sebou přináší několik nepříjemností spojených s časováním. Většina dnešních přehrávačů zobrazuje bez dalších informací (např. XING rámec) špatně délku VBR souborů a nelze se v nich tedy dost dobře pohybovat. VBR se také nemusí hodit pro online vysílání, protože v průběhu přehrávání mění nároky na přenosovou kapacitu, což může být někdy nežádoucím jevem.

2.3 Způsoby uložení kanálů

Standard definuje čtyři způsoby uložení kanálů:

- jeden kanál (anglicky Single Channel) – ukládá se pouze jeden kanál, jedná se tedy o typický mono signál,
- dva kanály (anglicky Dual Channel) – oba kanály jsou ukládány samostatně, jedná se tedy o „dvojité mono“ a každý používá přesně polovinu zvoleného datového toku. Většina přehrávačů tento režim interpretuje jako stereo, ale někdy to nemusí být vhodné (např. při uložení dvou komentářů v různých jazycích),
- stereo – typický stereo signál, větší část datového toku je věnována tomu kanálu, který to potřebuje,
- joint stereo – tento režim si všímá redundancí mezi oběma kanály a snaží se jich využít.

První tři způsoby nepotřebují dalšího komentáře, podívejme se ale trochu blíže na joint stereo. Nabízí dvě techniky k docílení úspory, ty se jmenují middle/side stereo (MS stereo) a intensity stereo. Zřejmě není rozumné překládat tyto názvy, proto je ponechme v angličtině.

MS stereo se hodí v situacích, kdy jsou si oba původní kanály hodně podobné. Do levého kanálu se uloží jejich součet a do pravého jejich rozdíl. Pokud byly oba kanály opravdu podobné, pak kanál obsahující součet bude větší než rozdílový a proto mu bude věnována větší část daného datového toku. Je vhodné ještě poznamenat, že počítání MS sterea je bezztrátovou operací.

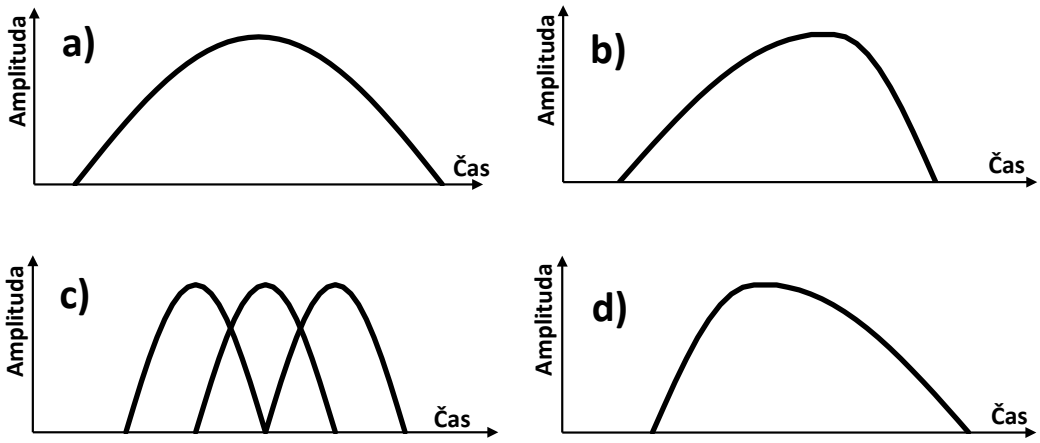
Intensity stereo přenáší audio informace pouze v jediném kanálu a informace o prostorovém rozložení jednotlivých frekvenčních skupin (pásem) se uchovávají v rámci *Side Information*. V tomto režimu ale vznikají odlišnosti oproti původnímu signálu – zvuky, které byly jen v jednom kanálu, se objeví v obou. Většina odlišností ale nebude lidským uchem postřehnutelná.

Některé kodéry umí využít kombinaci obou.

2.4 Volba okénka a antialiasing

Slyšitelné spektrum je při kódování rozděleno na 32 stejně velikých pásem. Následně je použit filtr, který z celého spektra propustí do každého pásma jen frekvence jemu náležící. Bohužel takový filtr má pouze omezenou přesnost (nelze udělat filtr s přesně čtvercovým průběhem) a proto při takovém dělení vzniknou nepřesnosti. Budeme je nazývat aliasy.

Jako ochrana proti vzniku aliasů (tzv. antialiasing) se signál rozdělí na tzv. okénka, která se přes sebe překládají a redukují tak nepřesnosti na okrajích pásem. Obrázek 5 ukazuje různé typy okének.



Obrázek 5: Typy okének, a) normální, b) start, c) tři krátká, d) stop (9)

Jak lze z obrázku vytušit, okénko je, velmi laicky řečeno, signál vynásobený první polovinou sinusoidy. Standard přesně koeficienty sinusoid definuje a při dekódování jsou okénka překládána přes sebe, čímž je docíleno plynulého napojení.

Podle množství změn v signálu se psychoakustický model rozhodne, který typ okének využít. Porovnávají se vždy dva následující výstupy FFT, a pokud se audio signál rychle mění, jsou použita krátká okénka, aby poskytla vyšší rozlišení v časové doméně, které je nutné pro kontrolu různých časových artefaktů, např. pre-echo. V opačném případě jsou použita dlouhá okénka, která poskytují lepší spektrální rozlišení ve frekvenční doméně. Start a stop okénka řeší přechody mezi dvěma předchozími, z čehož vyplývá, že koeficienty sinu se v jeho průběhu mění, čímž výsledná křivka docílí jistého „nesinusového“ vzhledu.

Takto zpracovaný signál je následně poslán do modifikované diskrétní kosinové transformace (MDCT), kterou popisuje Lincoln Bosse v (10), a kde je každé pásmo rozděleno na 18 frekvenčních vzorků, čímž získáme $32 * 18 = 576$ hodnot, tzv. granule. Granule je anglické slovo, které by se dalo přeložit jako malá část nebo částice, my ho však překládat nebudeme a budeme používat anglický originál.

Bližší studií tohoto jevu bylo zjištěno, že frekvenční aliasy vznikají symetricky na hranicích jednotlivých pásem. Například mezi prvním pásmem (frekvenční vzorky 0 až 17) a druhým pásmem (vzorky 18 až 35) se alias 17. frekvence spáruje s aliasem 18. frekvence, alias 16. frekvence s aliasem 19. atd. Kodér se snaží redukovat tento efekt tak, že z obou frekvencí vypočítá očekávaný alias a tato čísla porovnává s reálným aliasem, který kompenzuje (11). Této technice se říká série křížových výpočtů (anglicky butterfly computations) a dochází tedy k přidání vážených koeficientů zrcadlově převrácených verzí přilehlých pásem ke každému pásmu. Používá se hlavně proto, aby se při následném Huffmanově kódování ušetřilo místo, které aliasy zabírají.

3 Anatomie MP3 souboru

V této kapitole si popíšeme, z čeho se skládá MP3 soubor a jaké jsou významy jednotlivých datových polí.

Všechny MP3 soubory jsou rozděleny na malé fragmenty zvané rámce. Každý rámec obsahuje 1152 audio vzorků a trvá 26 ms. Z toho vyplývá, že během každé vteřiny přehrávač přehraje skoro 39 rámců. Navíc je každý rámec rozdělen do dvou částí, kterým se říká granule, z nichž každý obsahuje 576 vzorků. Protože datový tok určuje přesnost každé navzorkované hodnoty, je jasné, že se vzrůstajícím datovým tokem se bude zvětšovat i velikost rámce. Lze ji spočítat podle následujícího vzorce:

$$\left\lfloor \frac{144 * datov\acute{y}\_tok}{vzork\,ovac\acute{i}\_frekvence} + patička \right\rfloor \text{ kB}$$

Patička se používá pouze pro prodloužení rámce pro potřeby správného výpočtu délky. Více je patička popsána v následující kapitole. Informace o tom, zda je potřeba patičku přičíst, je uložena v hlavičce.

Dále je potřeba si uvědomit, že délka rámce je vždy celé číslo. Například pro MP3 soubor s datovým tokem 128 kb/s a vzorkovací frekvencí 44,1 kHz dostáváme po dosazení do vzorce:

$$\lfloor (144 * 128000)/44100 \rfloor = 417 \text{ kB}$$

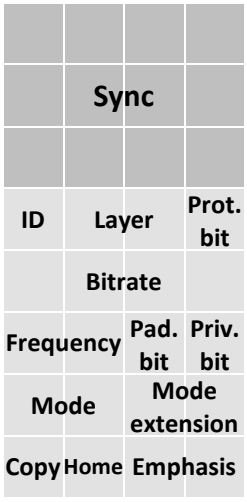
Rámec se skládá z pěti částí: hlavičky, CRC, Side Information, audio dat a dodatečných dat (anglicky Ancillary Data), tak jak to ukazuje Obrázek 6.

Hlavička	CRC	Side Information	Audio data	Dodatečná data

Obrázek 6: Rozložení rámce

3.1 Hlavička rámce

Hlavička rámce má 32 bitů a obsahuje synchronizační slovo spolu s popisem rámce. Synchronizační slovo se nachází na začátku každého rámce a umožňuje snadnou synchronizaci při čtení audio dat z jiné pozice než jejich začátku. Díky tomu lze data v MP3 formátu vysílat online. Přijímač pouze hledá synchronizační slovo, a když ho najde, má skoro jistotu, že našel začátek validního rámce. Z důvodů robustnosti se doporučuje provádět synchronizaci na dva rámce, protože se v datech v praxi někdy vyskytují falešná synchronizační slova. Celou hlavičku MP3 rámce znázorňuje Obrázek 7.



Obrázek7: Hlavička MP3 rámce (18)

3.1.1 Sync (12 bitů)

Je synchronizační slovo popisované výše. Všechny bity musí být nastaveny na 1. Synchronizační slovo MP3 rámce tedy vypadá následovně: 1111 1111 1111.

3.1.2 ID (1 bit)

Určuje verzi MPEG rámce. Pokud je bit nastaven na 1, jedná se o verzi MPEG-1, pokud ne, jde o verzi MPEG-2.

3.1.3 Layer (2 bity)

Jak ukazuje Tabulka 3, tyto dva bity slouží pro výběr aktuální vrstvy.

Layer	Použitá vrstva
00	Rezervováno
01	Vrstva III
10	Vrstva II
11	Vrstva I

Tabulka 3: Definice aktuální vrstvy

3.1.4 Protection bit (1 bit)

Tento bit signalizuje, zda je použita ochrana rámce pomocí CRC. Pokud je příznak nastaven na 1, CRC není použit, jinak použit je.

3.1.5 Bitrate (4 bity)

Říká dekodéru, jaký datový tok byl při kódování zvolen pro tento rámec. Pokud je použito CBR, pak všechny rámce mají tuto hodnotu stejnou. Přípustné hodnoty včetně hodnoty bitového příznaku ukazuje Tabulka 4.

Bity	MPEG-1			MPEG-2		
	Vrstva I	Vrstva II	Vrstva III	Vrstva I	Vrstva II	Vrstva III
0000	Není určeno			Není určeno		
0001	32	32	32	32	32	8
0010	64	48	40	64	48	16
0011	96	56	48	96	56	24
0100	128	64	56	128	64	32
0101	160	80	64	168	80	64
0110	192	96	80	192	96	80
0111	224	112	96	224	112	56
1000	256	128	112	256	128	64
1001	288	160	128	228	160	128
1010	320	192	160	320	192	160
1011	352	224	192	352	224	112
1100	284	256	224	284	256	128
1101	416	320	256	416	320	256
1110	448	384	320	448	384	320
1111	Rezervováno			Rezervováno		

Tabulka 4: Povolené hodnoty datového toku v kb

3.1.6 Frequency (2 bity)

Uchovává informaci o zvolené frekvenci. Tabulka 5 zobrazuje povolené kombinace dvou frekvenčních bitů a jejich významy.

Bity	MPEG-1	MPEG-2	MPEG-2.5
00	44100	22050	11025
01	48000	24000	12000
10	32000	16000	8000
11		rezervováno	

Tabulka 5: Povolené samplovací frekvence v Hz

3.1.7 Padding bit (1 bit)

Určuje, zda má být rámec prodloužen o patičku o délce jeden byte či nikoliv. Používá se tehdy, kdy chceme přesně dodržet průměrný datový tok. Již dříve jsme si řekli, podle jakého vzorce se počítá délka rámce a také to, že je to celočíselný údaj. Pro rámec o datovém toku 128 kb/s a frekvenci 44100 Hz dostáváme délku 417 bytů. Při neceločíselném dělení však dostáváme 417,96 bytů. Některé rámce budou proto muset být o jeden byte delší, aby byla kompenzována chyba vzniklá při celočíselném dělení.

3.1.8 Private bit (1 bit)

Jeden bit pro vlastní použití. Standard garantuje, že nebude v budoucnu pro nic použit.

