FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Vkládání interaktivních prvků a multimédií do PDF souborů prostřednictvím TeXu
Student:	Michal Vlasák
Vedoucí:	RNDr. Petr Olšák
Studijní program:	Informatika
Obor / specializace:	Bezpečnost a informační technologie
Katedra:	Katedra počítačových systémů
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Novější verze formátu PDF podporují interaktivní prvky a vkládání, respektive přehrávání multimédií (audio, video, Adobe U3D a PRC). V rámci práce se zaměřte na generování takových PDF souborů TeXem včetně vlastní implementace.

1) Srovnejte možnosti vkládání interaktivních prvků a multimédií, které nabízí nejnovější volně dostupná specifikace formátu PDF, s možnostmi, které využívá ConTeXt a LaTeXový balíček media9.
2) Vyhodnoťte a shrňte podporu zobrazení interaktivních prvků a přehrávání multimédií v různých PDF prohlížečích. Zaměřte se také na bezpečnostní aspekty.
3) Analyzujte možnost vkládání interaktivních prvků a multimédií na úrovni TeXových primitiv pdfTeXu nebo LuaTeXu.
4) Implementujte vkládání interaktivních prvků a multimédií pro použití v plainTeXu nebo OpTeXu a zveřejněte výsledný balíček na CTAN.

Elektronicky schválil/a prof. Ing. Pavel Tvrdík, CSc. dne 14. listopadu 2020 v Praze.

ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F8

Fakulta informačních technologií
Katedra počítačových systémů

Bakalářská práce

Vkládání interaktivních prvků a multimédií do PDF souborů prostřednictvím TEXu

Michal Vlasák

12. 5. 2021

Vedoucí práce: RNDr. Petr Olšák

Poděkování / Prohlášení

V první řadě děkuji svému vedoucímu, RNDr. Petru Olšákovi, za jeho trpělivost, cenné rady a připomínky.

Díky patří také mým rodičům za jejich neustálou podporu.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Proboštově dne 12. 5. 2021

.....

Abstrakt / Abstract

Zaměřením této bakalářské práce je oblast interaktivních prvků a multimédií ve formátu PDF, obzvlášť pak ve spojitosti s TEXem. Kromě analýzy a diskuze toho, co specifikuje norma formátu PDF nebo co implementují existující TEXové balíčky, bylo zaměření práce také praktické – otestování co doopravdy funguje v dnešních PDF prohlížečích.

Jak se ukázalo, tak interaktivních a multimediálních možností nabízí formát PDF celou řadu. Špatná je ale jejich podpora v PDF prohlížečích, která v této oblasti silně zaostává. Výjimkou je de facto referenční prohlížeč Acrobat a jím inspirovaný prohlížeč Foxit. Jsou ale i open-source prohlížeče (Evince, Okular), které mohou být pro některá využití, např. prezentace, naprosto vyhovující.

Na základě všech získaných znalostí byl poté vytvořen balíček pro nový TEXový formát OpTEX. Ten implementuje v oblasti TEXu smysluplné a zároveň v praxi funkční interaktivní prvky a multimédia. Balíček je veřejný a volně dostupný.

Výsledný balíček nabízí v oblasti multimédií možnosti vkládání audia, videa a 3D děl. V oblasti interaktivity pak například komplexně řeší akce, události nebo animované přechody.

Klíčová slova: TeX, PDF, interaktivní prvky, multimédia, audio, video, 3D

This bachelor thesis concerns itself with the area of interactive features and multimedia in PDF files, especially in connection with TEX. Apart from the analysis and discussion of what PDF standard offers, or what is implemented by existing TEX packages, the focus was also practical—testing what really works in PDF viewers available today.

It turns out, that PDF offers many interactive and multimedia possibilities. The support in PDF viewers is however strongly lacking. The exception is the de-facto reference viewer Acrobat and a viewer strongly influenced by it—Foxit. However, there are open-source viewers (Evince, Okular) whose use in some areas may be completely satisfactory.

The gained knowledge was used to create a package for a new TEX format OpTEX. The package implements those interactive features and multimedia capabilities that have sense in the context of TEX and also work in practice. The package is publicly and freely available.

In the area of multimedia, the resulting package offers the possibility to insert audio, video, and 3D art. In the area of interactive features, it for example complexly handles actions, trigger events, or transitions.

Keywords: TeX, PDF, interactive elements, multimedia, audio, video, 3D

Title translation: Embedding interactive elements and multimedia into PDF files using TEX

Obsah /

Tabulky / Obrázky

Kapitola 1
Úvod

Při zmínce formátu PDF (Portable Document Format) nebo typografického systému TEX si asi málokdo představí interaktivní prvky nebo multimédia. Přesto formát PDF nabízí v tomto ohledu nemalé množství možností. Logicky se potom v průběhu let objevilo několik rozšíření TEXu, která toho dokázala v různé míře využít. Druhým koncem spektra je ovšem schopnost PDF prohlížečů multimédia a interaktivní prvky správně zobrazit a interpretovat. S tou zahýbalo také nedávné ukončení podpory technologie Flash, kterou některé prohlížeče využívaly pro přehrávání multimédií.

Motivací pro vypracování této práce byla snaha zdokumentovat možnosti, které v tomto ohledu nabízí norma PDF [9] a také snaha zmapovat, jaká je podpora v PDF prohlížečích. Dalším popudem byla absence balíčku, který by umožnil uživatelům plain-TEXových formátů tyto možnosti využít. V souvislosti s tím byly stanoveny následující cíle práce:

- Analýza možností nabízených normou PDF a jejich srovnání s možnostmi existujících TEXových balíčků.
- Otestování skutečné podpory interaktivních prvků a multimédií v dnes používaných PDF prohlížečích.
- Analýza a diskuze možností TEXu a jeho rozšíření v oblasti výstupu do formátu PDF.
- Na základě získaných znalostí implementovat balíček cílený na uživatele plainTEXu. Implementována by měla být smysluplná a v prohlížečích fungující funkcionalita z oblasti interaktivních prvků a multimédií. Samozřejmostí je zveřejnění balíčku na portál CTAN[1], tedy místo kde TEXová komunita sdílí užitečné materiály v podobě balíčků.

Text této práce je koncipován pro čtenáře, který má konkrétnější povědomí o fungování TEXu, PDF a částečně i multimédií. V TEXové části je předpokládána znalost fungování (plain) TEXu zhruba na úrovni dokumentu „TEX in a Nutshell“[2]. U PDF se předpokládá alespoň základní povědomí o PDF objektech [9, část 7.3].

Práce v rámci svých kapitol postupuje od teoretických částí k praktičtějším. Následující dvě kapitoly se zabývají obsahem PDF souborů s interaktivními prvky a multimédii, a také možnostmi jejich tvorby TEXem. Další kapitola je praktičtější a spojuje obě předchozí – rozebírá možnosti existujících TEXových balíčků a částečně diskutuje i jejich implementace. Následující kapitola o podpoře interaktivních prvků a multimédií v PDF prohlížečích shrnuje výsledky testu PDF prohlížečů. Poslední významná kapitola se zabývá návrhem balíčku. Prakticky tak dává dohromady závěry všech předchozích kapitol. Součástí přílohy je podstatná část výsledného balíčku – vysázená uživatelská a technická dokumentace i zdrojový kód. Implementace a technická dokumentace jsou protkány díky použití techniky „Literate programming“, která je historicky s TEXem úzce spjata.

[1] `https://www.ctan.org/`
[2] `https://mirrors.ctan.org/info/tex-nutshell/tex-nutshell.pdf`

Kapitola 2
Interaktivní prvky a multimédia ve formátu PDF

Tato kapitola má především teoretický charakter. Její účel je shrnout, jaké možnosti norma PDF [9] nabízí z hlediska tzv. interaktivních prvků a multimédií.

Součástí této kapitoly jsou také ukázky PDF struktur v prakticky využitelné podobě a širších souvislostech. To je něco, co v normě místy chybí.

2.1 Interaktivní prvky

Pod slovním spojením „interaktivní prvky“ si lze představit mnoho různých věcí. V souvislosti s formátem PDF jistou definici nabízí přímo norma PDF, která věnuje interaktivním prvkům celou jednu kapitolu („Interactive features“, *interaktivní možnosti*, [9, část 12]). Ta říká, že „*interaktivní prvky umožňují interakci uživatele s dokumentem za pomoci myši a klávesnice*“. Konkrétněji jsou různé druhy možných interaktivních prvků rozepsány v podsekcích této kapitoly.

Bohužel ne všechny možnosti popsané v zmíněné kapitole normy lze chápat jako opravdové interaktivní prvky. Je to případ především elektronických podpisů a možností odměřování skutečných rozměrů ze stránek PDF dokumentu (což je v běžné technické praxi stejně zavrhováno). Ani jedním se tato práce nebude zaobírat. Dále v této práci nebude pojednáno o formulářích, které norma také řadí do interaktivních prvků.

Oborem na hraně skutečné interaktivity je také možnost nastavení výchozího stavu PDF prohlížeče. Některá z dostupných nastavení dokáží interaktivitu uzpůsobit (například pro prezentace), proto se jimi práce dále zabývá.

Konečně hlavními možnostmi interakce, které jsou především tématem této práce, jsou tzv. „akce“ (*actions*), „anotace“ a s nimi související možnosti jako pohyb v rámci dokumentu či stránky a multimédia. Po vzoru normy PDF je i v tomto textu multimédiím věnována zvláštní sekce (2.2).

2.1.1 Nastavení prohlížeče

Různá nastavení výchozího stavu PDF prohlížeče lze pro konkrétní dokument přizpůsobit pomocí slovníku `/ViewerPreferences` v katalogu dokumentu[1]. Protože se nastavení pomocí `/ViewerPreferences` objevily až v PDF verzi 1.2 [9, část 12.2], tak některá dřívější nastavení, která by tam logicky patřila, jsou o úroveň výše – v katalogu dokumentu, ale má větší smysl je uvažovat zároveň.

Změnitelná nastavení jsou v podstatě dvojího druhu:

- uzpůsobení zobrazení na obrazovce (užitečné především pro prezentace),
- nastavení tisku.

[1] *Document catalog*, „kořenový objekt“ PDF souboru, mimo jiné třeba nepřímo odkazuje na všechny stránky v dokumentu [9, část 7.7.2].

Obě katogerie jsou užitečné pro typicky různé druhy dokumentů, je však možné je nastavit oboje.

Například následující (vcelku samovysvětlující) katalog by mohl patřit prezentaci, u které je vyžadováno, aby se zobrazila na na celou obrazovku:

```
<<
  /Type /Catalog
  /Pages 1 0 R
  /PageMode /FullScreen
>>
```

Případně podobný „nerušivý“ styl, ale pro zobrazení v okně (ne na celou obrazovku), lze docílit složitěji:

```
<<
  /Type /Catalog
  /Pages 1 0 R
  /ViewerPreferences <<
    /HideToolbar true
    /HideMenubar true
    /HideWindowUI true
    /FitWindow true
  >>
>>
```

Například pro tuto práci může být vhodné zobrazit „záložky“ (*outlines*) a nastavit jak zobrazení, tak tisk pro oboustrannou sazbu:

```
<<
  /Type /Catalog
  /Pages 1 0 R
  /PageMode /UseOutlines
  /PageLayout /TwoPageRight
  /ViewerPreferences <<
    /Duplex /DuplexFlipLongEdge
  >>
>>
```

2.1.2 Anotace

Anotace [9, část 12.5] je v podstatě obdélníková oblast na stránce. Její druh (`/Subtype`) ale přidává konkrétní význam – například existuje anotace typu `/Link`, která odkazuje na jiné umístění v PDF dokumentu, nebo umožňuje vyvolání nějaké obecné akce (detailněji viz sekci 2.1.4). Následující příklad ukazuje různé aspekty anotace:

```
6 0 obj
<<
  /Type /Annot
  /Subtype /Link
  /Rect [ 100 100 250 250 ]
  /Border [ 0 0 0 ]
  /AP <<
    /N 7 0 R
    /R 8 0 R
    /D 9 0 R
  >>
```

```
    /M (8. listopadu 1620)
    /Contents (Link annotation.)
    /A <<
      /Type /Action
      /S /URI % URI = Uniform Resource Identifier
      /URI (https://fit.cvut.cz/)
    >>
  >>
  endobj
```

První dva údaje ve slovníku jsou přímočaré – udávají o jaký typ objektu, respektive anotace se jedná. Poté následuje několik položek, které jsou platné pro všechny druhy anotací. Jedná se především o specifikaci obdélníkové části stránky, která je s anotací spjata. To je velmi důležitý koncept. Anotace totiž stojí úplně mimo normální obsah stránky (viz také sekci 2.2.1). Na jejich umístění se tak například nevztahuje transformační matice – jsou opravdu umístěny absolutně. Má to své určité výhody a nevýhody. Jednou výhodou je jednoduchost výpočtu, kterým PDF prohlížeč určuje zda kliknutí myší patří některé z anotací na stránce. Na druhou stranu je to problematické v situacích, kdy je potřeba zvláštní, neobdélníkový tvar anotace. To v praxi například nastává, když je anotací realizován klikací odkaz, třeba do webu, ale URL (*Uniform Resource Locator*) je dlouhá a dojde k jejímu rozlomení do dvou řádků. V takových situacích je potřeba definovat několik anotací lišících se jen obdélníkem, který pokrývají. To také souvisí s tím, že u podobného odkazu musí program tvořící PDF zajistit jistou synchronizaci mezi obsahem stránky (např. text uvádějící URL) a anotací, protože z pohledu PDF je neváže nic dohromady.

Je také nutno zdůraznit, jak se anotace na stránku dostane. Stránka je totiž přeci jen koncipována jako „soběstačný“ objekt – PDF prohlížeč má možnost načíst jakoukoliv individuální stránku a vykreslit jí bez toho, aniž by zkoumal jakýkoliv kontext. Pro potřeby vložení anotací je tak ve slovníku stránky možno uvést položku `/Annots`, která je polem obsahujícím nepřímé odkazy na všechny použité anotace. I když by to mohlo znamenat, že by anotace mohly být využity vícekrát (ovšem vždy na stejných souřadnicích, například pro hyperlinkový odkaz opakující se v záhlaví stránek), tak to je zakázáno – každá anotace může být spjata maximálně s jednou stránkou.

Zobrazení ohraničujícího obdélniku lze vypnout pomocí nulových hodnot `/Border`. nulových parametrů. Dále jsou definovány tři vzhledy anotace, prostřednictvím tzv. *Appearance streams*. Vzhledy určují jak bude anotace vypadat ve výchozím stavu (`/N`), při najetí kurzorem (`/R`) a při kliknutí (`/D`). Většina anotací má ale svůj výchozí vzhled, takže položku `/AP` není potřeba uvádět vůbec. Platí také, že výchozí hodnota `/R` a `/D` je hodnota položky `/N`. Pod všemi se skrývá nějaká forma (*form XObject*, viz sekci 2.2.1).

Dalšími atributy anotace jsou datum vytvoření a textový obsah. To u tohoto konkrétního typu anotace nedává moc velký smysl. Anotace jsou rozpolcený mechanismus – na jednu stranu umožňují autorům dokumentu vkládat prvky, u kterých to jinak nejde (akční tlačítka, videa, 3D grafiku, . . .). Na druhou stranu je ale většina anotací koncipována spíš pro čtenáře dokumentu, kterým dovoluje mimo obsah stránky přidávat různé značky a poznámky, které mohou sloužit třeba pro zpětnou vazbu autorovi. V kontextu PDF jsou anotace tohoto druhého typu nazývány „značkovací“ (*markup annotations*).

Pro účel této práce je postačující zaměřit se pouze na několik málo druhů anotací. Především jsou to anotace pro multimédia (viz sekci 2.2) a také velmi praktické Link anotace (související především s akcemi, viz sekci 2.1.4). Velká část zbylých anotací obzvláště v kontextu dokumentů vytvářených TEXem nemá moc smysl. Anotace jako „Square“, „Circle“, „Underline“, „Highlight“ a mnoho dalších grafických prvků lze totiž

vytvářet v TEXu přímo bez potřeby použít anotace. Tyto anotace jsou také označeny jako „značkovací“ a jsou tak především určeny pro dodatečné značkování dokumentu.

2.1.3 Umístění v dokumentu

Základem pro pohyb v dokumentu jsou takzvaná „umístění“ (*destinations*, [9, část 12.3.2]), která určují jedno konkrétní místo v PDF dokumentu a také styl přiblížení (*zoom*).

Dva nejobvyklejší typy umístění vypadají takto:

```
[ 2 0 R /XYZ 72 72 ]
[ 2 0 R /Fit ]
```

První říká, že umístění je bod vzdálený jeden palec od levého i spodního okraje stránky skrývající se pod nepřímým objektem číslo 2. Aktuální přiblížení je zachováno. Druhé umístění není spjato s konkrétním bodem, ale celou stranou. Zároveň toto umístění nastaví přiblížení tak, aby se strana vešla na obrazovku.

Místo toho, aby bylo vždy nutné umístění udávat přímo (PDF polem), nebo klasickým nepřímým objektem („nepřímým“ PDF polem) je možné také v dokumentu pomocí jmenného stromu definovat zobrazení jmen (PDF řetězce) na nepřímé objekty umístění. To by se ze začátku mohlo zdát jako zbytečná komplikace, když lze rovnou použít nepřímé objekty. Má to ale své použití, protože to dovoluje při jednoprůchodovém zpracování dokumentu zavést pohodlnou formu asynchronity, kdy je možné se odvolat na umístění, které bude definováno až později.

2.1.4 Akce a skoky

Široké možnosti interakce nabízí ve formátu PDF takzvané „akce“ (*actions*, [9, část 12.6]). Podobně jako anotací, je i akcí více druhů. Nejtypičtější akcí v dokumentech jsou `/GoTo` akce, jejichž vyvolání způsobí skok na určené místo v dokumentu (tzv. „umístění“). Pro ilustraci je zde uvedena jednoduchá ukázka (`/GoTo`) akce, která realizuje skok na pojmenované umístění a pokračuje řetězovým vyvoláním další akce:

```
<<
  /Type /Action
  /S /GoTo
  /D (chapter1)
  /Next <<
    /Type /Action
    /S /Trans
    /Trans << /S /Split >>
  >>
>>
```

Již na tomto jednoduchém příkladu lze ukázat všechny obecné vlastnosti akcí. Akce je PDF objekt typu slovník, který obsahuje minimálně jednu položku: `/S`, kterou je určen druh akce. Na základě konkrétního druhu akce je pak možné (nebo spíš nutné) uvést další položky. V příkladu výše se jedná o `/GoTo` akci, která vyžaduje umístění jako položku `/D` s cílem skoku. Ten může být udán všemi různými způsoby, blíže popsanými v sekci 2.1.3. Zjednodušeně řečeno se po provedení tohoto skoku vykoná akce z položky `/Next`. V tomto případě je další akcí tzv. přechodová akce (*transition action*, detailněji dále v sekci 2.1.6). Interakce přechodů v řetězu akcí je malinko složitější, ale toto pořadí dává očekávaný výsledek skoku s přechodem. Položka `/Type` se ze slovníku akce typicky vynechává, protože druh slovníku lze odvodit z kontextu. V ukázce je uveden pouze pro úplnost.

PDF akce lze víceméně typově rozdělit do několik skupin:

- *Skokové akce.* Do této skupiny se řadí akce `/GoTo` (skok na umístění v tomto dokumentu), `/GoToR` (skok na umístění v jiném dokumentu), `/URI` (skok do umístění určeného pomocí URI – *Uniform Resource Identifier*). Také sem lze zařadit přechodové (`/Trans`) akce, které dávají smysl ve zřetězení se skokovými akcemi.

  `/GoTo` akce poměrně kompletně již představila předchozí ukázka. `/GoToR` akce se liší tím, že navíc odkazuje na externí dokument určený pomocí „specifikace souboru“ (*file specification*, detailněji v sekci 2.2.4). Preferovaným způsobem určení umístění v tomto externím dokumentu je pojmenované umístění, které vyhodnotí PDF prohlížeč. Pro manuálnější určení umístění je možné použít i pole s tím rozdílem, že místo nepřímého odkazu na stránku je potřeba uvést *číslo* stránky (stránky jsou číslovány globálně od nuly).

  Ukázka `/URI` akce byla vidět v sekci 2.1.2. Jediným obvyklým parametrem je `/URI`, který udává konkrétní identifikátor.

