LilyPond

A kottaszedő program

Használat

A LilyPond fejlesztőcsapata

Ez a dokumentáció ismerteti, hogyan kell a LilyPond 2.18.2 verziójához tartozó programokat futtatni, valamint tanácsokat ad azok hatékony használatához.

A teljes dokumentáció a http://www.lilypond.org/ honlapon található.

Copyright © 1999–2012 a szerzők.

Ezt a dokumentumot a GNU Free Documentation License (GNU Szabad Dokumentációs Licenc) 1.1-es vagy frissebb, a Free Software Foundation (Szabad Szoftver Alapítvány) által kiadott verziójának megfelelően lehet másolni, terjeszteni és/vagy módosítani, nem változtatható szakaszok nélkül. A licenc másolata a „GNU Free Documentation License” című függelékben található.

A LilyPond 2.18.2 verziójához

Tartalomjegyzék

1 A `lilypond` használata ... 1

1.1 Egyszerű használat ... 1
1.2 Parancssori használat ... 1
A `lilypond` futtatása ... 1
A `lilypond` parancssori paraméterei ... 1
Környezeti változók ... 5
1.3 Hibaüzenetek ... 5
1.4 Gyakori hibák ... 6
A kotta nem fér ki az oldalra ... 6
Egy kottasorral több van a kelleténél ... 6
Hiba a `../ly/init.ly` fájlban ... 7
`Unbound variable %` hibaüzenet ... 8
`FT_Get_Glyph_Name` hibaüzenet ... 8

2 A `convert-ly` használata ... 9

2.1 Miért változik a szintaxis? ... 9
2.2 A `convert-ly` futtatása ... 9
2.3 A `convert-ly` parancssori paraméterei ... 10
2.4 Problémák a `convert-ly` futtatása közben ... 10
2.5 Kézi frissítés ... 10

3 A `lilypond-book` használata ... 12

3.1 Egy kottapéldákat tartalmazó dokumentum ... 12
3.2 Zene és szöveg integrációja ... 15
3.2.1 LaTeX ... 15
3.2.2 Texinfo ... 16
3.2.3 HTML ... 17
3.2.4 DocBook ... 18
3.3 Kottapéldák paraméterei ... 18
3.4 A `lilypond-book` futtatása ... 21
3.5 Fájlkiterjesztések ... 23
3.6 lilypond-book sablonok ... 23
3.6.1 LaTeX ... 24
3.6.2 Texinfo ... 24
3.6.3 html ... 24
3.6.4 xelatex ... 25
3.7 Közös tartalomjegyzék ... 26
3.8 További módszerek zene és szöveg kombinálására ... 27

4 External programs ... 28

4.1 Point and click ... 28
4.1.1 Configuring the system ... 28
Using Xpdf ... 28
Using GNOME 2 ... 28
Using GNOME 3 ... 29
Extra configuration for Evince ... 29
Enabling point and click ... 29

Selective point-and-click 30
4.2 Text editor support 30
Emacs mode 30
Vim mode 31
Other editors 31
4.3 Converting from other formats 31
4.3.1 Invoking `midi2ly` 31
4.3.2 Invoking `musicxml2ly` 32
4.3.3 Invoking `abc2ly` 33
4.3.4 Invoking `etf2ly` 34
4.3.5 Other formats 35
4.4 LilyPond output in other programs 35
Many quotes from a large score 35
Inserting LilyPond output into OpenOffice and LibreOffice 35
Inserting LilyPond output into other programs 35
4.5 Independent `includes` 36
4.5.1 MIDI articulation 36

5 Suggestions for writing files 37
5.1 General suggestions 37
5.2 Typesetting existing music 38
5.3 Large projects 38
5.4 Troubleshooting 39
5.5 Make and Makefiles 39

függelék A GNU Free Documentation License 46

függelék B LilyPond index 53

1 A lilypond használata

Ez a fejezet a LilyPond használatának technikai vonzatait részletezi.

1.1 Egyszerű használat

A legtöbb felhasználó grafikus felületről indítja a LilyPondot; ennek módját az rész "Első lecke" in *Tankönyv* írja le. Kényelmi szolgáltatásokat nyújtó szövegszerkesztők használatának leírása a saját dokumentációjukban található.

1.2 Parancssori használat

Ez a szakasz a LilyPond parancssori futtatásáról tartalmaz plusz információkat, arra az esetre, ha a programnak plusz paramétereket szeretnénk átadni. Ráadásul bizonyos segédprogramok (mint pl. a `midi2ly`) csak parancssorból érhetőek el.

Parancssor alatt az operációs rendszer megfelelő parancssorát értjük. A Windows-felhasználók ezt „DOS-parancssor" néven, a Mac OS X felhasználók „Terminal" néven ismerhetik.

Az operációs rendszer parancssorának használatának leírása kívül esik a LilyPond dokumentációjának hatáskörén; az ebben kevésbé járatos felhasználók az operációs rendszerhez tartozó dokumentációban olvashatnak erről.

A lilypond futtatása

A `lilypond` program a következő módon futtatható parancssorból:

```
lilypond [opció]... fájlnév...
```

Ha nem adunk meg kiterjesztést, az alapértelmezett '`.ly`' kiterjesztéssel próbálkozik a LilyPond. A szabványos bemenetről való beolvasáshoz a `-` karakter használandó *fájlnév* gyanánt.

Amikor a '`fájlnév.ly`' fájl feldolgozásra kerül, egy '`fájlnév.ps`' és egy '`fájlnév.pdf`' fájlt kapunk kimenetként. Több fájlt is feldolgoztathatunk egyszerre; ezek egymástól függetlenül kerülnek feldolgozásra.[1]

Ha a '`fájlnév.ly`' több `\book` blokkot tartalmaz, minden blokkból egy-egy, számozott kimeneti fájl keletkezik, '`fájlnév.pdf`', '`fájlnév-1.pdf`', '`fájlnév-2.pdf`' stb. formában. Az `output-suffix` változó értéke fog szerepelni a fájlnév és a számozás között. Például a következő bemeneti fájlból:

```
#(define output-suffix "violino")
\score { ... }
#(define output-suffix "cello")
\score { ... }
```

egy '*fájlnév*`-violino.pdf`' és egy '*fájlnév*`-cello-1.pdf`' nevű fájl keletkezik.

A lilypond parancssori paraméterei

A következő parancssori opciók támogatottak:

`-e, --evaluate=`*kifejezés*

A Scheme *kifejezés* kiértékelése az '`.ly`' fájlok beolvasása előtt. Több `-e` opció is megadható, ezek a megadott sorrendben lesznek végrehajtva.

A kifejezés kiértékelése a `guile-user` modulban történik, így ha definíciókat kell használni a *kifejezés*ben, a parancssorban a következőt kell megadni:

[1] A GUILE megelőző állapota nem áll vissza feldolgozás után, így elővigyázatosnak kell lenni, hogy ne változtassuk meg a rendszer alapbeállításait Scheme kódból.

```
lilypond -e '(define-public a 42)'
```

a forrásfájl elejére pedig a következőt kell beszúrni:

```
#(use-modules (guile-user))
```

`-f, --format=`*formátum*
: A kimenet formátuma. Lehetőségek: `ps`, `pdf` vagy `png`.

  Példa: `lilypond -fpng` *fájlnév*`.ly`

`-d, --define-default=`*azonosító*`=`*érték*
: Az *azonosító* nevű belső változó beállítása az *érték* Scheme értékre. Ha az *érték* nincs megadva, az alapértelmezett `#t` lesz a változó értéke. Egy opció kikapcsolásához a `no-` prefixumot kell az azonosító elé írni, pl.

  ```
  -dno-point-and-click
  ```

  ugyanaz, mint

  ```
  -dpoint-and-click='#f'
  ```

  Íme pár hasznos opció:

  ‘`help`’
  : A `lilypond -dhelp` parancs futtatása kilistázza az összes elérhető `-d` opciót.

  ‘`paper-size`’
  : Az alapértelmezett papírméret beállítása.

    ```
    -dpaper-size=\"letter\"
    ```

    Ügyelni kell arra, hogy a méretet `\"` jelek közé írjuk.

  ‘`safe`’
  : A LilyPond futtatása biztonsági módban, megbízhatatlan bemenet esetén.

    Amikor a LilyPond egy webszerveren keresztül érhető el, vagy a `-dsafe`, vagy a `--jail` opciót **MINDENKÉPPEN KÖTELEZŐ** megadni. A `-dsafe` opcióval megelőzhető, hogy a forrásfájlban szereplő rosszindulatú Scheme kód kárt okozzon. Például:

    ```
    #(system "rm -rf /")
    {
      c4^$(ly:gulp-file "/etc/passwd")
    }
    ```

    `-dsafe` módban a Scheme kifejezések kiértékelése egy speciális biztonsági modulban történik. Ez a modul a GUILE ‘`safe-r5rs`’ modulján alapul, de a LilyPond API néhány függvényének meghívását lehetővé teszi. Ezek a függvények a ‘`scm/safe-lily.scm`’ fájlban találhatóak.

    Ezenkívül biztonsági módban tilos az `\include` parancsok alkalmazása és a \ karakter használata TEX karakterláncokban.

    Biztonsági módban ezenfelül nem lehetséges LilyPond változók importálása Scheme-be.

    A `-dsafe` mód *nem* figyeli az erőforrások túlzott használatát. Továbbra is elérhető, hogy a program tetszőlegesen hosszú ideig fusson, például ciklikus adatstruktúrák használatával. Így ha a LilyPond publikus webszerveren fut, a folyamat processzor- és memóriafelhasználását korlátozni kell!

    Biztonsági módban sok hasznos LilyPond kódrészlet nem fog lefordulni. A `--jail` mód egy több lehetőséget biztosító alternatíva, de előkészítése több munkát igényel.

‘`backend`’ A szedés kimeneti formátuma. Lehetőségek:

`ps` PostScript.

A PostScript fájlok teljes egészükben tartalmazzák a megjelenítéshez szükséges TTF, Type1 és OTF betűkészleteket. Keleti karakterkészletek használata esetén ez nagy fájlokhoz vezethet.

`eps`

Encapsulated PostScript.

Minden oldal külön ‘`EPS`’ fájlba kerül, betűtípusok nélkül, valamint egy összesített ‘`EPS`’ fájl is létrejön, amely az összes oldalt tartalmazza betűtípusokkal együtt.

A `lilypond-book` alapértelmezetten ezt a formátumot használja.

`svg`

SVG (Scalable Vector Graphics).

Oldalanként egy SVG fájl keletkezik, beágyazott betűtípusok nélkül. Így megtekintésükhöz érdemes feltelepíteni a Century Schoolbook betűtípusokat. Ezeket tartalmazza a LilyPond. Például UNIX alatt egyszerűen csak be kell másolni ezeket a program könyvtárából (tipikusan ‘`/usr/share/lilypond/VERZIÓ/fonts/otf/`’) a ‘`~/.fonts/`’ könyvtárba. Az SVG kimenet szabványos, így bármilyen, ezt a formátumot olvasni képes programmal megnyitható.

`scm`

A belső Scheme rajzolóparancsok szó szerinti kiírása.

`null`

Nincs kimenet; ugyanaz a hatása, mint a `-dno-print-pages` opciónak.

Példa: `lilypond -dbackend=svg` *`fájlnév`*`.ly`

‘`preview`’

A fejléc és az első szisztéma fog szerepelni a kimenetben.

‘`print-pages`’

Teljes oldalak generálása, ez az alapbeállítás. A `-dno-print-pages` opció a `-dpreview` opcióval együtt hasznos.

`-h, --help`

Összegzés az alkalmazás használatáról.

`-H, --header=`*`mező`*

A megadott fejlécmező kiírása a ‘*`fájlnév.mező`*’ nevű fájlba.

`--include, -I=`*`könyvtár`*

A *könyvtár* hozzáadása a bemeneti fájlok keresési útvonalához.

`-i, --init=`*`fájl`*

Az inicializáló fájl beállítása a megadott *fájl*ra. (Alapértelmezett: ‘`init.ly`’.)

`-o, --output=`*fájl*
: Kimeneti fájl megadása. A megfelelő kiterjesztés automatikusan hozzáfűzésre kerül (pl. `.pdf` PDF kimenet esetén).

`--ps`
: PostScript kimenet generálása.

`--png`
: Oldalanként egy-egy PNG kép létrehozása. Ez a `--ps` opció hatását vonja maga után. A kép DPI-ben mért felbontása (alapértelmezett értéke 110) a következőképpen állítható be:

```
-dresolution=110
```

`--pdf`
: PDF generálása. A `--ps` opció hatását vonja maga után.