3.1.9 Mode (2 bity)

Pomocí těchto bitů je nastaven způsob uložení kanálů. Jednotlivé způsoby uložení jsme si popsali v kapitole 2.3 Způsoby uložení kanálů. Přípustné hodnoty a jejich význam ukazuje Tabulka 6.

Bity	Způsob uložení kanálů
00	Stereo
01	Joint Stereo
10	Dual Channel
11	Single Channel

Tabulka 6: Volba uložení kanálů

3.1.10 Mode Extension (2 bity)

Tyto dva bity jsou použity pouze v případě, kdy je zvoleno joint stereo a tehdy slouží k výběru kódovací techniky. V průběhu kódování lze měnit techniky měnit či dokonce zapínat a vypínat. Přípustné hodnoty a jejich význam ilustruje Tabulka 7.

První bit	MS Stereo
0	Ne
1	Ano

Druhý bit	Intensity stereo
0	Ne
1	Ano

Tabulka 7: Význam mode extension bitů

3.1.11 Copyright bit (1 bit)

Pokud je tento bit nastaven na 1, pak je obsah rámce chráněn copyrightem a neměl by být kopírován.

3.1.12 Home bit (1 bit)

Pokud je tento bit nastaven na 1, pak se jedná o originální data, jinak se jedná o kopii.

3.1.13 Emphasis (2 bity)

Tento indikátor říká dekodéru, zda jsou potřeba v souboru provést nějaké další korekce spektra, například je-li potřeba znovu přepočítat zvukové vzorky po některých prostorových efektech. Používá se jen velmi zřídka. Tabulka 8 ukazuje významy jednotlivých možností.

Emphasis	Význam
00	žádné korekce
01	50/15 ms
10	rezervováno
11	CCITT J.17

Tabulka 8: Význam datového pole Emphasis

3.2 CRC

CRC (anglicky cyclic redundancy check) je speciální hashovací funkce, používaná pro detekci chyb vzniklých během přenosu dat. Jedná se o dělení polynomů se zbytkem v tělese GF(2), kde jednotlivé koeficienty dělence reprezentují bit 1 nebo 0 v rámci přenášené zprávy a dělitel je definován jako $x^{16} + x^{12} + x^5 + 1$. Příjemce ověří korektnost doručené zprávy tak, že oba vydělí a zbytek mu vyjde nula. Pro lepší porozumění by bylo dobré prostudovat např. (12). V dnešní době se však CRC u MP3 v podstatě nepoužívá.

Pokud je v hlavičce nastaven *Protection bit* na 0, tak bezprostředně za hlavičkou následuje 16 bitů CRC kódu, který ověřuje, zda byla důležitá data přenesena správně. Za důležitá data považuje standard bity 16 až 31 v hlavičce a v *Side Information*. Pokud by byla některá data z tohoto rozsahu porušena, není možné správně dekódovat obsah rámce a takový by měl být buď nahrazen tichem, nebo předchozím rámcem.

3.3 Side Information

Tato část rámce obsahuje informace, které jsou nutné pro dekódování audio dat. I nadále v textu budeme používat jejich anglické označení. Jejich velikost záleží na způsobu uložení kanálů. Pokud rámec obsahuje pouze jeden kanál, délka je 17 bytů, jinak 32 bytů. Jednotlivé díly zobrazuje Obrázek 8.

Main_data_begin	Private_bits	Scfsi	Side_info_granule1	Side_info_granule2

Obrázek 8: Side Information

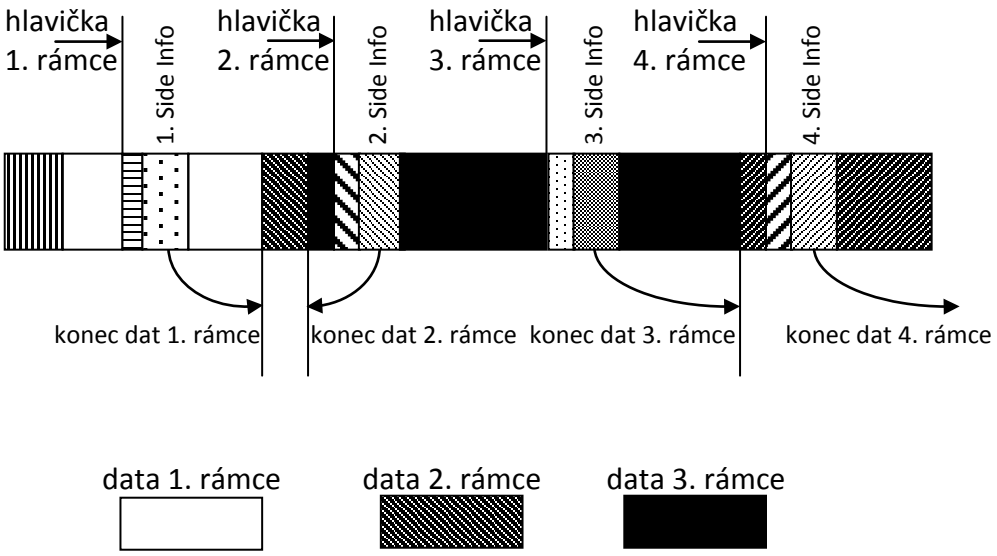
V následujícím popisu je vždy název datové položky a za ním v závorkách velikost. Pokud je v závorkách jediné číslo, pak je velikost vždy stejná. Pokud jsou uvedeny čísla dvě, tak první značí velikost v mono módu a druhé ve všech ostatních.

3.3.1 Main_data_begin (9 bitů)

Třetí vrstva jako jediná využívá bitového rezervoáru. Je to technika, která umožňuje použít nevyužité místo z předchozích rámců pro data rámců následujících. K nalezení začátku audio dat aktuálního rámce slouží dekodéru tato položka, které vyjadřuje negativní offset vzhledem k začátku rámce. Vzhledem k tomu, že pro ni bylo vyhrazeno devět bitů, může posouvat data až o $(2^9 - 1) * 8 = 4088$ bitů, tedy až o sedm rámců zpět.

Je důležité si uvědomit, že statické části rámce jako hlavička nebo *Side Information* nejsou v tomto offsetu započítány.

Obrázek 9, zobrazuje jednu takovou situaci. Audio data třetího rámce (znázorněna černou barvou) zabírají většinu místa ve třetím rámci, celou datovou část druhého rámce a zasahují ještě částečně i do datové části prvního rámce.



Obrázek 9: Využití bitového rezervoáru (13)

3.3.2 Private_bits (5 bitů, 3 bity)

Dalších několik bitů pro vlastní použití, ISO opět garantuje, že v budoucnosti nebudou k ničemu využity.

3.3.3 Scfsi (4 bity, 8 bitů)

Zkratka znamená *ScaleFactor Selection Information* a dala by se přeložit jako informace o výběru rozsahových koeficientů. Určuje, zda se tyto koeficienty budou přenášet pro každý granule zvlášť nebo ne. Koeficienty jsou rozděleny do čtyř skupin, jak ukazuje Tabulka 9.

Skupina	Rozsahová pásma
0	0, 1, 2, 3, 4, 5
1	6, 7, 8, 9, 10
2	11, 12, 13, 14, 15
3	16, 17, 18, 19, 20

Tabulka 9: Skupiny rozsahových koeficientů

Jsou přenášeny čtyři bity pro každý kanál, a pokud je příslušný bit nastaven na nulu, jsou koeficienty přenášeny pro každý granule. V opačném případě jsou koeficienty uložené pro *granule0* platné i pro *granule1*, což logicky znamená, že ušetřené místo lze použít pro samotná audio data.

Pokud jsou použita krátká časová okénka, jsou koeficienty vždy přenášeny pro každý granule.

3.4 Side Information pro každý granule

Poslední dvě části *Side Information* jsou důležité dekódovací informace pro každý granule, jak ukazuje Obrázek 10. Jejich struktura je stejná, proto ji nebudeme popisovat dvakrát.

Part2_3_length	Big_values	Global_gain	Scalefac_compress
Block_split_flag	Block_type	Mixed_block_flag	Table_select
Subblock_gain	Region0_count	Region1_count	Preflag
Scalefac_scale	Count1table_select		

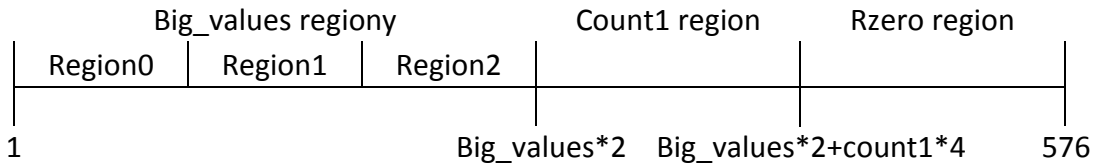
Obrázek 10: Datová pole Side Information pro každý granule

3.4.1 Part2_3_length (12 bitů, 24 bitů)

Udává počet bitů, které dohromady zabírají rozsahové koeficienty (*part2*) a Huffmanovým kódováním zakódovaná audio data (*part3*), jinými slovy udává velikost jednoho kanálu aktuálního granule.

3.4.2 Big_values (9 bitů, 18 bitů)

Všech 576 frekvencí každého granule nejsou kódovány stejnými Huffmanovými tabulkami, ale jsou rozděleny na pět regionů, jak to ukazuje Obrázek 11. Důvod takového rozdělení je v tom, že různé části audio spektra obsahují různé rozsahy čísel a proto je výhodnější použít pro každý úsek jinou tabulku kódů.



Obrázek 11: Regiony frekvenčního spektra

Rozdělení je provedeno na základě rozsahu hodnot, za předpokladu, že vyšší frekvence mají menší amplitudu nebo nepotřebují být kódovány vůbec.

Region jménem *Rzero* reprezentuje nejvyšší frekvence, jejichž velikost amplitudy je nula. V regionu *Count1* jsou zakódovány kvantizované čtveřice čísel, které jsou pouze -1, 0 nebo 1. Konečně, v regionu *Big_values* jsou uloženy dvojice čísel, které charakterizují frekvenční rozsah pokračující až k nule. Jak si již jistě pečlivý čtenář domyslel, datové pole *Big_values* tedy určuje délku *Big_values* regionu.

3.4.3 Global_gain (8 bitů, 16 bitů)

Určuje kvantizační krok pro daný rámec, kterým je při dekódování potřeba přenásobit všechny hodnoty, abychom získali hodnoty z původního rozsahu.

3.4.4 Scalefac_compress (4 bity, 8 bitů)

Určuje počet bitů použitých pro přenos rozsahových koeficientů. Každý granule může být rozdělen do 12 (při použití krátkých časových okének) nebo 21 (při použití dlouhých okének) pásem, které jsou následně rozděleny na dvě skupiny, 0 až 6, 7 až 11 pro krátká okénka a 0 až 10, 11 až 20 pro dlouhá.

Scalefac_compress	Slen1	Slen2
0	0	0
1	0	1
2	0	2
3	0	3
4	3	0
5	1	1
6	1	2
7	1	3
8	2	1
9	2	2
10	2	3
11	3	1
12	3	2
13	3	3
14	4	2
15	4	3

Tabulka 10: Předdefinovaná tabulka pro scalefac_compress index

Scalefac_compress pole je index do standardem definované tabulky (viz Tabulka 10), kde *Slen1* a *Slen2* říkají, kolik bitů má být přiděleno první a kolik druhé skupině koeficientů.

3.4.5 Block_split_flag (1 bit, 2 bity)

Pokud je tento příznak nastaven na jedna, pak je použito jiné, než normální okénko. Datová pole *Block_type*, M*ixed_block_flag* a S*ubblock_gain* jsou používána pouze, pokud je B*lock_split_flag* nastaven na jedna. Jeho hodnota má také vliv na *Big_values* regiony, kde se vůbec nepoužívá *region2* a tedy všechny hodnoty, které nejsou v *regionu0* patří do *regionu1*.

3.4.6 Block_type (2 bity, 4 bity)

Toto pole indikuje typ použitého okénka pro aktuální granule. Je použito pouze ve chvíli, kdy je B*lock_split_flag* nastaveno na jedna. Tabulka 11 ukazuje možnosti nastavení a jejich význam.

Block_type	Použité okénko
00	Nepovolená hodnota
01	Start
10	tři krátká
11	Stop

Tabulka 11: Definice pole block_type

3.4.7 Mixed_block_flag (1 bit, 2 bity)

Tento příznak určuje, zda jsou pro všechna frekvenční pásma použita stejná okénka. Pokud je nastaven na jedna, tak nejspodnější dvě pásma jsou transformována za použití normálního okénka a pro ostatní je použito takové okénko, jaké je specifikováno pomocí B*lock_type*.