  Následující srovnání `/GoToR` a `/URI` akce ukazuje, že obě dokáží v určitých kontextech poskytnout podobný výsledek (skok do konkrétního pojmenovaného umístění v externím dokumentu nacházejícím se na internetu):

```
<<
  /S /GoToR
  /F <<
    /FS /URL
    /F (http://pdf.example/example.pdf)
  >>
  /D (ref:chapter1)
>>
```

```
<<
  /S /URI
  /URI (http://pdf.example/example.pdf#chapter1)
>>
```

- *Multimediální akce.* Ty jsou popsány odděleně v příslušných sekcích (2.2.3, 2.2.4, 2.2.5, 2.2.6 a 2.2.7).
- *Formulářové akce.* Jsou mimo rozsah této práce.
- *JavaScriptové akce.* Umožňují spuštění kusu JavaScriptu v globálním kontextu celého dokumentu. Tato instance interpretu JavaScriptu je úplně nezávislá od dalších instancí interpretu JavaScriptu, které jsou použity pro 3D díla (viz sekci 2.2.6). Jako doplňující prostředek k těmto akcím je také možné do PDF dokumentů vkládat tzv. „document level JavaScript“. Tyto kusy kódu se spustí při otevření dokumentu a je v nich tak možné provést inicializaci a definovat funkce používané později v JavaScript akcích. Dostupné JavaScript API (*application programming interface*) je popsáno v [1].
- *Speciální akce.* Jedná se o specializované akce, kterémi se tato práce dále zabývat nebude. Výjimku tvoří `/Launch` akce a `/Named` akce. `/Launch` akce umožňuje spuštění libovolných externích programů! To může mít důležité bezpečností následky. Podporou této akce v PDF prohlížečích se věnuje sekce 5.1.6. „Pojmenovaných“ (`/Named`) akcí může být v podstatě neomezeně – každý prohlížeč může definovat svoje. Norma definuje čtyři základní – skok na další (`/NextPage`), předchozí, první a poslední stranu.

Na závěr je nutné poznamenat, že na některých místech, kde je možné uvést akci, je jako alternativu možno uvést přímo umístění. Chování je pak identické, jako kdyby

byla použita `/GoTo` akce. Detaily jsou o trochu složitější, ale v podstatě nezajímavé. Mnohem výhodnější je totiž pro lepší abstrakci uvažovat o skocích v dokumentu jen jako o speciálním případu akce – je tak možné je zaměnit, řetězit, atd.

Před konkrétnějším uvážením jednotlivých druhů akcí je dobré zmínit, jak vlastně akce vypadá a jak může dojít k jejímu vyvolání. Akce je PDF objekt typu slovník, který obsahuje minimálně jednu položku: `/S`, kterou je určen druh akce. Na základě konkrétního druhu akce je pak možné (nebo spíš nutné) uvést další položky. K vyvolání akce potom může dojít několika různými způsoby:

- Úvodní akce dokumentu. Je nastavena položkou `/OpenAction` v katalogu dokumentu. Touto akcí lze předefinovat výchozí chování otevření první stránky PDF dokumentu.
- Kliknutí na záložkou (outline). Záložky jsou detailněji popsány v sekci 2.1.5.
- Kliknutí na anotaci typu Link. Jediný účel `/Link` anotací je při aktivaci anotace spustit asociovanou akci.
- Reakce na události. Stránky a některé druhy anotací mohou mít definovány prostřednictvím položky `/AA` tzv. „přídavné akce“. Ty reagují na „události“ (*trigger events*), například „kurzor se objevil na ploše anotace“ (`/E`), stránka s anotací byla otevřena (`/PO`), stránka přestala být viditelná (`/PI`), atd.

2.1.5 Záložky

Záložky (*outlines*, [9, část 12.3.3]) jsou určeny především jako možnost snadné navigace v PDF dokumentu pomocí hierarchické struktury. Pojmem z teorie grafů je možné záložky charakterizovat jako *strom*. Každý uzel stromu je potom v PDF reprezentován slovníkem s odkazy na nadřazený uzel, předcházející uzel na stejné úrovni, prvního a posledního potomka. Kořen stromu je odkázán z katalogu dokumentu, položkou `/Outlines`.

Typicky uzly v první hladině (přímí potomci kořenu) představují kapitoly dokumentu a potomci v dalších hladinách různě zanořené (pod)sekce. Akce asociované se záložkami přitom většinou představují skok na místo, kde kapitola/sekce začíná. Nic ale nebrání při tvorbě PDF vytvořit outlines jakkoliv jinak.

2.1.6 Přechody

Kromě přechodů dostupných jako akce pro řetězení se skoky (viz sekci 2.1.4) slouží přechody především jako atributy *stránek* [9, část 12.4.4]. Každá stránka tedy může mít vlastní určený přechod (vizuální efekt), který se pro ni použije, ať už se na ni člověk dostane jakkoliv. Co se stane v případě skoku s přechodem na stránku, která má nastavený vlastní přechod, není nijak specifikováno.

Další atribut stránek, podobně určený především pro prezentace přes celou obrazovku, je možnost automatického přepínání na další stránku. To je možné díky parametru stránky `/Dur`, kterým lze určit na kolik sekund zůstane stránka viditelná, poté se přepíná na další. Do času zobrazení stránky se nepočítá stav „polozobrazení“ při přechodu.

Konečně ještě k možnostem různých přechodů. Přechody jsou určeny slovníkem typu `/Trans`, ve kterém je možno nastavit konkrétní styl animace (`/S`), např. `/Split` (přes stránku přejedou dvě linky, které postupně odhalují novou stránku) nebo `/Slides` (nová stránka přijede z jedné strany a nahradí stávající stránku). Každý styl může mít další uzpůsobitelné parametry, zejména směr pohybu. Některé parametry jsou shodné pro více stylů. Konečně i pro přechody lze nastavit dobu po kterou trvá. Lze tak stejnou animaci zrychlit/zpomalit oproti výchozímu času jedné sekundy.

Ukázka stránky (detailněji o stránkách pojednává sekce 2.2.1), která má nastavený přechod typu `/Dissolve` v době trvání 4 sekund a samotná stránka zůstane na obrazovce jen 3 sekundy po dokončení přechodu. (Dohromady tedy bude stránka alespoň částečně viditelná 7 sekund).

```
<<
  /Type /Page
  /Parent 18 0 R
  /Contents 190 0 R
  /MediaBox [ 0 0 595.276 419.528 ]
  /Trans <<
    /S /Dissolve
    /D 4
  >>
  /Dur 3
>>
```

Přechod s animací typu `/Box` ve směru „ven“ (`/M /O`), tentokrát v podobě akce by mohl vypadat následovně:

```
<<
  /S /Trans
  /Trans <<
    /S /Box
    /M /O
  >>
>>
```

Stránky v PDF jsou listy ve stromové struktuře. Některé atributy stránek nastavené ve vyšších úrovních se dědí. Bohužel to není případ atributů `/Trans` a `/Dur`, ty je nutno nastavit pro každou stránku explicitně (tedy v listech, samotných objektech typu `/Page`).

2.1.7 Komentáře a přílohy

Anotace typu `/Text` („komentáře“) a `/FileAttachment` (souborové přílohy) patří do skupiny „markup anotací“, tedy těch, které jsou určeny především pro dodatečné značkování čtenářem. Obě anotace proto mohou mít položky jako například jméno uživatele, který je přidal, čas přidání nebo stav schválení. Třebaže to nesouvisí přímo s účelem, pro který používám TEX, jsou často implementovány v TEXových balíčích.

Tyto anotace mají každá svůj výchozí vzhled – čtvercovou ikonku s nějakým symbolem (například připínaček pro přílohy). Existuje pro ně ale více předdefinovaných vzhledů, které lze nastavit pomocí `/Name` (například `/Name /Key` pro ikonu klíče v případě komentáře). Nebo lze nastavit vzhled klasicky pomocí „appearance“ (viz sekci 2.1.2). Použití výchozího nebo pojmenovaného vzhledu je z principu problematické, protože jsou závislé na konkrétním PDF prohlížeči.

Komentářové anotace představují komentář – nějaký text, který je schovaný pod ikonkou. Po rozkliknutí se anotace „rozbalí“ a dojde k zobrazení textu. Lze nastavit, že ve výchozím stavu je anotace rozbalená.

Přílohové anotace odkazují na specifikaci souboru (viz sekci 2.2.4, kde jsou detailněji popsány). Mohou tedy odkazovat na soubory vložené v PDF nebo soubory určené pomocí cesty/URL. Při rozkliknutí této anotace by se měl soubor otevřít.

2.2 Multimédia

Multimédia ve formátu PDF jsou poměrně nešťastný příběh. Během historie tohoto formátu se objevilo několik různých způsobů jak je vkládat, každý složitější a ne striktně lepší. V této sekci jsou jednotlivé způsoby popsány s cílem jejich srovnání z teoretické stránky. I když se tato práce zaměřuje především na audio, video a 3D grafiku, má cenu podívat se na to, jak v PDF funguje vkládání klasické vektorové i rastrové (bitmapové) grafiky. To dovolí následující srovnání různých mechanismů. Ukáže se tak například, jaké nároky to klade na implementaci programů, které PDF soubory čtou nebo vytvářejí.

2.2.1 Formy, stránky

Jako doma je v PDF vektorová grafika. Stránka samotná [9, část 7.7.3.3] je totiž v podstatě také jen vektorový „obrázek“. Typicky vypadá následovně:

```
6 0 obj
<<
  /Type /Page
  /MediaBox [ 0 0 200 100 ]
  /Resources 3 0 R
  /Contents 5 0 R
  /Parent 2 0 R
>>
endobj
```

V tomto slovníku jsou následující zajímavé položky:

- `/MediaBox` – udává obdélník stránky (většinou od bodu $(0, 0)$ do (*šířka*, *výška*));
- `/Resources` – slovník obsahující odkazy na všechny externí objekty potřebné k vykreslení stránky (např. fonty);
- `/Contents` – obsah stránky, tj. stream grafických instrukcí pro vykreslení stránky;
- `/Parent` je povinný odkaz na rodiče ve stromové struktuře stránek, pro nás aktuálně nezajímavé.

Obsah stránky (`/Contents`) může vypadat například následovně:

```
5 0 obj
<<
  /Length 84
>>
stream
q
0.5 G
0 0 m
200 100 l
0 100 m
200 0 l
S
Q
BT
/F1 18 Tf
20 40 Td
(PAGE TEXT) Tj
ET
endstream
endobj
```

Na této stránce se nejprve poloviční šedou vykreslí obě úhlopříčky. Poté se nastaví písmo `/F1` ve velikosti 18 bodů a na souřadnicích $(20, 40)$ se vykreslí text „PAGE TEXT“. Pro úplnost je potřeba také definovat `/Resources`, kde je popsáno, co vlastně `/F1` znamená:

```
3 0 obj
<<
  /Font <<
    /F1 4 0 R % odkaz na objekt typu /Font, mimo téma této kapitoly
  >>
>>
endobj
```

Každý externí zdroj v `/Resources` dostává svůj název. Žádný jiný zdroj použít nelze. Výhodou je, že PDF prohlížeče tak dokáží rychle pro jakoukoliv stránku najít vše potřebné.

Podobně jako stránku lze definovat takzvanou „formu“ (*form XObject*, [9, část 8.10]), klidně se stejným obsahem a dokonce se stejnými externími zdroji:

```
7 0 obj
<<
  /Type /XObject
  /Subtype /Form
  /BBox [ 0 0 200 100 ]
  /Resources 4 0 R
  /Length 84
>>
stream
[...]
endstream
endobj
```

Tuto formu je možné využít jako externí zdroj v jiných stranách (nebo formách) jejím pojmenováním a odkázáním v příslušných `/Resources`. Formu je pak možné vyvolat operátorem `Do`.

Další stránka, která čtyřikrát vloží tu samou formu v poloviční velikosti může vypadat takto:

```
8 0 obj
<<
  /Type /Page
  /MediaBox [ 0 0 200 100 ]
  /Resources <<
    /XObject <<
      /X1 7 0 R
    >>
  >>
  /Contents 9 0 R
  /Parent 2 0 R
>>
endobj

9 0 obj
<<
  /Length 126
```

```
>>
stream
q 0.5 0 0 0.5 0 0 cm /X1 Do Q
q 0.5 0 0 0.5 100 0 cm /X1 Do Q
q 0.5 0 0 0.5 100 50 cm /X1 Do Q
q 0.5 0 0 0.5 0 50 cm /X1 Do Q
endstream
endobj
```

Popsaný mechanismus nám tedy v podstatě umožňuje definovat si vektorovou grafiku odděleně a vkládat ji do stránek jako externí objekt. U stránek je potom možné zabývat se pouze samotnou sazbou textu.

2.2.2 Obrázkové XObjekty

Rastrová grafika funguje velice podobně jako formy. Také se jedná o XObject, tentokrát se `/Subtype /Image` (*image Xobject*, [9, část 8.9.5]). Podobně jako u formy jsou obrázková data uložena v streamu. Není to ale tak, že by mezi klíčovými slovy `stream` a `endstream` byla například data ve formátu JPEG nebo PNG, která by musel PDF prohlížeč umět interpretovat (rozpoznat typ souboru; přečíst hlavičku s údaji jako rozměry obrázku, barevný prostor, počet bitů na barevnou komponentu; ale hlavně extrahovat samotná obrázková data).

Ve skutečnosti formát PDF nepředepisuje podporu pro žádný formát rastrové grafiky. Místo toho jsou metadata obsažena přímo jako údaje ve slovníku obrázkového XObjektu a stream samotný pak obsahuje syrová obrázková data – pixel po pixelu, barevnou komponentu po barevné komponentě. To pochopitelně zabere spoustu místa. Je ale nutné mít na paměti, že na tato data je možné aplikovat kompresi, jako na každý jiný stream. Konečně, protože v PDF formátu jsou definovány komprese DCT („Discrete cosine transform“) a Flate, používané v JPEGu, respektive PNG, je tak možné tyto obrázky vkládat téměř přímo. Úkolem tvůrců PDF je tak přečíst hlavičky souborů, extrahovat obrázková data a vytvořit z nich odpovídající obrázkový XObjekt s metadaty ve slovníku a obrázkovými daty ve streamu.[2]

Slovník obrázkového XObjektu by mohl vypadat například takto:

```
<<
  /Type /XObject
  /Subtype /Image
  /Width 512
  /Height 512
  /BitsPerComponent 8
  /Length 106738
  /ColorSpace /DeviceRGB
  /Filter /DCTDecode
>>
```

2.2.3 Sound objekty

Sound objekty [9, část 13.3] se objevily ve specifikaci PDF 1.2. Jedná se o poměrně jednoduchý koncept, který umožňuje přehrávat audio. Funguje na třech úrovních:

1. sound objekty – streamy, které definují přehratelnou audio stopu;

[2] Ve skutečnosti to u PNG formátu tak jednoduché není, protože obsahem souboru nejsou přímo obrázková data.

2. sound anotace, které asociují sound objekt s ikonkou na stránce, která umožňuje zahájení přehrávání;
3. sound akce, které při svém vyvolání přehrají sound objekt.

Sound objekty jsou streamy prinicipiálně podobné obrázkovým XObjektům. V streamu jsou syrová audio data ve formátu PCM (*Pulse-code modulation*, pulzně kódová modulace). Ve slovníku streamu potom jsou konkrétní parametry kódování, která PDF prohlížeči umožní audio stopu dekódovat. Takový Sound objekt může vypadat třeba následovně:

```
6 0 obj
<<
  /Type /Sound
  /Length 60480
  /R 11025
  /C 1
  /B 8
  /E /Raw
>>
stream
[...]
endstream
```

```
7 0 obj
<<
  /Type /Sound
  /Length 967680
  /R 44100
  /C 2
  /B 16
  /E /Signed
>>
stream
[...]
endstream
```

Platí, že PCM data obsažená v streamech odpovídají vzorkování analogového audio signálu. Přitom u levé ukázky každý vzorek nabývá hodnoty v rozmezí 0 až 255 (parametry `/B` a `/E`) a vzorkovací frekvence (počet vzorků pro jednu vteřinu audia) činí 11 025 Hz. Blíže skutečným formátům je ukázka vpravo. V ní je pro kvantování použito rozpětí čísel −32768 až 32767. Čísla samotná jsou zakódovaná ve znamenénkové 16bitové Big Endian hodnotě. Dále pravá ukázka využívá dva kanály a vzorkovací frekvenci 44 100 Hz.

Hned několik audio formátů ukládá data takto přímo, navíc přidávají jen hlavičku (případně další metadata) a častěji ukládají čísla v Little Endian formátu. Jedná se například o formáty `.wav`, `.au` a `.aiff`.

Pro převod audia do podoby, která by se dala použít v předchozích ukázkách, je možné použít svobodný nástroj `ffmpeg`, který umí vstupní soubor v cestě `IN` převést do odpovídajícího formátu a uložit do umístění určeného cestou `OUT` například následovně:

```
ffmpeg -i IN -f u8    -c:a pcm_u8    -ar 11025 -ac 1 OUT
ffmpeg -i IN -f s16be -c:a pcm_s16be -ar 44100 -ac 2 OUT
```

Obecně může každý stream objekt odkazovat pomocí slovníkové položky `/F` na soubor, který může být buď úplně mimo PDF (pak je udaný svou cestou či dokonce obecnou URL adresou) nebo na takzvaně vložený soubor (stream objekt typu `/EmbeddedFile`). Sound objekt je definován tak, že v případě výskytu položky `/F` musí tato odkazovat na externí soubor a všechna metadata k přehrání tohoto souboru by měla být obsažena přímo v něm, nikoliv jako položky ve slovníku sound objektu. Norma uvádí jako příklady vhodných formátů právě dříve jmenované (`.wav`, `.au` a `.aiff`), ale skutečná podpora jakéhokoliv konkrétního formátu nijak vymezena není.

Menší nejistota se týká také předešlého „raw“ formátu. Podpora konkrétních hodnot parametrů `/R`, `/C`, `/B`, `/E` není zaručena. Vůbec nejzvláštnější jsou další dva parametry „raw“ formátu, `/CO`, `/CP`, které mohou udat formát zvukové komprese. Tato komprese by měla proběhnout úplně nezávisle na standardních možnostech komprese pomocí položky `/Filter` (viz sekci 2.2.2). Žádná konkrétní komprese ale normou definována

není. Obecně nejjistější se tak zdá použití „raw“ formátu s doporučenými parametry (tomu odpovídá třeba levá ukázka sound objektu).

V případě připraveného sound objektu je možné ho použít v anotaci nebo akci. Sound anotace, která zabírá čtvercovou oblast v levé dolní oblasti stránky, jejíž vzhled je ikona reproduktoru a po aktivaci (kliknutí myší) začne přehrávat audio stopu ze sound objektu, může vypadat takto:

```
5 0 obj
<<
  /Type /Annot
  /Subtype /Sound
  /Rect [ 0 0 72 72 ]
  /Contents (Textovy popis anotace.)
  /Name /Speaker
  /Sound 6 0 R
>>
endobj
```

Nebo je možné v katalogu dokumentu definovat akci, která po otevření PDF souboru začne ve smyčce přehrávat sound objekt:

```
1 0 obj
<<
  /Type /Catalog
  /Pages 2 0 R
  [...]
  /OpenAction <<
    /Type /Action
    /S /Sound
    /Sound 6 0 R
    /Volume 0.1
    /Repeat true
  >>
>>
```

Je vidět, že koncept Sound objektů/anotací/akcí je docela omezený. Například není jednoduše možné předčasně zastavit přehrávanou audio stopu. Jediná možnost je vyvolání jiné sound akce a to tak, že minimálně jedna z nich bude mít zakázáno slučování audia (parametr `/Mix` sound anotace, výchozí hodnota je `false`).

2.2.4 Movie objekty

Movie objekty [9, část 13.4] se objevily současně se sound objekty ve specifikaci PDF 1.2. Od sound objektů se ale koncepčně liší. Neintuitivně totiž neslouží pouze k přehrávání videí, ale mohou sloužit i k přehrávání audia bez jakéhokoliv obrazu. (Pod „movie“ bude v této práci myšleno jakékoliv video+audio, video či audio). Movie objekty jsou označeny jako překonané již ve specifikaci PDF verze 1.5, kdy se objevil koncept multimédií, viz sekci 2.2.5.

Pro movie mechanismus jsou definovány čtyři relevantní PDF objekty:

1. movie slovníky,
2. aktivační movie slovníky,
3. movie anotace,
4. movie akce.

Prostřednictvím movie slovníku je definován soubor, který bude přehrán, a také tzv. statické parametry (neovlivňující přehrávání, jako poměr stran, rotace, či „plakát“, který bude tvořit náhledový obrázek). Aktivační movie slovník naopak definuje parametry přehrávání (začátek, doba, rychlost, hlasitost, ...). Movie anotace potom obsahuje oba slovníky a asociuje takto definované movie s konkrétní oblastí stránky. Movie akce může přehrávání ovlivňovat – její hlavní parametr je `/Operation`, který přepne movie do konkrétního stavu přehrání (`/Play`, `/Pause`, `/Stop`, `/Resume`). V akci je možné ale i zopakovat některé z parametrů aktivačního movie slovníku a změnit tím jejich hodnoty.