`-j, --jail=`*felhasználó*`,`*csoport*`,`*börtön*`,`*könyvtár*
: A `lilypond` futtatása ún. börtönben.

  A `--jail` opció egy rugalmasabb alternatíva a `-dsafe` módnál abban az esetben, amikor a LilyPond forrás megbízhatatlan forrásból származik, pl. amikor webszerveren keresztül érhető el a LilyPond szolgáltatásként.

  A `--jail` módban a `lilypond` gyökere a *börtön* lesz, mielőtt a fordítási folyamat elkezdődne. Ezután a LilyPond átvált a megadott felhasználóra, csoportra és könyvtárba. Ezáltal garantálható, hogy (legalábbis elméletben) lehetetlen kitörni a börtönből. A `--jail` mód csak akkor működik, ha a `lilypond` alkalmazást root felhasználóként futtatjuk. Ez általában biztonságosan történik, pl. a `sudo` parancs használatával.

  A börtön előkészítése egy bonyolult folyamat, mivel biztosítani kell, hogy a LilyPond *a börtönben* mindent megtaláljon, ami a fordításhoz szükséges. Egy tipikus előkészítés a következő lépésekből áll:

  Különálló fájlrendszer létrehozása
  : A LilyPond számára létre kell hozni egy fájlrendszert, amelyet a biztonságos `noexec`, `nodev` és `nosuid` opciókkal tudunk felcsatolni. Így lehetetlen a LilyPondból programokat futtatni vagy közvetlenül eszközökre írni. Ha egy külön partíció létrehozása nem kívánatos, egy elegendően nagy fájl létrehozása és loop eszközként való használata is megfelelő. A külön fájlrendszer azt is megelőzi, hogy a LilyPond többet írjon a lemezre, mint amennyi megengedett.

  Különálló felhasználó létrehozása
  : Egy, kevés jogosultsággal rendelkező (pl. `lily/lily` nevű) felhasználó és csoport nevében kell, hogy fusson a LilyPond. Ennek a felhasználónak csak egy könyvtárhoz lehet írási joga, amit a *könyvtár* paraméterben kell megadni.

  A börtön előkészítése
  : A LilyPond futásához szükséges összes fájlt be kell másolni a börtönbe, megtartva az eredeti elérési utakat. Az egész LilyPond telepítés (pl. a '`/usr/share/lilypond`' könyvtár tartalmának) másolása szükséges.

    Ha mégis probléma lépne fel, a forrását legegyszerűbben az `strace` paranccsal határolhatjuk be, amellyel meghatározható, hogy mely fájlok hiányoznak.

  A LilyPond futtatása
  : A `noexec` kapcsolóval csatolt börtönben lehetetlen külső programot futtatni. Így csak olyan kimeneti formátumok érhetőek el, amelyek ezt nem igénylik. Mint már említettük, superuser privilégiumokkal

kell futtatni a LilyPondot (amelyeket természetesen egyből elveszít), lehetőleg `sudo` használatával. Ajánlott a LilyPond által elfoglalt processzoridő korlátozása (pl. az `ulimit -t` parancs segítségével), illetve a memóriafoglalásáé is.

`-v, --version`
: Verzióinformáció kijelzése.

`-V, --verbose`
: Bőbeszédűség bekapcsolása: az összes beolvasott fájl elérési útjának, futásidőknek és egyéb információknak a kijelzése.

`-w, --warranty`
: A GNU LilyPond garanciavállalásának kijelzése. (A LilyPond fejlesztői **SEMMIFÉLE GARANCIÁT** nem vállalnak!)

Környezeti változók

A `lilypond` a következő környezeti változókat veszi figyelembe:

`LILYPOND_DATADIR`
: Annak a könyvtárnak a megadására szolgál, ahol a LilyPond üzeneteit és adatfájljait keresni fogja. Tartalmaznia kell a szükséges alkönyvtárakat (‘`ly/`’, ‘`ps/`’, ‘`tex/`’ stb.).

`LANG` A program kimeneti üzeneteinek nyelve.

`LILYPOND_GC_YIELD`
: A program memóriaigénye és futásideje közötti finomhangolást lehet elvégezni ezzel a változóval. Százalékos érték; minél nagyobb, annál több memóriát használ a program, minél alacsonyabb, annál több processzoridőt. Az alapértelmezett érték `70`.

1.3 Hibaüzenetek

Egy fájl fordítása során különböző hibaüzenetek jelenhetnek meg:

Figyelmeztetés
: Valami gyanúsnak tűnik. A figyelmeztetések azt jelzik, hogy valamit nagy valószínűséggel nem úgy írt le a felhasználó, ahogy azt gondolta. De ha tudatosan valami rendkívülit kérünk, akkor általában figyelmen kívül hagyhatóak.

Hiba Valami határozottan helytelen. A feldolgozás aktuális lépése (beolvasás, értelmezés vagy formázás) befejeződik, de a következő lépés ki fog maradni.

Végzetes hiba
: Olyan hiba történt, amitől a LilyPond nem tud tovább futni. Ez ritkán fordul elő. A leggyakoribb ok a rosszul telepített betűtípusok.

Scheme hiba
: A Scheme kód végrehajtása során előforduló hibák, amelyeket a Scheme interpreter kap el. Ha bőbeszédű módban fut a LilyPond, akkor a hibás függvényhez vezető hívások kiírásra kerülnek.

Programozási hiba
: Belső inkonzisztencia lépett fel. Ezek a hibaüzenetek a fejlesztőknek és hibakeresőknek segítenek. Általában figyelmen kívül hagyhatóak. Néha olyan nagy mennyiségben fordulnak elő, hogy nehéz tőlük észrevenni a többi kimeneti üzenetet.

A futás megszakadt (core dumped)
: Kritikus hiba lépett fel, amely a program futását azonnal megszakította. Az ilyen hibákat jelenteni kell a fejlesztőknek.

Ha a figyelmeztetések vagy hibák a bemeneti fájl egy konkrét részére vonatkoznak, akkor az üzenet a következő formátummal bír:

```
fájlnév:sorszám:oszlopszám: üzenet
hibás sor
```

A hibás soron belül a hiba helyét sortörés jelzi. Például:

```
test.ly:2:19: error: not a duration: 5
  { c'4 e'
          5 g' }
```

A probléma helye csak egy becslés, mely olykor pontatlan lehet, hiszen természetüknél fogva a problémák nem várt bemenetnél lépnek fel. Ha nem található hiba a megadott helyen, érdemes a környékén keresni.

A hibákról bővebben a rész 1.4 [Gyakori hibák], oldal 6 c. szakaszban olvashatunk.

1.4 Gyakori hibák

Az alábbi hibajelenségek gyakran előfordulnak, ugyanakkor az okuk nem mindig egyértelmű vagy könnyen megtalálható. Ha azonban egyszer megértjük a természetüket, gyorsan meg lehet rájuk találni a megoldást.

A kotta nem fér ki az oldalra

Ha a kotta jobb oldalra „lefolyik” az oldalról, vagy rendkívül össze van sűrítve, szinte mindig hibás hanghosszúságról van szó, amely miatt egy ütemben az utolsó hang túlnyúlik az ütemvonalom. Ez nem számít hibának, de ha sok ilyen van egymás után, akkor a sor nem tud megtörni, mert sortörés csak olyan ütemek végén helyezkedhet el, amelyek végén nem nyúlik túl hang.
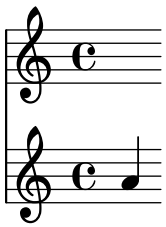
A hibás ritmus könnyen megtalálható ütemhatár-ellenőrzésekkel: ld. a rész “Bar and bar number checks” in *A kottaírás kézikönyve* c. szakaszt.

Ha sok ilyen rendhagyó ütemre van szükség, akkor láthatatlan ütemvonalat kell oda beszúrni, ahol a sortörés megengedett. Ennek módját a rész “Bar lines” in *A kottaírás kézikönyve* c. szakasz írja le.

Egy kottasorral több van a kelleténél

Ha a kontextusokat nem explicite hozzuk létre a `\new` paranccsal, akkor minden figyelmeztetés nélkül létrejön egy új kontextus ott, ahol olyan parancs fordul elő, amely a létező kontextusban nem alkalmazható. Egyszerű kottákban a kontextusok automatikus létrehozása hasznos, és a legtöbb példa hasznát veszi ennek az egyszerűsítésnek. De olykor ez nem várt kottasorok vagy tételek megjelenését eredményezheti. Például a következő kódtól azt várnánk, hogy a kottasorban minden kottafej piros lesz, miközben valójában az eredmény két kottasor, mely közül az alsóban alapértelmezett színű, fekete kottafejek lesznek.

```
\override Staff.NoteHead.color = #red
\new Staff { a }
```
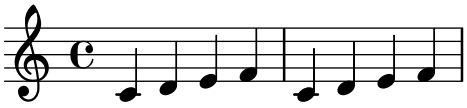


Ez azért történik, mert a Staff kontextus nem létezik az \override parancs helyén, így létrejön, a finomhangolás pedig az így létrehozott kottasorra fog vonatkozni, nem a \new Staff paranccsal létrehozott kottasorra. A példa helyesen:

```
\new Staff {
  \override Staff.NoteHead.color = #red
  a
}
```
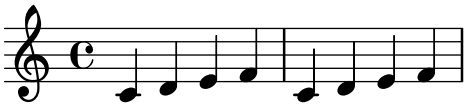


Másik példánkban egy \relative blokk szerepel egy \repeat blokkon belül, ami két kottasort eredményez, amely közül a második később kezdődik, mint az első, mert a \repeat parancs hatására két \relative blokk keletkezik, amik implicit módon létrehoznak egy-egy Staff és Voice kontextust.

```
\repeat unfold 2 {
  \relative c' { c d e f }
}
```



A megoldás a \repeat és a \relative parancsok felcserélése, a következő módon:

```
\relative c' {
  \repeat unfold 2 { c d e f }
}
```



Hiba a ../ly/init.ly fájlban

Különféle rejtélyes hibaüzenetek jelenhetnek meg, melyek a '../ly/init.ly' fájlban található szintaktikai hibára utalnak, ha a forrásfájl nem jól formált, például nem egyezik a nyitó és csukó kapcsos zárójelek vagy idézőjelek száma.

A leggyakoribb hiba a hiányzó } karakter egy blokk, pl. \score blokk végén. A megoldás kézenfekvő: ellenőrizni kell, hogy minden kapcsos zárójelnek megvan-e a párja. A rész "Hogyan működnek a LilyPond bemeneti fájlok?" in *Tankönyv* lecke írja le a forrásfájlok helyes szerkezetét. Egy olyan szövegszerkesztő használatával, mely kiemeli a zárójelpárokat, elkerülhetőek az ilyen hibák.

Egy másik gyakori ok az, hogy nincs szóköz a dalszöveg utolsó szótagja és a dalszöveg blokk záró kapcsos zárójele között. Enélkül az elválasztás nélkül a kapcsos zárójel a szótag részének számít. Emellett *minden* kapcsos zárójel körül érdemes szóközt vagy sortörést hagyni. A jelenség magyarázata a rész "Lyrics explained" in *A kottaírás kézikönyve* c. szakaszban olvasható.

A hiba akkor is előfordulhat, amikor egy záró idézőjel (") hiányzik. Ebben az esetben a hiba egy közeli sorban jelentkezik. A pár nélküli idézőjel általában néhány sorral feljebb található.

`Unbound variable %` hibaüzenet

Ez a hiba akkor fordul elő (egy „GUILE signaled an error ...” hibaüzenettel együtt), amikor a LilyPondba ágyazott Scheme kód *LilyPond* formátumú megjegyzést tartalmaz *Scheme* formátumú helyett.

A LilyPondban a megjegyzések százalékjellel (`%`) kezdődnek, és nem használhatóak Scheme kódon belül. A Scheme kódban a megjegyzések pontosvesszővel (`;`) kezdődnek.

`FT_Get_Glyph_Name` hibaüzenet

Ez a hiba azt jelzi, hogy a bemeneti fájl egy nem ASCII karaktert tartalmaz, ugyanakkor nem UTF-8 karakterkódolással lett elmentve. Részletekért ld. a rész “Text encoding” in *A kottaírás kézikönyve* c. szakaszt.

2 A convert-ly használata

A LilyPond nyelvtana rendszeresen változik, hogy egyszerűsödjön és fejlődjön. Ennek mellékhatásaként a LilyPond olykor nem tudja értelmezni a régebbi forrásfájlokat. Ezt az inkompatibilitást hidalja át a `convert-ly` segédprogram, mely a verziók közötti nyelvváltozások legtöbbjét lekezeli.