3.4.8 Table_select (10 bitů, 20 bitů nebo 15 bitů, 30 bitů)

Tento údaj říká, která z Huffmanových tabulek je použita v různých částech *Big_values* regionu. Pro každý region je vyhrazeno pět bitů, z čehož vyplývá, že kodér i dekodér mají k dispozici 32 předpočítaných Huffmanových tabulek z nichž volí vždy tu nejvýhodnější. První dva velikostní údaje jsou platné v případě, kdy je nastaven B*lock_split_flag*, jinak jsou platná druhá dvě čísla. V prvním případě se totiž nepoužívá R*egion2* a tudíž není potřeba pro něj přenášet informace o Huffmanově tabulce.

3.4.9 Subblock_gain (9 bitů, 18 bitů)

Toto pole je použito pouze v případě, kde se používají tři krátká okénka a určuje o kolik je potřeba zvětšit či zmenšit hodnotu *Global_gain* pro jednotlivá okénka.

3.4.10 Region0_count (4 bity, 8 bitů), Region1_count (3 bity, 6 bitů)

Region0_count obsahuje o jedna méně než je počet frekvenčních pásem v *Regionu0*, *Region1_count* pak obsahuje stejnou informaci pro *Region1*.

3.4.11 Preflag (1 bit, 2 bity)

Tento bit rozhoduje o dalším zesílení kvantizovaných frekvenčních vzorků. Pokud je nastaven na jedna, pak jsou k hodnotám vzorků přidány hodnoty z připravené tabulky. Nepoužívá se u krátkých okének.

3.4.12 Scalefac_scale (1 bit, 2 bity)

Rozsahové koeficienty (*Scale Factors*) jsou logaritmicky kvantizovány s krokem 2 nebo $\sqrt{2}$. Při dekódování je potřeba provést inverzní operaci se správným krokem. Tento bit informuje o tom, jaký krok byl použit.

Scalefac_scale	Velikost kroku
0	$\sqrt{2}$
1	2

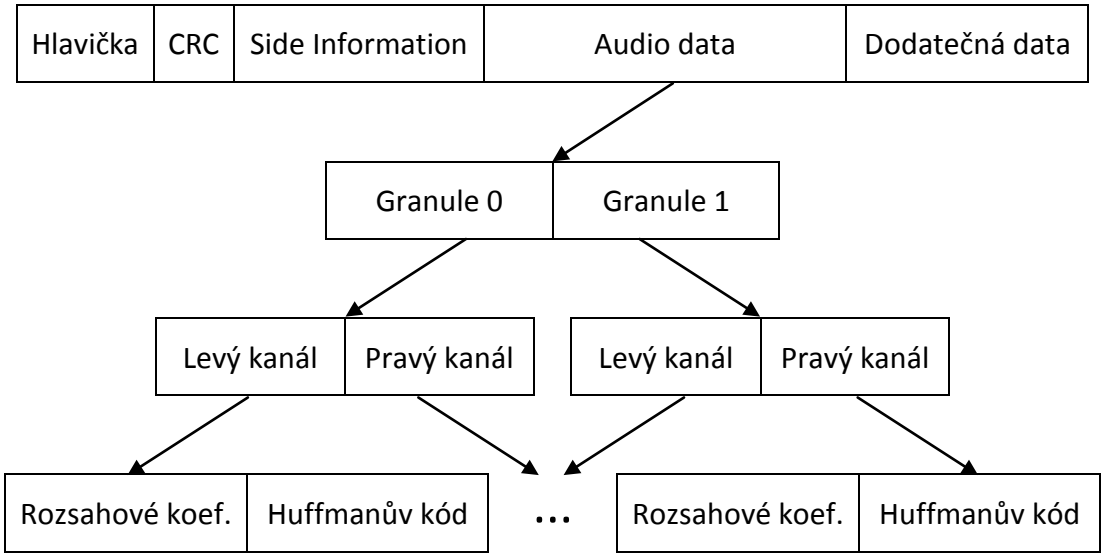
Tabulka 12: Kvantizační krok rozsahových koeficientů

3.4.13 Count1table_select (1 bit, 2 bity)

Pro region *Count1* jsou dostupné dvě Huffmanovy tabulky. Tento bit říká, kterou zvolit.

3.5 Audio data

Audio data sestávají z rozsahových koeficientů a Huffmanovým kódováním zakódovaných dat. Jejich rozložení ukazuje Obrázek 12.



Obrázek 12: Organizace audio dat

3.5.1 Rozsahové koeficienty

V běžném životě potřebují být velká čísla ukládána pomocí velkého množství bitů, kdežto pro menší stačí menší rozsahy. Ve chvíli, kdy jsou bity pro ulo-

žení dat cenným prostředkem, je nutné tuto strategii trochu přehodnotit. Ve skutečnosti je totiž potřeba velkých bitových rozsahů jen pro zachování přesnosti, nikoliv rozsahu. Například staří říct jedna hodina místo 3600 sekund. Kodér volí rozsah bitů pro hodnoty na základě zvolené přesnosti (datového toku určeného uživatelem) a potom podle toho určuje jaké „měřítko“ bude zvoleno pro uložení jednotlivých skupin hodnot. Rozsahové koeficienty jsou toto měřítko.

Koeficienty jsou přenášeny vždy jeden pro každé frekvenční pásmo. Datové pole *scfsi*, které jsme si popsali dříve, určuje, zda se koeficienty sdílí mezi oběma granule, či nikoliv. Skutečný rozsah koeficientů záleží na poli *scalefac_compress*. Způsob, jakým se má spektrum na frekvenční pásma rozdělit, mají kodér i dekodér uloženo v předpřipravených tabulkách pro každou kombinaci vzorkovací frekvence a typu okénka.

3.5.2 Huffmanův kód

V této části jsou uloženy samotné audio vzorky, zakódované Huffmanovým kódováním. Informace o tom, jak je dekódovat je uložena v *Side Information*. V *Big_values* regionu jsou hodnoty kódovány vždy po dvojicích a v regionu *Count1* po čtveřicích. Vzhledem k tomu, že region *Rzero* obsahuje samé nuly, nekódují se.

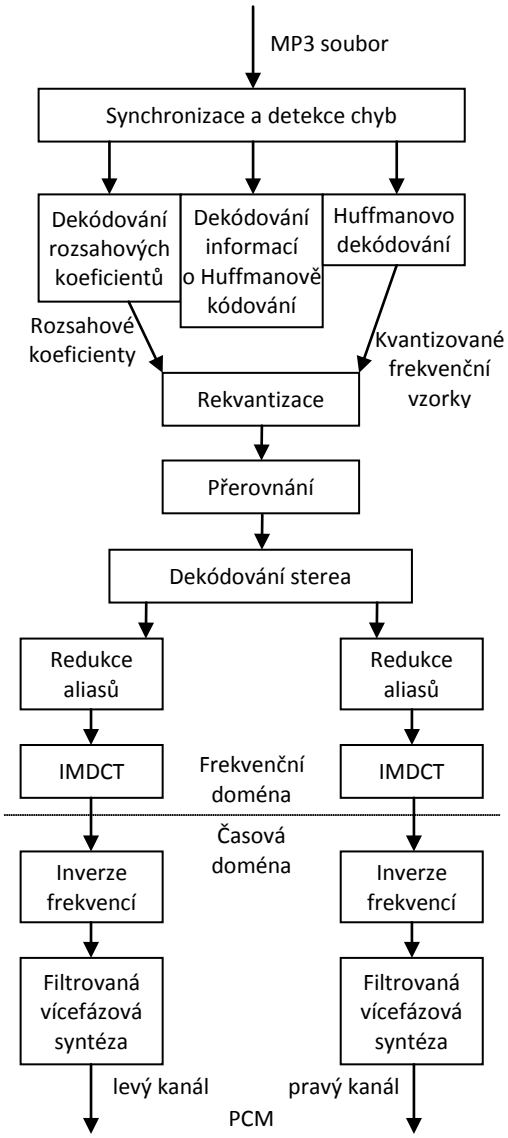
Podle toho, jestli jsou použita krátká nebo dlouhá okénka, liší se pořadí Huffmanových dat. Pokud jsou použita dlouhá okénka, Huffmanova data jsou seřazena podle frekvence.

3.6 Dodatečná data

Tato data jsou volitelná a jejich počet se mění. Leží bezprostředně za audio daty a končí se začátkem dalšího rámce. Často obsahují data bitového rezervoáru následujících rámců. Pro dekódování rámce, ve kterém se nacházejí, jsou však zbytečná.

4 Proces dekódování

Dekódovací proces je velmi komplikovaný a proto si zde pouze rámcově popíšeme jeho nejdůležitější součásti, abychom si dokázali udělat lepší představu o tom, co se při něm děje. Schéma celého procesu ukazuje Obrázek 13.



Obrázek 13: Dekódovací proces

4.1 Synchronizace a kontrola chyb

V této první části dekodér obdrží proud dat, ve kterém musí identifikovat jednotlivé rámce. Vyhledává synchronizační slovo, jímž každý rámec začíná. Když ho najde, může ověřit, že se opravdu jedná o rámec tím, že z údajů v hlavičce spočítá délku aktuálního rámce a ověří, že na nové pozici začíná další rámec. Pokud je v hlavičce poznamenáno, že rámec má CRC ochranu, dekodér by měl ověřit, že pole chráněná kontrolním součtem jsou neporušena. Pokud nejsou data v pořádku, nelze rámec úspěšně dekódovat a měl by být nahrazen tichem nebo předchozím rámcem. V opačném případě jsou uložena datová pole hlavičky i *Side Information* do příslušných datových struktur a dekódovací proces může začít.

4.2 Huffmanovo dekódování

Vzhledem k tomu, že Huffmanovo kódování má proměnnou délku slova, nelze začít s dekódování jen tak někde. Dekodér musí začít tam, kde začíná první zakódované slovo. Tento poznatek získá ze *Side Information*, kde jsou všechny nutné parametry pro Huffmanův dekodér.

Navíc musí dekodér zajistit, že z dat bude získáno všech 576 frekvenčních vzorků a to i tehdy, když tam nejsou všechny uloženy. Již dříve jsme si řekli, že region *Rzero* obsahuje jen nuly, které se nikam neukládají. Zde proto musí dekodér nulami doplnit frekvenční vzorky na správný počet.

4.3 Dekódování rozsahových koeficientů

V této části jsou podle parametrů v *Side Information* načteny rozsahové koeficienty. Jsou načítány po granulích a nejdříve pro levý, pak pro pravý kanál. Může nastat situace, kdy se koeficienty nenačítají, ale jsou použity z minulého granule. Koeficienty budou potřeba hned v dalším kroku při rekvantizaci frekvenčních vzorků.

4.4 Rekvantizace

Jak probíhá kvantizace vzorků jsme si popsali již v kapitole 3.5.1 Rozsahové koeficienty. Při rekvantizaci je potřeba tento proces otočit, což znamená, že před jakýmikoliv výpočty je potřeba vynásobit dekódované frekvenční vzorky odpovídajícím měřítkem (rozsahovým koeficientem), aby měly všechny vzorky opět stejný rozsah.

Rekvantizace využívá datových polí *Global_gain*, *Scalefac_scale* a *Preflag* a samozřejmě rozsahových koeficientů, získaných při dekódování v minulé fázi, i frekvenčních vzorků, získaných Huffmanovým dekódováním, ze kterých obnoví původní koeficienty vygenerované pomocí MDCT části v kodéru. Použitý postup se liší podle typu okénka a výsledky vždy jsou umocněny na 4/3, protože je potřeba korigovat výsledky mocnin aplikovaných v kodéru.

4.5 Přerovnání

Frekvenční koeficienty generované při rekvantizaci nejsou vždy ve stejném pořadí. Pokud jsou použita dlouhá okénka, MDCT v kodéru generuje vzorky seřazené nejdříve podle pásem a pak podle frekvencí. Při použití krátkých okének jsou vzorky generovány v pořadí podle pásma, pak okénka a nakonec podle frekvence. Kvůli zvýšení efektivity Huffmanova kódování jsou výsledky krátkých okének přerovnány podle pásem, frekvence a až nakonec podle okénka, protože bližší frekvence mívají podobné hodnoty.

Přerovnávací část tedy zjišťuje, zdali byla použita krátká okénka a pokud ano, změní pořadí jednotlivých vzorků.