Jedná se tedy o univerzálnější mechanismus, který zároveň obsáhne video a audio. Navíc je mnohem lépe ovladatelný a přitom stále relativně jednoduchý. Přesto není bezproblémový. To co se vlastně přehraje, není určeno stream objektem (jako v případě vektorové/rastrové grafiky a audia), ale položkou `/F` v movie slovníku. Ta má typ „specifikace souboru“ (*file specifiaction*). To v nejjednodušším případě může znamenat řetězec s cestou k souboru (např. `(movie.mp4)`). Ve složitějším případě se může jednat o slovník, který může odkazovat i na soubor vložený do PDF souboru jako stream nebo soubor na internetu (obdobně je to také s položkou `/F` u streamů, která byla popsána v sekci 2.2.3). V případě sound objektů norma přímo zakazovala, aby položka `/F` směřovala na vložený soubor. U movie objektů tomu tak není.

Z celého návrhu těchto objektů lze odvodit, že movie objekty spadají do té kategorie anotací, která je určena hlavně pro primární autory dokumentu. Není moc myšleno na to, aby movie anotaci vkládali čtenáři PDF souboru dodatečně. Definice takové anotace při jednoduchém odkazu na externí soubor může vypadat následovně:

```
5 0 obj
<<
  /Type /Annot
  /Subtype /Movie
  /Rect [ 0 0 480 270 ]
  /Contents (Movie annotation.)
  /Movie <<
    /F (video.mp4)
    /Aspect [ 480 270 ]
  >>
  /A <<
    /ShowControls true
  >>
>>
endobj
```

Pod položkou `/Movie` se nachází movie slovník, kde je povinný parametr odkazu na soubor (ve zjednodušené řetězcové variantě) a také rozměry movie (požadované pro video). Položka `/A` pak udává aktivační slovník movie, kde je vyžádáno zobrazení ovládacích prvků přehrávače videa (ostatní parametry jako čas začátku, hlasitost, rychlost přehrávání, atd. získávají výchozí hodnoty).

Pro vložení video souboru z předchozí ukázky do PDF musíme použít složitější variantu specifikace souboru pomocí slovníku. Ten může být například v nepřímem objektu s číslem 6 a odkazovat na soubor vložený jako `/EmbeddedFile` v stream objektu s číslem 7:

```
5 0 obj
<<
  /Type /Annot
```

```
  /Subtype /Movie
  [...]
  /Movie <<
    /F 6 0 R
    /Aspect [ 480 270 ]
  >>
  [...]
>>
endobj

6 0 obj
<<
  /Type /Filespec
  /F (video.mp4)
  /EF <<
    /F 7 0 R
  >>
>>
endobj

7 0 obj
<<
  /Type /EmbeddedFile
  /Subtype /video#2Fmp4
  /Length 52523
>>
stream
[...]
endstream
```

V objektu vloženého souboru (`7 0 obj`) je v této ukázce uveden i „MIME type“[3]. To pro použití s movie není vyžadováno, ale je to užitečné v případě, že je stejný vložený soubor prezentován jako „příloha“ (*attachment*). PDF prohlížeč totiž v tomto jiném kontextu nemusí vědět, že se jedná o soubor s přehratelným videem.

Další možnost je odkázat na soubor z internetu. To je také realizováno slovníkovou variantou specifikace souboru. Nepřímý objekt č. 6 je proto možné nahradit následovně:

```
6 0 obj
<<
  /Type /Filespec
  /FS /URL
  /F (https://fit.cvut.cz/video.mp4)
>>
endobj
```

Pro zastavení přehrávání ve chvíli, kdy čtenář opustí stránku s movie anotací, lze definovat nejprve (nepřímý) objekt s movie akcí, která zařídí samotné zastavení videa:

```
8 0 obj
<<
  /Type /Action
  /S /Movie
  /Annotation 5 0 R
  /Operation /Stop
```

[3] `https://tools.ietf.org/html/rfc2045`

```
>>
endobj
```

Spuštění této akce například při opuštění stránky (*page close*) lze docílit pomocí slovníku přídavných akcí (*additional actions*, `/AA`). Přídavné akce lze definovat na stránce nebo na některých druzích anotací. Výhodnější je akci definovat jako přídavnou akci přímo u anotace – pak je anotace soběstačný objekt. Bohužel movie anotace nemůže mít přídavné akce, proto je potřeba se v tomto místě spokojit s přidáním obsahu do objektu stránky:

```
2 0 obj
<<
  /Type /Page
  [...]
  /AA <<
    /C 8 0 R
  >>
>>
endobj
```

2.2.5 Multimédia (Renditions)

„Multimedia“ je název pro mechanismus [9, část 13.2] uvedený v PDF verzi 1.5. Ten je mnohem obecnější, než dříve popsané starší movie a sound objekty. Dokonce je tak obecný, že teoreticky podporuje „přehrávání“ jakéhokoliv multimédia – explicitně jsou zmíněny kromě audia a videa třeba obrázky nebo flash soubory. Podporováno je přehrávání vložených i externích médií. Samotné přehrávání může probíhat na stránce nebo v nově vytvořeném okně. Je specifikován rozsáhlý mechanismus, který umožňuje specifikovat více médií a jejich parametrů – PDF prohlížeč potom může vybrat nejvhodnější (např. podle rychlosti připojení k internetu).

Kromě rozsáhlých možností konfigurace tohoto mechanismu přes PDF slovníky, existuje i možnost programovat tyto objekty za pomoci JavaScriptu. Konkrétní API definuje dokument společnosti Adobe [1].

Nepříjemná část mechanismu multimédií je koncepce „přehrávačů“ (*media players*). Myšlenka je taková, že přehrávačem multimédií ve většině případů není přímo PDF prohlížeč samotný, ale nějaký jeho zásuvný modul (*plug-in*). Norma tak definuje celou řadu dalších vnořených slovníkových struktur, které umožňují v rendition objektu specifikovat, jaký přehrávač použít pro přehrání. To je podle mého názoru velmi nešťastné řešení. Nejen proto, že norma vůbec žádné přehrávače nespecifikuje. To dělá dokumenty velmi nepřenositelné. Následující ukázky koncept přehrávačů obchází trochu pochybným způsobem – norma stanovuje, že pokud je specifikován MIME typ multimédia, pak by měly (*should*) být specifikovány povolené a zakázané přehrávače – požadavek tedy není striktní a v praxi tato neposlušnost prochází. Společnost Adobe mechanismus multimédií již poměrně dlouho považuje za zastaralý a nedokonalý právě kvůli závislosti na pluginech – to se svými vlastními chybami mění Rich Media popsaná v sekci 2.2.7. Dalšími problematickými aspekty multimédií jsou složité struktury slovníků, které obsahují položky s kryptickými názvy (především jedno a dvoupísmenné). Navíc někdy je vyžadováno duplicitní uvedení jedné informace, nebo některé slovníky neodpovídají zvyklostem formátu PDF (např. použití řetězců místo symbolických názvů, *names*).

Opět jsou ve věci tři základní druhy objektů – *rendition objekty*, *screen anotace* a *rendition akce*. V rendition objektech je schována téměř veškerá složitost. Screen anotace v podstatě jen rezervují místo, na kterém dojde k přehrávání multimédií (jako u všech

anotací pomocí položky /Rect). Navíc ale může definovat akce, které se spustí při kliknutí na anotaci (její *aktivaci*, položka /A) a některých dalších událostech (položka /AA). Jako akce se přitom přímo nabízí rendition akce, prostřednictvím které lze přehrávání multimédia spustit, pozastavit či zastavit (položka /OP). Jako alternativa může být místo položky /OP uvedena položka /JS, která spustí přiložený JavaScript kód.

Jednoduchý příklad screen anotace, který připraví prostor na stránce a zároveň prostřednictvím rendition akce po aktivaci začne přehrávat odkazovaný rendition objekt, může vypadat takto:

```
6 0 obj
<<
  /Type /Annot
  /Subtype /Screen
  /Rect [ 0 0 480 270 ]
  /P 4 0 R
  /AP <<
    /N 5 0 R
  >>
  /A <<
    /Type /Action
    /S /Rendition
    /R 7 0 R
    /AN 6 0 R
    /OP 0
  >>
>>
endobj
```

Na ukázce je zajímavá položka /P, která je specialitou pro screen anotace a musí odkazovat na objekt stránky, kde je tato anotace umístěna. Dále je této anotaci přiřazen konkrétní „normální“ (/N) vzhled (*appearence stream*, /AP), který je zobrazen před aktivováním anotace a v tomto případě slouží jako náhled videa.

Rendition akce obecně může asociovat jakýkoliv rendition objekt („co a jak se má přehrát“) s jakoukoliv screen anotací („kde se to má přehrát“). Právě proto rendition akce obsahuje nepřímé odkazy na oba z nich. To se v této zjednodušené ukázce, kdy je navíc akce uvedena jako přímý (*direct*) objekt, jeví poněkud zbytečné. Hodnota 0 položky /OP v tomto případě docílí zahájení přehrávání rendition objektu.

Rendition objektů existují dva druhy – médiový rendition objekt (*media rendition*) a výběrový rendition objekt (*selector rendition*). Obecně dohromady mohou tvořit stromovou strukturu – výběrové objekty totiž pomocí položky /R odkazují na pole dalších výběrových či médiových rendition objektů. Listy jsou potom vždy médiové rendition objekty. Je možný i jednoduchý případ, kdy místo obecného stromu je použit pouze jeden médiový rendition objekt. Smysl celé této struktury spočívá v tom, že každý rendition objekt specifikuje svoji přijatelnost (*viability*). PDF prohlížeč v případě, že dojde k požadavku přehrát rendition objekt, provede na stromu prohledávání do hloubky (DFS, *Depth First Search*) a přehraje první přijatelný objekt.

Pro přijatelnost rendition objektů slouží především ve slovníkové struktuře různě zanořené položky /BE („zkus dodržet“, *best effort*) a /MH („musíš dodržet“, *must honour*). Obě tyto položky jsou slovníky a na stejné úrovni ve slovníkové struktuře mohou obecně obsahovat stejné podpoložky, které rozhodují o přijatelnost objektů. V případě, že PDF prohlížeč nedokáže splnit některou podpoložku ze slovníku /MH, je objekt ihned označený za nepřijatelný (*non-viable*). Naopak v případě nemožnosti splnit něco z /BE,

nedojde k ovlivnění přijatelnosti, ale i tak by se PDF prohlížeč měl pokusit požadavek dodržet.

Tato bakalářská práce se nebude přijatelností objektů příliš zabývat. Všechny položky budou uváděny pouze v `/BE`. Hlavní důvod je, že v obvyklých případech, jakožto i v následujících ukázkách, je použit jen jeden médiový rendition objekt a žádný výběrový. Proto nedává moc smysl ztratit přijatelnost takového objektu nastavením nutně splnitelných kritérií. Obzvlášť pak protože rozsáhlost tohoto „multimedia“ mechanismu (20 stran normativního textu oproti např. 2 pro „movie“) nedává velké naděje na plnou podporu v prohlížečích.

Pro doplnění předchozí ukázky je zde definován nepřímý media renditon objekt č. 7:

```
7 0 obj
<<
  /Type /Rendition
  /S /MR
  /C <<
    /Type /MediaClip
    /S /MCD
    /D 8 0 R
    /CT (video/mp4)
    /P <<
      /Type /MediaPermissions
      /TF (TEMPALWAYS)
    >>
  >>
  /P <<
    /Type /MediaPlayParams
    /BE <<
      /C true
    >>
  >>
  /SP <<
    /Type /MediaScreenParams
    /BE <<
      /O 0.0
    >>
  >>
>>
endobj
```

Důležitá je položka `/S`, která uvádí, že se jedná o médiový renditon objekt (`/MR`), nikoliv výběrový (`/SR`). Protože je tento objekt odkazován přímo z rendition akce, tak vlastně tvoří kořen i list stromu a je tak jediný rendition objekt, který pro přehrání připadá v úvahu. Základem médiového slovníku jsou potom tři základní slovníky `/C` („co přehrát“), `/P` („jak přehrát“) a `/SP` („kde přehrát“). Bohužel určení těchto slovníků není úplně dokonalé a tak se mírně prolínají.

Nejzajímavější položkou je `/C`, ta specifikuje konkrétní multimédium k přehrání. Existují zde dvě možnosti – buď přehrání konkrétního „média klipu“ (*media clip*) nebo jeho části (*media clip section*). V této ukázce je jednodušší varianta, která k přehrání specifikuje celý jeden video soubor. V takovém případě je nutná položka `/D` – odkaz na specifikaci souboru (detailně popsanou v sekci 2.2.4). Až zde také dochází k rozlišení, jaké multimédium se vlastně bude přehrávat, a to pomocí položky `/CT`, která uvádí

„MIME type“[4] odkazovaného souboru. Do slovníku /C je také poněkud nevhodně zařazen i vnořený slovník /P, který určuje práva, jež PDF prohlížeč dostává pro přehrání tohoto média. V této ukázce je dovoleno neomezené vytváření dočasných souborů.

Položka /P říká, jak bude probíhat přehrávání. Lze nastavit hlasitost, opakování, nebo třeba jak se multimédium zobrazí, pokud jeho obdélník neodpovídá poměrem stran obdélníku screen anotace. V této ukázce je vyžádáno zobrazení ovládacích prvků (/C).

Konečně položka /SP umožňuje nastavit parametry okna, ve kterém se bude multimédium přehrávat. To je celkem nezajímavé v případě, že je požadováno výchozí přehrávání v obdélníku definovaném screen anotací – to se nepočítá jako zvláštní okno. Nastavením položek tohoto slovníku (nebo správněji vnořeného slovníku /BE nebo /MH) jde vynutit přehrávání v novém okně, mimo stránku a PDF prohlížeč. Toto okno má své další parametry. V ukázce k tomu nedochází, místo toho je nastavena plná průhlednost pozadí umístěného do zbytku obdélníku screen anotace, který multimédium nevyužije. To je užitečné, protože v předchozí položce /P nebylo pro multimédium nastaveno roztažení ani oříznutí.

Pro úplnost je třeba ještě uvést jednu další položku, kterou může mít médiový i výběrový rendition objekt. Jedná se o tzv. slovník kritérií. Jeho účel je především výběr jednoho z více médiových objektů ve stromové struktuře a to pomocí dotazů na nastavení, které provedl čtenář PDF dokumentu. Je tak možné například požadovat, že dojde k přehrání multimédia pouze pokud uživatel vyžaduje titulky (položka /S). Rozšíření médiového rendition objektu by pak bylo následující:

```
7 0 obj
<<
  /Type /Rendition
  [...]
  /BE <<
    /C <<
      /Type /MediaCriteria
      /S true
    >>
  >>
>>
endobj
```

2.2.6 3D díla

Od verze 1.6 je součástí specifice formátu PDF koncept 3D děl (*3D Artwork*, [9, část 13.6]). Principem 3D děl je vyhrazení části stránky, ve které je prostřednictvím virtuální kamery a projekce do 2D možné i interaktivně zobrazit 3D scénu. Taková scéna může obsahovat libovolný počet 3D objektů (nemají nic společného s PDF objekty). Je dovoleno nastavovat některé parametry pro vykreslování jak jednotlivých 3D objektů, tak celé scény. Důležité pro 3D dílo je také tzv. pohled (*view*). Dílo má vždy jeden výchozí pohled, ale může jich mít i více – čtenář mezi těmito pohledy může přepínat buď pomocí uživatelského rozhraní poskytnutého PDF prohlížečem nebo se tak může stát prostřednictvím akcí přepínajících 3D pohled.

Kromě základních definic pomocí klasických PDF objektů tento mechanismus umožňuje využít i skriptování v JavaScriptu, které je popsáno v dokumentu zvlášť vydaném společností Adobe [3] (možné je například naskriptovat základní nastavení pohledu

[4] https://tools.ietf.org/html/rfc2045

nebo pohyb, rotaci a mizení 3D objektů). Je nutno podotknout, že každý 3D objekt má vlastní instanci interpretu JavaScriptu a zvlášť definované API. Jedná se v podstatě o úplně jiný svět než u klasických PDF JavaScript skriptů. Díky rozšířením z PDF verze 1.7 si lze při běžném použití vystačit pouze s PDF objekty a není potřeba psát skripty – právě na tuto variantu se zaměřuje tato sekce. Ani obyčejné zobrazení bez JavaScriptu není statické – čtenář může ve 3D scéně různě měnit pozici virtuální kamery i další parametry a zobrazení si tak měnit.

Základem vložení 3D děl do PDF souboru jsou čtyři druhy objektů:

1. 3D anotace, které na stránce vymezují obdélníkovou oblast, na které lze prohlížet 3D dílo skrze virtuální kameru;
2. 3D streamy, které obsahují samotné 3D dílo a parametry jeho zobrazení;
3. 3D pohledy, které popisují konkrétní pohled na 3D dílo;
4. akce typu `/GoTo3DView`, které umožňují změnit aktuální 3D pohled.

Relativně jednoduchá ukázka 3D anotace může vypadat takto:

```
6 0 obj
<<
  /Type /Annot
  /Subtype /3D
  /Rect [ 100 100 495 742 ]
  /3DA <<
    /A /PV
    /D /PI
  >>
  /3DD 7 0 R
  /3DV 2
>>
endobj
```

Zde je prostřednictvím slovníku aktivace 3D anotace (*3D activation dictionary*, `/3DA`) nastaveno, že se 3D anotace automaticky aktivuje (`/A`) při zobrazení stránky a deaktivuje (`/D`) při zneviditelnění stránky. Není tak uveden výchozí vzhled (podpoložka `/N` slovníku `/AP`), který je zobrazen pro neaktivní anotaci. Dále je už jen odkázáno na 3D stream (položka `/3DD`) a na třetí z 3D pohledů, který je v něm definovaný (položka `/3DV`).

3D stream potom míchá 3D data samotná s některými metadaty a údaji o zobrazení. Zjednodušená definice pro potřeby ukázky může vypadat takto:

```
7 0 obj
<<
  /Type /3D
  /Subtype /PRC
  /Length 0
  /F 8 0 R
  /VA [ 9 0 R 10 0 R 11 0 R ]
  /OnInstantiate 13 0 R
>>
stream
endstream
endobj
```

Jedná se o klasický stream objekt. Jeho obsahem může být U3D soubor (hned od PDF verze 1.6) nebo PRC soubor (prvně v Adobe rozšíření [2]; později také v PDF

2.0 [10]). V této ukázce je odkazováno (`/F`) na specifikaci souboru v nepřímém objektu, proto nejsou žádná data mezi `stream` a `endstream`, a délka je nulová. 3D stream také odkazuje na tři pohledy (pole `/VA`, *view array*), které jsou opět definovány v nepřímých objektech. Protože není explicitně řečeno, který pohled je výchozí (položka `/DV`, *default view*), tak se výchozím pohledem stane ten s indexem `0` ve `/VA` – v našem případě nepřímý objekt s číslem 9.

Položka `/OnInstantiate` odkazuje na stream obsahující JavaScript, který se spustí při vytvoření 3D díla. Tento JavaScript je spuštěn přímo v kontextu 3D díla. To je rozdíl oproti *JavaScript akcím*, které jsou spouštěny v jiné instanci interpretu JavaScriptu. Také ale mohou programovat 3D díla, jen musí explicitně přistupovat ke *3D kontextu*. Jde to ale i naopak – JavaScript z kontextu 3D díla může přistupovat ke stavu globálního JavaScript interpretu.

Skript pro změnu režimu vykreslování (viz dále) na tzv. „drátěný model“ může vypadat třeba takto:

```
13 0 obj
<<
  /Length 31
>>
stream
scene.renderMode = "wireframe";
endstream
endobj
```

Něco podobného bylo potřeba v dřívějších verzích PDF, kdy nebylo možné nastavit režim vykreslování a další parametry v strukturách PDF.

Následuje ukázka jak může vypadat 3D pohled:

```
9 0 obj
<<
  /Type /3DView
  /XN (Narys)
  /IN (Narys)
  /MS /M
  /C2W [ 1 0 0 0 1 0 0 0 1 0 0 -200 ]
  /CO 200
  /P <<
    /Subtype /O
  >>
  /BG <<
    /Type /3DBG
    /Subtype /SC
    /C [ .8 .8 .8 ]
  >>
  /RM <<
    /Type /3DRenderMode
    /Subtype /Solid
  >>
  /LS <<
    /Type /3DLightingScheme
    /Subtype /Cube
  >>
>>
endobj
```

Kromě názvu 3D pohledu, který se zobrazí v uživatelském rozhraní (`/XN`) má 3D pohled také interní název (`/IN`), který je použit při skriptování atd. Ostatní parametry lze rozdělit do několika skupin:

1. parametry definující umístění a orientaci virtuální kamery;
2. parametry, které určují, jak je 3D scéna vykreslena;
3. pole řezových slovníků (*cross section dictionaries*).