2.1 Miért változik a szintaxis?

Ahogy a LilyPond maga fejlődik, a szintaxis (azaz a bemenet nyelve) is ennek megfelelően változik. Ezek a változások azért mennek végbe, hogy a bemenetet könnyebb legyen olvasni és írni, vagy a LilyPond új képességeihez igazodnak.

Például minden `\paper` és `\layout` blokkbeli tulajdonság nevében a szavak konvenció szerint kötőjelekkel kerülnek elválasztásra. A 2.11.60-as verzióban azonban észrevettük, hogy a `printallheaders` tulajdonság nem követi ezt a konvenciót. Felmerült a kérdés: úgy hagyjuk, ahogy eddig volt (így inkonzisztenciával megzavarva az új felhasználókat), vagy megváltoztassuk (így arra kényszerítve a régi felhasználókat, hogy meglévő kottáikat frissítsék)? Ebben az esetben amellett döntöttünk, hogy megváltoztatjuk `print-all-headers`-re. Szerencsére ezt a változás automatikusan kezelhető a `convert-ly` parancssori eszközzel.

Sajnos a `convert-ly` nem képes a nyelvtan minden változását lekezelni. Például a LilyPond 2.4-es és korábbi verzióiban az ékezetes és egyéb, nem angol ábécébe tartozó karaktereket a LaTeX-ben megszokott módszerrel kellett megadni (pl. a francia Noël szót a következőképpen: `No\"el`). De a LilyPond 2.6-os verziója óta minden ilyen karakter, pl. az ë is közvetlenül beleírható a bemeneti fájlba UTF-8 karakterkódolással. A `convert-ly` nem képes minden LaTeX szintaxissal megadott speciális karaktert átkonvertálni az UTF-8 megfelelőjébe; ezeket kézzel kell frissíteni.

2.2 A convert-ly futtatása

A `convert-ly` a forrásfájlban található `\version` parancs alapján állapítja meg a fájl verziószámát. A legtöbb esetben a forrásfájl frissítéséhez elegendő kiadni a

```
convert-ly -e fájlnév.ly
```

parancsot abban a könyvtárban, ahol a fájl található. Ez a parancs helyben frissíti a *fájlnév*`.ly` fájlt, az eredetit pedig megőrzi *fájlnév*`.ly~` néven.

> **Figyelem:** A `convert-ly` parancs alapesetben csak arra a verzióra frissít, amelyikben a legutóbbi szintaxisváltozás történt. Így általában a frissített fájl verziószáma kisebb lesz, mint az éppen használt programé.

Egy könyvtárban található összes bemeneti fájl frissítéséhez a következő parancs használható:

```
convert-ly -e *.ly
```

Amennyiben az újabb fájlnak más nevet szeretnénk adni, és az eredeti fájlt változatlanul szeretnénk hagyni, a következő parancsot adjuk ki:

```
convert-ly fájlnév.ly > újfájlnév.ly
```

Futása során a program kiírja a verziószámokat, amelyekre frissítés történt. Ha egy verziószám sincs kiírva, akkor a fájl teljesen friss.

A Mac OS X-felhasználók ezt a parancsot a grafikus felületen is elérhetik a `Compile > Update syntax` menüpontból.

A Windows-felhasználóknak ezeket a parancsokat a DOS parancssorba kell beírni, amit tipikusan a Start menüben a `Programok > Kellékek > Parancssor` kiválasztásával lehet elindítani.

2.3 A convert-ly parancssori paraméterei

A program meghívása a következő módon történik:

```
convert-ly [opció]... fájlnév...
```

A következő opciók adhatóak meg:

`-e, --edit`
: A fájl helyben frissítése.

`-f, --from=`*forrásverzió*
: A forrásfájl verziójának megadása. Ha nincs megadva, a `convert-ly` a fájlban található `\version` parancs alapján kitalálja. Példa: `--from=2.10.25`

`-n, --no-version`
: Alapesetben a `convert-ly` ellátja a kimenetét a megfelelő `\version` paranccsal. Ez az opció ezt tiltja le.

`-s, --show-rules`
: Nem történik frissítés, csak a frissítési szabályok kiírása.

`--to=`*célverzió*
: Azt adja meg, hogy melyik verzióra frissüljön a fájl. Alapértéke a legfrissebb elérhető verzió. Példa: `--to=2.12.2`

`-h, --help`
: Segítség kiírása az alkalmazás használatához.

Texinfo fájlokban található LilyPond részletek frissítéséhez az alábbi parancs használatos:

```
convert-ly --from=... --to=... --no-version *.itely
```

A LilyPond két verziója közötti, a nyelvtanban bekövetkezett változások megtekintéséhez pedig a következő:

```
convert-ly --from=... --to=... -s
```

2.4 Problémák a convert-ly futtatása közben

Amikor olyan forrásfájlt frissítünk a `convert-ly` segédprogrammal Windows alatt parancssorból, amelynek elérési útja szóközt tartalmaz, a forrásfájl elérési útját három-három (!) idézőjel közé kell írni:

```
convert-ly """D:/Az én kottáim/Óda.ly""" > "D:/Az én kottáim/Óda - új.ly"
```

Ha az egyszerű `convert-ly -e *.ly` parancs futása meghiúsul a fájlok nagy mennyisége miatt, a másik lehetőség a `convert-ly` futtatása ciklusban. A következő, UNIX alatt használható példa minden `.ly` fájlt frissít az aktuális könyvtárban:

```
for f in *.ly; do convert-ly -e $f; done;
```

A Windows parancssorában a megfelelő parancs:

```
for %x in (*.ly) do convert-ly -e """%x"""
```

A program nem minden változást képes kezelni. A Scheme kód és a LilyPond Scheme felületének frissítése nem történik meg, a Scheme kódrészleteket kézzel kell átírni.

2.5 Kézi frissítés

Ideális esetben a `convert-ly` minden változás kezelésére képes lenne. Elvégre ha a régi verzió képes volt értelmezni a régi nyelvtant, az új verzió pedig az újat, akkor elvileg létezhetne egy másik program, amelyik a kettő közötti konverziót elvégzi[1].

[1] Legalábbis ez abban az esetben lehetséges, ha a LilyPond fájl nem tartalmaz Scheme kódot. Ha viszont tartalmaz, akkor egy Turing-teljes nyelvvel van dolgunk, és az algoritmuselméletben jól ismert „megállási problémába” ütközünk.

A gyakorlatban azonban a LilyPond erőforrásai korlátosak: nem minden konverzió történik meg automatikusan. Íme az ismert problémák listája.

```
1.6 -> 2.0:
- A számozott basszus frissítése nem tökéletes, főleg a {< >} esetében.
Ez úgy kerülhető meg, hogy a '{<' karakterlánc összes előfordulását egy
ideiglenes másik karakterláncra cseréljük, pl. '{#'-re. Hasonlóképpen a
'>}' előfordulásai '&}'-re cserélendőek. A frissítés után pedig a következő
cseréket kell végrehajtani: '{ #' -> '{ <' és '& }' -> '> }'.
- A formázott szövegek frissítése sem mindig jó. Eddig zárójelekkel
csoportosítani lehetett több formázó parancsot, pl.:
   -#'((bold italic) "string")
Ez sajnos helytelenül a következővé alakul:
   -\markup{{\bold italic} "string"}
A helyes ez lenne:
   -\markup{\bold \italic "string"}

2.0 -> 2.2:
- A \partcombine frissítése nem támogatott.
- Az \addlyrics => \lyricsto frissítés nem történik meg, ez több versszakkal
rendelkező kották esetében problémát okozhat.

2.0 -> 2.4:
A következő konverziók nem támogatottak:
- \magnify #m => \fontsize #f, ahol f = 6ln(m)/ln(2))
- \applyMusic #(remove-tag '...) => \keepWithTag #'...
- first-page-number no => print-first-page-number = ##f
- "Első sor" \\\\ "Második sor" =>
  \markup \center-align < "Első sor" "Második sor" >
- \rced => \!
- \rc => \!

2.2 -> 2.4:
A \turnOff parancs (pl. a következő esetben:
\set Staff.VoltaBracket = \turnOff) frissítése helytelen.

2.4.2 -> 2.5.9
A \markup{ \center-align <{ ... }> } parancs a frissítés után
\markup{ \center-align {\line { ... }} } kellene, hogy legyen, de a \line
jelenleg hiányzik.

2.4 -> 2.6
A speciális LaTeX karakterek (pl. $~$) nem alakulnak át az UTF-8
megfelelőjükre.

2.8
A \score{} blokknak innentől kezdve egy zenei kifejezéssel kell kezdődnie.
Minden más (pl. a \header{} blokk) a zene után jöhet csak.
```

3 A lilypond-book használata

Amennyiben egy dokumentumba kottapéldákat szeretnénk beszúrni, megtehetjük, hogy azok képeit egyesével létrehozzuk a LilyPond segítségével PostScript vagy PNG formátumban, és mint bármilyen más képeket, beillesztjük azokat egy LaTeX vagy HTML dokumentumba.

A lilypond-book ennek a folyamatnak az automatizálására szolgál: ez a program kiszedi a LilyPond kódrészleteket egy dokumentumból, lefordítja őket a lilypond segítségével, és az így kapott képeket beilleszti az eredeti kódrészletek helyére. A kottakép méretei igazodnak a dokumentum elrendezéséhez.

A lilypond-book egy különálló parancssori program; a parancssoros programok futtatásának módját a rész 1.2 [Parancssori használat], oldal 1 írja le bővebben.

A lilypond-book jelenleg a LaTeX, HTML, Texinfo és DocBook formátumokat támogatja.

3.1 Egy kottapéldákat tartalmazó dokumentum

Bizonyos dokumentumok kottapéldákat tartalmaznak. Ezek között vannak zenetudományi értekezések, énekeskönyvek, vagy ehhez hasonló kézikönyvek. Ezeket úgy is el lehet készíteni, hogy a szövegbe beillesztjük a kottaábrákat. Azonban ahhoz, hogy ne kelljen minden egyes kottarészlet szedését külön elvégezni, a HTML, LaTeX, Texinfo és DocBook formátumú dokumentumok esetén mód nyílik ennek automatizálására.

Egy lilypond-book nevű parancsfájl a LilyPond nyelvén írt kódrészleteket szépen formázott kottapéldákká alakítja át. Íme egy rövid, magyarázatokkal ellátott LaTeX példa.

Bemenet

```
\documentclass[a4paper]{article}

\begin{document}

A \verb+lilypond-book+ segítségével feldolgozott dokumentumok
kottapéldákat tartalmazhatnak. Például:

\begin{lilypond}
\relative c' {
  c2 e2 \tuplet 3/2 { f8 a b } a2 e4
}
\end{lilypond}

A beállításokat szögletes zárójelbe kell tenni:

\begin{lilypond}[fragment,quote,staffsize=26,verbatim]
  c'4 f16
\end{lilypond}

A nagyobb kottapéldákat ki lehet emelni külön fájlba, majd beilleszteni
ket a \verb+\lilypondfile+ paranccsal:

\lilypondfile[quote,noindent]{screech-and-boink.ly}

\end{document}
```

Feldolgozás

A fenti dokumentumot egy ‘lilybook.lytex’ nevű fájlba mentve futtassuk le a következő parancsokat:

```
lilypond-book --output=out --pdf lilybook.lytex
lilypond-book (GNU LilyPond) 2.18.2

Reading lilybook.lytex...
...
Compiling lilybook.tex...
cd out
pdflatex lilybook
...
xpdf lilybook
(az xpdf helyére értelemszerűen tetszőleges PDF-nézegető
kerülhet)
```
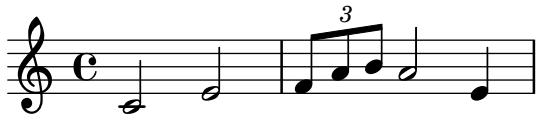
A lilypond-book és a latex rengeteg ideiglenes fájlt hoznak létre. Annak érdekében, hogy ezek külön alkönyvtárba kerüljenek, a --output=*alkönyvtár* opciót kell megadni.

Lent látható a fenti LaTeX példa kimenete.[1] Ezzel elsajátítottuk a lilypond-book használatának alapjait.

[1] Ezt a dokumentumot a Texinfo generálta, így apró eltérések lehetnek.