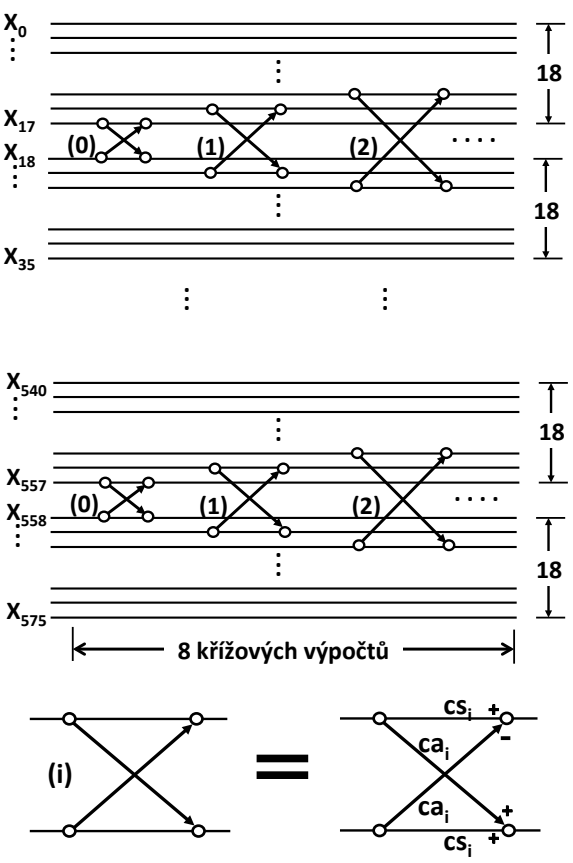
4.6 Dekódování sterea

Účel tohoto bloku je již z názvu zřejmý: oddělit od sebe informace pro levý a pravý kanál, pokud je v nahrávce uložen stereo signál. Způsob uložení stereo signálu lze vyčíst z příslušných datových polí, a jak víme z dřívějška, existují čtyři způsoby uložení audio kanálů (viz kapitola 2.3 Způsoby uložení kanálů). Nebudeme se tedy o tom znovu rozepisovat, jen poznamenejme, že kromě režimů jednoho nebo dvou kanálů není získání informace o kanálech úplně triviální.

4.7 Redukce aliasů (tzv. antialiasing)

V kapitole 2.4 Volba okénka a antialiasing jsme si již řekli, že po aplikaci MDCT je ještě použit algoritmus křížových výpočtů kvůli redukci aliasů vzniklých při rozdělení audio spektra na pásma a kvůli úspoře místa při Huffmanově kódování.

Kvůli tomu musí pro korektní rekonstrukci audio signálu nyní nastoupit opačná technika a artefakty k signálu opět přidat. Rekonstrukce aliasů se provádí osmi křížovými výpočty pro každé pásmo tak, jak to ukazuje Obrázek 14. Koeficienty ca_i a cs_i, jimiž jsou jednotlivé frekvenční vzorky násobeny, jsou definovány přímo ve standardu. Redukce i následná rekonstrukce aliasů se provádí pouze při použití krátkých okének.



Obrázek 14: Křížová redukce aliasů (18)

4.8 Inverzní modifikovaná diskrétní kosinová transformace (IMDCT)

Analytické vyjádření IMDCT nám ukazuje Obrázek 15. Za n je potřeba dosadit počet vzorků (pro krátká okénka 12, pro dlouhá 36). Pokud jsou transformována krátká okénka (jsou tři, jak víme), je transformováno každé zvlášť. Celkem $n/2$ hodnot x_k je transformováno na n hodnot x_i. Při použití krátkých okének musí být všechny výsledky přeloženy přes sebe a sečteny.

$$x_i = \sum_{k=0}^{\frac{n}{2}-1} x_k * \cos\left(\frac{\pi}{2n}\left(2i + 1 + \frac{n}{2}\right)(2k+1)\right) \; pro\; i \text{ od } 0 \; do \; n-1$$

Obrázek 15: Analytické vyjádření IMDCT (14)

Nakonec, prvních 18 vzorků (čili polovina) aktuálního bloku je sečtena s druhou polovinou minulého bloku a vrácena jako výsledek. Druhá polovina aktuálního bloku je uschována pro spojení s následujícím blokem.

4.9 Inverze frekvencí

Protože je potřeba kompenzovat změny frekvencí provedené v rámci syntézy, je každý lichý frekvenční vzorek každého lichého pásma vynásoben minus jedničkou.

4.10 Filtrovaná vícefázová syntéza

Syntéza postupně převede 32 pásem z 18 časových domén v každém granule na 18 bloků po 32 PCM samplech. Výsledné vzorky potřebují ještě přerovnat do správného pořadí. Po přerovnání dostáváme kýžený výsledek, 576 vzorků, které můžeme předat zvukové kartě a ta je bude schopna přehrát.

5 Editační knihovna

Vždy, když v běžném editoru upravujeme MP3 soubor, dochází při jeho uložení k dalším ztrátám kvality. Znovu nastupuje psychoakustický model a perceptuální kodek a znovu je celá nahrávka analyzována a kódována. Problém je o to horší, že kódovací proces není zcela standardizován, a tak každý psychoakustický model má trochu jiné parametry a za slyšitelný může považovat i zvuk, který jiný model za slyšitelný nepovažuje a stejně tak obráceně. Často také záleží na hardwarových prostředcích dostupných kodéru – mobilní zařízení obsahující slabší procesor a menší operační paměť bude v rámci kodéru obsahovat jednodušší psychoakustický model než výkonnější stroje. Při několikanásobném uložení můžeme tedy ztratit pokaždé jiné informace a v součtu může být celková chyba slyšitelná.

Knihovna, kterou jsem měl za úkol navrhnout a implementovat řeší tento problém jednoduše tak, že editovaný soubor nedekóduje a následně znovu nekóduje, ale operuje přímo nad zkomprimovanými daty, čímž nedochází ke ztrátě žádných dalších informací, a to ani lokálně v místě úprav (např. spoje dvou souborů). Cenou za bezztrátovost jsou ale omezené možnosti úprav a hlavně jejich přesnost. Veškeré operace probíhají s přesností na jeden audio rámec (cca 26 ms), což nemusí vždy zcela dostačovat.

Knihovnu lze začlenit do jiného projektu využitím jejího API, lze využít samostatné jednoúčelové utility spouštěné z příkazového řádku, které jsou připraveny pro každou editační operaci, anebo lze využít jednoduché grafické rozhraní, které přehledně sdružuje všechnu funkčnost v jednoduchém okně.

5.1 Koncept intervalů

Tento koncept je poměrně zásadním pro práci s knihovnou. Interval reprezentuje časový úsek jednoho souboru, kterému lze přidělit některou editační akci. Při otevření souboru je automaticky vytvořen interval, který ho celý pokrývá. O otevírání souborů si řekneme více v následující kapitole.

Intervaly lze rozdělovat v libovolném místě s přesností na jeden rámec a sousedící intervaly, které na sebe v rámci jednoho souboru přímo navazují, lze opět spojit. Pořadí intervalů lze libovolně měnit, vytvořené intervaly lze duplikovat a libovolný interval také mazat.

Dále je možné posouvat hranice jednotlivých intervalů, přičemž knihovna kontroluje, zda je časový údaj, kam chceme hranici posunout, uvnitř souboru. Předpokládejme, že chceme posunout horní hranici nějakého intervalu (např. chceme prodloužit úsek, kterému jsme nastavili akci „odstranit z výsledného souboru“). Za ním však typicky leží další interval, který na něj přesně navazuje

(v našem konkrétním případě jde o část nahrávky, kterou chceme ve výsledném souboru zachovat). Pokud posuneme horní mez našeho intervalu, knihovna automaticky posune i spodní mez intervalu následujícího, čímž se nám snaží ušetřit práci.

5.2 Otevření souboru

Při otvírání každého souboru je provedena jeho analýza. Při ní jsou zjištěny a uloženy základní informace o daném souboru (např. počet rámců, jejich pozice v souboru, datový tok, počet bytů potřebných pro zvuková data atd.), přičemž v případě detekce nějaké zásadní chyby (např. nekompatibilní audio vrstva, změna vzorkovací frekvence apod.) je proces otevírání ukončen.

Poté, co úspěšně proběhne analýza souboru, je vytvořen první interval, který pokrývá celou délku souboru, a je přidán jako poslední do seznamu intervalů.

5.3 Akce zeslabit či zesílit

Několikrát jsme si již zdůraznili, že všechny prováděné operace jsou bezztrátové, podívejme se tedy na to, jak funguje bezztrátové zesílení souboru. V rámci intervalu, kterému chceme změnit hlasitost je ve všech rámcích změněno datové pole *Global_gain*, které se používá pro dekvantizaci frekvenčních vzorků. Jeho rozsah hodnot je 0 – 255 a typická hodnota uložená při vytvoření souboru je 127, případně je to hodnota jí blízká. Uživatelem zadaná hodnota změny hlasitosti je jednoduše přičtena do tohoto datového pole. Máme tedy dostatek místa pro zesilování i zeslabování. Dodejme, že jednotka zesílení či zeslabení je decibel.

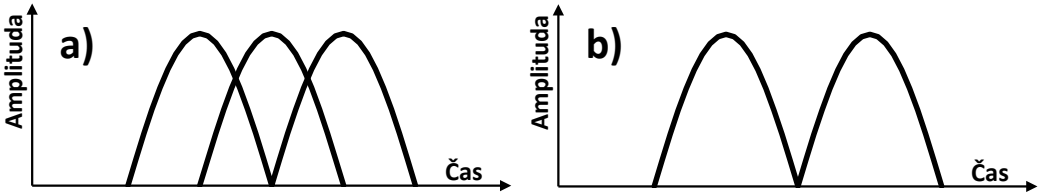
5.4 Akce odstranit z výsledného souboru

Tato akce je z technického hlediska poněkud zajímavější. Bylo zde potřeba vyřešit, jak napojit dvě spolu nesouvisející části signálu. Pokud totiž spojíme dvě části audio signálu, které na sebe přesně nenavazují, uslyšíme ve výsledku nepříjemné a výrazné lupnutí. Pozorného čtenáře jistě ihned napadne, že stačí detekovat v obou signálech místo, se stejnou hodnotou (ideálně se pro tento záměr hodí nula) a napojit signály tam. Tento postup bohužel není možné provést, protože i kdybychom byli schopni ve výsledném signálu identifikovat vhodné místo pro spojení, museli bychom odstranit další část dat a provést nové kódování. Když odhlédneme od toho, že tento postup není bezztrátový, přináší s sebou jeden nemalý problém. Kódování je totiž řádově komplikovanější než dekódování a je proto zcela mimo rozsah této práce.

Nabízí se ale jiná, velmi elegantní cesta, kterou umožňuje technická realizace kodéru. V kapitole 4.8 Inverzní modifikovaná diskrétní kosinová transformace (IMDCT) jsme si řekli, že po aplikaci IMDCT jsou přes sebe přesně v polovině přeloženy dva sousední rámce a že je to možné právě díky okénkům, která zajiš-

ťují návaznost takového spoje. Je tedy nasnadě, vložit mezi obě nesouvisející části signálu, jeden rámec ticha, který, díky přeložení rámců přes sebe, nejen, že nebude slyšet, ale vytvoří onen společný bod v nule, který zajistí plynulé napojení obou částí. Přesnější popis technické realizace spoje, včetně popisu práce s bitovým rezervoárem, lze nalézt v kapitole 6.1.3 Metody akcí.

Pro lepší představu se podívejme na Obrázek 16, jehož část a) ukazuje napojení běžných po sobě jdoucích rámců po aplikaci IMDCT a část b) pak využití tohoto jevu pro plynulé napojení dvou částí souboru (či dvou souborů).



Obrázek 16: Vyřešení řezu pomocí okénka, a) běžný signál, b) řez

5.5 Přehrávání mezivýsledků

Knihovna umožňuje přehrát rozpracovaný soubor či soubory, bez nutnosti ukládání na disk. Uživatel tak má možnost poslechnout si výsledky své práce, aniž by musel pokaždé vytvářet nový soubor a spouštět ho v externím přehrávači. Knihovna přehraje vše od zvoleného místa až do posledního rámce posledního intervalu.

Pro přehrávání je použita veřejná verze kódu dekodéru od IIS Fraunhofer (15). Tento kód byl zvolen proto, že se na rozdíl od ostatních nesnaží dělat desítky dalších věcí, zběsilými optimalizacemi počínaje a podporou ekvalizéru či nejrůznějších filtrů konče. Pro náš účel postačuje základní funkčnost, díky které je často vidět, co daná část kódu opravdu dělá a že to opravdu odpovídá tomu, co definuje standard. Dekodér sice není nejefektivnější, ale při výkonu dnešních počítačů není tento nedostatek poznat.

5.6 Uložení souboru

Při uložení souboru jsou načítány, v pořadí nastaveném uživatelem, postupně od prvního až po poslední, vytvořené intervaly a jsou rámec po rámci kopírovány do výstupního souboru. Pokud je potřeba s daným rámcem provést nějakou operaci (zesílit či neuložit do výstupního souboru), je ihned po načtení provedena. Všechny intervaly jsou spojeny do jednoho tak, jak jsme si popsali při řezání a napojování dvou částí signálu a výsledkem je tedy jeden MP3 soubor, který obsahuje všechny původně vložené soubory/intervaly.

5.7 XING rámec

XING rámec není součástí standardu MPEG-1, ale vyskytuje se v drtivé většině MP3 souborů, a proto je potřeba ho brát v potaz. Je zařazen na začátek

souboru a chová se jako běžný rámec ticha. Dekodér, který ho dokáže rozpoznat, toto ticho nepřehrává, ale získá z něj několik informací, z nichž nejdůležitější je celková délka audio dat v bytech a počet audio rámců.