Nastavení virtuální kamery je složitější a matematičtější a bude popsáno později. Možnosti nastavení pozadí (`/BG`) jsou v celku omezené a jedinou možností je nastavit RGB barvu pomocí položky `/C`. Pro vykreslovací režim (`/RM`, (*rendering mode*) lze vybrat z několika předdefinovaných režimů určených symbolickými názvy – např. klasické pevné těleso (`/Solid`) nebo třeba `/Wireframe` (drátový model). Některé vykreslovací režimy umožňují nastavení dalších parametrů (barev, průhledností, ...), to ale značně záleží na konkrétním režimu. Poslední parametr 3D pohledu ovlivňující vykreslení je nastavení osvětlení scény. To nelze nastavit detailně, ale lze pouze vybrat jedno z předdefinovaných osvětlovacích schémat – např. `/None`, `/Day`, `/Cube`, `/CAD`.

Konečně je potřeba se vrátit k virtuální kameře. Pro pochopení její koncepce je nejprve potřeba vysvětlit, jak v PDF fungují 3D souřadnice. Jedná se o celkem přímočaré rozšíření fungování dvourozměrného kartézského systému souřadnic z obsahu stránky (*content streamu*) nebo formy. 3D souřadnice jsou uspořádaná trojice čísel, (x, y, z), které udávají polohu vůči odpovídající ose. Přitom počátkem se myslí bod $(0, 0, 0)$, osa x roste doprava, osa y nahoru a osa z směrem *do* stránky („od pozorovatele“). Jedná se tedy o levotočivou soustavu souřadnic. Pro zobrazení 3D díla ale připadá v úvahu hned několik systémů souřadnic, mezi kterými probíhají transformace pomocí transformačních matic. Transformace bodu (x, y, z) do bodu (x', y', z') je výsledkem maticového násobení maticí s 12 parametry:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} a & d & g & t_x \\ b & e & h & t_y \\ c & f & i & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} ax + dy + gz + t_x \\ bx + ey + hz + t_y \\ cx + fy + iz + t_z \\ 1 \end{pmatrix}$$

V PDF je tato matice zapsána jako pole: `[`$a\ b\ c\ d\ e\ f\ g\ h\ i\ t_x\ t_y\ t_z$`]`.

Transformací pro zobrazení 3D díla ve 2D probíhá hned několik:

- 3D transformace ze souřadnicového systému 3D díla do světového (*world*) souřadnicového systému (transformační matice je součást 3D souboru, je tak stejná pro všechny 3D pohledy);
- 3D transformace ze světového souřadnicového systému do systému souřadnic kamery (invertovanou transformační matici lze určit pro každý 3D pohled);
- projekce do 2D, viz dále.

Druhá z 3D transformačních matic není zadána sama o sobě. Místo toho je očekávána její inverze, tzv. *camera-to-world* matice. Vzhledem k povaze transformace si tak lze představit, že maticí je vlastně zadána poloha a orientace kamery ve světovém systému souřadnic. V kamerovém systému souřadnic se kamera nachází v bodu $(0, 0, 0)$ a směřuje do kladného směru osy z. Pro přímé uvedení této matice lze ve 3D pohledu použít položku `/C2W`. Je pak ale nutné uvést `/MS /M`, tedy že matice je uvedena přímo, narozdíl od možnosti `/MS /U3D`, která říká, že bude uvedena další položka, která specifikuje jak tuto matici nalézt ve 3D souboru. To je o něco méně flexibilnější možnost – nefunguje pro PRC soubory a předpokládá, že je známa vnitřní struktura U3D souboru, což při exportu z různých programů nemusí být pravda.

Posledním důležitým aspektem zobrazení 3D děl je projekce. Ta je specifikována slovníkem v položce `/P` 3D pohledu. Jsou podporovány dva druhy (`/Subtype`) projekce:

1. Pravoúhlé promítání (`/O`, *ortographic*). Průmětna (rovina, na kterou je promítáno) je rovnoběžná s rovinou XY – souřadnice z je zahozena.
2. Perspektivní/středové promítání (`/P`, *perspective*). Obecně zkresluje (vzdálenosti, rovnoběžnosti,...), ale je přirozenější – odpovídá pohledu lidského oka. Středem je zde virtuální kamera. Parametrem tohoto promítání je `/FOV` (*field of view*), zorný úhel, který může nabývat hodnoty 0 až 180 stupňů.

Oba druhy projekce podporují koncept blízké a vzdálené ořezové roviny, který umožňují zamezit zobrazení objektů, které jsou ke kameře příliš blízko nebo naopak příliš daleko. Ve výchozím stavu jsou tyto roviny určeny automaticky.

3D pohledy mají konečně ještě poslední zajímavý parametr – vzdálenost kamery od středu orbitu (`/CO`). Ta se uplatní, pokud čtenář PDF dokumentu začne s 3D scénou interagovat v orbitovém režimu.

Teprve se znalostí předešlých konceptů se lze vrátit k nastavení virtuální kamery a projekce z předchozí ukázky:

```
/MS /M
/C2W [ 1 0 0 0 1 0 0 0 1 0 0 -200 ]
/CO 200
/P <<
  /Subtype /O
>>
```

Tato transformační matice neprovádí žádnou lineární transformaci. Pouze „posouvá kameru“ směrem od počátku, stejná vzdálenost je také nastavena jako vzdálenost středu orbitu – ten tak efektivně padne do počátku. Tento pohled nelze nijak konkrétně pojmenovat bez znalosti toho, jak je model umístěn ve světovém souřadnicovém systému. Pokud ale autor modelu zvolil rovinu XY jako průmětnu pro hlavní pohled (obvykle nazývaný „nárys“), pak ho může popisovat právě výše zmíněná matice.

Pro 3D díla je dostupná akce `/GoTo3DView`. Prostřednictvím té je možné přepnout aktuální zobrazení 3D scény na jeden z pohledů vyjmenovaných v poli `/VA`. Přitom lze konkrétní pohled zvolit nejen obyčejně pomocí indexu (číslování je od nuly), ale i pomocí (interního) jména 3D pohledu (řetězec) nebo některého symbolického jména (třeba `/N`, *next*, „další pohled“).

Akce pro přepnutí do v této kapitole definovaného pohledu „Narys“ může vypadat takto:

```
<<
  /S /GoTo3DView
  /TA 6 0 R
  /V (Narys)
>>
```

Přepnutí probíhá v 3D dílu zobrazeném anotací skrývající se pod nepřímým objektem č. 6.

2.2.7 Rich Media

První specifikace Rich Media („bohatá média“) pochází z rozšíření [2] PDF verze 1.7, kterou vydala společnost Adobe v roce 2008. Bohatost Rich médií pravděpodobně odkazuje na široké možnosti propojení s technologií Flash – přehrávání SWF a Flash videa,

oboucestné skriptování mezi PDF (JavaScript) a Flashem (ActionScript) a spoustu dalšího. Kromě toho je přes Rich Media možné vkládání obecného videa, audia a také 3D děl.

Konec Flashe byl oznámen v roce 2017 a koncem roku 2020 skutečně nastal. Poměrně překvapivě se Rich Media stala součástí PDF 2.0 (2017, pozdější revize 2020 [10]). Podle volně dostupných zdrojů ale není součástí normy možnost použití Flashe ve spojitosti s Rich Media anotacemi. Tato práce vychází ze starší (volně dostupné) specifikace Rich Media. Neperspektivní využití Flashe není dále uvažováno. Zde se hodí poznamenat, že i když Rich Media anotace dokáží odkazovat a přehrát video/audio přímo, tak se tak dříve v praxi v podstatě nikdy nedělo. Místo toho bylo zvykem do Rich Media anotace vložit Flash přehrávač a video/audio předat jemu. Právě tento přehrávač zajišťoval dnes již vytracenou bohatost – ovládací prvky, možnost skriptování, atd.

Základem mechanismu Rich Media je anotace typu `/RichMedia`. Ta sama o sobě je vcelku jednoduchá, obsahuje pouze dva další slovníky, ve kterých je teprve schována veškerá složitost. Tyto slovníky jsou `/RichMediaSettings` („jak přehrát“) a `/RichMediaContent` („co přehrát“). Toto rozdělení dává naději na možnost využít jedno nastavení přehrávání s více multimédii nebo naopak. Ve skutečnosti to možné není, protože ve slovníkové struktuře `/RichMediaSettings` se často nachází odkazy na nepřímé objekty, které zároveň musí být zároveň odkazovány z `/RichMediaContent`. Pro názornost následuje ukázka, jak pomocí Rich Media anotace vložit do PDF dokumentu 3D PRC soubor a napodobit tak ukázku ze sekce 2.2.6:

```
6 0 obj
<<
  /Type /Annot
  /Subtype /RichMedia
  /Rect [ 100 100 495 742 ]
  /AP <<
    /N 5 0 R
  >>
  /RichMediaSettings <<
    /Activation <<
      /Condition /PV
      /Scripts [ 14 0 R ]
    >>
    /Deactivation <<
      /Condition /PI
    >>
  >>
  /RichMediaContent <<
    /Assets <<
      /Names [ (kladka.prc) 9 0 R (wireframe.js) 14 0 R ]
    >>
    /Configurations [ 7 0 R ]
    /Views [ 10 0 R 11 0 R 12 0 R ]
  >>
>>
endobj
```

Na první pohled je viditelná složitost Rich Media anotací. Ta vyvstává hlavně z toho, že obecně jeden `/RichMediaContent` definuje více konfigurací (`/Configurations`). Přitom v aktivačním slovníku může být řečeno, která konfigurace má být spuštěna (výchozí je první z pole konfigurací). Zde je jen jedna. Součástí aktivačního slovníku je i položka

`/Scripts`, obdoba `/OnInstantiate`, která umožňuje spuštění více skriptů (relevantní pouze pro 3D díla) – změna je zde ta, že stream skriptu je zabalen ve specifikaci souboru a uveden je také mezi zdroji.

Rich Media konfigurace je pouze množina instancí:

```
7 0 obj
<<
  /Type /RichMediaConfiguration
  /Subtype /3D
  /Instances [ 8 0 R ]
>>
endobj
```

```
8 0 obj
<<
  /Type /RichMediaInstance
  /Subtype /3D
  /Asset 9 0 R
>>
endobj
```

Instance potom odkazuje na objekt specifikace souboru, v tomto případě blíže neurčený nepřímý objekt č. 9 (stejný jako v sekci 2.2.6), který popisuje PRC 3D soubor. Zatímco specifikace souboru může obecně obsahovat místo vloženého souboru jen cestu k němu nebo jeho umístění na internetu, tak to není pro Rich Media dovoleno – podporovány jsou pouze vložené soubory (streamy typu `/EmbeddedFile`). Zároveň je odkaz na stejný objekt (zde `9 0 R`) součástí „databáze zdrojů“ (*asset name tree*) slovníku `/RichMediaContent`. Obecně je tato databáze strom, zde má pouze jednu položku s názvem `kladka.prc`.

Celá specifikace se tak zdá být velmi obfuskovaná a hlavně vyžaduje dvojí odkaz na stejný objekt. To vše je bohužel pozůstatek Flashe. Hlavní myšlenka byla, že objekt konfigurace bude obsahovat více instancí – Rich médií (video, audio, 3D) – a také hlavní instanci, Flash aplikaci, která se postará o přehrávání ostatních instancí. Pro tento účel mohly být přehrávači předány různé parametry, pomocí kterých jej bylo možné nakonfigurovat. Výběr konfigurace probíhá v aktivačním slovníku zanořeném v `/RichMediaSettings`. Bez Flashe se tato složitá struktura zdá komplikovaná a zbytečná. Jediné využití více zdrojů je v případě 3D děl, kde připadá v úvahu více souborů s JavaScriptem, což na zbytečnosti databáze zdrojů nic nemění – postačuje jednoduchá nepřímá reference ze `/Scripts`.

Součástí mechanismu Rich Media je ještě *Rich Media Execute* akce, která umožňuje zavolání funkce v kontextu konkrétní Rich Media instance. V případě Flashe by to byla ActionScript funkce, v případě 3D modelu to může být JavaScript funkce. JavaScript pro 3D modely je ale možné spouštět klasickými JavaScript akcemi, takže Rich Media Execute akce v podstatě dnes pozbývá smyslu.

Téměř stejně jako v předchozí ukázce lze docílit vložení videa nebo audia. Stačilo by změnit jen `/Subtype` ze `/3D` na `/Video` či `/Sound`. Samozřejmě by nemělo smysl udávat `/Scripts` nebo `/Views`. Z toho je vidět, že neexistuje žádná možnost jak docílit zobrazení ovládacích prvků nebo ovládat přehrávání pomocí PDF akcí. Důležité poznámky k praxi obsahuje sekce 5.2.5.

Kapitola 3
Tvorba PDF s interaktivními prvky a multimédii TEXem

TEX jako program má své kořeny již v roce 1977. Od roku 1990 zůstal v podstatě nezměněn [11]. Formát PDF sice částečně vychází ze staršího PostScriptu, sám ale pochází z roku 1993 [9]. Potřeba vytvářet soubory PDF donutila komunitu TEXu přijít s nějakým řešením. Ta řešení jsou principiálně dvě. Jejich základní filozofii popisuje následující sekce.

Hlavním cílem této teoretické kapitoly je ale představit způsob, jakým lze vytvářet PDF dokumenty, které obsahují interaktivní prvky nebo multimédia představené v kapitole 2.

3.1 DVI

V úplných počátcích byl výstupem TEXu kód, který dokázala interpretovat tiskárna, kterou měl Knuth zrovna k dispozici [18]. To bylo pro jeho původní využití v „malém“ dostačující. Pro případ použití s jinou tiskárnou by pak bylo potřeba upravit zdrojový kód TEXu. Naštěstí navrhl pro TEX David Fuchs formát DVI (*device independent*, „nezávislý na zařízení“), který je dodnes jediným formátem, jenž (původní) TEX generuje. Výhodou DVI je to, že pro použití s různými tiskárnami stačilo naprogramovat jen převodník DVI → kód tiskárny. Žádný jiný programátor tak nemusel zasahovat do vnitřního fungování TEXu.

Cesta sazby z `.tex` zdrojového souboru na papír probíhala tedy zhruba následovně:

- TEX postupně čte ze vstupního souboru text, který projde zpracováním čtyřmi TEXovými procesory [14]. Hlavní datovou strukturou je spojový seznam různých druhů uzlů. Například `char` uzly reprezentují znak v nějakém fontu, `glue` uzly reprezentují (obecně pružnou) mezeru a `hlist` uzly reprezentují horizontální seznam uzlů (klidně i dalších uzlů typu `hlist` nebo `vlist`). Struktura je tedy velice často rekurzivní a seznamy jsou mnohdy spíš stromy. Všechny druhy uzlů v podstatě přímo odpovídají některému primitivnímu konceptu TEXu.
- V momentě, kdy TEX usoudí, že má ve svém seznamu dost materiálu pro zaplnění jedné strany, zabalí jej do uzlu `vlist` („`\vboxu`“) a předá v TEXu naprogramované output rutině.
- Output rutina obvykle po nějakých úpravách (jako přidání hlavičky a patičky) předá box s výsledkem příkazu `\shipout`.
- `\shipout` je primitivní příkaz, který nepřímou rekurzí a procedurami `vlist_out` a `hlist_out` projde veškerý obsah předaného seznamu a převede ho do formy DVI stránky, kterou zapíše do výstupního DVI souboru.
- Konverzní program přečte DVI soubor a jeho výstupem je kód, kterému rozumí tiskárna.
- Tiskárna na základě poskytnutého kódu provede tisk.

Obvykle jsou programy, které zpracovávají DVI, nazývány výstupními ovladači, a obecně samozřejmě mohou místo generování kódu pro tiskárnu klidně DVI zobrazit na obrazovce [14]. Jedna z nevýhod generování DVI a následného pozdějšího zpracování ovladači je ta, že každý ovladač musí sám umět hledat a zpracovávat fonty. V DVI jsou totiž jen jména použitých fontů. To kdysi tolik nevadilo, protože metriky fontů (jediné co potřebuje TEX) a kresby znaků (co potřebuje výstupní ovladač) jsou v jiných souborech.

Knuth do TEXu prozíravě zabudoval mechanismus pro rozšíření TEXu v podobě uzlů typu `whatsit`. Uzlů `whatsit` může být více druhů, ale TEX je v podstatě ignoruje až do doby, než nastane `\shipout`. Pro každý poddruh uzlu typu `whatsit` je zde totiž naprogramováno co se stane při jeho zpracování. Knuthovou myšlenkou bylo, že to může být cokoliv a v případě rozšíření TEXu tímto způsobem není potřeba měnit velkou část TEXu, jen část zabývající se výstupem. Jako ukázku naprogramoval pomocí uzlů typu `whatsit` zápis do souborů a také příkaz `\special`. Právě příkaz `\special` je další možností rozšíření. Tento primitiv do aktuálního seznamu umístí uzel `whatsit` obsahující text, který je zapsán *přímo* do DVI souboru. V případě, že se makro programátor dohodne s programátorem výstupního ovladače na podobě „special“ řetězců, tak můžou obejít omezení TEXu a přidat například možnost barev.

Výstupních ovladačů vzniklo mnoho. Ovladač `dvipdfm`[1] Marka Wickse například umožňuje převod do formátu PDF. NeTEXové možnosti, jako je vytváření anotací nebo PDF objektů, jsou potom řešeny smluvenými `special` řetězci. Bohužel je ale také stále populární i vytváření PDF přes DVI a PostScript, popsané například v historickém [8]. Tento způsob obvykle zahrnuje pár převodníků (dvips[2] a ps2pdf[3]), a také odpovídající pár „special“ řetězců/příkazů, které nemají význam pro výstupní formát samotný, ale pouze pro daný převodník (TEXovský special a PostScriptový pdfmark).

Důležitý nedostatek dodatečného zpracování DVI souborů je to, že nepracují se spojovými seznamy, které používá TEX. Z toho vyplývají některá omezení, jako například to, že DVI ovladač nemůže například obarvit plochu určenou `\hbox`em. Jednoduše proto, že údaje o tomto boxu už neexistují.

3.2 pdfTEX

Úplně jinak šel na problém generování PDF TEXem Hàn Thế Thành [17]. Ten do TEXu přidal kód, který proceduře `\shipout`u umožňuje kromě DVI vytvářet také PDF soubory. Makro programátor může komunikovat přímo s programem, který PDF vytváří a to pomocí nových primitivů pro vytváření PDF objektů, anotací, atd. Ty obvykle vytvoří `whatsit` uzly nových typů. Není již potřeba vytvářet a na druhém konci zase zpracovávat strukturované textové řetězce.

Navíc má tento rozšířený TEX (později nazvaný pdfTEX) při generování PDF k dispozici i informace, které se dříve při výstupu do DVI ztrácely. Může tak například podporovat anotaci, která se roztáhne do velikosti boxu v němž se nachází. Anotace je totiž `whatsit` a při rekurzivním zpracování uzlu typu `hlist` nebo `vlist` (který odpovídá boxu v němž anotace je) TEX zná rozměry tohoto „boxu“. Navíc pdfTEX definuje i primitivy, které umožňují zpětnou vazbu. I v tomto ohledu je mocnější než alternativní řešení přes `dvipdfm` nebo pozdější `dvipdfmx`[4].

[1] `https://www.ctan.org/pkg/dvipdfm`
[2] `https://www.tug.org/texinfohtml/dvips.html`
[3] `https://www.ghostscript.com/`
[4] `http://project.ktug.org/dvipdfmx/`

3.3 LuaTEX

Další změny v tomto ohledu přinesl LuaTEX [12]. Ten totiž sice vychází z pdfTEXu, a tím pádem dědí i jeho možnosti tvorby PDF, ale přináší možnost rozšíření TEXu pomocí Lua kódů. Jedna z nejzajímavějších funkcí je možnost zasáhnout pomocí zpětně volaných (*callback*) funkcí, v určitých rozumných místech, do spojových seznamů, které zrovna TEX zpracovává. Dalším užitečným rozšířením LuaTEXu jsou *atributy*. Atributy mají svá čísla a samy nabývají číselných hodnot. Nastavení například atributu 50 na hodnotu 25 má za důsledek to, že všechny další uzly, které TEX vkládá do interních spojových seznamů budou mít v sobě uchováno „atribut 50 má hodnotu 25“. Velkou výhodou je, že nastavení atributů například respektuje TEXové skupiny, dochází tak k obnovení nastavení změn atributů, které byly provedeny lokálně ve skupině. Uzly, které mají nastavený určitý atribut, lze pak při pozdějším zpracováním Lua kódem odchytit a něco s nimi provést. Kromě široké škály jiných využití to má implikace i pro výstup do PDF.