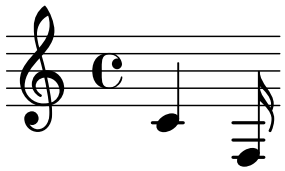
Kimenet

A `lilypond-book` segítségével feldolgozott dokumentumok kottapéldákat tartalmazhatnak. Például:

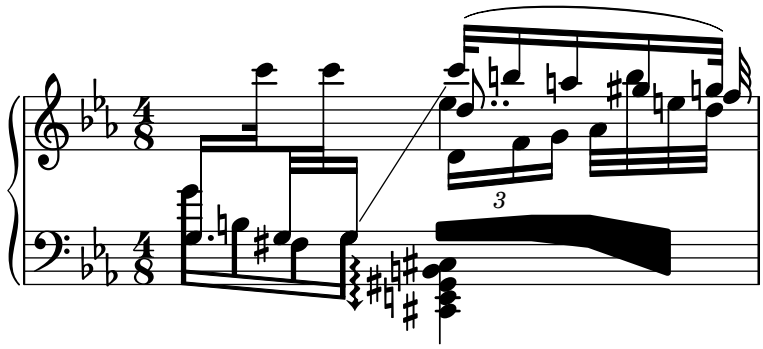


A beállításokat szögletes zárójelbe kell tenni:

```
c'4 f16
```



A nagyobb kottapéldákat ki lehet emelni külön fájlba, majd beilleszteni őket a `\lilypondfile` paranccsal:



3.2 Zene és szöveg integrációja

Here we explain how to integrate LilyPond with various output formats.

3.2.1 LaTeX

LaTeX is the de-facto standard for publishing layouts in the exact sciences. It is built on top of the TeX typesetting engine, providing the best typography available anywhere.

See *The Not So Short Introduction to LaTeX* for an overview on how to use LaTeX.

Music is entered using

```
\begin{lilypond}[options,go,here]
  YOUR LILYPOND CODE
\end{lilypond}
```

or

```
\lilypondfile[options,go,here]{filename}
```

or
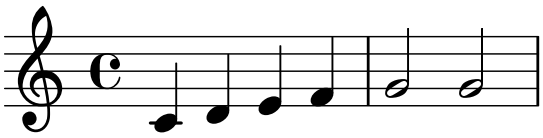
```
\lilypond[options,go,here]{ YOUR LILYPOND CODE }
```

Additionally, `\lilypondversion` displays the current version of lilypond. Running `lilypond-book` yields a file that can be further processed with LaTeX.

We show some examples here. The `lilypond` environment

```
\begin{lilypond}[quote,fragment,staffsize=26]
  c' d' e' f' g'2 g'2
\end{lilypond}
```

produces



The short version

```
\lilypond[quote,fragment,staffsize=11]{<c' e' g'>}
```

produces



Currently, you cannot include `{` or `}` within `\lilypond{}`, so this command is only useful with the `fragment` option.

The default line width of the music will be adjusted by examining the commands in the document preamble, the part of the document before `\begin{document}`. The `lilypond-book` command sends these to LaTeX to find out how wide the text is. The line width for the music fragments is then adjusted to the text width. Note that this heuristic algorithm can fail easily; in such cases it is necessary to use the `line-width` music fragment option.

Each snippet will call the following macros if they have been defined by the user:

- `\preLilyPondExample` called before the music,
- `\postLilyPondExample` called after the music,
- `\betweenLilyPondSystem[1]` is called between systems if `lilypond-book` has split the snippet into several PostScript files. It must be defined as taking one parameter and will be passed the number of files already included in this snippet. The default is to simply insert a `\linebreak`.

Válogatott kódrészletek

Sometimes it is useful to display music elements (such as ties and slurs) as if they continued after the end of the fragment. This can be done by breaking the staff and suppressing inclusion of the rest of the LilyPond output.

In LaTeX, define `\betweenLilyPondSystem` in such a way that inclusion of other systems is terminated once the required number of systems are included. Since `\betweenLilyPondSystem` is first called *after* the first system, including only the first system is trivial.

```
\def\betweenLilyPondSystem#1{\endinput}

\begin{lilypond}[fragment]
  c'1\( e'( c'~ \break c' d) e f\)
\end{lilypond}
```

If a greater number of systems is requested, a TeX conditional must be used before the `\endinput`. In this example, replace '2' by the number of systems you want in the output.

```
\def\betweenLilyPondSystem#1{
    \ifnum#1<2\else\expandafter\endinput\fi
}
```

(Since `\endinput` immediately stops the processing of the current input file we need `\expandafter` to delay the call of `\endinput` after executing `\fi` so that the `\if-\fi` clause is balanced.)

Remember that the definition of `\betweenLilyPondSystem` is effective until TeX quits the current group (such as the LaTeX environment) or is overridden by another definition (which is, in most cases, for the rest of the document). To reset your definition, write

```
\let\betweenLilyPondSystem\undefined
```

in your LaTeX source.

This may be simplified by defining a TeX macro

```
\def\onlyFirstNSystems#1{
    \def\betweenLilyPondSystem##1{%
      \ifnum##1<#1\else\expandafter\endinput\fi}
}
```

and then saying only how many systems you want before each fragment,

```
\onlyFirstNSystems{3}
\begin{lilypond}...\end{lilypond}
\onlyFirstNSystems{1}
\begin{lilypond}...\end{lilypond}
```

Lásd még

There are specific `lilypond-book` command line options and other details to know when processing LaTeX documents, see ⟨undefined⟩ [Invoking lilypond-book], oldal ⟨undefined⟩.

3.2.2 Texinfo

Texinfo is the standard format for documentation of the GNU project. An example of a Texinfo document is this manual. The HTML, PDF, and Info versions of the manual are made from the Texinfo document.

In the input file, music is specified with

```
@lilypond[options,go,here]
  YOUR LILYPOND CODE
@end lilypond
```

or

```
@lilypond[options,go,here]{ YOUR LILYPOND CODE }
```

or

```
@lilypondfile[options,go,here]{filename}
```
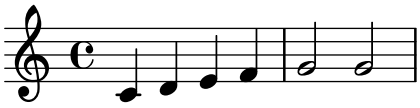
Additionally, `@lilypondversion` displays the current version of lilypond.

When `lilypond-book` is run on it, this results in a Texinfo file (with extension '`.texi`') containing `@image` tags for HTML, Info and printed output. `lilypond-book` generates images of the music in EPS and PDF formats for use in the printed output, and in PNG format for use in HTML and Info output.

We show two simple examples here. A `lilypond` environment

```
@lilypond[fragment]
c' d' e' f' g'2 g'
@end lilypond
```

produces



The short version

```
@lilypond[fragment,staffsize=11]{<c' e' g'>}
```

produces
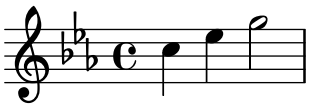


Contrary to LaTeX, `@lilypond{...}` does not generate an in-line image. It always gets a paragraph of its own.

3.2.3 HTML

Music is entered using

```
<lilypond fragment relative=2>
\key c \minor c4 es g2
</lilypond>
```

`lilypond-book` then produces an HTML file with appropriate image tags for the music fragments:



For inline pictures, use `<lilypond ... />`, where the options are separated by a colon from the music, for example

```
Some music in <lilypond relative=2: a b c/> a line of text.
```

To include separate files, say

```
<lilypondfile option1 option2 ...>filename</lilypondfile>
```

For a list of options to use with the `lilypond` or `lilypondfile` tags, see ⟨undefined⟩ [Music fragment options], oldal ⟨undefined⟩.

Additionally, `<lilypondversion/>` displays the current version of lilypond.

3.2.4 DocBook

For inserting LilyPond snippets it is good to keep the conformity of our DocBook document, thus allowing us to use DocBook editors, validation etc. So we don't use custom tags, only specify a convention based on the standard DocBook elements.

Common conventions

For inserting all type of snippets we use the `mediaobject` and `inlinemediaobject` element, so our snippets can be formatted inline or not inline. The snippet formatting options are always provided in the `role` property of the innermost element (see in next sections). Tags are chosen to allow DocBook editors format the content gracefully. The DocBook files to be processed with `lilypond-book` should have the extension '`.lyxml`'.

Including a LilyPond file

This is the most simple case. We must use the '`.ly`' extension for the included file, and insert it as a standard `imageobject`, with the following structure:

```
<mediaobject>
  <imageobject>
    <imagedata fileref="music1.ly" role="printfilename" />
  </imageobject>
</mediaobject>
```

Note that you can use `mediaobject` or `inlinemediaobject` as the outermost element as you wish.

Including LilyPond code

Including LilyPond code is possible by using a `programlisting`, where the language is set to `lilypond` with the following structure:

```
<inlinemediaobject>
  <textobject>
    <programlisting language="lilypond" role="fragment verbatim staffsize=16 ragged-rig
\context Staff \with {
  \remove "Time_signature_engraver"
  \remove "Clef_engraver"}
  { c4( fis) }
    </programlisting>
  </textobject>
</inlinemediaobject>
```

As you can see, the outermost element is a `mediaobject` or `inlinemediaobject`, and there is a `textobject` containing the `programlisting` inside.

Processing the DocBook document

Running `lilypond-book` on our '`.lyxml`' file will create a valid DocBook document to be further processed with '`.xml`' extension. If you use dblatex, it will create a PDF file from this document automatically. For HTML (HTML Help, JavaHelp etc.) generation you can use the official DocBook XSL stylesheets, however, it is possible that you have to make some customization for it.

3.3 Kottapéldák paraméterei

In the following, a 'LilyPond command' refers to any command described in the previous sections which is handled by `lilypond-book` to produce a music snippet. For simplicity, LilyPond commands are only shown in LaTeX syntax.

Note that the option string is parsed from left to right; if an option occurs multiple times, the last one is taken.

The following options are available for LilyPond commands:

`staffsize=`*ht*
: Set staff size to *ht*, which is measured in points.

`ragged-right`
: Produce ragged-right lines with natural spacing, i.e., `ragged-right = ##t` is added to the LilyPond snippet. This is the default for the `\lilypond{}` command if no `line-width` option is present. It is also the default for the `lilypond` environment if the `fragment` option is set, and no line width is explicitly specified.

`noragged-right`
: For single-line snippets, allow the staff length to be stretched to equal that of the line width, i.e., `ragged-right = ##f` is added to the LilyPond snippet.

`line-width`
`line-width=`*size*`\`*unit*
: Set line width to *size*, using *unit* as units. *unit* is one of the following strings: `cm`, `mm`, `in`, or `pt`. This option affects LilyPond output (this is, the staff length of the music snippet), not the text layout.

  If used without an argument, set line width to a default value (as computed with a heuristic algorithm).

  If no `line-width` option is given, `lilypond-book` tries to guess a default for `lilypond` environments which don't use the `ragged-right` option.

`notime`
: Do not print the time signature, and turns off the timing (time signature, bar lines) in the score.

`fragment`
: Make `lilypond-book` add some boilerplate code so that you can simply enter, say,

  ```
  c'4
  ```

  without `\layout`, `\score`, etc.

`nofragment`
: Do not add additional code to complete LilyPond code in music snippets. Since this is the default, `nofragment` is redundant normally.

`indent=`*size*`\`*unit*
: Set indentation of the first music system to *size*, using *unit* as units. *unit* is one of the following strings: `cm`, `mm`, `in`, or `pt`. This option affects LilyPond, not the text layout.

`noindent`
: Set indentation of the first music system to zero. This option affects LilyPond, not the text layout. Since no indentation is the default, `noindent` is redundant normally.

`quote`
: Reduce line length of a music snippet by 2∗0.4 in and put the output into a quotation block. The value '0.4 in' can be controlled with the `exampleindent` option.

`exampleindent`
: Set the amount by which the `quote` option indents a music snippet.

`relative`
`relative=`*n*
: Use relative octave mode. By default, notes are specified relative to middle C. The optional integer argument specifies the octave of the starting note, where the default `1` is middle C. `relative` option only works when `fragment` option is set, so `fragment` is automatically implied by `relative`, regardless of the presence of any `(no)fragment` option in the source.

LilyPond also uses lilypond-book to produce its own documentation. To do that, some more obscure music fragment options are available.

verbatim The argument of a LilyPond command is copied to the output file and enclosed in a verbatim block, followed by any text given with the intertext option (not implemented yet); then the actual music is displayed. This option does not work well with \lilypond{} if it is part of a paragraph.