Při změnách délky souboru, či spojování více souborů do jednoho se samozřejmě mění délka původního souboru a proto je potřeba upravit informace o délce i v XINGu. Knihovna vezme XING prvního otevřeného souboru a zapíše do něj nový údaj o celkovém počtu rámců a smaže údaj o délce dat. Jsou k tomu dva důvody: pro zjištění aktuální délky dat by musela opět projít všechny soubory/intervaly a sečíst délku všech rámců a vypočítat délky spojovacích rámců, což není úplně žádoucí a pak hlavně údaj o počtu rámců stačí přehrávači pro korektní výpočet délky souboru. Vzhledem k tomu, že délka audio dat se od délky souboru liší o případné ID3 tagy, které nejsou typicky delší než několik běžných rámců, může dekodér spočítat průměrný datový tok z délky celého souboru a chyba bude minimální.

5.8 ID3 tag

Další věcí, která sice nepatří do standardu, ale audio soubory provází už od prvopočátku, je ID3 tag, a proto jsem rozhodl i jemu věnovat pozornost. Jedná se o kontejner pro metadata (typicky jimi jsou např. autor písně, její název, CD ze kterého pochází apod.), který je uložen na začátku nebo na konci MP3 souboru a uchovává relevantní informace pro daný soubor. Existují dvě majoritní verze ID3 tagu, ID3v1 a ID3v2, přičemž obě jsou knihovnou částečně podporovány.

Knihovna sice zatím neumožňuje jednotlivé tagy upravovat, ale alespoň je zachovává v souboru. Pokud editujeme jeden soubor, zůstane původní tag, beze změny, pokud spojujeme více souborů do jednoho, pak výsledek bude mít tag souboru odpovídajícího prvnímu intervalu. Toto chování zatím nelze měnit, ale další vývojová verze bude rozšiřovat podporu tagů o některé základní operace.

6 Implementace

Již od úplného začátku bylo naplánováno použití nízkoúrovňového jazyka pro výkonný kód knihovny a interpretovaného jazyka pro tvorbu grafického rozhraní. Kombinace C a Pythonu nebyla ani zdaleka bezproblémová, ale nakonec se podařilo oba jazyky úspěšně spojit. Nemalou mírou pomohl generátor bindingů SWIG (16), který ze zdrojového kódu v C dokáže vytvořit modul pro Python a postará se o to, aby byly na straně Pythonu použity odpovídající datové typy pro parametry a návratové hodnoty metod.

Kromě samotného API knihovny jsou k dispozici čtyři jednoúčelové utility použitelné z příkazového řádku a jednoduché grafické rozhraní, které sdružuje veškerou funkčnost do jednoho okna.

Při psaní knihovny jsem se snažil využít pouze standardní C a API daného operačního systému. Pod Windows jsou použity eventy, semafor a funkce z rodiny Waveform, přičemž autoři v dokumentaci zaručují zcela bezproblémový chod pro Windows 2000 a novější. Pod Linuxem knihovna využívá modul pthread a z něj mutex a condition variable a ALSA modul, což značí, že by měla být schopna pracovat pod libovolným Linuxem, protože se jedná o dva z nejběžnějších systémových modulů, které by měly přítomny u všech běžných distribucí.

6.1 Implementace knihovny

V této kapitole si popíšeme implementaci důležitých částí knihovny. Z metod tvořících rozhraní budeme explicitně mluvit pouze o jedné metodě (metoda `do_file`), protože celé rozhraní knihovny je popsáno v kapitole Dodatek B: Dokumentace a není nutné ho tu rozebírat dvakrát.

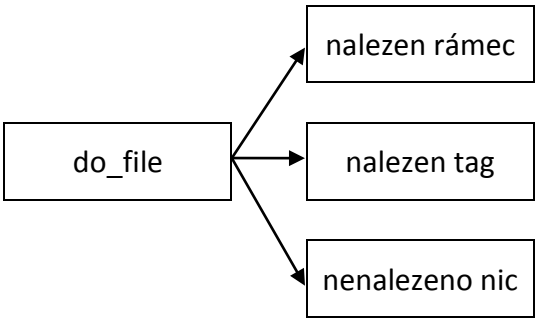
Za zmínku pouze stojí, že všechny veřejné metody publikované v rámci API informují o svém průběhu pomocí návratových kódů. Pokud při vykonávání metody dojde k chybě, je uživatel informován pomocí chybového kódu některé z definovaných chybových konstant. Jejich rozsah byl záměrně zvolen záporný, aby bylo možno rozlišit, kdy došlo k chybě a kdy se jedná o regulérní návratovou hodnotu metody.

6.1.1 Metoda do_file

Tato metoda je ústřední metodou celé knihovny, neboť se stará o rozhodování, jak zpracovat jednotlivé rámce. Jedná se v podstatě o jeden velký switch, který nasměruje vykonávání do příslušné metody. Rozhodnutí udělat tuto metodu prakticky bezestavovou s sebou přináší několik výhod, ale také několik komplikací. Bezestavovostí se myslí hlavně to, že metoda zpracovává jen jeden rámec a je volána pro každý rámec zvlášť, dokud nenarazí na konec souboru.

Jistý stav knihovny vytváří ostatní proměnné, které si pamatují např. aktuální soubor, ale tato metoda sama o sobě nerozlišuje ani soubory ani intervaly.

Její funkcí je to, že z aktuálního souboru (který je potřeba ručně nastavit jako aktuální a otevřít) přečte čtyři byty a zjistí, zda se jedná o audio rámec, tag či ani jedno, jak to demonstruje Obrázek 17. Pokud se jedná o validní hlavičku rámce, zjistí podle knihovních proměnných, zda je aktuálně zvolen režim analýzy, přehrávání, či ukládání a zavolá příslušnou metodu, která se postará o započtení důležitých údajů, dekódování rámce a jeho odeslání na zvukovou kartu, nebo uložení rámce do výstupního souboru. Pokud se jedná o tag, zavolá metodu pro jeho zpracování, která tag buď uloží do výstupního souboru, nebo prostě přeskočí. Pokud se nejedná ani o jedno, podle zvoleného režimu buď zapíše do výstupního souboru, nebo započte a zahodí jeden byte z načtených čtyř a načte ze souboru další jeden byte, přičemž nastaví do stavových proměnných, že příště není potřeba načítat nové čtyři byty, ale že jsou již načteny.

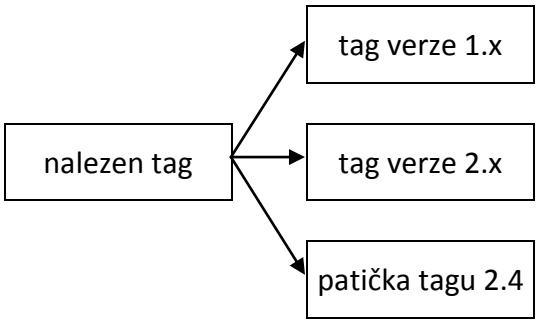


Obrázek 17: Možné situace v metodě do_file

6.1.2 Metody pro práci s ID3 tagy

Situace, které mohou při identifikaci tagu nastat, jsou tři, jak ukazuje Obrázek 18. Metody, které mají práci s tagy na starosti, jsou taktéž tři a jmenují se `deal_with_id31tag`, `deal_with_id32tag` a poslední z nich `deal_with_id32padding`. Jejich názvy dobře vypovídají o tom, co metody dělají a proto se nebudeme zabývat opakováním jejich jmen.

Vzhledem k tomu, že knihovna zatím nepodporuje přímo úpravu tagů, všechny tři metody dělají v podstatě stejnou věc. U tagu verze 1 i u patičky tagu verze 2.4 známe dopředu jejich velikost. Je to 128 resp. 10 bytů a proto příslušné metody načtou dalších 128-4 resp. 10-4 bytů a podle zvoleného režimu je buď uloží do výstupního souboru, nebo přeskočí. Poslední metoda má o trochu složitější práci, musí nejdříve spočítat, jak dlouhý je tag, protože jeho velikost není fixní. Autoři ID3.2 tagu (17) se rozhodli, že pro uložení velikosti tagu budou sloužit byty 7 až 10 a že z každého bytu budou použity bity 0 až 7. Velikost tagu je proto spočítána ze vztahu:



Obrázek 18: Možnosti knihovny při identifikaci tagu

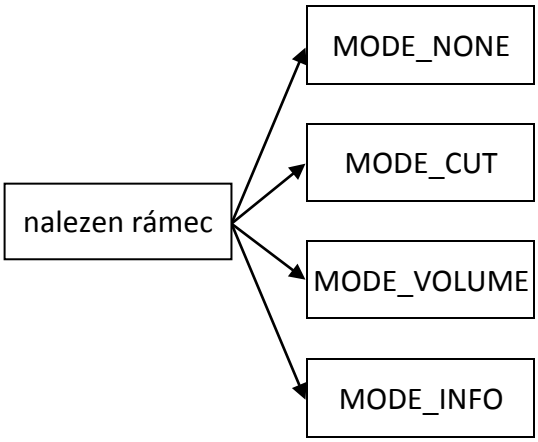
$$buffer[6] \ll 21 \mid buffer[7] \ll 14 \mid buffer[8] \ll 7 \mid buffer[9]$$

Obrázek 19: Vztah pro výpočet velikosti ID3.2 tagu

Daný počet bytů mínus čtyři je, stejně jako u minulých metod, načten do paměti a buď uložen do výstupního souboru, nebo přeskočen.

6.1.3 Metody akcí

Při identifikaci validního audio rámce záleží na tom, jaký je aktuálně nastavený režim průchodu. Tyto režimy jsou identifikovány konstantami, z nichž tři může uživatel nastavit.



Obrázek 20: Režimy průchodu při identifikaci rámce

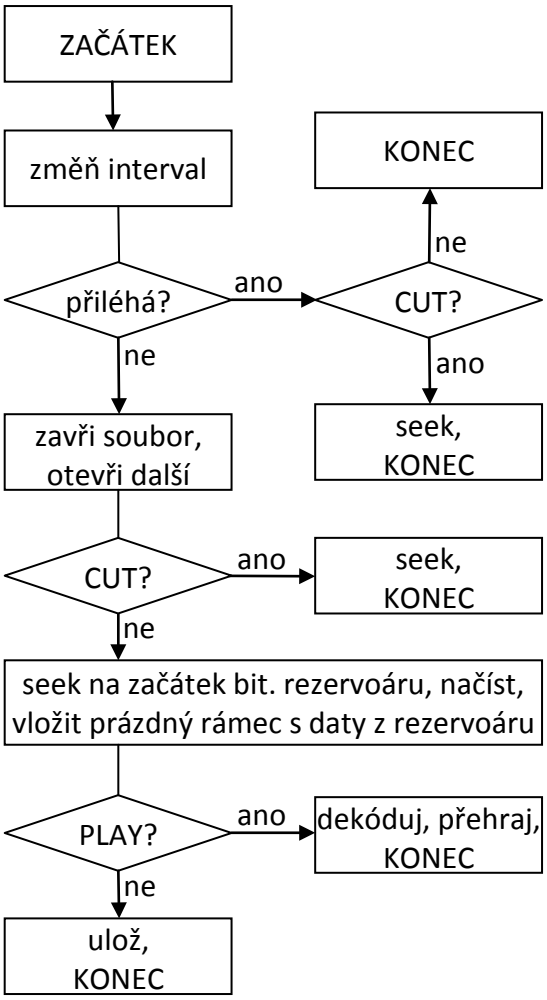
Režim *MODE_INFO* není pro použití uživatelem a slouží pro počáteční analýzu souboru. V tomto režimu je zavolána metoda `gather_info`, která uloží aktuální pozici rámce v souboru do vnitřní datové struktury. Kvůli rychlosti vyhledávání byl pro ukládání pozic zvolen červeno-černý strom. Metoda dále přičte délku rámce a jeho datový tok do adekvátních proměnných a zvětší počítadlo rámců pro aktuální soubor o jedna.

Režim *MODE_VOLUME* volí uživatel ve chvíli, kdy nastaví intervalu akci „změnit hlasitost“. V tomto režimu je volána metoda `do_volume`, která v aktuálním rámci načte hlavičku i *Side Information* a do pole *Global_gain* pro každý kanál a každý granule přičte uživatelem definovaný rozsah změny hlasitosti. Pokud je při zpracovávání rámce nastaven příznak *play*, je upravený rámec poslán k dekódování a přehrání. V opačném případě je uložen do výstupního souboru.

Režim *MODE_CUT* je nastaven podobně jako předchozí. Jedná se o akci “odstranit ze souboru“. Je zavolána metoda `do_cut`, která všechny rámce z aktuálního intervalu přeskočí, tedy je ani nepředá k dekódování a přehrání, ani je neuloží do výstupního souboru.