Místo použití primitivů pdfTEXu, které vkládají do spojových seznamů TEXu uzly typu `whatsit` je možné nastavit atribut a ten zpracovat až později. Například až po tom co jsou vytvořeny odstavce a je proveden stránkový zlom. Až v tuto chvíli lze atributy nahradit vhodně vloženými `whatsit` uzly. To se na první pohled neliší od vložení `whatsit` uzlů hned, jak to dělají primitivy. Má to ale některé zajímavé implikace. Předně není totiž úplná pravda, že TEX `whatsit` uzly vždy ignoruje. Jejich přítomnost v některých místech může ovlivnit některé algoritmy TEXu. Jindy je výhodné použití atributů kvůli tomu, že respektují skupiny. To je zajímavé třeba pro vkládání barev, které je v pdfTEXu podporováno jen primitivy, které nerespektují skupiny.

O něco dál než LuaTEX jde LuaMetaTEX [6] od vývojářů ConTEXtu. Ten odstranil téměř veškerý kód pro generování výstupu. Jeho myšlenka je, že lze naprogramovat vlastní `\shipout` v Lue. Ten přitom může využívat atributy nebo uživatelem definované `whatsit` uzly a jeho výstup tím pádem nemusí být omezen na historicky obvyklé formáty DVI a PDF.

3.4 Primitivy pdfTEXu a LuaTEXu

V předchozích sekcích byly představeny základní principy fungování primitivů pdfTEXu a LuaTEXu[5] v kontextu výstupu do PDF. Účelem této sekce je nastínit jejich praktické využití pro vkládání interaktivních prvků a multimédií, kterým se věnovala kapitola 2.

Často vzniká potřeba vytvořit PDF objekt a vytvořit na něj nepřímý odkaz. To řeší primitivní `\pdfobj`, kterému lze předat nějaký text a on ho uloží do PDF souboru jako objekt, jehož číslo je k dispozici v `\pdflastobj`. Důležité je, že pdfTEX tento text zpracovává jen částečně – provádí sice tokenizaci a expanzi, poté ale údaje o tokenech zahodí a převede je zpět na text. Při použití Olšákovy terminologie [14] lze říci, že se tento text nikdy nedostane do *hlavního procesoru* a žádné jeho povely tedy nebudou fungovat. Při práci s výstupem do PDF je tedy potřeba si dát pozor na použití pouze takových maker a primitivů, které se expandují víceméně pouze na znaky kategorií 10, 11 a 12.

Takto lze vytvářet všechny různé druhy PDF objektů, protože k jejich rozeznání se používá v podstatě jen to, kterými znaky začínají (např. znakem „`(`“ pro řetězec). `\pdfobj` ale nabízí velice šikovnou možnost vytvářet streamy. To je výhodné, protože

[5] Velká část primitivů pdfTEXu byla v LuaTEXu přejmenována. Většina makrobalíků ale emuluje i stará jména, takže lze použít i ta. Jak provést emulaci ukazuje i manuál LuaTEXu [12].

sám spočítá velikost v bajtech. Přidat streamu další atributy (např. `/Type`) je možné pomocí `attr` specifikace. Také existuje možnost vytvoření objektu, jehož obsah se získá ze souboru daného jména. To funguje jak pro „syrové“ objekty, tak i pro streamy. Například pro vložení videa ze souboru `video.mp4` je možné použít:

```
\catcode`\#=12 % usnadnění použití znaku '#'
\pdfobj stream attr{/Type /EmbeddedFile /Subtype /video#2Fmp4} file {video.mp4}
```

Výsledný objekt je v podstatě shodný s tím, který byl zmíněn a vysvětlen v sekci 2.2.4. K zápisu tohoto objektu do PDF zatím ale nedojde. Objekty umožňují jejich vícenásobné využití. V případě, že objekt není využit vůbec, nemá smysl ho do PDF vůbec zapisovat. Vynutit zapsání ihned je možné pomocí prefixu `\immediate`. Výhodnější je ale provést „referenci“ na objekt pomocí `\pdfrefobj` ⟨*číslo objektu*⟩. `\pdfrefobj` vytváří uzel typu `whatsit`. To je výhodné například proto, že reference, které se ocitnou v nevyužitých boxech se nikdy nezpracují a je možné, že se tak ušetří místo v PDF souboru.

V podstatě analogicky k `\pdfobj` existuje `\pdfannot` pro tvorbu anotací. Sice by bylo možné vypočítat polohu a velikost anotací ručně a vytvářet je tak pomocí `\pdfobj`. Není to ale potřeba, protože `\pdfannot` vypočítá a zapíše `/Rect` automaticky. Příkazu `\pdfannot` lze zadat rozměry manuálně (podobně jako u `\vrule` nebo `\hrule`). Nezadané rozměry jsou určeny automaticky (roztažením do boxu ve kterém se `\pdfannot` nachází, nebo přesněji `\pdfannot whatsit` v uzlu typu `hlist` či `vlist`).

Pro případ především anotací typu `/Link` je k dispozici také alternativní možnost vytváření anotací pomocí příkazů `\pdfstartlink` a `\pdfendlink`. Je jimi dokonce možné vytvářet libovolné anotace, ne jen ty typu `/Link`. Tyto primitivy umožňují řešit případ, kdy je potřeba plochu anotace sestavit z více obdélníků (typicky případ, kdy dojde k řádkovému zlomu). Oba příkazy vkládají do aktuálního seznamu `whatsit` uzly, které (zjednodušeně řečeno) při tvorbě stránky způsobí vytvoření identických anotací[6] pro všechny obsažené hboxy (přesněji uzly typu `hlist`). Tento mechanismus tvorby anotací také respektuje nastavení aktuální transformační matice, ale pouze v případech, kdy je poctivě užito příkazů `\pdfsetmatrix` (PDF příkaz `cm`), `\pdfsave` (`q`), `\pdfrestore` (`Q`). PDF příkazy `cm`, `q` a `Q` vložené manuálně pomocí příkazů `\pdfliteral` pdfTEX totiž nijak neinterpretuje.

Anotacím je možné nastavit vzhled (`/AP`, *appearance*, viz sekci 2.1.2). U multimediálních anotací, které nemají žádný předdefinovaný vzhled, je to dokonce nutnost. Vzhledy jsou formy (*form Xobject*, viz sekci 2.2.1). Ty je možno vytvářet pomocí primitivu `\pdfxform`, který má jako argument číslo boxu. Ten interně funguje téměř stejně jako `\shipout`, jen se místo objektu stránky zapíše PDF objekt typu forma.

Předchozí primitivy dovolovaly vytvářet *nové* PDF objekty. V některých případech je ale potřeba přidat něco do objektů, které obvykle vytváří už sám pdfTEX. Je to případ třeba atributů stránky, které jsou parametry v objektu stránky. Zrovna v tomto případě je možné přidat vlastní parametry pomocí registru `\pdfpageattr`, jehož obsah je přidán k ostatním parametrům vytvořených pdfTEXem (např. `/Contents`).

Výše zmíněné je vše, co je potřeba pro vkládání všech multimédií a interaktivních prvků. Důležité je to, že konkrétně audio/video a 3D díla TEX vůbec neřeší (narozdíl třeba od obrázků). PDF objekty s nimi je tak potřeba vytvářet a vkládat manuálně – mimo vkládání souborů tedy především na textové bázi. Jsou sice k dispozici i další primitivy (např. `\pdfoutline` pro tvorbu záložek). Ty ale striktně nejsou potřeba. Objekty, které tyto primitivy vytváří je totiž možné konstruovat přímo pomocí `\pdfobj`.

[6] Samozřejmě až na položku `/Rect`.

Kapitola 4
Existující TEXová řešení vkládání interaktivních prvků a multimédií

V průběhu let se objevilo hned několik řešení, která umožňovala uživatelům TEXu vkládat multimédia do PDF souborů. Přístup těchto řešení se lišil v závislosti na konkrétním makrobalíku, tak jak to u nich bývá zvykem. V LATEXovém světě vznikaly balíčky, kdežto v ConTEXtu byla funkcionalita přidána přímo do formátu.

Pozdější řešení využívala také novější možnosti formátu PDF. I když na konci roku 2020 nastala významná změna ve fungování nejběžnějšího způsobu vkládání multimédií (viz také sekce 2.2.7), tak se toho ve světě TEXu moc nezměnilo. V ConTEXtu dokonce považují tuto oblast za mrtvou a další vývoj na poli multimédií neplánují, dokud nepřinese norma PDF nějaké novinky.

Cílem této teoretické kapitoly je alespoň základně představit existující balíčky, zjistit co umožňují a rozebrat principy jejich fungování.

4.1 LATEXové balíčky

Myšlenka LATEXu je postavena na tom, že existuje jádro, které definuje programátorské rozhraní a také značkovací jazyk pro koncového uživatele. Kromě základů jako je třeba správa fontů je většina dalších možností implementována v třídách a hlavně balíčcích – jádro samotné například neumožňuje vkládat do dokumentů ani obrázky. Přirozeně se tak podpora vkládání multimédií k LATEXovým uživatelům dostala prostřednictvím balíčků.

4.1.1 hyperref

Cílem balíčku hyperref[1] je především dodat LATEXu možnost hyperlinkových odkazů. Některé detaily jeho klasického použití i interního fungování v případě použití pdfTEXu popisuje [4]. Za tím účelem předefinovává velké množství interních příkazů LATEXu. To je velká nevýhoda tohoto i dalších LATEXových balíčků – často je nutné pro implementace nějaké funkcionality zasáhnout do jádra, které je ale striktně zpětně kompatibilní. Autorům balíčků nezbývá nic jiného než definice jádra předefinovat a doufat, že uživatel nenačte jiný balíček, který definice mění jinak. To se poslední dobou začíná měnit v souvislosti s rozšířením LATEXového jádra o prostředky z projektu LATEX3. Dochází například k přidání míst, do kterých se můžou balíčky zaháčkovat (*hooks*). Jestli dojde k změnám i v balíčku hyperref se teprve musí ukázat.

Jako u dalších LATEXových balíčků se i tento snaží o co nejširší pokrytí výstupních ovladačů a TEXových rozšíření. Podporovány jsou tak i jiné výstupní formáty než PDF.

Základním výsledkem po načtení balíčku hyperref je, že příkazy jako `\label`, `\ref`, `\cite` a další začnou automaticky vytvářet umístění respektive skoky na umístění. Také jsou při výchozím nastavení automaticky vytvářeny PDF záložky (viz sekci 2.1.5).

[1] `https://www.ctan.org/pkg/hyperref`

Přizpůsobení možností se provádí pomocí key-value syntaxe. Samotné parametry je možné buď předat při načítání balíčku, nebo kdykoliv později pomocí `\hypersetup`. Tento příkaz zároveň umožňuje volby kdykoliv později změnit. Možností přizpůsobení je mnoho. Kromě vzhledu hyperlinkových odkazů je možné nastavit třeba různá nastavení prohlížeče (popsaná v sekci 2.1.1), informaci o dokumentu a jeho autorovi, přechody (viz sekci 2.1.6), atd. Konkrétně v případě přechodů lze například nastavit všechny parametry jednotlivých stylů animací.

K dispozici je i několik uživatelských příkazů. Ty dovolují například vytvářet vlastní záložky, nebo hyperlinkové odkazy na libovlná URL (`\href`). Příkaz `\href` dokonce umožňuje zadání další zřetězené akce (`/Next`), nedává ale žádnou příjemnou možnost, jak ji definovat, zcela to nechává na uživateli.

4.1.2 multimedia (beamer)

Balíček beamer[2] je známý především díky stejnojmenné třídě, která umožňuje v LaTeXu vytvářet prezentace. Od verze 2.20 (rok 2004) je součástí tohoto většího balíku také balíček Multimedia (především pak soubor `multimedia.sty`). I když je balíček součástí beameru, je na něm zcela nezávislý – je možné ho použít s jakoukoliv třídou. Omezena je ale možnost zpracování dokumentu. Podporováno je pouze přímé zpracování pdfLaTeXem (později také LuaLaTeXem) a nepřímé zpracování přes dvips.

Balíček Multimedia v dokumentaci inzeruje možnost vkládat do dokumentů „movie“ a „sound“ – je totiž založen na movie a sound anotacích a dokonce i akcích (detailně popsány v sekci 2.2.3 a sekci 2.2.4). I když by to napovídal název „Multimedia“ nemá balíček nic společného s mechanismem multimédií („renditions“), který se objevil v PDF verze 1.5 (rok 2003) a který je také představen v sekci 2.2.5.

Uživatelské rozhraní je založeno víceméně na dvou párech uživatelských příkazů:

- `\movie` (vytváří movie anotaci),
- `\hyperlinkmovie` (vytváří movie akci),
- `\sound` (vytváří sound objekt a anotaci),
- `\hyperlinksound` (vytváří sound akci).

Přitom použití příkazů vypadá zhruba takto:

`\movie[`⟨*options*⟩`]{`⟨*poster horizontal material*⟩`}{`⟨*movie filename*⟩`}`

Tyto uživatelské příkazy s výhodou využívají způsob „key-value“ (klíč-hodnota) pro upřesnění chování konkrétní anotace anebo akce. Jednoduchost mechanismu movie a sound anotací se zde poměrně příjemně projevila. Každý klíč celkem jednoduše ovládá nezávislou vlastnost movie anotace. Například přidání ovládacích prvků pomocí `showcontrols` nebo složitější přidání akce, která spustí přehrávání při otevření stránky pomocí `autostart`. Speciálním klíčem je `externalviewer`, který způsobí, že nebudou použity movie a sound anotace. Místo nich je vložena `/Link` anotace s `/Launch` akcí, která způsobí otevření souboru v externím prohlížeči.

Kromě vynechatelných přepínačů chování mají `\movie` a `\sound` příkazy ještě dva další parametry. První parametr určuje *poster* („plakát“), tedy to, co se zobrazí před tím, než se anotace aktivuje. Tímto plakátem může v tomto případě být jakýkoliv horizontální materiál – typicky se jedná například o náhledový obrázek vložený pomocí `\includegraphics`. Rozměry plakátu jsou zároveň použity jako implicitní rozměry videa, pokud není stanoveno jinak v nepovinných parametrech. Druhý parametr slouží k určení, co vlastně bude přehráno (podrobněji později).

[2] `https://www.ctan.org/pkg/beamer`

Balíček multimedia je striktnější než norma PDF – omezuje přehrávání pouze na externí soubory nebo vložené audio soubory. Omezení na externí soubory přitom nutné není – launch akce i movie/sound objekty mohou totiž pracovat i s URL adresami.

Vkládání audio souborů přímo do PDF souboru, tak jak to činí balíček multimedia při volbě `inlinesound`, je principiálně špatně. Jak je vysvětleno v sekci 2.2.3, chování vloženého audia a externího souboru je docela odlišné. Vložené audio znamená vložení syrových PCM dat do stream části sound objektu, což balíček multimedia neřeší a jednoduše vloží uživatelem jmenovaný soubor, který obvykle kromě syrových PCM dat obsahuje i metadata. V rámci maker se ale balíček vůbec nesnaží PCM data oddělit a minimálně by tak byla žádoucí změna dokumentace. Další chybou je opomenuté nastavení parametru endianity, což v případech použití nevýchozí hodnoty způsobuje nesprávné přehrání audia (typicky velmi hlasitě a zkresleně). Nesprávně je také řešen případ odkázání na externí audio soubor. V takovém případě nejsou v PDF vyžadována metadata, která má obsahovat přímo externí soubor. Balíček multimedia je ale uvádí minimálně nastavené na jeho výchozí hodnoty, což je v lepším případě PDF prohlížečem ignorováno.

Multimedia také pro některé situace spoléhá na balíček „hyperref“ (viz sekci 4.1.1), jehož načtení ale explicitně neprovádí. Uživatele tak někdy může překvapit ne zcela napovídající chybová hláška `Undefined control sequence.`

Implementace se opírá víceméně o primitivní příkazy TEXu. Jsou ale dodrženy některé typicky LATEXové idiomy – definování uživatelských příkazů pomocí `\newcommand`, nebo použití balíčku „keyval“ pro key-value parametry.

4.1.3 movie15

Balíček movie15[3] vytvořil Alexander Grahn v roce 2004. Důležitější část názvu je číslovka „15“, která odkazuje na formát PDF ve verzi 1.5 Právě v této verzi byl přidán mechanismus multimédií („renditions“), na kterém je tento balíček založen. Movie15 totiž vůbec vůbec nepracuje se staršími „movie“ anotacemi, jak by mohla napovídat první část názvu. Později byl balíček rozšířen také o podporu vkládání 3D objektů, která je založena na možnostech PDF verze 1.6 (viz sekci 2.2.6).

Ústředním příkazem je `\includemovie`, prostřednictvím kterého může uživatel vkládat audio, video a 3D díla. Jeho použití je následující:

`\includemovie[`⟨*options*⟩`]{`⟨*width*⟩`}{`⟨*height*⟩`}{`⟨*media file*⟩`}`

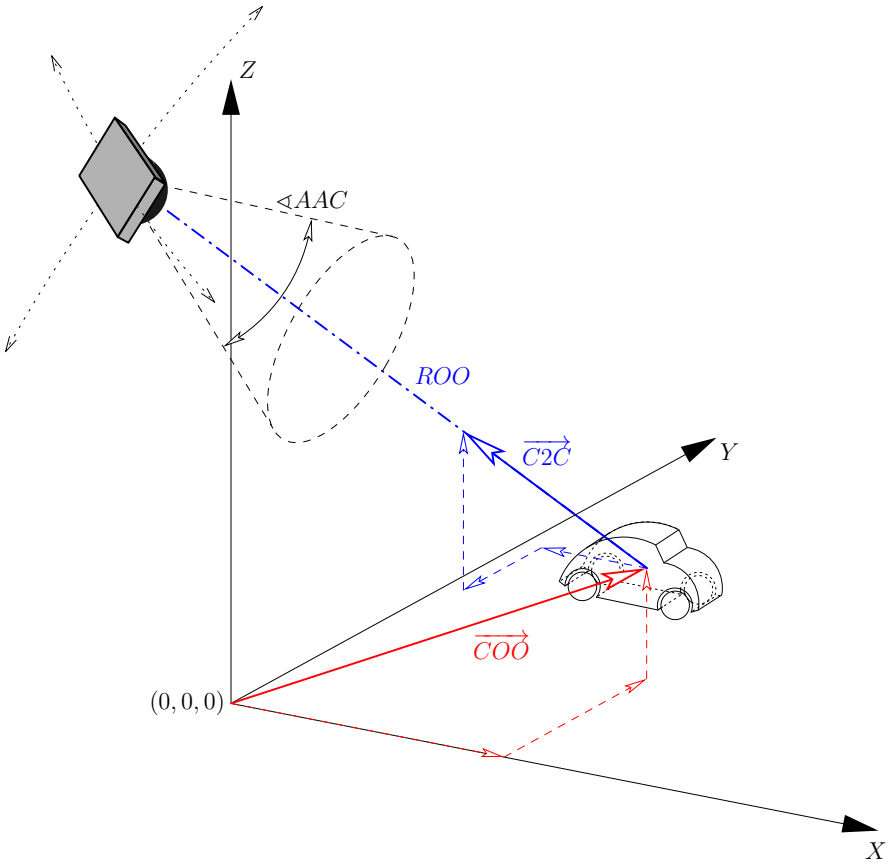
První volitelný argument může obsahovat key-value parametry a poslední jméno souboru s multimédiem. To je velice podobné příkazům `\movie` a `\sound` z balíčku Multimedia (viz sekci 4.1.2). Zásadní rozdíl ale spočívá v nastavení plakátu. Zatímco balíček multimédia preferoval jako plakát nějaký TEXový horizontální materiál, tak movie15 využívá možnost PDF mechanismu multimédií zobrazit jako plakát první snímek videa (volba `poster`, funguje jen při vkládání videí) nebo nějaký konkrétní obrázek (volba `poster=`⟨*soubor*⟩). Použití libovolného horizontálního materiálu je také možné (volba `text=`⟨*horizontální materiál*⟩).

Pomocí volitelných key-value parametrů pokrývá movie15 téměř vše, co je možné docílit s mechanismem multimédií. Například podporuje automatický začátek přehrávání (nebo jeho pozastavení) prostřednictvím akcí, které se vykonají při otevření/zavření stránky s multimédiem. Pro některé funkce, jako třeba určení začátku a konce přehrávání konkrétní stopy, je ale použit JavaScript, což redukuje podporu v prohlížečích.

[3] `https://www.ctan.org/pkg/movie15`

Určení začátku a konce by zrovna šlo implementovat i bez JavaScriptu, je ale zapotřebí sestavit složitější strukturu PDF objektů.