If verbatim is used in a lilypondfile command, it is possible to enclose verbatim only a part of the source file. If the source file contain a comment containing 'begin verbatim' (without quotes), quoting the source in the verbatim block will start after the last occurrence of such a comment; similarly, quoting the source verbatim will stop just before the first occurrence of a comment containing 'end verbatim', if there is any. In the following source file example, the music will be interpreted in relative mode, but the verbatim quote will not show the relative block, i.e.

```
\relative c' { % begin verbatim
  c4 e2 g4
  f2 e % end verbatim
}
```

will be printed with a verbatim block like

```
  c4 e2 g4
  f2 e
```

If you would like to translate comments and variable names in verbatim output but not in the sources, you may set the environment variable LYDOC_LOCALEDIR to a directory path; the directory should contain a tree of '.mo' message catalogs with lilypond-doc as a domain.

addversion (Only for Texinfo output.) Prepend line \version @w{"@version{}"} to verbatim output.

texidoc (Only for Texinfo output.) If lilypond is called with the '--header=texidoc' option, and the file to be processed is called 'foo.ly', it creates a file 'foo.texidoc' if there is a texidoc field in the \header. The texidoc option makes lilypond-book include such files, adding its contents as a documentation block right before the music snippet.

Assuming the file 'foo.ly' contains

```
\header {
  texidoc = "This file demonstrates a single note."
}
{ c'4 }
```

and we have this in our Texinfo document 'test.texinfo'

```
@lilypondfile[texidoc]{foo.ly}
```

the following command line gives the expected result

```
lilypond-book --pdf --process="lilypond \
  -dbackend=eps --header=texidoc" test.texinfo
```

Most LilyPond test documents (in the 'input' directory of the distribution) are small '.ly' files which look exactly like this.

For localization purpose, if the Texinfo document contains @documentlanguage *LANG* and 'foo.ly' header contains a texidoc*LANG* field, and if lilypond is called with '--header=texidoc*LANG*', then 'foo.texidoc*LANG*' will be included instead of 'foo.texidoc'.

`doctitle` (Only for Texinfo output.) This option works similarly to `texidoc` option: if `lilypond` is called with the '`--header=doctitle`' option, and the file to be processed is called '`foo.ly`' and contains a `doctitle` field in the `\header`, it creates a file '`foo.doctitle`'. When `doctitle` option is used, the contents of '`foo.doctitle`', which should be a single line of *text*, is inserted in the Texinfo document as `@lydoctitle` *text*. `@lydoctitle` should be a macro defined in the Texinfo document. The same remark about `texidoc` processing with localized languages also applies to `doctitle`.

`nogettext`
(Only for Texinfo output.) Do not translate comments and variable names in the snippet quoted verbatim.

`printfilename`
If a LilyPond input file is included with `\lilypondfile`, print the file name right before the music snippet. For HTML output, this is a link. Only the base name of the file is printed, i.e. the directory part of the file path is stripped.

3.4 A `lilypond-book` futtatása

`lilypond-book` produces a file with one of the following extensions: '`.tex`', '`.texi`', '`.html`' or '`.xml`', depending on the output format. All of '`.tex`', '`.texi`' and '`.xml`' files need further processing.

Format-specific instructions

LaTeX

There are two ways of processing your LaTeX document for printing or publishing: getting a PDF file directly with PDFLaTeX, or getting a PostScript file with LaTeX via a DVI to PostScript translator like `dvips`. The first way is simpler and recommended[1], and whichever way you use, you can easily convert between PostScript and PDF with tools, like `ps2pdf` and `pdf2ps` included in Ghostscript package.

To produce a PDF file through PDFLaTeX, use

```
lilypond-book --pdf yourfile.lytex
pdflatex yourfile.tex
```

To produce PDF output via LaTeX/`dvips`/`ps2pdf`, you should do

```
lilypond-book yourfile.lytex
latex yourfile.tex
dvips -Ppdf yourfile.dvi
ps2pdf yourfile.ps
```

The '`.dvi`' file created by this process will not contain note heads. This is normal; if you follow the instructions, they will be included in the '`.ps`' and '`.pdf`' files.

Running `dvips` may produce some warnings about fonts; these are harmless and may be ignored. If you are running `latex` in twocolumn mode, remember to add `-t landscape` to the `dvips` options.

Texinfo

To produce a Texinfo document (in any output format), follow the normal procedures for Texinfo; this is, either call `texi2pdf` or `texi2dvi` or `makeinfo`, depending on the output format you want to create. See the documentation of Texinfo for further details.

[1] Note that PDFLaTeX and LaTeX may not be both usable to compile any LaTeX document, that is why we explain the two ways.

Command line options

lilypond-book accepts the following command line options:

-f *format*
--format=*format*
: Specify the document type to process: html, latex, texi (the default) or docbook. If this option is missing, lilypond-book tries to detect the format automatically, see ⟨undefined⟩ [Filename extensions], oldal ⟨undefined⟩. Currently, texi is the same as texi-html.

-F *filter*
--filter=*filter*
: Pipe snippets through *filter*. lilypond-book will not –filter and –process at the same time. For example,

```
lilypond-book --filter='convert-ly --from=2.0.0 -' my-book.tely
```

-h
--help
: Print a short help message.

-I *dir*
--include=*dir*
: Add *dir* to the include path. lilypond-book also looks for already compiled snippets in the include path, and does not write them back to the output directory, so in some cases it is necessary to invoke further processing commands such as makeinfo or latex with the same -I *dir* options.

-o *dir*
--output=*dir*
: Place generated files in directory *dir*. Running lilypond-book generates lots of small files that LilyPond will process. To avoid all that garbage in the source directory, use the '--output' command line option, and change to that directory before running latex or makeinfo.

```
lilypond-book --output=out yourfile.lytex
cd out
...
```

--skip-lily-check
: Do not fail if no lilypond output is found. It is used for LilyPond Info documentation without images.

--skip-png-check
: Do not fail if no PNG images are found for EPS files. It is used for LilyPond Info documentation without images.

--lily-output-dir=*dir*
: Write lily-XXX files to directory *dir*, link into --output directory. Use this option to save building time for documents in different directories which share a lot of identical snippets.

--info-images-dir=*dir*
: Format Texinfo output so that Info will look for images of music in *dir*.

--latex-program=*prog*
: Run executable prog instead of latex. This is useful if your document is processed with xelatex, for example.

--left-padding=*amount*
: Pad EPS boxes by this much. *amount* is measured in millimeters, and is 3.0 by default. This option should be used if the lines of music stick out of the right margin.

The width of a tightly clipped system can vary, due to notation elements that stick into the left margin, such as bar numbers and instrument names. This option will shorten each line and move each line to the right by the same amount.

`-P` *command*
`--process=`*command*
: Process LilyPond snippets using *command*. The default command is `lilypond`. `lilypond-book` will not `--filter` and `--process` at the same time.

`--pdf` : Create PDF files for use with PDFLaTeX.

`--use-source-file-names`
: Write snippet output files with the same base name as their source file. This option works only for snippets included with `lilypondfile` and only if directories implied by `--output-dir` and `--lily-output-dir` options are different.

`-V`
`--verbose`
: Be verbose.

`-v`
`--version`
: Print version information.

Ismert problémák és figyelmeztetések

The Texinfo command `@pagesizes` is not interpreted. Similarly, LaTeX commands that change margins and line widths after the preamble are ignored.

Only the first `\score` of a LilyPond block is processed.

3.5 Fájlkiterjesztések

You can use any filename extension for the input file, but if you do not use the recommended extension for a particular format you may need to manually specify the output format; for details, see ⟨undefined⟩ [Invoking lilypond-book], oldal ⟨undefined⟩. Otherwise, `lilypond-book` automatically selects the output format based on the input filename's extension.

extension	**output format**
'`.html`'	HTML
'`.htmly`'	HTML
'`.itely`'	Texinfo
'`.latex`'	LaTeX
'`.lytex`'	LaTeX
'`.lyxml`'	DocBook
'`.tely`'	Texinfo
'`.tex`'	LaTeX
'`.texi`'	Texinfo
'`.texinfo`'	Texinfo
'`.xml`'	HTML

If you use the same filename extension for the input file than the extension `lilypond-book` uses for the output file, and if the input file is in the same directory as `lilypond-book` working directory, you must use `--output` option to make `lilypond-book` running, otherwise it will exit with an error message like „Output would overwrite input file".

3.6 lilypond-book sablonok

These templates are for use with `lilypond-book`. If you're not familiar with this program, please refer to fejezet 3 [lilypond-book], oldal 12.

3.6.1 LaTeX

You can include LilyPond fragments in a LaTeX document.

```
\documentclass[]{article}

\begin{document}

Normal LaTeX text.

\begin{lilypond}
\relative c'' {
  a4 b c d
}
\end{lilypond}

More LaTeX text, and options in square brackets.

\begin{lilypond}[fragment,relative=2,quote,staffsize=26,verbatim]
d4 c b a
\end{lilypond}
\end{document}
```

3.6.2 Texinfo

You can include LilyPond fragments in Texinfo; in fact, this entire manual is written in Texinfo.

```
\input texinfo @node Top
@top

Texinfo text

@lilypond
\relative c' {
  a4 b c d
}
@end lilypond

More Texinfo text, and options in brackets.

@lilypond[verbatim,fragment,ragged-right]
d4 c b a
@end lilypond

@bye
```

3.6.3 html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<!-- header_tag -->
<HTML>
<body>

<p>
Documents for lilypond-book may freely mix music and text.  For
example,
```

```
<lilypond>
\relative c'' {
  a4 b c d
}
</lilypond>
</p>

<p>
Another bit of lilypond, this time with options:

<lilypond fragment quote staffsize=26 verbatim>
a4 b c d
</lilypond>
</p>

</body>
</html>
```

3.6.4 xelatex

```
\documentclass{article}
\usepackage{ifxetex}
\ifxetex
%xetex specific stuff
\usepackage{xunicode,fontspec,xltxtra}
\setmainfont[Numbers=OldStyle]{Times New Roman}
\setsansfont{Arial}
\else
%This can be empty if you are not going to use pdftex
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage{mathptmx}%Times
\usepackage{helvet}%Helvetica
\fi
%Here you can insert all packages that pdftex also understands
\usepackage[ngerman,finnish,english]{babel}
\usepackage{graphicx}

\begin{document}
\title{A short document with LilyPond and xelatex}
\maketitle

Normal \textbf{font} commands inside the \emph{text} work,
because they \textsf{are supported by \LaTeX{} and XeteX.}
If you want to use specific commands like \verb+\XeTeX+, you
should include them again in a \verb+\ifxetex+ environment.
You can use this to print the \ifxetex \XeTeX{} command \else
XeTeX command \fi which is not known to normal \LaTeX .

In normal text you can easily use LilyPond commands, like this:
```

```
\begin{lilypond}
{a2 b c'8 c' c' c'}
\end{lilypond}

\noindent
and so on.

The fonts of snippets set with LilyPond will have to be set from
inside
of the snippet.  For this you should read the AU on how to use
lilypond-book.

\selectlanguage{ngerman}
Auch Umlaute funktionieren ohne die \LaTeX -Befehle, wie auch alle
anderen
seltsamen Zeichen: __ ______, wenn sie von der Schriftart
unterst__tzt werden.
\end{document}
```

3.7 Közös tartalomjegyzék

These functions already exist in the OrchestralLily package:

http://repo.or.cz/w/orchestrallily.git

For greater flexibility in text handling, some users prefer to export the table of contents from lilypond and read it into LaTeX.

Exporting the ToC from LilyPond

This assumes that your score has multiple movements in the same lilypond output file.