Režim *MODE_NONE* pokrývá situace, kdy není intervalu nastavena žádná akce. O tento režim se stará metoda `do_nothing`, která zcela nedostává svému názvu a přece jen dělá to, že podle příznaku *play* buď pošle rámec na dekódování a přehrání nebo uloží do výstupního souboru.

Každá z metod, kromě `gather_info`, kontroluje, zda aktuálně zpracovávaný rámec není posledním rámcem intervalu. Pokud ano, je zavolána metoda `do_merge`, která spojí interval s následujícím. Její průběh ukazuje Obrázek 21. Nejdříve se podívá, zda následující interval přesně nepřiléhá k aktuálnímu. Pokud ano, tak buď jen změní aktuální interval na následující, nebo, v situaci, kdy má následující interval nastaven režim *MODE_CUT*, ještě seekne na předposlední rámec dalšího intervalu. Pokud následující interval přímo nepřiléhá, je uzavřen aktuální soubor a otevřen následující a pokud se jedná o režim *MODE_CUT* je, stejně jako v minulém případě, seeknuto na předposlední rámec a metoda končí. V opačném případě se seekuje na začátek bitového rezervoáru (sedm rámců před začátkem intervalu), který je celý načten. Z prvního rámce nového intervalu se zjistí, kolik je z bitového rezervoáru opravdu potřeba a následně je vytvořen prázdný rámec, jenž, kromě toho, že zajistí napojení obou okolních rámců, také pojme a do svých dodatečných dat uloží všechna data z rezervoáru. Délka prázdného rámce je přizpůsobena délce rezervoáru, což, vzhledem k tomu, že prázdný rámec nemá žádná svá audio data, není problém. Následně se opět podle příznaku *play* rozhodne, zda se mají data předat k dekódování a přehrání, či se uložit do souboru.



Obrázek 21: Průběh funkce do_merge

6.1.4 Metody pro dekódování a přehrávání

Jak jsme si řekli již dříve, dekódovací metody patří do veřejné verze kódu od Fraunhoferova Institutu a vzhledem k tomu, že jejich složení a průběh skoro přesně odpovídá popisu z kapitoly 4 Proces dekódování, nebudeme je zde detailně probírat.

Přehrávání využívá API operačního systému, a proto knihovna obsahuje separátní modul pro přehrávání pod Windows i pod Linuxem. Idea je však u obou modulů podobná. Je vytvořeno audio vlákno, kterému jsou postupně předávána

audio data, a vlákno je předává zvukové kartě k přehrání ve chvíli, kdy je informováno, že zvuková karta je připravena zpracovat další data.

Windows API vyžaduje, aby aplikace používala několik bufferů, které bude postupně zvukové kartě dávat. Ve chvíli, kdy je jeden buffer zpracován, je zavolána callback funkce, která zvedne semafor o jedna, čímž dává najevo, že zvuková karta je připravena přijmout další data, na což kód čekající na daný semafor okamžitě reaguje tím, že předá zvukové kartě pomocí funkce `waveOutWrite` další, již připravený buffer. Knihovna je schopna dodávat rámce mnohokrát rychleji, než je zvuková karta přehrává, proto dekodér čeká na zmíněném semaforu a po odeslání bufferu naplní další a opět čeká na jeho zpracování.

ALSA která se stará o přehrávání pod Linuxem, nevyžaduje několik bufferů, ale přijímá pouze pointer na audio data. Zda jsou v jednom bufferu či v různých je jí jedno. Nevýhodou je, že po dobu zpracovávání nelze do daného bufferu přistupovat, proto data čekají v bufferu v knihovně a ve chvíli, kdy je zvuková karta připravena zpracovat další data (o čemž ALSA informuje nastavením condition variable), je tento buffer překopírován do bufferu pro zvukovou kartu, nastavena druhá condition variable, která signalizuje modulu ALSA aktuálnost bufferu a pokračuje se v přípravě dalšího bufferu, který opět čeká v knihovní proměnné, aby byl ve správné chvíli překopírován do vyhrazeného bufferu, jenž obhospodařuje ALSA.

Počet bufferů a počet rámců v každém bufferu (čili zpoždění mezi odesláním dat zvukové kartě a jejich přehráním), je možné nastavit pomocí konstant PLAY_BUFFERS_NUMBER a PLAY_BLOCKS_IN_SAMPLE.

6.2 Utility pro práci z příkazového řádku

Tyto utility jsou čtyři a jsou to jednoduché a jednoúčelové programy, které umožňují využít funkce knihovny z příkazového řádku. V této kapitole si je popíšeme.

První utilita se jmenuje volumer a slouží pro zesílení či zeslabení části souboru a jako parametry očekává: název vstupního souboru, název výstupního souboru, časový rozsah a údaj o tom, zda zesilovat nebo zeslabovat a o kolik. Časový rozsah se skládá ze dvou údajů (od a do, přičemž oba jsou včetně), ve formátu (H:)MM:SS:MS. Při nesprávném počtu nebo formátu parametrů je nahlášena chyba a program skončí. Pokud jsou parametry v pořádku, je otevřen zadaný soubor, provedena jeho analýza, vytvořeny dva nebo tři intervaly, podle toho, kde se nalézá zadaný rozsah a odpovídajícímu intervalu nastavena akce "změň hlasitost" a rozsah změny. Pokud během analýzy dojde k nějaké zásadní chybě (viz kapitola 5.1 Koncept intervalů), je program ukončen, vypíše tuto chybu na obrazovku a vrátí číslo chyby jako svůj návratový kód. V opačném případě je vytvořen nový soubor, který bude mít název zadaný pomocí druhého

parametru a do něj je překopírován zdrojový soubor, přičemž ve zvoleném intervalu je provedena požadovaná úprava hlasitosti.

Druhá utilita se jmenuje cutter a nabízí možnost odstranit zvolený úsek ze souboru. Bere čtyři parametry: název vstupního souboru, název výstupního souboru, čas od a čas do a stejně jako volumer projde vstupní soubor, vytvoří intervaly, jednomu nastaví akci "odstranit" a následně vytvoří nový soubor s názvem získaným z druhého parametru, který nebude obsahovat zvolený interval.

Třetí se jmenuje merger a bere tři parametry: název prvního souboru, název druhého a název třetího souboru. Provede nezbytnou analýzu prvního a druhého souboru, vytvoří jeden interval přes každý soubor a sloučí je do souboru, jehož název dostane jako třetí parametr.

Poslední utilita se jmenuje player a bere jen jeden povinný parametr a jeden volitelný. Povinným parametrem je název souboru, který má přehrát, volitelným pak časový údaj ve standardním formátu, určující odkud chceme přehrávat.

6.3 Grafické rozhraní

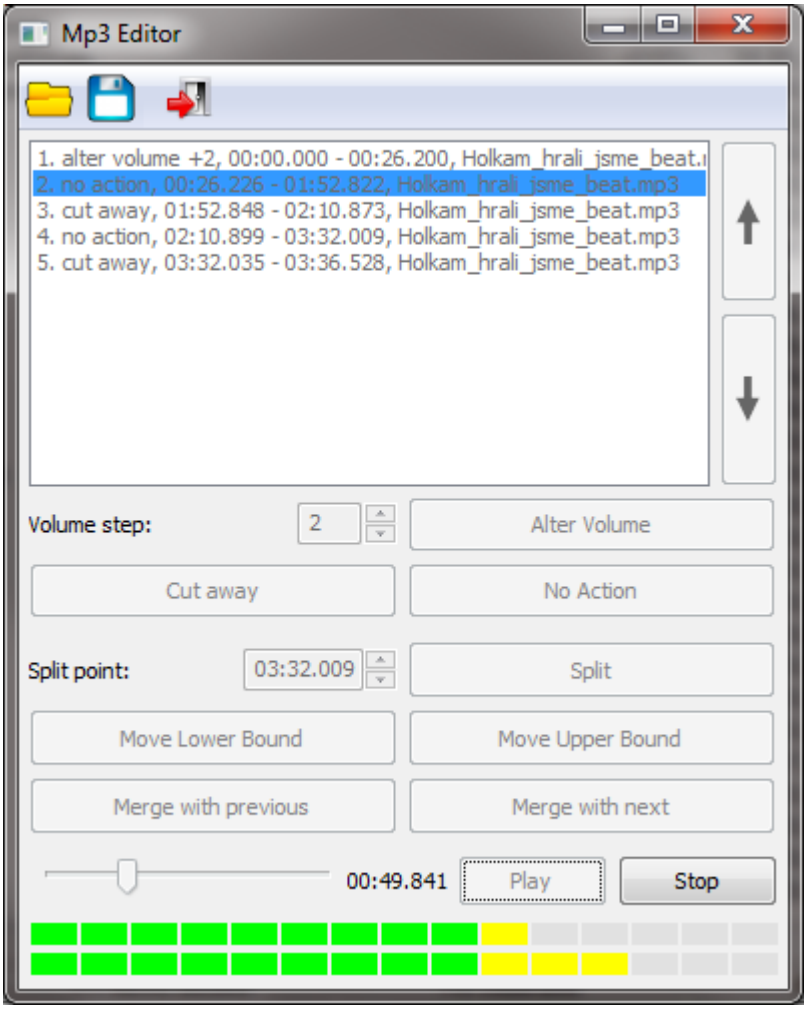
Již dříve jsme si řekli, že pro grafické rozhraní je použit Python. Python byl nečekaně jedním z největších zdrojů problémů. Původně jsem plánoval použití grafické knihovny Tkinter a to z toho důvodu, že je přímou součástí Pythonu. Tkinter bohužel nedostačoval, co se týče vzhledu nebo pohodlnosti práce, a tak jsem se začal ohlížet po jiných grafických knihovnách.

Po rychlém opuštění knihovny Qt kvůli její velikosti, jsem hodně času strávil s GTK. Až po kompletním návrhu okna jsem zjistil, že moje knihovna není schopna v kombinaci s GTK přehrát soubor. Ve chvíli, kdy jsem do pythoního zdrojového kódu napsal "import gtk", aniž bych z tohoto modulu cokoliv použil, přestala knihovna zcela produkovat poslouchatelný zvuk a nahradila ho pískáním, cvakáním a šustěním.

Posledním zastavením na trnité cestě za grafickým rozhraním je WXPython. Pomocí něho se mi funkční GUI povedlo vytvořit celkem jednoduše a pouze s několika drobnými problémy. Problémem je značná nekompatibilita tohoto interpretovaného jazyka při přechodu platforem. Některé postupy a metody, které fungují pod Windows, nefungují pod Linuxem a obráceně.

Každý, kdo někdy programoval GUI aplikaci, jistě ví, že není dobrý nápad aktualizovat elementy v grafickém okně z více než jednoho vlákna. Při takovém snažení pod Linuxem aplikace buď zamrzne, nebo (a to spíše) je zabita. Pythoní kód použitý pod Linuxem vytváří události, které jsou posílány hlavnímu vláknu, jež, na základě jejich parametrů, aktualizuje grafické elementy. Tento přístup má jednu nevýhodu a tou je měřitelné a měnící se zpoždění mezi odesláním a

přijetím události. Pod Windows nefunguje tento postup pro změnu vůbec. Pod Windows se tedy dopouštím hrubé nerozvážnosti a aktualizuji obsah elementů z více vláken. Během testování na několika strojích se neobjevil žádný problém a aplikace ani jednou nespadla.



Obrázek 22: Grafické rozhraní programu, probíhá přehrávání

Pro správnou funkčnost GUI je potřeba WXPython 2.8 nebo novější. Obrázek 22 ukazuje jeho vzhled, který si v rychlosti popíšeme.

V horní části okna jsou tlačítka pro otevření a uložení souboru a pro ukončení programu. Hned pod nimi je výčet existujících intervalů, spolu s tlačítky pro posun intervalu nahoru nebo dolu. Pro jakoukoliv akci je nutné, aby byl zvolen některý z intervalů, aby nedošlo k omylu. Pokud není vybrán žádný interval, nejsou také aktivní žádná tlačítka.

Pro nastavení akce slouží tři tlačítka zhruba uprostřed. První z nich (Alter Volume) nastaví zvolenému intervalu akci změny hlasitosti, přičemž je použit takový krok, jaký je v danou chvíli nastaven v poli Volume step. Další tlačítko (Cut away) odstraní zvolený interval z výsledného souboru a poslední tlačítko (No Action) zruší případnou akci, kterou už má interval nastavenu.

Dalších šest tlačítek slouží pro práci s intervaly. První tlačítko (Split) rozdělí interval na pozici, která je zadána v poli Split point. Knihovna kontroluje správnost formátu a rozsahu vkládané hodnoty, přičemž se vždy jedná o absolutní čas v rámci souboru. Další dvě tlačítka (Move Lower Bound a Move Upper Bound) slouží k posunu spodní resp. horní hranice intervalu, která je opět posunuta na čas uvedený v poli Split point. Jak jsme si již řekli, pokud interval vedle posouvané hranice přesně na náš aktuální navazuje, bude jeho mez posunuta také. Ve stejné situaci (intervaly leží těsně za sebou) lze použít další dvě tlačítka (Merge with previous a Merge with next), která spojí aktuálně vybraný interval s intervalem před ním popř. za ním.