Hned několik klíčů volitelných parametrů je určeno pro nastavení zobrazení 3D děl. Tyto parametry slouží k nastavení pohledu a pokud jsou uvedeny, nastaví tzv. výchozí pohled. Vytvoření dalších pohledů je také možné a to pomocí odkázání na externí soubor, z kterého movie15 údaje vyčte. Některé nastavitelné parametry v podstatě přímo odpovídají parametrům 3D pohledů definovaným v normě PDF (třeba `BGCOLOR` nebo `LIGHTS`). Větší abstrakce je ale u nastavení pozice a orientace kamery. Formát PDF standardně očekává matici „C2W“ (*camera to world*, inverze transformační matice zobrazující 3D scénu do prostoru kamery). Movie15 tuto matici, jejíž význam nemusí být na první pohled zřejmý, ani neumožňuje zadat přímo. Místo toho ji počítá z několika jiných parametrů, které jsou voleny tak, aby byly lépe představitelné pro běžného uživatele. Důmyslný výběr těchto parametrů i jejich výchozích hodnot má za důsledek to, že například základní strojírenské pohledy na součástky umístěné v počátku lze specifikovat jen vektorem a vzdáleností (4 čísla oproti dvanácti pro 3D transformační matici).



Obrázek 4.1. 3D parametry, které používá balíček movie15 pro stanovení pozice a orientace kamery ve 3D scéně [5]

Protože stejné parametry pro nastavení 3D pohledů převzaly i další implementace, vyplatí se podívat na to, jak konkrétně jsou tyto parametry koncipovány. Znázornění používaných parametrů obsahuje obrázek 4.1. Základem je *poziční* vektor ($\overrightarrow{COO}$), kterým jsou určeny souřadnice středu orbitu. Ten má ještě druhou funkci protože zároveň představuje bod na který směruje pohled kamery. Pozice kamery je přitom určena

směrovým vektorem ($\overrightarrow{C2C}$, *center of orbit to camera vector*) a vzdáleností kamery od orbitu (*ROO*). Protože je balíček movie15 omezen pouze na perspektivní promítaní, umožňuje nastavit i zorný úhel ($\angle AAC$, „FOV“). Posledním volitelným parametrem je natočení kamery (*roll*), které není na obrázku 4.1 znázorněno.

Pro ilustraci následující ukázka kódu obsahuje definici dodatečného pohledu tak, jak je dokáže balíček movie15 vyčíst:

```
VIEW=Izometricky
  COO=50 20 70
  C2C=-1 -1 1
  ROO=500
  PART=<...>
    VISIBLE=false
  END
END
```

Takto může být v jednom souboru za sebou definováno více pohledů. Kromě toho mají pohledy definovány v externím souboru další možnost – mohou definovat parametry zobrazení jednotlivých objektů ve 3D scéně. V ukázce je například pro jeden objekt zakázána viditelnost.

Dalším podporovaným příkazem je `\movieref`. Tímto příkazem je možné definovat akční tlačítka, která budou provádět akci nad objektem vloženým pomocí `\includemovie`. Provázání mezi tlačítky a cílovými objekty je realizováno pomocí labelů. Tlačítko samotné je určeno nějakým horizontálním materiálem. Pro audio/video jsou k dispozici akce přímo odpovídající rendition akcím. Zajímavé ale je, že je preferováno provedení akce JavaScriptem, ne jednoduchou symbolickou hodnotou v PDF objektu akce, které je použito až jako náhradní možnost. Je to kvůli tomu, že se movie15 snaží o trochu odlišné chování, než u těchto jednoduchých akcí (například možnost použití stejného tlačítka pro akci pozastavení i pokračování v přehrávání). Hlavní důvod je ale ten, že movie15 umožňuje definovat parametry přehrávání, které nelze v rendition objektech ani rendition akcích specifikovat (třeba rychlost přehrávání). Aby je bylo možné konzistentně dodržet, je potřeba při každé akci (i té „hlavní“, interně definované na Screen anotaci) provést manipulaci pomocí JavaScriptu.

Pro 3D díla jsou kromě základní akce přepnutí 3D pohledu, která je přímo specifikována normou PDF, k dispozici také dvě speciální akce, které movie15 realizuje JavaScriptem. První po aktivaci zobrazí dialogové okno, které obsahuje parametry aktuálního pohledu (jenž mohl interaktivně změnit uživatel) ve formátu externího souboru. Druhá akce provede výpočet optimálních hodnot parametrů *ROO* a $\overrightarrow{COO}$, které také zobrazuje dialogovým oknem. Tyto akce nejsou určeny pro finální verze dokumentů, ale jako pomůcka autorům pro možnost interaktivního/automatického stanovení všech parametrů pohledu (včetně možností jako jsou viditelnsti jednotlivých součástek nebo nastavení osvětlení 3D scény).

Celkově je balíček movie15 mnohem sofistikovanější než multimedia. To bohužel přináší i vyšší složitost. Podobně jako u multimedia se významná část implementace zabývá definováním key-value parametrů a dvojím řešením téhož – definování PDF objektů pro vysokoúrovnější rozhraní primitivů pdfTEXu nebo omezenější rozhraní `\pdfmark` pro zpracování přes dvips a ps2pdf. Kvůli podpoře 3D objektů ve starší PDF verzi 1.6 jsou také některé parametry zobrazení 3D děl nastaveny přes JavaScript. Své problémy také přináší výpočet matice „C2W“. Obecně totiž výpočet takové matice probíhá v oboru reálných čísel, do formátu PDF lze pak jednotlivé složky této matice zapsat jako dese-

tinná čísla v libovolné přesnosti. Movie15 pro výpočty používá balíček „fp“[4] Michaela Mehlicha, který dovoluje v TEXu provádět výpočty s pevnou desetinnou čárkou. Podporovány jsou všechny parametry multimédií (renditions), které dávají smysl. U 3D pohledů ale chybí možnost přidat řezy, které by bylo ve verzi formátu PDF, na kterou movie15 cílí, nutné realizovat JavaScriptem.

4.1.4 media9

Balíček media9[5] je pokračováním movie15 a má i stejného autora. I když je pomocí obou balíčků možno dosáhnout podobných výsledků, je mezi nimi mnoho rozdílů. Hlavní část uživatelského rozhraní je stále postavena na dvou příkazech (vložení multimédia, respektive akčního tlačítka), ale jinak pojmenovaných. Také nepřijímají úplně stejné key-value parametry. To je dáno hlavně tím, že media9 je postaven na Rich Media anotacích (blíže popsány v sekci 2.2.7), jejichž chování a možnosti jsou jiné, hlavně v oblasti audia a videa.

V případě 3D objektů je změn oproti movie15 minimum. Je to dáno i tím, že pomocí Rich Media anotací lze dosáhnout velice podobné výsledky jako s původním mechanismem vkládání 3D děl. Přibyla možnost pro pohledy definovat řezy nebo možnost asociovat s 3D dílem více skriptů. Proto JavaScript, který umožňoval zobrazení parametrů aktuálního pohledu ve formátu externího souboru, tentokrát není spouštěn jako akce přes vložené akční tlačítko. Místo toho je možné při vkládání 3D díla zadat `3Dmenu`, což způsobí přidání skriptu mezi `/Scripts` (viz sekci 2.2.7). Ten zajistí zobrazení několika dodatečných položek v kontextové nabídce interaktivní práce s 3D dílem. Rozkliknutí těchto položek způsobí spuštění dalšího JavaScriptu, který zjistí či případně dopočítá odpovídající parametry a zobrazí je v dialogovém okně.

Vkládání audia a videa je ale naprosto odlišné. Místo toho, aby souborem pro vložení byl přímo audio/video soubor, je jím Flash přehrávač. Přehrávané soubory jsou do PDF vloženy jako streamy, zpřístupněné Flash přehrávači jako zdroje (*assets*). Ovládací prvky a možnosti `/RichMediaExecute` akcí jsou pak závislé na konkrétním Flash přehrávači.

Výhodou balíčku media9, oproti jeho předchůdci movie15, je jeho implementace v jazyce `expl3` a použití množství dalších maker z projektu LaTeX3. Díky tomu, je velká část implementace abstrahována od specifik jednotlivých DVI výstupních ovladačů nebo TEXových rozšíření (viz sekci 3). S výhodou jsou použity také další rozhraní projektu LaTeX3: výpočty s plovoucí desetinnou čárku a zaháčkování (například k tvorbě poslední strany). Některé abstrakce balíčku media9 jsou k dispozici i ostatním makro programátorům v podobě souboru `pdfbase.sty`.

Bohužel tento balíček zatím nebyl aktualizován v souvislosti s ukončením podpory Flashe. Alexander Grahn, autor balíčku, ale zveřejnil kolem listopadu 2020 na `stackexchange.com`[6] kód, který lze chápat jako alternativu balíčku pro jednoduché vkládání videa bez Flashe. Tento kód ale stále využívá Rich Media anotace. Zdrojem nejnovějších informací je v tomto ohledu Issue č. 9[7] v repozitáři, kde probíhá vývoj balíčku media9. Právě v rámci tohoto i dalšího Issue[8] jsem za pomoci uživatelů balíčku media9 přišel s možností rozšíření o podporu audia, automatického přehrávání a ovládacích prvků. Detaily jsou v sekci 5.2.5.

[4] `https://www.ctan.org/pkg/fp`
[5] `https://www.ctan.org/pkg/media9`
[6] `https://tex.stackexchange.com/a/516102/125126`
[7] `https://gitlab.com/agrahn/media9/-/issues/9`
[8] `https://gitlab.com/agrahn/media9/-/issues/15`

4.1.5 rmannot

Balíček rmannot[9] je dílem Donalda P. Storyho. Ten o něm dokonce publikoval článek v TUGboatu [16]. Předmětem článku jsou ale především Rich Media anotace, na kterých je balíček založen, a také spojení s Flashem, které dnes již není možné. Narozdíl od balíčku media9 (viz sekci 4.1.4) byl balíček rmannot aktualizován a obsahuje i informace o použití bez Flashe.

Kromě audia a videa je možné i vkládání 3D děl, zabalených opět v Rich Media anotacích. Kód pro vytváření 3D pohledů byl převzat z balíčku movie15 (viz sekci 4.1.3).

Značnou nevýhodou tohoto balíčku je jeho dostupnost a použitelnost. Balíček totiž není součástí distribuce TEX Live (údajně na žádost autora[10]). I kdyby dostupný byl, pravděpodobně by i tak nebyl pro většinu uživatelů použitelný – vyžaduje totiž zpracování přes dvips a komerční Adobe Distiller.

4.2 ConTEXt

ConTEXt[11] je samostaný formát. Je dílem nizozemské společnosti „PRAGMA Advanced Document Engineering“.

I mimo oblast multimédií a interaktivních prvků má tento formát několik významných specifik:

- většina funkcionality je v něm implementována přímo, není potřeba spoléhat se na externí balíčky;
- konzistentní využití „key-value“ syntaxe pro předávání parametrů;
- dědičnost a možnost uzpůsobení příkazů;
- důsledné oddělení různých „backendů“, tedy částí kódu, které se starají o výstup do konkrétního formátu.

I když je to nepopiratelnou výhodou ConTEXtu, tak poslední bod je pro účel této kapitoly zanedbán – zájmem této práce je jen výstup do PDF.

S příchodem verze ConTEXt MkIV se navíc přidává ještě jedno specifikum – velká část implementace, mimo jiné i kód obsluhující vytváření PDF struktur, je napsána v jazyku Lua, nikoliv v TEXu. To je běžnému uživateli ale v podstatě skryto. Většina uživatelského rozhraní navíc pochází z doby, kdy bylo vše implementováno pouze v TEXu.

4.2.1 Interaktivní prvky

Velká část interaktraktivních možností ConTEXtu je popsána v zatím nedokončeném dokumentu „Interaction“ [7]. Jakoukoliv formu interakce je v ConTEXtu potřeba explicitně vyžádat. Podporovány jsou v všechny možnosti zmíněné v kapitole 2. Některé možnosti jsou „nízkoúrovňovější“ (jako například PDF akce řešené univerzálně pomocí příkazu `\goto`), jiné jsou „vysokoúrovnější“ (například možnost vytváření tlačítek, která vyvolávají PDF akce). Tvorba záložek (*outlines*) odvozených ze struktury dokumentu (kapitoly, sekce, ...) je zcela automatická, ale je i možné definovat vlastní záložky.

PDF akce v ConTEXtu je možné vyvolat příkazem `\goto`. Ten obstarává (téměř) všechny druhy akcí. Jeho jeden argument určuje text k sazbě, druhý akci. Samotná

[9] `https://www.ctan.org/pkg/rmannot`
[10] `https://tug.org/svn/texlive/trunk/Master/tlpkg/libexec/ctan2tds?r1=19906&r2=19905&pathrev=19906`
[11] `https://wiki.contextgarden.net/Main_Page`

akce potom může mít vlastní argumenty (např. pro akci zahájení přehrávání videa je potřeba určit které video). Chybí implementace některých akcí, jako například přepínání 3D pohledů, nebo příjemná možnost přístupu k 3D kontextu 3D anotací z JavaScript akcí.

Výjimkou z příkazu `\goto` jsou skoky v dokumentu. V případě nastavení interakce totiž místo toho dochází jen k rozšíření příkazů `\in` a `\at`, které obstarávají odkazy na místa respektive stránky těchto míst v dokumentu.

Jsou dostupné i dvě „markup“ anotace – komentáře (typ `/Comment`) a přílohy (typ `/FileAttachment`). V typickém stylu ConTEXtu jsou k dispozici různé automatické způsoby jejich umístění (například do okrajů) nebo nastavení ikonek jejich vzhledu.

Za zmínku stojí také řešení skriptů v JavaScriptu. Kromě klasických JavaScript akcí přes `\goto` je možno definovat i „document level“ JavaScript akce. K jejich spuštění dochází při inicializaci dokumentu. V ConTEXtu jsou nazývány preambule a je možno je definovat s různou úrovní struktury.

ConTEXt podporuje i přechody (viz sekci 2.1.6). Myšlenkou je, že místo úplné kontroly nad animací přechodu, má uživatel na výběr jen sadu předdefinovaných vzhledů. Tento přechod se potom použije pro všechny aktuální stránky dokud není vypnut nebo změněn.

4.2.2 Figure mechanismus

„Figure“ mechanismus ConTEXtu, je určený primárně pro vkládání obrázků (JPG, PNG, ...), ale i PDF souborů. Obecný mechanismus ConTEXtu umožňuje úplně stejným způsobem vkládat i audio, video, 3D a Flash soubory. Typ souboru je automaticky rozpoznán podle přípony. Uživatel si tak vystačí pouze s příkazem `\externalfigure` (případně jinými variantami odvozenými dědičností). Například pro video ze souboru `video.mp4`:

```
\externalfigure[video.mp4]
[
  width=\textwidth,
  height=.461\textwidth,
]
```

Audio a video jsou vkládány pomocí movie anotací (viz sekci 2.2.4). 3D soubory jsou řešeny 3D anotacemi, Flash Rich Media anotacemi. Zatímco pro audio/video je dostačující stávající sada key-value parametrů, určená pro všechny „figures“, tak 3D modely mívají parametrů mnohem více. Z toho důvodu jsou tyto parametry řešeny odděleně pomocí množin Lua parametrů (`luaparameterset`). Ty existují pro 3D modely dvě:

1. `display`, nastavení parametrů 3D anotace;
2. `controls`, nastavení parametrů 3D streamu.

Obě množiny mají místy překryv – i 3D anotace i 3D stream mohou totiž obsahovat nastavení 3D pohledu/pohledů. Zpracování těchto množin parametrů je provedeno v Lue, a jak je v ConTEXtu obvyklé, tak je brán ohled na to, které parametry byly zadány a pomocí jakých *typů*. To má za následek velmi příjemné a odpouštějící uživatelské rozhraní.

Pozice a orientace kamery je pro 3D modely určena buď jménem U3D pohledu, nebo pomocí několika číselných parametrů. Na základě toho, které parametry a jak jsou zadány, je určena metoda výpočtu pozice a orientace 3D kamery. Je k dispozici stejný výpočet, jaký provádějí balíčky movie15 a media9 (viz sekci 4.1.3), ale i alternativní výpočty pomocí výšky a azimutu, případně pomocí rotací.

Zajímavé je i řešení Flash souborů. Kromě toho, že je možné vložit vlastní Flash soubor (rozpoznaný příponou `.swf`) je možné použít i již definovaný[12] alias pro některý z Flash video přehrávačů. Ovládání těchto Flash přehráváčů zajišťují JavaScript akce, dostupné klasicky přes příkaz `\goto`.

4.2.3 Renderings

V ConTEXtu je zabudována také podpora mechanismu multimédií („renditions“, viz sekci 2.2.5). Tato podpora ale v podstatě není vůbec zdokumentována a ani dokončena. Protože v ConTEXtu považují tento mechanismus za slepou uličku, tak pravděpodobně dokončena ani nebude.

Přesto se vyplatí se na tuto implementaci alespoň zhruba podívat. Předně se do uživatelského rozhraní promítly dvě hlavní částí mechanismu multimédií: `/Screen` anotace a `/Rendition` objekty. Ty lze vytvořit pomocí příkazu `\definerenderingwindow`, respektive `\useexternalrendering`. Oba vytvoří „pojmenované objekty“, které lze následně dát dohromady pomocí `\placerenderingwindow`.

To se značně liší od přístupu balíčku movie15, který tyto detaily od uživatele úplně skrývá a tím pádem nedokáže například využít jednu `/Screen` anotaci pro přehrání více multimédií, apod.

Aktuální implementace obsahuje několik chyb:

- striktnější nastavení oprávnění zbytečně znemožňuje přehrávání v některých prohlížečích,
- špatná synchronizace umístění anotace vůči poloze sazby,
- nesprávná strukutra PDF objektů, která znemožňuje přehrání multimédií.

Kód obsahuje dokonce zakomentovanou nápravu první chyby. Z toho lze usoudit, že se jedná o velmi málo používaný mechanismus, protože chyby jsou očividné. Vývojáři ConTEXtu obvykle nahlášené chyby rychle opravují, ale protože tento mechanismus považují za zastaralý, nelze říci jestli budou ochotni věnovat čas opravám nebo radši nepoužívaný kód úplně odstraní.

[12] Od novějších verzí ConTEXtu jsou tyto definice (stejně jako i jako některé další inicializace) součástí volitelných modulů a je nutné v dokumentu uvést například `\useJSscripts[vplayer]`.

Kapitola 5
Podpora interaktivních prvků a multimédií v PDF prohlížečích

Kapitoly 2 a 3 byly převážně teoretické, a zabývaly se tím, jak PDF obsahující multimédia či interaktivní prvky vypadají nebo jak by je šlo vytvořit v TEXu. Tato kapitola je praktičtější. Shrnuje skutečnost, tedy jak dané funkcionality formátu PDF zvládají různé prohlížeče.

Důležitá je zde terminologie. Norma PDF rozlišuje tři základní druhy programů pracujících s formátem PDF:

- *reader* (software, který dokáže číst PDF soubory)
- *writer* (software, který dokáže PDF soubory zapisovat)
- *product* (software, který dokáže PDF soubory číst i zapisovat)

Pro potřeby této kapitoly neplatí, že prohlížečem je míněn reader podle normy. Některé programy, které jsou dnes typicky nazývány prohlížeči, mají totiž navíc i (omezené) možnosti „úpravy“ PDF dokumentů. Jedná se především o možnosti přidávat do dokumentů *anotace* nebo do nich různě kreslit. Někdy PDF prohlížeče umožňují také PDF dokumenty kryptograficky podepsat či zašifrovat.

PDF prohlížečů dnes existuje mnoho, ale ne všemi má cenu se zabývat. A to obzvlášť v oblasti multimédií nebo interaktivních prvků. Nebyly tedy například testovány prohlížeče určené pro mobilní zařízení, kde je formát PDF už ze své podstaty problematický – pevné rozměry stránky a absulutní umístění jejího obsahu značně omezují čitelnost a použitelnost na těchto zařízeních s malými displeji.

Jasnou volbou pro zařazení do testu je Adobe Acrobat Reader DC (nástupce Adobe Acrobat Reader). Jako dílo společnosti Adobe, původce formátu PDF, ho lze v podstatě chápat jako „referenční implementaci“. To byla pravda obzvlášť v dobách, kdy to byl v podstatě prohlížeč jediný a veřejně dostupné specifikace neobsahovaly popis všeho, čeho tento prohlížeč byl schopný. Tento prohlížeč je volně dostupný a pravděpodobně jeden z nejpoužívanějších.

Acrobat Reader DC je zdarma dostupná a částečně osekaná varianta licencovatelného programu Acrobat Pro DC. Ten kromě části odpovídající Acrobat Readeru (a tomu co bychom nazvali prohlížeč) umí mimo jiné PDF dokumenty i vytvářet, nebo do nich přidávat nový *obsah*, což už je něco mimo obvyklý rámec prohlížečů. Oba programy by se v rámci prohlížečové části měly chovat stejně, i když, jak se ukáže v sekci 5.2.5, to není vždy pravda.