```
#(define (oly:create-toc-file layout pages)
  (let* ((label-table (ly:output-def-lookup layout 'label-page-table)))
    (if (not (null? label-table))
      (let* ((format-line (lambda (toc-item)
             (let* ((label (car toc-item))
                    (text  (caddr toc-item))
                    (label-page (and (list? label-table)
                                     (assoc label label-table)))
                    (page (and label-page (cdr label-page))))
               (format #f "~a, section, 1, {~a}, ~a" page text label))))
             (formatted-toc-items (map format-line (toc-items)))
             (whole-string (string-join formatted-toc-items ",\n"))
             (output-name (ly:parser-output-name parser))
             (outfilename (format "~a.toc" output-name))
             (outfile (open-output-file outfilename)))
        (if (output-port? outfile)
            (display whole-string outfile)
            (ly:warning (_ "Unable to open output file ~a for the TOC information") outfilename))
        (close-output-port outfile)))))

\paper {
  #(define (page-post-process layout pages) (oly:create-toc-file layout pages))
}
```

Importing the ToC into LaTeX

In LaTeX, the header should include:

```
\usepackage{pdfpages}
```

```
\includescore{nameofthescore}
```

where \includescore is defined as:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% \includescore{PossibleExtension}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Read in the TOC entries for a PDF file from the corresponding .toc file.
% This requires some heave latex tweaking, since reading in things from a file
% and inserting it into the arguments of a macro is not (easily) possible

% Solution by Patrick Fimml on #latex on April 18, 2009:
% \readfile{filename}{\variable}
% reads in the contents of the file into \variable (undefined if file
% doesn't exist)
\newread\readfile@f
\def\readfile@line#1{%
{\catcode`\^^M=10\global\read\readfile@f to \readfile@tmp}%
\edef\do{\noexpand\g@addto@macro{\noexpand#1}{\readfile@tmp}}\do%
\ifeof\readfile@f\else%
\readfile@line{#1}%
\fi%
}
\def\readfile#1#2{%
\openin\readfile@f=#1 %
\ifeof\readfile@f%
\typeout{No TOC file #1 available!}%
\else%
\gdef#2{}%
\readfile@line{#2}%
\fi
\closein\readfile@f%
}%


\newcommand{\includescore}[1]{
\def\oly@fname{\oly@basename\@ifmtarg{#1}{}{_#1}}
\let\oly@addtotoc\undefined
\readfile{\oly@xxxxxxxxx}{\oly@addtotoc}
\ifx\oly@addtotoc\undefined
\includepdf[pages=-]{\oly@fname}
\else
\edef\includeit{\noexpand\includepdf[pages=-,addtotoc={\oly@addtotoc}]
{\oly@fname}}\includeit
\fi
}
```

3.8 További módszerek zene és szöveg kombinálására

Other means of mixing text and music (without lilypond-book) are discussed in rész 4.4 [LilyPond output in other programs], oldal 35.

4 External programs

LilyPond can interact with other programs in various ways.

4.1 Point and click

Point and click lets you find notes in the input by clicking on them in the PDF viewer. This makes it easier to find input that causes some error in the sheet music.

4.1.1 Configuring the system

When this functionality is active, LilyPond adds hyperlinks to the PDF file. These hyperlinks are sent to a 'URI helper' or a web-browser, which opens a text-editor with the cursor in the right place.

To make this chain work, you should configure your PDF viewer to follow hyperlinks using the '`lilypond-invoke-editor`' script supplied with LilyPond.

The program '`lilypond-invoke-editor`' is a small helper program. It will invoke an editor for the special `textedit` URIs, and run a web browser for others. It tests the environment variable `EDITOR` for the following patterns,

`emacs` this will invoke

```
emacsclient --no-wait +line:column file
```

`gvim` this will invoke

```
gvim --remote +:line:normcolumn file
```

`nedit` this will invoke

```
nc -noask +line file'
```

The environment variable `LYEDITOR` is used to override this. It contains the command line to start the editor, where `%(file)s`, `%(column)s`, `%(line)s` is replaced with the file, column and line respectively. The setting

```
emacsclient --no-wait +%(line)s:%(column)s %(file)s
```

for `LYEDITOR` is equivalent to the standard emacsclient invocation.

Using Xpdf

For Xpdf on UNIX, the following should be present in '`xpdfrc`'. On UNIX, this file is found either in '`/etc/xpdfrc`' or as '`$HOME/.xpdfrc`'.

```
urlCommand     "lilypond-invoke-editor %s"
```

If you are using Ubuntu, it is likely that the version of Xpdf installed with your system crashes on every PDF file: this state has been persisting for several years and is due to library mismatches. Your best bet is to install a current '`xpdf`' package and the corresponding '`libpoppler`' package from Debian instead. Once you have tested that this works, you might want to use

```
sudo apt-mark hold xpdf
```

in order to keep Ubuntu from overwriting it with the next 'update' of its crashing package.

Using GNOME 2

For using GNOME 2 (and PDF viewers integrated with it), the magic invocation for telling the system about the '`textedit:`' URI is

```
gconftool-2 -t string -s /desktop/gnome/url-handlers/textedit/command "lilypond-invoke-
gconftool-2 -s /desktop/gnome/url-handlers/textedit/needs_terminal false -t bool
gconftool-2 -t bool -s /desktop/gnome/url-handlers/textedit/enabled true
```

After that invocation,

```
gnome-open textedit:///etc/issue:1:0:0
```

should call '`lilypond-invoke-editor`' for opening files.

Using GNOME 3

In GNOME 3, URIs are handled by the 'gvfs' layer rather than by 'gconf'. Create a file in a local directory such as '`/tmp`' that is called '`lilypond-invoke-editor.desktop`' and has the contents

```
[Desktop Entry]
Version=1.0
Name=lilypond-invoke-editor
GenericName=Textedit URI handler
Comment=URI handler for textedit:
Exec=lilypond-invoke-editor %u
Terminal=false
Type=Application
MimeType=x-scheme-handler/textedit;
Categories=Editor
NoDisplay=true
```

and then execute the commands

```
xdg-desktop-menu install ./lilypond-invoke-editor.desktop
xdg-mime default lilypond-invoke-editor.desktop x-scheme-handler/textedit
```

After that invocation,

```
gnome-open textedit:///etc/issue:1:0:0
```

should call '`lilypond-invoke-editor`' for opening files.

Extra configuration for Evince

If `gnome-open` works, but Evince still refuses to open point and click links due to denied permissions, you might need to change the Apparmor profile of Evince which controls the kind of actions Evince is allowed to perform.

For Ubuntu, the process is to edit the file '`/etc/apparmor.d/local/usr.bin.evince`' and append the following lines:

```
# For Textedit links
/usr/local/bin/lilypond-invoke-editor Cx -> sanitized_helper,
```

After adding these lines, call

```
sudo apparmor_parser -r -T -W /etc/apparmor.d/usr.bin.evince
```

Now Evince should be able to open point and click links. It is likely that similar configurations will work for other viewers.

Enabling point and click

Point and click functionality is enabled by default when creating PDF files.

The point and click links enlarge the output files significantly. For reducing the size of PDF and PS files, point and click may be switched off by issuing

```
\pointAndClickOff
```

in a '`.ly`' file. Point and click may be explicitly enabled with

```
\pointAndClickOn
```

Alternately, you may disable point and click with a command-line option:

```
lilypond -dno-point-and-click file.ly
```

> **Figyelem:** You should always turn off point and click in any LilyPond files to be distributed to avoid including path information about your computer in the .pdf file, which can pose a security risk.

Selective point-and-click

For some interactive applications, it may be desirable to only include certain point-and-click items. For example, if somebody wanted to create an application which played audio or video starting from a particular note, it would be awkward if clicking on the note produced the point-and-click location for an accidental or slur which occurred over that note.

This may be controlled by indicating which events to include:

- Hard-coded in the '`.ly`' file:

```
\pointAndClickTypes #'note-event
\relative c' {
  c2\f( f)
}
```

or

```
#(ly:set-option 'point-and-click 'note-event)
\relative c' {
  c2\f( f)
}
```

- Command-line:

```
lilypond -dpoint-and-click=note-event   example.ly
```

Multiple events can be included:

- Hard-coded in the '`.ly`' file:

```
\pointAndClickTypes #'(note-event dynamic-event)
\relative c' {
  c2\f( f)
}
```

or

```
#(ly:set-option 'point-and-click '(note-event dynamic-event))
\relative c' {
  c2\f( f)
}
```

- Command-line:

```
lilypond \
  -e"(ly:set-option 'point-and-click '(note-event dynamic-event))" \
  example.ly
```

4.2 Text editor support

There is support for different text editors for LilyPond.

Emacs mode

Emacs has a '`lilypond-mode`', which provides keyword autocompletion, indentation, LilyPond specific parenthesis matching and syntax coloring, handy compile short-cuts and reading LilyPond manuals using Info. If '`lilypond-mode`' is not installed on your platform, see below.

An Emacs mode for entering music and running LilyPond is contained in the source archive in the '`elisp`' directory. Do `make install` to install it to *elispdir*. The file '`lilypond-init.el`' should be placed to *load-path*'`/site-start.d/`' or appended to your '`~/.emacs`' or '`~/.emacs.el`'.

As a user, you may want add your source path (e.g. '`~/site-lisp/`') to your *load-path* by appending the following line (as modified) to your '`~/.emacs`'

```
(setq load-path (append (list (expand-file-name "~/site-lisp")) load-path))
```

Vim mode

For Vim, a filetype plugin, indent mode, and syntax-highlighting mode are available to use with LilyPond. To enable all of these features, create (or modify) your '`$HOME/.vimrc`' to contain these three lines, in order:

```
filetype off
set runtimepath+=/usr/local/share/lilypond/current/vim/
filetype on
syntax on
```

If LilyPond is not installed in the '`/usr/local/`' directory, change the path appropriately. This topic is discussed in rész "Other sources of information" in *Tankönyv*.

Other editors

Other editors (both text and graphical) support LilyPond, but their special configuration files are not distributed with LilyPond. Consult their documentation for more information. Such editors are listed in rész "Easier editing" in *Általános információk*.

4.3 Converting from other formats

Music can be entered also by importing it from other formats. This chapter documents the tools included in the distribution to do so. There are other tools that produce LilyPond input, for example GUI sequencers and XML converters. Refer to the website for more details.

These are separate programs from `lilypond` itself, and are run on the command line; see ⟨undefined⟩ [Command-line usage], oldal ⟨undefined⟩ for more information. If you have MacOS 10.3 or 10.4 and you have trouble running some of these scripts, e.g. `convert-ly`, see rész "MacOS X" in *Általános információk*.

Ismert problémák és figyelmeztetések

We unfortunately do not have the resources to maintain these programs; please consider them „as-is". Patches are appreciated, but bug reports will almost certainly not be resolved.

4.3.1 Invoking `midi2ly`

`midi2ly` translates a Type 1 MIDI file to a LilyPond source file.

MIDI (Music Instrument Digital Interface) is a standard for digital instruments: it specifies cabling, a serial protocol and a file format. The MIDI file format is a de facto standard format for exporting music from other programs, so this capability may come in useful when importing files from a program that has a converter for a direct format.

`midi2ly` converts tracks into rész "Staff" in *A belső működés referenciája* and channels into rész "Voice" in *A belső működés referenciája* contexts. Relative mode is used for pitches, durations are only written when necessary.

It is possible to record a MIDI file using a digital keyboard, and then convert it to '`.ly`'. However, human players are not rhythmically exact enough to make a MIDI to LY conversion trivial. When invoked with quantizing ('`-s`' and '`-d`' options) `midi2ly` tries to compensate for

these timing errors, but is not very good at this. It is therefore not recommended to use `midi2ly` for human-generated midi files.

It is invoked from the command-line as follows,

```
midi2ly [option]... midi-file
```

Note that by 'command-line', we mean the command line of the operating system. See rész 4.3 [Converting from other formats], oldal 31, for more information about this.

The following options are supported by `midi2ly`.

`-a, --absolute-pitches`
: Print absolute pitches.

`-d, --duration-quant=`*DUR*
: Quantize note durations on *DUR*.

`-e, --explicit-durations`
: Print explicit durations.

`-h, --help`
: Show summary of usage.

`-k, --key=`*acc*`[:`*minor*`]`
: Set default key. *acc* $>$ 0 sets number of sharps; *acc* $<$ 0 sets number of flats. A minor key is indicated by `:1`.

`-o, --output=`*file*
: Write output to *file*.

`-s, --start-quant=`*DUR*
: Quantize note starts on *DUR*.

`-t, --allow-tuplet=`*DUR*`*`*NUM*`/`*DEN*
: Allow tuplet durations *DUR***NUM*/*DEN*.

`-v, --verbose`
: Be verbose.

`-V, --version`
: Print version number.

`-w, --warranty`
: Show warranty and copyright.

`-x, --text-lyrics`
: Treat every text as a lyric.

Ismert problémák és figyelmeztetések

Overlapping notes in an arpeggio will not be correctly rendered. The first note will be read and the others will be ignored. Set them all to a single duration and add phrase markings or pedal indicators.

4.3.2 Invoking `musicxml2ly`

MusicXML is an XML dialect for representing music notation.

`musicxml2ly` extracts the notes, articulations, score structure, lyrics, etc. from part-wise MusicXML files, and writes them to a '`.ly`' file. It is invoked from the command-line.

It is invoked from the command-line as follows,

```
musicxml2ly [option]... xml-file
```

Note that by 'command-line', we mean the command line of the operating system. See rész 4.3 [Converting from other formats], oldal 31, for more information about this.

If the given filename is '-', `musicxml2ly` reads input from the command line.

The following options are supported by `musicxml2ly`:

`-a, --absolute`
: convert pitches in absolute mode.