Poslední dvě tlačítka slouží ke spuštění a zastavení přehrávání a pomocí posuvníku můžeme nastavit, odkud chceme ve vybraném intervalu začít přehrávat. Po zastavení přehrávání se posuvník přesune zpět na pozici, od které začalo přehrávání a je označen původní interval, pokud došlo k jeho změně.

Posledním grafickým prvkem je peak meter, který zobrazuje úrovně audio signálu pro oba kanály a díky kterému je vyžadován WXPython 2.8+. Je zobrazována maximální úroveň za několik posledních rámců, protože jeden audio rámec je příliš krátký na to, aby bylo možné okem postřehnout všechny změny, pokud by byl peak meter aktualizován po každém přehraném rámci. Ve chvíli, kdy se objeví červená barva, došlo při dekódování ke zkreslení signálu kvůli jeho nadbytečné hlasitosti. Kvůli výše popsanému zpoždění při doručování událostí je indikátor pod Linuxem o několik milisekund opožděn. Pro příští verzi knihovny je plánováno přidání možnosti zanalyzovat najednou úrovně hlasitosti v celém souboru. Pro vyhledání špiček v souboru tedy nebude nutné během přehrávání sledovat peak meter.

Pro usnadnění práce jsou k dispozici čtyři klávesové zkratky. Jsou definovány pomocí *AcceleratorTable*, díky které nezáleží na tom, která komponenta okna je ve chvíli stisku aktivní. Ctrl+C přesune aktuální čas posuvníku do pole Split point, což je vhodné pro hrubé určení místa začátku intervalu během přehrávání. Ctrl+V přenese čas z pole Split point do aktuálního času posuvníku, přičemž od něj odečte několik vteřin. To se může hodit pro poslech začátku intervalu (změny). Poslední dvě zkratky Ctrl+D a Ctrl+F posouvají čas v poli Split point o jeden audio rámec dozadu a dopředu.

Závěr

V práci byl popsán standard MPEG-1 a detailněji rozebrána jeho audio část. Dále jsme si ukázali strukturu MP3 souboru a nahlédli do spletitých zákoutí dekódovacího procesu. Tento proces je velmi komplikovaný, ale dosahuje výborných kompresních výsledků při zachování v podstatě stejné audio kvality. K dosažení takových výsledků je nutné zkombinovat ztrátovou i bezztrátovou kompresi, protože sama není ani jedna schopna docílit požadovaného kompresního poměru.

Požadavky zadání se, i přes několik obtíží, podařilo zcela splnit a knihovna nabízí základní bezztrátové operace řez a změna hlasitosti. Podporuje však pouze třetí vrstvu MPEGu a proto se ve výjimečných situacích může stát, že některý soubor nepůjde otevřít, protože bude i přes svou příponu značící třetí vrstvu uložen ve vrstvě jiné. Oproti zadání byl do práce navíc zabudován dekodér a je tedy možné přehrát soubor v rozpracovaném stavu bez nutnosti ukládat mezivýsledky a spouštět je v externím přehrávači. Nejedná se však o vlastní kód, ale o veřejnou verzi kódu vydanou Fraunhoferovým Institutem, jenž byl zvolen hlavně kvůli své jednoduchosti.

Aktuální verze knihovny nese hrdě označení 1.0. Ve fázi návrhu je hned několik rozšíření a vylepšení do dalších verzí. Za všechny jmenujme např. lepší podporu tagů a jejich editaci, možnost zanalyzovat úrovně hlasitosti v souboru, či zachytávat online streamy a ještě před uložením měnit některé jejich klíčové parametry.

Seznam použitých zdrojů

1. **Ruckert, Martin.** *Understanding MP3.* Wiesbaden : Vieweg & Son Verlag/GWV Fachverlage GmbH, 2005. stránky 155-158. ISBN 3-528-05905-2.

2. **Bender, Ryan.** Example: Huffman Encoding Trees. *The MIT Press.* [Online] 17. duben 2000. [Citace: 19. červenec 2011.] http://mitpress.mit.edu/sicp/full-text/sicp/book/node41.html.

3. **Ruckert, Martin.** *Understanding MP3.* Wiesbaden : Vieweg & Son Verlag/GWV Fachverlage GmbH, 2005. stránky 17-22. ISBN 3-528-05905-2.

4. **Skarnitzl, Radek.** Psychoakustika. *Fonetický ústav Filozofické fakulty Univerzity Karlovy.* [Online] 2011. [Citace: 14. červenec 2011.] http://fu.ff.cuni.cz/vyuka/akustika/3_psychoakustika.pdf.

5. **Heijligers, Marc.** A Short Introduction to Audio Encoding. *An evaluation of AAC and MP3 compression.* [Online] 2006. [Citace: 14. červenec 2011.] http://homepage.mac.com/marc.heijligers/audio/ipod/compression/encoding/files/page10_1.gif.

6. **Heijligers, Marc.** A Short Introduction to Audio Encoding. *An evaluation of AAC and MP3 compression.* [Online] 2006. [Citace: 14. červenec 2011.] http://homepage.mac.com/marc.heijligers/audio/ipod/compression/encoding/files/page10_2.gif.

7. **Mareš, Martin.** Zápis přednášky z 27.11.2009: Fourierova transformace. *Algoritmy a datové struktury II.* [Online] [Citace: 21. červenec 2011.] http://mj.ucw.cz/vyuka/0910/ads2/9-fft.pdf.

8. **Moving Picture Experts Group.** ISO/IEC 11172 - Coding Of Moving Pictures And Associated Audio For Digital Storage Media At Up To About 1,5 Mbit/s - Part 3: Audio. *MP3' Tech.* [Online] 22. prosinec 1991. [Citace: 27. říjen 2008.] http://www.mp3-tech.org/programmer/docs/iso11172-3.zip.

9. **Duque, Alexander Vargas y Alejandro.** CODIFICADOR MP3 EN MATLAB. [Online] [Citace: 8. červenec 2011.] http://members.fortunecity.com/alex1944/mp3coding/modobloq.gif.

10. **Bosse, Lincoln.** Center for Computer Research in Music and Acoustics. *Modified Discrete Cosine Transform (MDCT).* [Online] 7. duben 1998. [Citace: 23. červenec 2011.] https://ccrma.stanford.edu/~bosse/proj/node27.html.

11. **Ruckert, Martin.** *Understanding MP3.* Wiesbaden : Vieweg & Son Verlag/GWV Fachverlage GmbH, 2005. stránky 151-152. ISBN 3-528-05905-2.

12. **Warren, Henry S.** Write-up: Cyclic Redundancy Check. *Hacker's Delight.* [Online] 2003. [Citace: 15. červenec 2011.] http://www.hackersdelight.org/crc.pdf. ISBN 978-0201914658.

13. **Moving Picture Experts Group.** ISO/IEC 11172 - Coding Of Moving Picture And Associated Audio For Digital Storage Media At Up To About 1,5 Mbit/s - Part 3: Audio. *MP3' Tech.* [Online] 22. prosinec 1991. [Citace: 27. říjen 2008.] http://www.mp3-tech.org/programmer/docs/iso11172-3.zip. soubor ANNEX_AB.DOC stránka 6.

14. **Moving Picture Experts Group.** ISO/IEC 11172 - Coding Of Moving Pictures And Associated Audio For Digital Storage Media At Up To About 1,5 Mbit/s - Part 3: Audio. *MP3' Tech.* [Online] 22. prosinec 1991. [Citace: 27. říjen 2008.] http://www.mp3-tech.org/programmer/docs/iso11172-3.zip. soubor MPGAUDIO.DOC stránka 38.

15. **Fraunhofer, IIS.** Decoding engines source codes. *MP3'Tech.* [Online] [Citace: 16. prosinec 2008.] http://www.mp3-tech.org/programmer/sources/mpeg1_iis.tgz.

16. *Simplified Wrapper and Interface Generator.* [Online] [Citace: 15. září 2010.] http://www.swig.org/.

17. **Nilsson, Martin.** id3v2.3.0. *ID3.org.* [Online] 1999. [Citace: 16. duben 2009.] http://www.id3.org/id3v2.3.0.

18. **Moving Picture Experts Group.** ISO/IEC 11172 - Coding Of Moving Picture And Associated Audio For Digital Storage Media At Up To About 1,5 Mbit/s - Part 3: Audio. *MP3' Tech.* [Online] 22. prosinec 1991. [Citace: 27. říjen 2008.] http://www.mp3-tech.org/programmer/docs/iso11172-3.zip. soubor ANNEX_AB.DOC stránka 5.

19. **O'Neil, Dan.** mp3Frame. *ID3.org.* [Online] 2006. [Citace: 7. červenec 2011.] http://www.id3.org/mp3Frame?action=AttachFile&do=get&target=mp3frame_blocks.gif.

Seznam tabulek

Seznam obrázků

Dodatek A: Obsah přiloženého CD

Struktura disku se skládá z následujících adresářů:

- **sources**: obsahuje zdrojové kódy knihovny, grafického rozhraní i utilit
- **texts**: obsahuje text této práce ve formátu pdf a docx
- **windows**: obsahuje instalační soubory Pythonu, WXPythonu a přeložené soubory jak utilit, tak celé knihovny, coby modulu pro Python.

Dodatek B: Dokumentace

V této kapitole si popíšeme rozhraní knihovny. Obecně u všech metod platí, že pokud vrací hodnotu typu *int*, pak nula (odpovídající konstantě *E_OK*) značí správně ukončené vykonávání. Kladná hodnota znamená, že metoda proběhla správně a vrací výsledek, kdežto záporná hodnota značí, že došlo k chybě a jedná se o chybový kód.

`int state_init(void)`

Inicializuje vnitřní stav knihovny na výchozí hodnoty. Je potřeba zavolat před začátkem práce s knihovnou.

`int add_file(char *filename)`

Přidá do knihovny nový soubor, provede jeho analýzu a vytvoří nový interval pokrývající celý soubor, a přidá ho na konec seznamu intervalů.

Parametr *filename* určuje název souboru, který má být otevřen.

Vrací chybový kód, pokud došlo k chybě při otevření nebo analýze souboru, jinak *E_OK*.

`int open_current_file(void)`

Otevře aktuální soubor pro zpracování a inicializuje vnitřní datovou strukturu. Je potřeba zavolat před začátkem každé práce se soubory (přehrávání, uložení).

Vrací chybový kód, pokud došlo k chybě při otevření souboru, jinak *E_OK*.

`int close_current_file(void)`

Zavře aktuálně otevřený soubor. Je potřeba zavolat po ukončení každé práce se soubory (přehrávání a uložení).

Vrací chybový kód, pokud došlo k chybě při zavření souboru, jinak *E_OK*.

`int set_output_file(char *filename)`

Nastaví zadaný soubor jako výstupní a otevře ho. Je potřeba zavolat před každým uložením souboru.

Parametr *filename* určuje, jak se má výstupní soubor jmenovat.

Vrací chybový kód, pokud došlo k chybě při otevření souboru, jinak *E_OK*.

`int close_output_file(void)`

Zavře výstupní soubor. Je potřeba zavolat po každém uložení souboru.

Vrací chybový kód, pokud došlo k chybě při zavření souboru, jinak *E_OK*.

`int split_interval(int interval_id, int frame_where)`

Rozdělí interval na zadaném místě na dva. Nový interval je zařazen za starý, nemá žádnou akci a začíná od následujícího rámce. Dojde také k přečíslování všech následujících intervalů.

Parametr *interval_id* identifikuje interval k rozdělení a parametr *frame_where* určuje, kde se má interval rozdělit, přičemž tento rámec bude ponechán ve starém intervalu.

Vrací chybový kód, pokud je identifikace intervalu neplatná nebo pokud je dělící rámec mimo rozsah intervalu, jinak *E_OK*.

`int merge_with_next_interval(int interval_id)`

Sloučí zvolený interval s následujícím intervalem a přečísluje všechny následující intervaly.

Parametr *interval_id* identifikuje interval, který chceme sloučit s následujícím.

Vrací chybový kód, pokud je identifikace intervalu neplatná a pokud následující interval neexistuje nebo neleží přímo za aktuálním či není ze stejného souboru, jinak *E_OK*.

`int merge_with_previous_interval(int interval_id)`

Sloučí zvolený interval s předchozím intervalem a přečísluje všechny následující intervaly.

Parametr *interval_id* identifikuje interval, který chceme sloučit s předchozím.

Vrací chybový kód, pokud je identifikace intervalu neplatná a pokud předchozí interval neexistuje nebo neleží přímo před aktuálním či není ze stejného souboru, jinak *E_OK*.

`int move_interval_up(int interval_id)`

Přehodí interval s předchozím. Pokud předchozí interval neexistuje, nic se nestane. Dojde také k záměně id intervalů.