Posledním proprietárním prohlížečem zahrnutým do testu byl Foxit Reader. Ten je do velké míry obdoba programu Acrobat Reader DC, tentokrát od společnosti Foxit Software Inc. Foxit také nabízí licencovatelnou, komerční variantu pod jménem Foxit PhantomPDF. Vztah mezi těmito dvěma programy je v podstatě stejný jako u produktů Adobe.

Do testu také bylo zařazeno několik open-source prohlížeče. U nich není podpora složitějších funkcí formátu PDF vždy jistá. Množství uživatelů, kteří tyto funkce potřebují, není až tak vysoké. Při uvážení náročnosti implementace je pak důsledkem to, že tyto

vlastnosti zůstavají neimplementovány. Výhodou open-source prohlížečů je ale možnost sdílení kódu. Často například různé prohlížeče pod pokličkou používají pro samotné čtení PDF souborů stejnou knihovnu (pro potřeby této kapitoly také „jádro“) a tato část implementace je za ně již vyřešena. Bohužel zrovna v případě multimédií a interaktivních prvků taková knihovna nepomůže vůbec, protože tyto funkce jsou na mnohem vyšší úrovni, než nízkoúrovňové čtení souboru a jeho syntaktická analýza. Proto má tedy cenu zařadit do testu i více prohlížečů sdílejících stejné „jádro“. Pro výběr konkrétních open-source PDF prohlížečů pro test byla hlavní kritéria jako množství uživatelů, dostupnost (například se může jednat o výchozí prohlížeč některého operačního systému) a také zařazení alespoň nějakého zástupce všech zajímavých PDF „jader“.

Tabulka 5.1 obsahuje seznam testovaných PDF prohlížečů a základní informace o nich. Také obsahuje údaje o konkrétní testované verzi PDF prohlížeče a operační systém, na kterém byly testovány. V případě open-source prohlížečů testovaných v distribuci Linuxu jsou pro reprodukovatelnost testu zajímavé také použité verze mnoha knihoven, které prohlížeč používá. Tranzitivně se ale může jednat o významné množství knihoven, které většinou nemají na základní funkcionalitu PDF prohlížeče takový vliv. Úplně identické zopakování testu je stejně velice obtížné kvůli nutnosti dohledat zpětně všechny knihovny ve správných verzích. Tabulka 5.2 tedy uvádí verze alespoň významných dynamicky linkovaných knihoven[1]. V případě prohlížečů testovaných na Windows není nic podobného nutné, použité instalátory totiž pochází z oficiálních stránek programů a ovlivnit lze tak v podstatě jen základní knihovny operačního systému. Jejich stav lze tedy zachytit jen verzí operačního systému, viz tabulka 5.3.

Tabulka 5.1. Základní informace o testovaných PDF prohlížečích

Prohlížeč	Zkratka	Verze	Jádro	Operační systém
Acrobat Reader DC[i]	AR	21.001.20145	-	Windows
Foxit Reader[ii]	FR	10.1.3.37598	-	Windows
SumatraPDF[iii]	SU	3.2	libmupdf	Windows
Evince[iv]	EV	40.1	poppler	Linux
Okular[v]	OK	21.04.0	poppler	Linux
Xpdf[vi]	XP	4.03	-	Linux
MuPDF-gl[vii]	MU	1.18.0	libmupdf	Linux
Firefox[viii]	FF	88.0	pdf.js	Linux
Google Chrome[ix]	CR	90.0.4430.72	pdfium	Windows

[i] `https://get.adobe.com/reader/`
[ii] `https://www.foxitsoftware.com/pdf-reader/`
[iii] `https://www.sumatrapdfreader.org/free-pdf-reader.html`
[iv] `https://wiki.gnome.org/Apps/Evince`
[v] `https://okular.kde.org/`
[vi] `https://www.xpdfreader.com/`
[vii] `https://mupdf.com/`
[viii] `https://www.mozilla.org/en-US/firefox/new/`
[ix] `https://www.google.com/chrome/`

[1] Tabulka proto například neobsahuje verzi knihovny libmupdf, protože ta obvykle bývá přilinkována staticky a jsou tak již součástí balíčku s prohlížečem. Podobně verze knihoven pdf.js a pdfium jsou jasně určeny konkrétní verzí daného prohlížeče.

Tabulka 5.2. Informace o verzích významných linuxových knihoven

Knihovna	Verze
poppler	21.05.0
poppler-glib	21.05.0
poppler-qt5	21.05.0
gstreamer	1.18.4
gst-plugins-base	1.18.4
gst-plugins-good	1.18.4
gst-plugins-ugly	1.18.4
gst-libav	1.18.4

Tabulka 5.3. Informace o verzích operačních systémů použitých při testování

Operační systém	Verze
Windows	Windows 10
Linux	Arch Linux

Kromě toho, že knihovnu poppler používají dva testované prohlížeče, je také nutné poznamenat, že poppler vychází z jádra prohlížeče Xpdf. Knihovna libmupdf je dílem společnosti Artifex Software, Inc. a prohlížeč MuPDF-gl je, dá se říct jejich referenční implementace prohlížeče používajícího tuto knihovnu. Firefox a Google Chrome jsou webové prohlížeče, ale oba mají ve výchozí instalaci v sobě zabudovanou funkcionalitu PDF prohlížeče. V obou případech dokonce znovuvyužitelnou. V případě Firefoxu je to knihovna pdf.js, která je napsaná v JavaScriptu a mimo její zabudování ve Firefoxu je možné ji použít na jakékoliv webové stránce, která tak získá možnost zobrazovat v sobě PDF dokumenty. Google Chrome používá knihovnu pdfium. Tu lze zkompilovat také bez podpory některých funkcí (XML formuláře, JavaScript), proto byl pro test nakonec zvolen prohlížeč Google Chrome, nikoliv jeho základ, tedy open source prohlížeč Chromium, který nemusí tuto podporu mít přikompilovanou (záleží na distributorovi).

Následující sekce shrnují podporu jednotlivých interaktivních prvků a multimediálních možností v PDF prohlížečích. Podpora je vyobrazena prostřednictvím tabulky, která pro každou sledovanou funkcionalitu (v řádcích) a každý prohlížeč (označený zkratkou z Tabulky 5.1, v sloupcích) obsahuje jeden ze symbolů: „✓“, „–“, „✗“. Tyto značky v tomto pořadí zkráceně označují plnou, částečnou a žádnou podporu sledovaného kritéria v daném prohlížeči. Důvod, proč byla podpora označena pouze jako částečná, je vždy vysvětlen. V zajímavých případech jsou komentovány i další implikace stavu podpory.

5.1 Interaktivní prvky

5.1.1 Nastavení prohlížeče

Praktická možná využití změny nastavení výchozího stavu PDF prohlížeče byla zmíněna v sekci 2.1.1. Jako nejvíce užitečné se zdají být nastavení určená pro prezentace – úvodní zobrazení na celou obrazovku nebo v okně bez uživatelského rozhraní (*user interface*, „UI“) PDF prohlížeče. Pro dokumenty „knižního“ charakteru, které jsou koncipovány pro oboustranný tisk, lze nastavit aby se „oboustrannost“ promítla do zobrazení, ale

také aby se nabídla jako výchozí možnost pro tisk (podpora tohoto přednastavení nemá tedy nic společného s podporou oboustranného tisku samotného). Pro všechny dokumenty může být zajímavé také vynutit zobrazení konkrétní lišty prohlížeče – nejzajímavější jsou lišta se záložkami nebo vloženými soubory. Podporu těchto možností v PDF prohlížečích shrnuje tabulka 5.4.

Tabulka 5.4. Podpora nastavení prohlížeče v PDF prohlížečích

	AR	FR	SU	EV	OK	XP	MU	FF	CR
Zobrazení na celou obrazovku	✓	✓	✗	✓	–	✗	✗	✗	✗
Zobrazení v okně bez UI	–	–	✗	✗	✗	✗	✓	✗	✗
Přednastavení oboustranného tisku	✓	✗	✗	✗	✗	✗	✗	✗	✗
Oboustranné zobrazení	✓	✓	✗	✗	✗	✗	✗	✓	✗
Vynucení konkrétní lišty	✓	✓	✗	✗	–	✗	✗	✓	✗

Prohlížeč Okular se sice pokusil na celou obrazovku zobrazit, v mých podmínkách se mu to bohužel nepodařilo. Okular také nezobrazí lištu vložených souborů (ale záložky zvládá), proto byla podpora označena jako částečná. Jediné dva prohlížeče, které se snaží o podporu zobrazení v okně bez uživatelského rozhraní jsou Acrobat Reader a Foxit Reader. Ani jeden ale neskryje úplně celé uživatelské rozhraní, jak by to napovídala jména odpovídajících parametrů. Prohlížeč MuPDF-gl je minimalistický a žádné uživatelské rozhraní nezobrazuje ani normálně, dalo by se tedy říci, že v tomto ohledu vyhověl, i když omylem.

5.1.2 Anotace

Jak bylo řečeno v sekcích 2.1.2 a 2.1.7, tak obzvlášť v kontextu TEXu a prvotního vytváření dokumentů nemá většina PDF anotací smysl. Jsou totiž určeny pro „značkování“ (*markup*) čtenářem dokumentu.

Při oddělení anotací sloužících pro multimédia (viz sekci 5.2) a také značkovacích anotací, které jsou často podporovány v TEXových balíčcích (viz sekci 5.1.5) zbývají už jen anotace typu `/Link`. Ty jsou ale velmi důležité, protože umožňují vytváření aktivních oblastí na stránce, které vyvolávají akce. Dá se říct, že se jedná o hlavní způsob, jakým jsou akce vyvolávány. Různé druhy PDF akce jsou detailněji popsány v sekci 2.1.4 a testovány v sekci 5.1.6.

Podpora Link anotací byla stanovena jednoduše – PDF prohlížeč podporuje `/Link` anotace, pokud zvládá `/Link` anotaci s alespoň jednou akcí. Skutečnou podporu `/Link` anotací v PDF prohlížečích shrnuje tabulka 5.5.

Tabulka 5.5. Podpora `/Link` anotací v PDF prohlížečích

	AR	FR	SU	EV	OK	XP	MU	FF	CR
`/Link` anotace	✓	✓	✓	✓	✓	✓	✓	✓	✓

`/Link` anotace jsou bez potíží podporovány ve všech testovaných PDF prohlížečích.

5.1.3 Záložky

Záložky (vysvětlené v sekci 2.1.5) jsou bez větších potíží zobrazeny ve všech PDF prohlížečích. To zda jsou správně respektovány všechny akce, je spíš záležitost sekce 5.1.6. Protože jsou ale záložky typicky spjaty se skoky v rámci dokumentu, byla v rámci této sekce testována i funkčnost alespoň jedné jiné akce (konkrétně URI). Automatické zobrazení záložek lze explicitně v PDF dokumentu vyžádat, to testovala již sekce 5.1.1. Skutečnou podporu záložek v PDF prohlížečích shrnuje tabulka 5.6.

Tabulka 5.6. Podpora záložek (*outlines*) v PDF prohlížečích

	AR	FR	SU	EV	OK	XP	MU	FF	CR
Zobrazení záložek	✓	✓	✓	✓	✓	–	✓	✓	✓
Podpora jiných akcí než skoků	✓	✓	✗	✓	✗	✓	✗	✓	✓

Xpdf nezvládá správné zobrazení řetězců kódovaných v UTF-16 Big Endian.

5.1.4 Přechody

Testování přechodů a automatického přepínání stránek (pro detaily viz sekci 2.1.6) je komplikované v tom, že je definováno 12 různých typů animací a většina z nich může být parametrizována. Testování všech možností tak není moc praktické. Proto byla zvolena metodika, kdy přechod je požadován za podporovaný v případě, kdy funguje bez parametrů. Následná funkčnost všech parametrů je předpokládána. Do tohoto testu je také zahrnuta související přechodová akce, která by mohla patřit i do sekce testující většinu ostatních akcí (2.1.4).

Skutečnou podporu přepínání stránek, přechodů, jejich akcí a animací v PDF prohlížečích shrnuje tabulka 5.7.

Tabulka 5.7. Podpora přechodů a přepínání stránek v PDF prohlížečích

	AR	FR	SU	EV	OK	XP	MU	FF	CR
Přepínání stránek	✓	✗	✗	✓	✓	✗	✗	✗	✗
Přechodová akce	✓	✗	✗	✗	✗	✗	✗	✗	✗
Animace Split	✓	✓	✗	✓	–	✗	✗	✗	✗
Animace Blinds	✓	✓	✗	✓	–	✗	✗	✗	✗
Animace Box	✓	✓	✗	✓	✓	✗	✗	✗	✗
Animace Wipe	–	✓	✗	✓	✓	✗	✗	✗	✗
Animace Dissolve	✓	✓	✗	✓	✓	✗	✗	✗	✗
Animace Glitter	✓	✓	✗	✗	✓	✗	✗	✗	✗
Animace Fly	–	✓	✗	✗	✗	✗	✗	✗	✗
Animace Push	✓	✓	✗	✓	✗	✗	✗	✗	✗
Animace Cover	✓	✓	✗	✓	✗	✗	✗	✗	✗
Animace Uncover	✓	✓	✗	✓	✗	✗	✗	✗	✗
Animace Fade	✓	✓	✗	✓	✓	✗	✗	✗	✗

Podpora dvou animací je u prohlížeče Evince označena pouze jako částečná. Důvod je ten, že Evince u ní provádí přechod vertikálně místo horizontálně (což je výchozí). Při bližším prozkoumání se ukázalo, že Evince parametr `/Dm` interpretuje přesně opačně.

Acrobat Reader jako jediný podporuje i přechodové akce. V případě, že je zároveň definován přechod stránky na kterou se provádí skok, tak použije přechod stránky. Bohužel norma zde nespecifikuje žádné konkrétní chování, takže lze tento přístup označit jako vyhovující. Acrobat Reader má ale pro změnu chybu, kdy přechod `/Fly` je sice poprvé proveden, ale přechod proběhne na stejnou stránku. Až podruhé přejde na další stranu (opět s přechodem). Tuto chyby se ale nepodařilo vždy reprodukovat. U přechodu `/Wipe` zase není respektován výchozí směr (zleva doprava), ten je potřeba pro tento prohlížeč vynutit explicitním zadáním `/Di 0`.

5.1.5 Komentáře a přílohy

Skutečnou podporu komentářových (`/Text`) a přílohových anotací (`/FileAttachment`) v PDF prohlížečích shrnuje tabulka 5.8.

Tabulka 5.8. Podpora komentářů, příloh a vložených souborů v PDF prohlížečích

	AR	FR	SU	EV	OK	XP	MU	FF	CR
Komentáře	✓	✓	✗	–	–	✗	–	–	–
Přílohy	✓	✓	✗	✓	✓	–	✓	✓	✗
EmbeddedFiles	✓	✓	–	✓	✓	✓	✗	✓	✗

Google Chrome u komentářů nezvládá řetězce kódované v UTF-16 Big Endian (přitom v jiných kontextech je umí). MuPDF-gl nesprávně zobrazuje název přílohy jako autora. Firefox neumí zobrazit výchozí ikonku přílohy. Chování všech prohlížečů se liší, velikostí a umístěním anotace / její ikonky. Například Firefox umístí výchozí ikonku (vždy o stejné velikosti) podle levého horního rohu anotace. Naopak, pokud je specifikován vlastní vzhled (*appearance stream*), roztáhne ho přes celý obdélník anotace. Google Chrome se chová podobně, ale výchozí ikonku umisťuje podle levého dolního rohu, atd. Všechny prohlížeče s částečnou podporou komentářů mají ale mnohem větší závadu – podle normy tato anotace nerespektuje přiblížení ani rotaci. To znamená, že ikonka je vždy stejná na stejném místě, ať se s okolím děje cokoliv. To je podle mého názoru velice nepraktické – jak je potom například určena prvotní velikost, která se poté už nemění? Především je to ale nepřirozené a tato anotace je jediná, která se tak chová. Mnohem lepší se proto jeví chování prohlížečů nerespektujících normu (v tabulce jsou tyto prohlížeče označeny částečnou podporou). Bohužel je chování napříč nimi v mnoha detailech nekonzistentní.

Prohlížeč Xpdf nezobrazuje výchozí vzhled komentářů ani příloh. Přesto je nejspíš umí zpracovat, protože minimálně u příloh nabízí možnost takto vložené soubory uložit (stejně jako u `/EmbeddedFiles`).

V případě příloh některé prohlížeče při aktivaci anotace nabízí možnost soubor otevřít, jiné uložit. Protože norma nespecifikuje v tomto ohledu nic, nezbývá než považovat všechna chování za přijatelná. Prohlížeč SumatraPDF sice dokáže zobrazit vložené soubory (`/EmbeddedFiles`), nedokáže je ale ani uložit ani otevřít. Proto byla podpora označena jako částečná.

Podle mého názoru jsou obě tyto „ikonové“ anotace nepoužitelné. Pro vložené soubory navíc existuje jistější mechanismus přes `/EmbeddedFiles`.

5.1.6 Akce a skoky

PDF akcí (blíže viz sekci 2.1.4) je definováno více. Ne všemi se tato práce zabývá. A i ty, kterými se tato práce zabývá bylo někdy vhodnější zařadit do jiných sekcí (např. akce

Tabulka 5.9. Podpora skokových akcí v PDF prohlížečích

	AR	FR	SU	EV	OK	XP	MU	FF	CR
GoTo	✓	✓	✓	✓	✓	✓	✓	✓	✓
GoToR	–	✓	✗	–	–	–	✗	–	✗
URI	✓	✓	✓	✓	✓	✓	✓	✓	✓

pro přepínání 3D pohledů jsou testovány v sekci 5.2.4). V této sekci jsou testovány zbývající druhy akcí.

Základní skoky pomocí `/GoTo` akce fungují bez problémů ve všech prohlížečích.

Prohlížeče Acrobat, Evince, Okular, Xpdf nedokáží pomocí `/GoToR` akcí řešit odkazy na dokumenty určené URL adresou. Firefox přesně naopak umí pouze dokumenty určené URL nikoliv cestou. Evince navíc pro odkazy na lokální soubory nezvládá odkaz na konkrétní číslo strany (vždy zobrazí jen první stranu). Okular a Xpdf shodně odkazovaný dokument neotevírají v novém okně ani když je to explicitně vyžádáno.

V případě `/URI` akcí byly testovány URI, které obsahovaly i tzv. fragment (například `#sekce1`). V případě odkazu na PDF dokument by tak mělo dojít k přechodu na pojmenované umístění (viz sekci 2.1.3). Při takovém použití mohou `/URI` akce být jistou náhradou za `GoToR` akce. Bylo testováno pouze obvyklé schéma `http` (případně `https`). Protože jsou URI v principu obecnější, než na co se omezuje tato práce, není stanoveno žádné konkrétní chování. Proto všechny prohlížeče byly označeny jako vyhovující. Přitom se jejich chování lišilo. Některé podle schématu `https` automaticky otevřely prohlížeč. Ale například Okular poznal PDF soubor a pokusil se ho otevřít v systémovém prohlížeči PDF, protože to byl on sám, tak navíc dokázal skočit i na konkrétní pojmenované umístění.

Launch („spouštěcí“) akce byly míněny pro „galantní“ použití – spuštění programů jako je video přehrávač nebo internetový prohlížeč. V případě, že soubor určený ke spuštění není program, měl by se PDF prohlížeč pokusit alespoň o otevření takového souboru v programu, který tento typ souborů zvládá. Tato náhradní možnost není moc užitečná, obzvlášť s přihlédnutím k tomu, že ve většině případů lépe poslouží možnost vložených souborů (viz sekci 5.1.5). Bohužel tvůrci formátu PDF přehlédli dopady, jaké může mít dovolení potenciálně nedůvěryhodným PDF souborům spouštět libovolné programy, v případě operačního systému Windows dokonce i s argumenty! V tomto je rozhodně na místě *absence* podpory v PDF prohlížečích. Tu shrnuje tabulka 5.10.

Tabulka 5.10. Podpora `/Launch` akce v PDF prohlížečích

	AR	FR	SU	EV	OK	XP	MU	FF	CR
Launch	✗	✓	✗	✗	–	✓	✗	✗	✗

Naštěstí žádný prohlížeč nedovolí spustit externí aplikaci jen tak. Foxit Reader a Xpdf sice podporují spouštění externích aplikací, ale napřed se zeptají uživatele jestli opravdu chce program spustit (viz také sekci 5.3). Okular umožňuje otevření externích souborů (umí přinejmenším PDF). Ani Okular ani žádný jiný prohlížeč se značkou „ne“ nedovolí žádné spuštění externí aplikace.

Podporu zbývajících druhů akcí v PDF prohlížečích shrnuje tabulka 5.11.