`-h, --help`
: print usage and option summary.

`-l, --language=LANG`
: use LANG for pitch names, e.g. 'deutsch' for note names in German.

`--loglevel=`*loglevel*
: Set the output verbosity to *loglevel*. Possible values are `NONE`, `ERROR`, `WARNING`, `PROGRESS` (default) and `DEBUG`.

`--lxml`
: use the lxml.etree Python package for XML-parsing; uses less memory and cpu time.

`-m, --midi`
: activate midi-block.

`-nd, --no-articulation-directions`
: do not convert directions (`^`, `_` or `-`) for articulations, dynamics, etc.

`--no-beaming`
: do not convert beaming information, use LilyPond's automatic beaming instead.

`-o, --output=`*file*
: set output filename to *file*. If *file* is '-', the output will be printed on stdout. If not given, *xml-file*'`.ly`' will be used.

`-r, --relative`
: convert pitches in relative mode (default).

`-v, --verbose`
: be verbose.

`--version`
: print version information.

`-z, --compressed`
: input file is a zip-compressed MusicXML file.

4.3.3 Invoking `abc2ly`

> **Figyelem:** This program is not supported, and may be remove from future versions of LilyPond.

ABC is a fairly simple ASCII based format. It is described at the ABC site:

http://www.walshaw.plus.com/abc/learn.html.

`abc2ly` translates from ABC to LilyPond. It is invoked as follows:

```
abc2ly [option]... abc-file
```

The following options are supported by `abc2ly`:

`-b, --beams=None`
: preserve ABC's notion of beams

`-h, --help`
: this help

`-o, --output=`*file*
: set output filename to *file*.

`-s, --strict`
: be strict about success

`--version`
: print version information.

There is a rudimentary facility for adding LilyPond code to the ABC source file. If you say:

```
%%LY voices \set autoBeaming = ##f
```

This will cause the text following the keyword 'voices' to be inserted into the current voice of the LilyPond output file.

Similarly,

```
%%LY slyrics more words
```

will cause the text following the 'slyrics' keyword to be inserted into the current line of lyrics.

Ismert problémák és figyelmeztetések

The ABC standard is not very 'standard'. For extended features (e.g., polyphonic music) different conventions exist.

Multiple tunes in one file cannot be converted.

ABC synchronizes words and notes at the beginning of a line; `abc2ly` does not.

`abc2ly` ignores the ABC beaming.

4.3.4 Invoking `etf2ly`

> **Figyelem:** This program is not supported, and may be remove from future versions of LilyPond.

ETF (Enigma Transport Format) is a format used by Coda Music Technology's Finale product. `etf2ly` will convert part of an ETF file to a ready-to-use LilyPond file.

It is invoked from the command-line as follows.

```
etf2ly [option]... etf-file
```

Note that by 'command-line', we mean the command line of the operating system. See rész 4.3 [Converting from other formats], oldal 31, for more information about this.

The following options are supported by `etf2ly`:

`-h, --help`
: this help

`-o, --output=`*FILE*
: set output filename to *FILE*

`--version`
: version information

Ismert problémák és figyelmeztetések

The list of articulation scripts is incomplete. Empty measures confuse `etf2ly`. Sequences of grace notes are ended improperly.

4.3.5 Other formats

LilyPond itself does not come with support for any other formats, but some external tools can also generate LilyPond files. These are listed in rész “Easier editing” in *Általános információk*.

4.4 LilyPond output in other programs

This section shows methods to integrate text and music, different than the automated method with `lilypond-book`.

Many quotes from a large score

If you need to quote many fragments from a large score, you can also use the clip systems feature, see rész “Extracting fragments of music” in *A kottaírás kézikönyve*.

Inserting LilyPond output into OpenOffice and LibreOffice

LilyPond notation can be added to OpenOffice.org and LibreOffice with OOoLilyPond.

Inserting LilyPond output into other programs

To insert LilyPond output in other programs, use `lilypond` instead of `lilypond-book`. Each example must be created individually and added to the document; consult the documentation for that program. Most programs will be able to insert LilyPond output in ‘`PNG`’, ‘`EPS`’, or ‘`PDF`’ formats.

To reduce the white space around your LilyPond score, use the following options

```
\paper{
  indent=0\mm
  line-width=120\mm
  oddFooterMarkup=##f
  oddHeaderMarkup=##f
  bookTitleMarkup = ##f
  scoreTitleMarkup = ##f
}

{ c1 }
```

To produce useful image files:

```
EPS

lilypond -dbackend=eps -dno-gs-load-fonts -dinclude-eps-fonts myfile.ly

PNG

lilypond -dbackend=eps -dno-gs-load-fonts -dinclude-eps-fonts --png myfile.ly

A transparent PNG

lilypond -dbackend=eps -dno-gs-load-fonts -dinclude-eps-fonts \
  -dpixmap-format=pngalpha --png myfile.ly
```

4.5 Independent `includes`

Some people have written large (and useful!) code that can be shared between projects. This code might eventually make its way into LilyPond itself, but until that happens, you must download and `\include` them manually.

4.5.1 MIDI articulation

LilyPond can be used to produce MIDI output, for „proof-hearing” what has been written. However, only dynamics, explicit tempo markings, and the notes and durations themselves are produced in the output.

The *articulate* project is one attempt to get more of the information in the score into MIDI. It works by shortening notes not under slurs, to ‘articulate’ the notes. The amount of shortening depends on any articulation markings attached to a note: staccato halves the note value, tenuto gives a note its full duration, and so on. The script also realises trills and turns, and could be extended to expand other ornaments such as mordents.

`http://www.nicta.com.au/people/chubbp/articulate`

Ismert problémák és figyelmeztetések

Its main limitation is that it can only affect things it knows about: anything that is merely textual markup (instead of a note property) is still ignored.

5 Suggestions for writing files

Now you're ready to begin writing larger LilyPond input files – not just the little examples in the tutorial, but whole pieces. But how should you go about doing it?

As long as LilyPond can understand your input files and produce the output that you want, it doesn't matter what your input files look like. However, there are a few other things to consider when writing LilyPond input files.

- What if you make a mistake? The structure of a LilyPond file can make certain errors easier (or harder) to find.
- What if you want to share your input files with somebody else? In fact, what if you want to alter your own input files in a few years? Some LilyPond input files are understandable at first glance; others may leave you scratching your head for an hour.
- What if you want to upgrade your LilyPond file for use with a later version of LilyPond? The input syntax changes occasionally as LilyPond improves. Most changes can be done automatically with `convert-ly`, but some changes might require manual assistance. LilyPond input files can be structured in order to be easier (or harder) to update.

5.1 General suggestions

Here are a few suggestions that can help you to avoid or fix problems:

- **Include `\version` numbers in every file**. Note that all templates contain `\version` information. We highly recommend that you always include the `\version`, no matter how small your file is. Speaking from personal experience, it's quite frustrating to try to remember which version of LilyPond you were using a few years ago. `convert-ly` requires you to declare which version of LilyPond you used.
- **Include checks**: rész "Bar and bar number checks" in *A kottaírás kézikönyve*, rész "Octave checks" in *A kottaírás kézikönyve*. If you include checks every so often, then if you make a mistake, you can pinpoint it quicker. How often is 'every so often'? It depends on the complexity of the music. For very simple music, perhaps just once or twice. For very complex music, perhaps every bar.
- **One bar per line of text**. If there is anything complicated, either in the music itself or in the output you desire, it's often good to write only one bar per line. Saving screen space by cramming eight bars per line just isn't worth it if you have to 'debug' your input files.
- **Comment your input files**. Use either bar numbers (every so often) or references to musical themes ('second theme in violins,' 'fourth variation,' etc.). You may not need comments when you're writing the piece for the first time, but if you want to go back to change something two or three years later, or if you pass the source over to a friend, it will be much more challenging to determine your intentions or how your file is structured if you didn't comment the file.
- **Indent your braces**. A lot of problems are caused by an imbalance in the number of `{` and `}`.
- **Explicitly add durations** at the beginnings of sections and variables. If you specify `c4 d e` at the beginning of a phrase (instead of just `c d e`) you can save yourself some problems if you rearrange your music later.
- **Separate tweaks** from music definitions. See rész "Saving typing with variables and functions" in *Tankönyv*, and rész "Style sheets" in *Tankönyv*.

5.2 Typesetting existing music

If you are entering music from an existing score (i.e., typesetting a piece of existing sheet music),

- Enter the manuscript (the physical copy of the music) into LilyPond one system at a time (but still only one bar per line of text), and check each system when you finish it. You may use the `showLastLength` or `showFirstLength` properties to speed up processing – see rész "Skipping corrected music" in *A kottaírás kézikönyve*.
- Define `mBreak = { \break }` and insert `\mBreak` in the input file whenever the manuscript has a line break. This makes it much easier to compare the LilyPond music to the original music. When you are finished proofreading your score, you may define `mBreak = { }` to remove all those line breaks. This will allow LilyPond to place line breaks wherever it feels are best.
- When entering a part for a transposing instrument into a variable, it is recommended that the notes are wrapped in

  ```
  \transpose c natural-pitch {...}
  ```

  (where `natural-pitch` is the open pitch of the instrument) so that the music in the variable is effectively in C. You can transpose it back again when the variable is used, if required, but you might not want to (e.g., when printing a score in concert pitch, converting a trombone part from treble to bass clef, etc.) Mistakes in transpositions are less likely if all the music in variables is at a consistent pitch.

  Also, only ever transpose to/from C. That means that the only other keys you will use are the natural pitches of the instruments - bes for a B-flat trumpet, aes for an A-flat clarinet, etc.

5.3 Large projects

When working on a large project, having a clear structure to your lilypond input files becomes vital.

- **Use a variable for each voice**, with a minimum of structure inside the definition. The structure of the `\score` section is the most likely thing to change; the `violin` definition is extremely unlikely to change in a new version of LilyPond.

  ```
  violin = \relative c'' {
  g4 c'8. e16
  }
  ...
  \score {
    \new GrandStaff {
      \new Staff {
        \violin
      }
    }
  }
  ```

- **Separate tweaks from music definitions**. This point was made previously, but for large projects it is absolutely vital. We might need to change the definition of `fthenp`, but then we only need to do this once, and we can still avoid touching anything inside `violin`.

  ```
  fthenp = _\markup{
    \dynamic f \italic \small { 2nd } \hspace #0.1 \dynamic p }
  violin = \relative c'' {
  g4\fthenp c'8. e16
  }
  ```

5.4 Troubleshooting

Sooner or later, you will write a file that LilyPond cannot compile. The messages that LilyPond gives may help you find the error, but in many cases you need to do some investigation to determine the source of the problem.

The most powerful tools for this purpose are the single line comment (indicated by `%`) and the block comment (indicated by `%{...%}`). If you don't know where a problem is, start commenting out huge portions of your input file. After you comment out a section, try compiling the file again. If it works, then the problem must exist in the portion you just commented. If it doesn't work, then keep on commenting out material until you have something that works.

In an extreme case, you might end up with only

```
\score {
  <<
    % \melody
    % \harmony
    % \bass
  >>
  \layout{}
}
```

(in other words, a file without any music)

If that happens, don't give up. Uncomment a bit – say, the bass part – and see if it works. If it doesn't work, then comment out all of the bass music (but leave `\bass` in the `\score` uncommented.

```
bass = \relative c' {
%{
  c4 c c c
  d d d d
%}
}
```

Now start slowly uncommenting more and more of the `bass` part until you find the problem line.

Another very useful debugging technique is constructing rész "Tiny examples" in *Általános információk*.

5.5 Make and Makefiles

Pretty well all the platforms Lilypond can run on support a software facility called `make`. This software reads a special file called a `Makefile` that defines what files depend on what others and what commands you need to give the operating system to produce one file from another. For example the makefile would spell out how to produce '`ballad.pdf`' and '`ballad.midi`' from '`ballad.ly`' by running Lilypond.

There are times when it is a good idea to create a `Makefile` for your project, either for your own convenience or as a courtesy to others who might have access to your source files. This is true for very large projects with many included files and different output options (e.g. full score, parts, conductor's score, piano reduction, etc.), or for projects that require difficult commands to build them (such as `lilypond-book` projects). Makefiles vary greatly in complexity and flexibility, according to the needs and skills of the authors. The program GNU Make comes installed on GNU/Linux distributions and on MacOS X, and it is also available for Windows.

See the **GNU Make Manual** for full details on using `make`, as what follows here gives only a glimpse of what it can do.