Parametr *interval_id* identifikuje interval, který chceme přesouvat nahoru.

Vrací chybový kód, pokud je identifikace intervalu neplatná, jinak *E_OK*.

`int move_interval_down(int interval_id)`

Přehodí interval s následujícím. Pokud následující interval neexistuje, nic se nestane. Dojde také k záměně id intervalů.

Parametr *interval_id* identifikuje interval, který chceme přesouvat dolů.

Vrací chybový kód, pokud je identifikace intervalu neplatná, jinak *E_OK*.

`int move_upper_bound(int interval_id, int new_frame_to)`

Posune horní hranici prvního intervalu na zadaný rámec. Pokud na něj přímo navazuje následující interval (tzn., že následující začíná o jeden rámec dále, než první interval končí a oba jsou ve stejném souboru), bude spodní mez následujícího intervalu posunuta na rámec *new_frame_to* + 1.

Parametr *interval_id* identifikuje interval, se kterým chceme pracovat, a parametr *new_frame_to* pak rámec, na který se má horní mez posunout.

Vrací chybový kód, pokud je identifikace intervalu neplatná, pokud je nová mez menší než spodní mez prvního intervalu, nebo pokud je mimo hranice souboru, anebo pokud je nová mez větší než hodní mez následujícího intervalu, pokud tento přímo navazuje, jinak *E_OK*.

`int move_lower_bound(int interval_id, int new_frame_from)`

Posune dolní hranici intervalu za zadaný rámec. Pokud přímo navazuje na interval předchozí (tzn., že začíná na následujícím rámci, než předchozí interval končí a oba jsou ve stejném souboru), bude horní mez předchozího posunuta na rámec *new_frame_from* - 1.

Parametr *interval_id* identifikuje interval, se kterým chceme pracovat a parametr *new_frame_from* pak rámec na který se má dolní mez posunout.

Vrací chybový kód, pokud je identifikace intervalu neplatná, pokud je nová mez větší než horní mez zvoleného, nebo pokud je mimo hranice souboru, anebo pokud je nová mez menší než dolní mez předchozího intervalu, pokud tento přímo navazuje, jinak *E_OK*.

`int duplicate_interval(int interval_id)`

Duplikuje zvolný interval a zařadí ho za zvolený interval. Nový interval má id o jedna vyšší a všechny následující intervaly jsou přečíslovány.

Parametr *interval_id* identifikuje interval, který chceme duplikovat.

Vrací chybový kód, pokud je identifikace intervalu neplatná, jinak *E_OK*.

`int delete_interval(int interval_id)`

Smaže zvolený interval a přečísluje následující intervaly.

Parametr *interval_id* identifikuje interval, který chceme smazat.

Vrací chybový kód, pokud je identifikace intervalu neplatná, jinak *E_OK*.

`int set_action(int interval_id, int mode, int volume_step)`

Nastaví zvolenému intervalu některou z akcí. Není kontrolována správnost ani rozsah čísla určujícího akci. Pokud se ale jedná o akci odpovídající konstantě *MODE_VOLUME*, je intervalu nastaven krok změny.

Parametr *interval_id* identifikuje interval, kterému chceme nastavit akci, parametr *mode* určuje, jaká akce se má nastavit, a parametr *volume_step* určuje, o kolik se má měnit hlasitost, v případě, že akcí je změna hlasitosti. Při jiných akcích nemá poslední parametr žádný význam.

Vrací chybový kód, pokud je identifikace intervalu neplatná, jinak *E_OK*.

`int get_number_intervals(void)`

Spočítá celkový počet intervalů.

Vrací počet intervalů, nikdy chybový kód.

`int get_current_interval_length(void)`

Spočítá délku aktuálního intervalu v rámcích.

Vrací chybový kód, pokud není zvolen žádný interval jako aktuální, jinak vrací délku intervalu v rámcích.

`int get_current_interval_start(void)`

Zjistí, na jakém rámci začíná aktuální interval.

Vrací chybový kód, pokud není zvolen žádný interval jako aktuální, jinak vrací rámec, na kterém začíná aktuální interval.

`int get_current_interval_end(void)`

Zjistí, na jakém rámci končí aktuální interval.

Vrací chybový kód, pokud není zvolen žádný interval jako aktuální, jinak vrací rámec, na kterém končí aktuální interval.

`int get_current_interval_number(void)`

Zjistí, jaký interval je zvolen jako aktuální.

Vrací chybový kód, pokud není zvolen žádný interval jako aktuální, jinak vrací id aktuálního intervalu.

`int set_current_interval(int interval_id)`

Nastaví zvolený interval jako aktuální a jako aktuální nastaví také soubor, ke kterému se interval vztahuje.

Parametr *interval_id* identifikuje interval, který chceme nastavit jako aktuální.

Vrací chybový kód, pokud je identifikace intervalu neplatná, jinak *E_OK*.

`char *interval_as_string(int interval_id)`

Vytvoří textovou reprezentaci zadaného intervalu. Textová reprezentace vypadá následovně: "číslo_intervalu. akce_a_její_parametr, čas_od – čas_do, název_souboru". Paměť obsazenou tímto řetězcem je explicitně potřeba uvolnit.

Parametr *interval_id* identifikuje interval, jehož reprezentace nás zajímá.

Vrací *NULL*, pokud je identifikace intervalu neplatná, jinak vrací pointer na textovou reprezentaci intervalu.

`int get_save_start(void)`

Zjistí, odkud má začít proces ukládání.

Vrací číslo prvního rámce aktuálního intervalu.

`char *help(int error_code)`

Zjistí, jaký textový popis odpovídá chybovému kódu.

Parametr *error_code* reprezentuje chybový kód.

Vrací pointer na řetězec obsahující textový popis chybového kódu.

`int write_tag_and_xing(void)`

Zapíše do aktuálního výstupního souboru ID3v2 tag a XING rámec z prvního intervalu, pokud je příslušný soubor obsahuje. Po ukončení operace je opět vstupní soubor zavřen.

Vrací chybový kód, pokud se nepodaří soubor otevřít či uzavřít, jinak *E_OK*.

`int write_id3v1_tag()`

Zapíše do aktuálního výstupního souboru ID3v1 tag z prvního intervalu, pokud ho příslušný soubor obsahuje. Po ukončení operace je opět vstupní soubor zavřen.

Vrací chybový kód, pokud se nepodaří soubor otevřít či uzavřít, jinak *E_OK*.

`int cast_time_to_frame(char *time)`

Převede časový údaj na číslo nejbližšího rámce. Je možné zadat B (anglicky begin) pro substituci začátku nebo E (anglicky end) pro substituci konce souboru.

Parametr *time* obsahuje řetězec ve formátu H:MM:SS.MS nebo M:SS.MS.

Vrací chybový kód, pokud je čas ve špatném formátu, pokud jsou minuty či sekundy větší než 59 nebo menší než nula, pokud jsou milisekundy větší než 999 nebo menší než nula, jinak vrací číslo nejbližšího rámce.

`char *cast_frame_to_time(int frame)`

Převede číslo rámce na čas, na kterém rámec začíná, rámce jsou číslovány od jedničky.

Parametr *frame* obsahuje číslo rámce.

Vrací textovou reprezentaci času, na kterém rámec začíná. Formátovací řetězec použitý pro výstup vypadá následovně: „%d:%02d:%02d.%03d“ v pořadí pro hodiny, minuty, vteřiny a milisekundy. Pokud čas není delší než hodina, je „%d:“ pro hodiny vynecháno.

`int get_currents_bitrate_sum(void)`

Zjistí součet datového toku všech rámců v aktuálním souboru.

Vrací chybový kód, pokud není žádný soubor nastaven jako aktuální, jinak vrací součet datových toků všech rámců.

`int get_currents_frame_number(void)`

Zjistí číslo aktuálního zpracovávaného rámce.

Vrací chybový kód, pokud není žádný soubor nastaven jako aktuální, jinak vrací číslo aktuálního zpracovávaného rámce.

`int get_current_file_frame_length(void)`

Zjistí počet rámců v aktuálním souboru.

Vrací chybový kód, pokud není žádný soubor nastaven jako aktuální, jinak vrací počet rámců v aktuálním souboru.

`int get_current_file_byte_length(void)`

Zjistí načtenou délku aktuálního souboru.

Vrací chybový kód, pokud není žádný soubor nastaven jako aktuální, jinak vrací počet bytů přečtených z aktuálního souboru.

`bool is_cbr(void)`

Zjistí, jaký datový tok soubor obsahuje.

Vrací *true*, pokud se jedná o konstantní datový tok, *false* pokud se jedná o variabilní.

`int set_cbr(void)`

Spočítá průměrný datový tok všech souborů a zjistí, zda ve výsledném souboru bude konstantní či variabilní. Tento údaj je uložen do vnitřního stavu knihovny pro správné rozlišení při zápisu XINGu. Měla by být proto zavolána před ukládáním XINGu a před voláním metody `is_cbr`.

Vrací chybový kód, pokud není žádný soubor nastaven jako aktuální, jinak *E_OK*.

`bool is_cbr(void)`

Zjistí, zda bude výsledný soubor obsahovat konstantní nebo variabilní datový tok.

Vrací *true*, pokud mají všechny soubory stejný a konstantní datový tok, jinak *false*.

`void set_play(bool b)`

Nastaví ve vnitřním stavu knihovny příznak říkající, zda se mají rámce při průchodu soubory přehrávat.

Parametr *b* určuje, zda zda aktivovat režim přehrávání či nikoliv.

`int initialize_soundcard(void)`

Inicializuje zvukovou kartu. Měla by být zavolána před každým přehráváním.

Vrací chybový kód, pokud se nepodaří nastavit a otevřít zvukovou kartu pro přehrávání, jinak *E_OK*.

`void uninitialize_soundcard(void)`

Odinicializuje zvukovou kartu a proměnné související s přehráváním, včetně všech bufferů zvukové karty (nejedná se o buffery uvolněné pomocí metody `clear_soubdcard_buffers`). Měla by být zavolána po každém přehrávání.

`void clear_soubdcard_buffers(void)`

Vyprázdní interní buffery používané při přehrávání (nejedná se o zvukové buffery uvolněné metodou `uninitialize_soundcard`). Měla by být spuštěna po každém přehrávání.

`void set_starts_working(bool b)`

Uloží příznak o tom, zda knihovna začíná zpracovávat první soubor. V takovém případě je potřeba seeknout v aktuálním souboru na příslušnou pozici. Je potřeba nastavit před každým začátkem zpracovávání.

Parametr *b* určuje, začíná práce s intervalem či nikoliv.

`int get_l_volume_level(void)`

Zjistí aktuální úroveň hlasitosti pro levý kanál.

Vrací nulu, pokud neprobíhá přehrávání, jinak vrací aktuální úroveň hlasitosti.

`int get_r_volume_level(void)`

Zjistí aktuální úroveň hlasitosti pro pravý kanál.

Vrací nulu, pokud neprobíhá přehrávání, jinak vrací aktuální úroveň hlasitosti.

`t_state *get_state(void)`

Vrací pointer na vnitřní stav knihovny.

`int do_file(int frame)`

Je hlavní metodou celé knihovny a zajišťuje přehrávání a ukládání souborů. Ze stavových proměnných, jejichž nastavování je popsáno výše, zjistí, zda se jedná o přehrávání či nikoliv a zpracuje odpovídajícím způsobem následující rámec aktuálního souboru. Samotné fungování metody se mění pouze podle stavových parametrů a je potřeba mít všechny důležité parametry správně nastaveny.

Zamýšlené použití je takové, že metoda dostane jako parametr rámec v aktuálním intervalu, od kterého má začít (při ukládání je to první rámec aktu-

álního intervalu – zde je potřeba, aby byl jako aktuální nastaven první interval ze seznamu intervalů; při přehrávání je potřeba za aktuální zvolit interval, od kterého chceme přehrávat a určit v jeho rozsahu počáteční rámec), načež tento rámec zpracuje a v případě, že nedojde k nějaké chybě, vrátí číslo následujícího rámce, který se má zpracovat. S tím by měla být metoda opět zavolána, dokud nevrátí konstantu *E_FILE_ENDED*.

V rámci jednoho intervalu je vraceno číslo následujícího fyzického rámce, při přechodu mezi intervaly je automaticky zakomponován spojovací rámec, uzavřen aktuální soubor, změněn aktuální interval na následující a otevřen příslušný soubor, který je nastaven jako nový aktuální, přičemž dojde i k případnému seeku na pozici začátku intervalu.

Parametr *frame* tedy určuje číslo rámce, který se má zpracovat.

Vrací chybový kód, pokud není nastaven aktuální interval, pokud soubor skončil nebo pokud při přehrávání nebyl nalezen soubor “huffdec“ anebo zvuková karta nezpracovala korektně audio data, jinak je vráceno číslo dalšího rámce, který se má zpracovat.