The commands to define rules in a makefile differ according to platform; for instance the various forms of GNU/Linux and MacOS use `bash`, while Windows uses `cmd`. Note that on MacOS X, you need to configure the system to use the command-line interpreter. Here are some example makefiles, with versions for both GNU/Linux/MacOS and Windows.

The first example is for an orchestral work in four movements with a directory structure as follows:

```
Symphony/
|-- MIDI/
|-- Makefile
|-- Notes/
|   |-- cello.ily
|   |-- figures.ily
|   |-- horn.ily
|   |-- oboe.ily
|   |-- trioString.ily
|   |-- viola.ily
|   |-- violinOne.ily
|   `-- violinTwo.ily
|-- PDF/
|-- Parts/
|   |-- symphony-cello.ly
|   |-- symphony-horn.ly
|   |-- symphony-oboes.ly
|   |-- symphony-viola.ly
|   |-- symphony-violinOne.ly
|   `-- symphony-violinTwo.ly
|-- Scores/
|   |-- symphony.ly
|   |-- symphonyI.ly
|   |-- symphonyII.ly
|   |-- symphonyIII.ly
|   `-- symphonyIV.ly
`-- symphonyDefs.ily
```

The '`.ly`' files in the '`Scores`' and '`Parts`' directories get their notes from '`.ily`' files in the '`Notes`' directory:

```
%%% top of file "symphony-cello.ly"
\include ../symphonyDefs.ily
\include ../Notes/cello.ily
```

The makefile will have targets of `score` (entire piece in full score), `movements` (individual movements in full score), and `parts` (individual parts for performers). There is also a target `archive` that will create a tarball of the source files, suitable for sharing via web or email. Here is the makefile for GNU/Linux or MacOS X. It should be saved with the name `Makefile` in the top directory of the project:

> **Figyelem:** When a target or pattern rule is defined, the subsequent lines must begin with tabs, not spaces.

```
# the name stem of the output files
piece = symphony
# determine how many processors are present
CPU_CORES=`cat /proc/cpuinfo | grep -m1 "cpu cores" | sed s/".*: "//`
```

```
# The command to run lilypond
LILY_CMD = lilypond -ddelete-intermediate-files \
                    -dno-point-and-click -djob-count=$(CPU_CORES)

# The suffixes used in this Makefile.
.SUFFIXES: .ly .ily .pdf .midi

# Input and output files are searched in the directories listed in
# the VPATH variable.  All of them are subdirectories of the current
# directory (given by the GNU make variable `CURDIR').
VPATH = \
  $(CURDIR)/Scores \
  $(CURDIR)/PDF \
  $(CURDIR)/Parts \
  $(CURDIR)/Notes

# The pattern rule to create PDF and MIDI files from a LY input file.
# The .pdf output files are put into the `PDF' subdirectory, and the
# .midi files go into the `MIDI' subdirectory.
%.pdf %.midi: %.ly
        $(LILY_CMD) $<; \           # this line begins with a tab
        if test -f "$*.pdf"; then \
            mv "$*.pdf" PDF/; \
        fi; \
        if test -f "$*.midi"; then \
            mv "$*.midi" MIDI/; \
        fi

notes = \
  cello.ily \
  horn.ily \
  oboe.ily \
  viola.ily \
  violinOne.ily \
  violinTwo.ily

# The dependencies of the movements.
$(piece)I.pdf: $(piece)I.ly $(notes)
$(piece)II.pdf: $(piece)II.ly $(notes)
$(piece)III.pdf: $(piece)III.ly $(notes)
$(piece)IV.pdf: $(piece)IV.ly $(notes)

# The dependencies of the full score.
$(piece).pdf: $(piece).ly $(notes)

# The dependencies of the parts.
$(piece)-cello.pdf: $(piece)-cello.ly cello.ily
$(piece)-horn.pdf: $(piece)-horn.ly horn.ily
$(piece)-oboes.pdf: $(piece)-oboes.ly oboe.ily
$(piece)-viola.pdf: $(piece)-viola.ly viola.ily
$(piece)-violinOne.pdf: $(piece)-violinOne.ly violinOne.ily
$(piece)-violinTwo.pdf: $(piece)-violinTwo.ly violinTwo.ily
```

```
# Type `make score' to generate the full score of all four
# movements as one file.
.PHONY: score
score: $(piece).pdf

# Type `make parts' to generate all parts.
# Type `make foo.pdf' to generate the part for instrument `foo'.
# Example: `make symphony-cello.pdf'.
.PHONY: parts
parts: $(piece)-cello.pdf \
       $(piece)-violinOne.pdf \
       $(piece)-violinTwo.pdf \
       $(piece)-viola.pdf \
       $(piece)-oboes.pdf \
       $(piece)-horn.pdf

# Type `make movements' to generate files for the
# four movements separately.
.PHONY: movements
movements: $(piece)I.pdf \
           $(piece)II.pdf \
           $(piece)III.pdf \
           $(piece)IV.pdf

all: score parts movements

archive:
        tar -cvvf stamitz.tar \       # this line begins with a tab
        --exclude=*pdf --exclude=*~ \
        --exclude=*midi --exclude=*.tar \
        ../Stamitz/*
```

There are special complications on the Windows platform. After downloading and installing GNU Make for Windows, you must set the correct path in the system's environment variables so that the DOS shell can find the Make program. To do this, right-click on "My Computer," then choose `Properties` and `Advanced`. Click `Environment Variables`, and then in the `System Variables` pane, highlight `Path`, click `edit`, and add the path to the GNU Make executable file, which will look something like this:

```
C:\Program Files\GnuWin32\bin
```

The makefile itself has to be altered to handle different shell commands and to deal with spaces that are present in some default system directories. The `archive` target is eliminated since Windows does not have the `tar` command, and Windows also has a different default extension for midi files.

```
## WINDOWS VERSION
##
piece = symphony
LILY_CMD = lilypond -ddelete-intermediate-files \
                    -dno-point-and-click \
                    -djob-count=$(NUMBER_OF_PROCESSORS)

#get the 8.3 name of CURDIR (workaround for spaces in PATH)
```

```
workdir = $(shell for /f "tokens=*" %%b in ("$(CURDIR)") \
          do @echo %%~sb)

.SUFFIXES: .ly .ily .pdf .mid

VPATH = \
  $(workdir)/Scores \
  $(workdir)/PDF \
  $(workdir)/Parts \
  $(workdir)/Notes

%.pdf %.mid: %.ly
        $(LILY_CMD) $<      # this line begins with a tab
        if exist "$*.pdf"  move /Y "$*.pdf"  PDF/ # begin with tab
        if exist "$*.mid" move /Y "$*.mid" MIDI/  # begin with tab

notes = \
  cello.ily \
  figures.ily \
  horn.ily \
  oboe.ily \
  trioString.ily \
  viola.ily \
  violinOne.ily \
  violinTwo.ily

$(piece)I.pdf: $(piece)I.ly $(notes)
$(piece)II.pdf: $(piece)II.ly $(notes)
$(piece)III.pdf: $(piece)III.ly $(notes)
$(piece)IV.pdf: $(piece)IV.ly $(notes)

$(piece).pdf: $(piece).ly $(notes)

$(piece)-cello.pdf: $(piece)-cello.ly cello.ily
$(piece)-horn.pdf: $(piece)-horn.ly horn.ily
$(piece)-oboes.pdf: $(piece)-oboes.ly oboe.ily
$(piece)-viola.pdf: $(piece)-viola.ly viola.ily
$(piece)-violinOne.pdf: $(piece)-violinOne.ly violinOne.ily
$(piece)-violinTwo.pdf: $(piece)-violinTwo.ly violinTwo.ily

.PHONY: score
score: $(piece).pdf

.PHONY: parts
parts: $(piece)-cello.pdf \
       $(piece)-violinOne.pdf \
       $(piece)-violinTwo.pdf \
       $(piece)-viola.pdf \
       $(piece)-oboes.pdf \
       $(piece)-horn.pdf

.PHONY: movements
```

```
movements: $(piece)I.pdf \
           $(piece)II.pdf \
           $(piece)III.pdf \
           $(piece)IV.pdf

all: score parts movements
```

The next Makefile is for a `lilypond-book` document done in LaTeX. This project has an index, which requires that the `latex` command be run twice to update links. Output files are all stored in the `out` directory for .pdf output and in the `htmlout` directory for the html output.

```
SHELL=/bin/sh
FILE=myproject
OUTDIR=out
WEBDIR=htmlout
VIEWER=acroread
BROWSER=firefox
LILYBOOK_PDF=lilypond-book --output=$(OUTDIR) --pdf $(FILE).lytex
LILYBOOK_HTML=lilypond-book --output=$(WEBDIR) $(FILE).lytex
PDF=cd $(OUTDIR) && pdflatex $(FILE)
HTML=cd $(WEBDIR) && latex2html $(FILE)
INDEX=cd $(OUTDIR) && makeindex $(FILE)
PREVIEW=$(VIEWER) $(OUTDIR)/$(FILE).pdf &

all: pdf web keep

pdf:
        $(LILYBOOK_PDF)  # begin with tab
        $(PDF)           # begin with tab
        $(INDEX)         # begin with tab
        $(PDF)           # begin with tab
        $(PREVIEW)       # begin with tab

web:
        $(LILYBOOK_HTML) # begin with tab
        $(HTML)          # begin with tab
        cp -R $(WEBDIR)/$(FILE)/ ./  # begin with tab
        $(BROWSER) $(FILE)/$(FILE).html &  # begin with tab

keep: pdf
        cp $(OUTDIR)/$(FILE).pdf $(FILE).pdf  # begin with tab

clean:
        rm -rf $(OUTDIR) # begin with tab

web-clean:
        rm -rf $(WEBDIR) # begin with tab

archive:
        tar -cvvf myproject.tar \ # begin this line with tab
        --exclude=out/* \
        --exclude=htmlout/* \
        --exclude=myproject/* \
```

```
        --exclude=*midi \
        --exclude=*pdf \
        --exclude=*~ \
        ../MyProject/*
```

TODO: make this thing work on Windows

The previous makefile does not work on Windows. An alternative for Windows users would be to create a simple batch file containing the build commands. This will not keep track of dependencies the way a makefile does, but it at least reduces the build process to a single command. Save the following code as `build.bat` or `build.cmd`. The batch file can be run at the DOS prompt or by simply double-clicking its icon.

```
lilypond-book --output=out --pdf myproject.lytex
cd out
pdflatex myproject
makeindex myproject
pdflatex myproject
cd ..
copy out\myproject.pdf MyProject.pdf
```

Lásd még

This manual: ⟨undefined⟩ [Command-line usage], oldal ⟨undefined⟩, fejezet 3 [lilypond-book], oldal 12

függelék A GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.
http://fsf.org/

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

   You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both

covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its

Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

függelék B LilyPond index

\

\header in LaTeX documents 15

A

ABC 33

B

biztonsági mód, parancssor 2

C

Coda Technology 34
coloring, syntax 30
convert-ly 9
core dumped 5

D

docbook 12
DocBook, kottapéldák 12
dokumentumok, kottapéldák 12
dvips 21

E

editors 30
előnézet, parancssor 3
emacs 30
Encapsulated PostScript 3
enigma 34
EPS (Encapsulated PostScript) 3
ETF 34
Evince 29
External programs, generating LilyPond files 35

F

fájlok frissítése 9
figyelmeztetés 5
file size, output 29
Finale 34

H

hiba 5
hibák formátuma 6
hibaüzenetek 5
html 12
HTML, kottapéldák 12

I

invoking dvips 21

K

keresési útvonal 3
kimeneti formátum 2

L

LANG 5
latex 12
LaTeX, kottapéldák 12
LibreOffice.org 35
LILYPOND_DATADIR 5
LILYPOND_GC_YIELD 5

M

make 39
makefiles 39
MIDI 31
modes, editor 30
MusicXML 32

O

OpenOffice.org 35
outline fonts 21

P

papírméret, parancssor 2
parancssori paraméterek 1
point and click 28
point-and-click, parancssor 2
Portable Document Format (PDF) kimenet 4
Portable Network Graphics (PNG) kimenet 4
PostScript kimenet 3, 4
preview image 17
programozási hiba 5

S

Scheme hiba 5
Scheme kiíratás 3
súgó, parancssor 2
SVG (Scalable Vector Graphics) 3
syntax coloring 30

T

texi 12
texinfo 12
Texinfo, kottapéldák 12
thumbnail 17
titling and lilypond-book 15
titling in HTML 17
type1 fonts 21

V

végzetes hiba 5
vim 30

X

Xpdf 28

Z

zenetudomány 12