Pojmenované (`/Named`) akce zahrnují několik základních akcí definovaných normou (viz sekci 2.1.4).

Tabulka 5.11. Podpora zbývajících akcí v PDF prohlížečích

	AR	FR	SU	EV	OK	XP	MU	FF	CR
Named akce	✓	✓	✓	✓	✓	✓	✓	✓	✗
DL JavaScript akce	✓	✓	✗	✗	–	✗	✓	✓	✓
JavaScript akce	✓	✓	✗	✗	–	✗	✗	✗	✗
Podpora JavaScript API	✓	✓	✗	✗	–	✗	–	–	–

S JavaScriptem v PDF prohlížečích je situace složitější. V praxi je nejčastěji využíván pro formuláře (mimo rozsah této práce). To se v praxi odráží i na podpoře v PDF prohlížečích. Některé prohlížeče totiž třeba podporu JavaScriptu mají, ale z velmi rozsáhlého API [1] implementují jen takovou podmnožinu, kterou vyžadují v praxi se vyskytující dokumenty. U prohlížečů Acrobat Reader a Foxit Reader nelze jednoduše ověřit úplnost podpory API, ale ve většině případů se tyto prohlížeče snaží o co největší kompatibilitu.

Formulářový JavaScript je obvykle spouštěn událostmi (např. uživatel něco napsal do textového políčka). Prohlížeče se tedy často omezují na podporu JavaScriptových akcí spouštěných těmito událostmi, nikoliv na obvyklou formu akcí. Rozsáhlé definice funkcí se ale obvykle neumisťují do reakcí na události, ale do inicializačních „dokumentových“ (*document level*) akcí, které prohlížeče také často podporují právě kvůli formulářům. Prohlížeče v této skupině jsou MuPDF-gl, Firefox a Google Chrome.

Jiné obvyklé využití JavaScriptu je přístup ke 3D kontextu 3D děl. Tím se zabývá sekce 5.2.4.

Prohlížeč Okular by měl JavaScript poměrně dobře podporovat. Alespoň to indikovalo nahlédnutí do zdrojového kódu. Nicméně se mi nepodařilo ho nijak zprovoznit. Podpora tak byla označena jako částečná, i když to testování neověřilo.

Kromě `/Link` anotací a reakcí na události jsou častými způsoby vyvolání akcí také záložky (testováno v sekci 5.1.3), ale i takzvaná úvodní akce dokumentu (`/OpenAction`, viz sekci 2.1.4). Její podporu v PDF prohlížečích shrnuje tabulka 5.12.

Tabulka 5.12. Podpora úvodní akce v PDF prohlížečích

	AR	FR	SU	EV	OK	XP	MU	FF	CR
Podpora OpenAction	✓	✓	✗	✗	✗	✗	✗	–	✓

Testování proběhlo s dvěma různými akcemi – `/GoTo` a `/JavaScript`. Výsledkem překvapivě je, že mnoho PDF prohlížečů nepodporuje úvodní akci vůbec. Firefox překvapivě zvládá jen `/JavaScript` akci.

5.2 Multimédia

5.2.1 Sound

Sound anotace nejsou příliš užitečné už ze svého principu (viz sekci 2.2.3. Skutečnou podporu Sounds v PDF prohlížečích shrnuje tabulka 5.13.

Sound objekty jsou podporovány pouze v prohlížečích Acrobat Reader a Foxit Reader. Navíc jen v případě „syrového audia“. To není v podstatě vůbec užitečné, protože vkládání takového audia vyžaduje předzpracování, což je velmi nepraktické. Foxit Reader navíc špatně dekóduje vícekanálové audio.

Tabulka 5.13. Podpora přehrávání audia pomocí Sound anotací v PDF prohlížečích

	AR	FR	SU	EV	OK	XP	MU	FF	CR
Nějaká podpora	✓	✓	✗	✗	✗	✗	✗	✗	✗
Sound akce	✓	✓	✗	✗	✗	✗	✗	✗	✗
Vložené RAW audio	✓	–	✗	✗	✗	✗	✗	✗	✗
Externí MP3 audio	✗	✗	✗	✗	✗	✗	✗	✗	✗
Externí Opus audio	✗	✗	✗	✗	✗	✗	✗	✗	✗

5.2.2 Movie

Movie anotace (viz sekci 2.2.4) jsou velmi jednoduché, přesto se zde najde několik zajímavých kategorií k otestování. Kromě toho, zda se PDF prohlížeč vůbec o podporu Movie snaží, je testovaným ohledem i to, zda umí zobrazit ovládací prvky a respektuje další parametry (jako plakát).

Movie (ale i další mechanismy z následujících sekcí) přehrávají soubor určený specifikací souboru, která může odkazovat v podstatě na 3 druhy souborů – vložený v PDF (dále také „Vložený soubor“), externí určený URL (dále také „URL soubor“), externí určený cestou (dále také „Externí soubor“), podrobněji viz sekci 2.2.4. Dalším kritériem jsou též přehratelné formáty. V rámci testu v této i dalších sekcích potom byly vyzkoušeny všechny tři varianty specifikace souboru a různé formáty audia/videa a to následující metodikou:

- Zkouška každého testovaného formátu se specifikací souboru typicky spjatou s daným mechanismem. (např. pro Movie je to externí soubor, pro Renditions vložený soubor).
- Pokud je daný formát podporován touto základní specifikací souboru, je označen jako podporovaný.
- Aby byla označená jiná možnost specifikace souboru jako podporovaná, stačí, aby fungovala s jedním podporovaným formátem.

Skutečnou podporu Movies v PDF prohlížečích shrnuje tabulka 5.14.

Tabulka 5.14. Podpora přehrávání audia/videa pomocí Movie anotací v PDF prohlížečích

	AR	FR	SU	EV	OK	XP	MU	FF	CR
Nějaká podpora	✗	✗	✗	✓	✓	✗	✗	✗	✗
Nastavení parametrů	✗	✗	✗	✗	✗	✗	✗	✗	✗
Ovládací prvky	✗	✗	✗	✓	✓	✗	✗	✗	✗
Movie akce	✗	✗	✗	✗	✗	✗	✗	✗	✗
Externí soubor	✗	✗	✗	✓	✓	✗	✗	✗	✗
Vložený soubor	✗	✗	✗	✓	✓	✗	✗	✗	✗
URL soubor	✗	✗	✗	✓	✓	✗	✗	✗	✗
MP3 audio	✗	✗	✗	–	–	✗	✗	✗	✗
Opus audio	✗	✗	✗	–	–	✗	✗	✗	✗
AVI video	✗	✗	✗	–	–	✗	✗	✗	✗
MP4 video	✗	✗	✗	–	–	✗	✗	✗	✗
MPG video	✗	✗	✗	–	–	✗	✗	✗	✗
MOV video	✗	✗	✗	–	–	✗	✗	✗	✗
WMV video	✗	✗	✗	–	–	✗	✗	✗	✗

Jediné prohlížeče podporující Movies jsou Evince a Okular. Ty sice používají stejnou knihovnu pro čtení PDF souborů (poppler), ale jejich implementace Movies jsou nezávislé. Důvod proč podporují v podstatě stejné formáty (a to s poznámkou „částečně“) je ten, že oba pro zpracování multimediálních souborů využívají GStreamer.

GStreamer je knihovna, která nabízí široké možnosti práce s audiem/videem – mixování audia, střih videa, ale i prosté přehrávání, které právě využívají Evince a Okular. Důležité ohledně GStreameru je jeho architektura. Ta je velmi obecná a práce s různými formáty je na abstraktní úrovni. Podpora jednotlivých formátů poté přichází ve formě plug-inů, většinou potom ve dvojicích (pro čtení a zápis). GStreamer tedy teoreticky dokáže přehrát jakýkoliv formát. Prakticky záleží na tom, které plug-iny jsou nainstalovány.

Které plug-iny GStreameru budou nainstalovány při instalaci Evince nebo Okularu, v podstatě záleží na dané distribuci Linuxu. V mém testu na distribuci Arch Linux (viz sekci 5) ani jeden prohlížeč neměl jako závislost žádný plug-in. Tím pádem nefungovalo přehrání žádného formátu. Plug-iny bylo třeba doinstalovat ručně. Obvykle jsou distribuovány ve skupinách (balících) a může se stát, že podporu pro čtení určitého formátu poskytuje i více plug-inů z různých balíků. Například podporu všech testovaných formátů lze docílit instalací následujících balíků:

- gst-plugins-good
- gst-plugins-ugly
- gst-libav

Podpora i základních formátů je dána přítomností správných plug-inů. V praxi záleží na distribucích a úplnosti instalací systémů, které mohou, ale nemusí, potřebné pluginy obsahovat. Z tohoto důvodu byla u prohlížečů Evince a Okular označena podpora všech sledovaných formátů pouze jako částečná – nelze se na ni spolehnout.

5.2.3 Multimédia (Renditions)

Mechanismus multimédií, detailně popsaný v sekci 2.2.5 je velice obecný a teoreticky podporuje přehrávání čehokoliv. Typické užití je ale zamýšleno pro audio/video a historicky také Flash. V úvahu připadá i použití s obrázky, ale tam již existuje tradiční řešení (viz sekci 2.2.2). Výchozí způsob specifikace souboru je pro tento mechanismus „Vložený soubor“.

Podpora přehrávání Flashových souborů nebyla testována vůbec, protože od Flashe již bylo úplně upuštěno. Testováno tak bylo pouze přehrávání videa a audia.

Dále nebyly testovány možnosti pro výběr jednoho konkrétního multimédia ze stromové struktury či širší možnosti nastavení pomocí „Best effort“ a „Must honor“. Oboje je problematické a v praxi v podstatě nevyužité (viz sekci 2.2.5). Účelem této práce je mimo jiné otestovat, které typy multimédií přehrát *lze*. Potom nemá smysl do PDF vkládat multimédií více a nechat PDF prohlížeč rozhodnout, které přehrát. V místech, kde Renditions umožňují stanovit omezení („Must honor“, nepoužití dočasných souborů, atd.) byly nastaveny co nejlaxnější hodnoty, aby nebyly prohlížeče zbytečně omezovány. Skutečnou podporu Multimédií (Renditions) v PDF prohlížečích shrnuje tabulka 5.15.

Podpora v prohlížečích Acrobat Reader a Foxit Reader je z hlediska tohoto testu v podstatě kompletní. Mírně zaostává jen podpora různých formátů audia/videa. To je dáno tím, že oba prohlížeče pro přehrávání používají vloženou instanci programu Windows Movie Player, z něhož omezení formátů vychází. Poměrně zajímavým poznatkem z tohoto testování je, že pro přehrání přes Windows Movie Player oba prohlížeče vy-

Tabulka 5.15. Podpora přehrávání audia/videa pomocí mechanismu multimédií (Renditions) v PDF prohlížečích

	AR	FR	SU	EV	OK	XP	MU	FF	CR
Snaha o podporu	✓	✓	✗	✓	✓	✗	✗	✗	✗
Podpora úseků	✓	✗	✗	✗	✗	✗	✗	✗	✗
Nastavení parametrů	✓	✓	✗	✗	✗	✗	✗	✗	✗
Ovládací prvky	✓	✓	✗	✓	✓	✗	✗	✗	✗
Rendition akce	✓	✓	✗	✗	✗	✗	✗	✗	✗
Vložený soubor	✓	✓	✗	✓	✓	✗	✗	✗	✗
URL soubor	✗	✓	✗	✓	✓	✗	✗	✗	✗
Externí soubor	✗	✓	✗	✗	✗	✗	✗	✗	✗
MP3 audio	✓	✓	✗	–	–	✗	✗	✗	✗
Opus audio	✗	✗	✗	–	–	✗	✗	✗	✗
AVI video	✓	✓	✗	–	–	✗	✗	✗	✗
MP4 video	✓	✓	✗	–	–	✗	✗	✗	✗
MPG video	✓	✓	✗	–	–	✗	✗	✗	✗
MOV video	✗	✗	✗	–	–	✗	✗	✗	✗
WMV video	✓	✓	✗	–	–	✗	✗	✗	✗

tváří dočasný soubor, což musí být explicitně povoleno v PDF struktuře Rendition objektu. To například opomíjí aktuální implementace v ConTEXtu, proto pro přehrávání audia/videa selhává. I když oba prohlížeče používají stejný přehrávač, oba se chovají jinak – například ovládací prvky umisťuje Acrobat Reader tak, že vystupují z oblasti anotace (směrem dolu). Foxit Reader umisťuje ovládací prvky do dolní části oblasti anotace – tedy ubírají prostor multimédiu, což zapříčiňuje jiný poměr stran.

Evince a Okular oba překvapivě podporují tento složitý mechanismus. Při pohledu do jejich implementací je ale vidět, že používají stejný kód pro podporu Movie i Renditions. Z toho plynou četná omezení, jako absence akcí (tím pádem i například automatické přehrávání) nebo úseků klipů. Jednoduše proto, že Movie nic takového neumí. Nefunkčností je zde ale více. V Okularu se multimédium začne *vždy* automaticky přehrávat. Důvodem je, že Okular interpretuje špatně položku `autoplay`, která neintuitivně pro tento účel neslouží). Evince nezvládá soubory testující přehrávání externích souborů a úseků ani otevřít (padá na „segmentation fault“). Další zádrhely podpopry Renditions jsou shodné s těmi popsanými v sekci 5.2.2. Chování obou prohlížečů z hlediska umístění ovládacích prvků je totožné s Acrobat Readerem.

5.2.4 3D díla

3D díla, jejichž podporou v PDF prohlížečích se zabývá tato sekce, jsou popsána v sekci 2.2.6. Ani ten popis bohužel nebyl kompletní a 3D díla mají velké množství dalších parametrů. Již jejich vyčerpávající výčet by byl dlouhý a jejich důsledné otestování velice náročné a pro potřeby této práce i zbytečné.

Pro otestování podpory v PDF prohlížečích jsem tak zvolil obecnější kritéria, která kritizují nějakou větší skupinu možností/funkcí spjatých s 3D díly. Základním kritériem je něco, co by se dalo nazvat základní podporou 3D děl. Tím je myšlena podpora 3D anotací a streamů s více (i složitějšími) pohledy a různými aktivačními možnostmi. Do zvláštní kategorie patří podpora rozšíření z PDF verze 1.7 (jako nastavení režimu vy-

kreslování, osvícení, ...). Odděleně je sledována podpora různých formátů 3D souborů (U3D, PRC) a složitější funkce:

- podpora ovládání JavaScriptem – úvodní skript i JavaScript akce,
- možnost uživatelské interakce pomocí myši,
- možnost tisku.

V podstatě stejné PDF struktury popisující 3D scénu lze zabalit jak v 3D streamu/anotaci, tak i v Rich Media anotaci. Jejich podporou v prohlížečích se zabývá sekce 5.2.5. Pro obojí by měly fungovat také `/GoTo3DView` akce, které byly testovány v této sekci (místo 5.1.6).

Skutečnou podporu 3D děl v PDF prohlížečích shrnuje tabulka 5.16.

Tabulka 5.16. Podpora 3D děl zabalených ve 3D anotacích v PDF prohlížečích

	AR	FR	SU	EV	OK	XP	MU	FF	CR
Základní podpora	✓	✓	✗	✗	✗	✗	✗	✗	✗
Externí stream	✗	✗	✗	✗	✗	✗	✗	✗	✗
PDF 1.7 rozšíření	✓	✓	✗	✗	✗	✗	✗	✗	✗
Ovládání JavaScriptem	✓	✗	✗	✗	✗	✗	✗	✗	✗
Interakce	✓	✓	✗	✗	✗	✗	✗	✗	✗
Tisk	✓	✗	✗	✗	✗	✗	✗	✗	✗
GoTo3DView akce	✓	✗	✗	✗	✗	✗	✗	✗	✗
Formát U3D	✓	✓	✗	✗	✗	✗	✗	✗	✗
Formát PRC	✓	✓	✗	✗	✗	✗	✗	✗	✗

Acrobat Reader zvládá všechny testy bez potíží. Jen pro tisk 3D anotací je potřeba nezapomenout povolit tisk (volbou *Document and Markups*). Prohlížeč Foxit Reader má podobnou volbu tisku anotací také, ale pro 3D anotace nefunguje. Tento prohlížeč navíc nepodporuje ani ovládání JavaScriptem – jak v případě akcí, tak pomocí inicializačních skriptů.

Bohužel ani jeden prohlížeč nepodopruje možnost externích streamů, tedy že by 3D data nebyla přímo ve streamu s typem `/3D`, ale v místě určeném specifikací souboru. To by umožnilo vícenásobné využití jedněch 3D dat s více různými nastaveními nebo zabalení 3D souboru také jako přílohu (`/EmbeddedFile`, viz sekci 2.1.7).

5.2.5 Rich Media

Bohatost Rich Médií (viz sekci 2.2.7) byla postavena na Flashi, který byl používán ve formě různých přehrávačů i pro přehrávání audia a videa. Alternativní cestou vždy bylo přehrávání videa a audia přímo. To je naopak velmi chudá možnost – neumožňuje zobrazení ovládacích prvků, akce, atd.

Přesto, že dnes logicky nelze očekávat podporu Flash *aplikací* v PDF prohlížečích, nic prohlížečům nebrání emulovat Flash *přehrávače* a zobrazit tak alespoň ovládací prvky. Takto by zůstalo možné alespoň částečné zobrazení starších dokumentů (mimo jiné i těch vytvořených pomocí balíčků media9 a starších verzí rmannot, viz sekci 4.1.4 a sekci 4.1.5).

Skutečnou podporu Rich Media v PDF prohlížečích shrnuje tabulka 5.17.

Situace je zde poměrně smutná, protože téměř žádný prohlížeč Rich Media anotace nezvládá. Ve Foxit Readeru dříve fungovaly díky Flashi, ale tento prohlížeč nemá žád-

Tabulka 5.17. Podpora Rich Media v PDF prohlížečích

	AR	FR	SU	EV	OK	XP	MU	FF	CR
Nějaká podpora	✓	✗	✗	✗	✓	✗	✗	✗	✗
Automatické přehrávání	✓	✗	✗	✗	✓	✗	✗	✗	✗
„Obyčejné“ audio/video	✓	✗	✗	✗	✗	✗	✗	✗	✗
Flash kompatibilita	✓	✗	✗	✗	✓	✗	✗	✗	✗
MP3 audio	✓	✗	✗	✗	–	✗	✗	✗	✗
Opus audio	✗	✗	✗	✗	–	✗	✗	✗	✗
AVI video	✓	✗	✗	✗	–	✗	✗	✗	✗
MP4 video	✓	✗	✗	✗	–	✗	✗	✗	✗
MPG video	✓	✗	✗	✗	–	✗	✗	✗	✗
MOV video	✗	✗	✗	✗	–	✗	✗	✗	✗
WMV video	✓	✗	✗	✗	–	✗	✗	✗	✗

nou vrstvu kompatibility pro starší dokumenty, a ani nefunguje s „obyčejným“ audiem a videem.

Zajímavá je ale situace prohlížeče Acrobat Reader. Ten totiž kromě toho, že zvládá přehrávání obyčejného audia a videa (stejným způsobem jako pro „multimedia“, viz sekci 5.2.3) má i kompatibilní vrstvu pro přehrání starších dokumentů, které vyžadují Flash. Tato vrstva není nijak zdokumentována. Navíc není úplně dokonalá, protože přehrávače, které měly zabudovány ovládací prvky jsou emulovány bez nich. Díky dalším uživatelům balíčku media9[2] se mi podařilo odhalit, že Acrobat Reader dokáže emulovat i ovládací prvky. K tomu nastává, když detekuje použití *skinu* („vzhledu přehrávače“). Protože jsou kontroly přítomnosti skinu a přehrávače prováděny nezávisle, lze tak docílit zobrazení ovládacích prvků i pro obyčejné ne-Flashové video/audio. Právě podle ovládacích prvků lze poznat, že pro přehrávání je použit Windows Media Player (jako v případě „multimedia“, viz sekci 2.2). V konečném výsledku tak v aktuální verzi Acrobat Readeru lze pomocí Rich Media docílit stejného výsledku jako s multimedia/renditions. Toto chování je ale výsadou pouze tohoto prohlížeče. Navíc aktuální verze obsahuje dva problémy, které jde v tuto chvíli obejít pomocí návodu zaměstnance Adobe Himanshu Sagara[3].

Prohlížeč Okular, který nikdy Flash nepodporoval, má také podobnou vrstu kompatibility jako Acrobat Reader. Bohužel neumí v tomto režimu zobrazit ovládací prvky, správně u něj narozdíl od Renditions (viz sekci 2.2.5) funguje automatické přehrávání. Přehrávání obyčejného, ne-Flashového videa/audia zabaleného v Rich Media anotacích Okular bohužel neumí.

3D díla zabalená v Rich Media anotacích by (alespoň teoreticky) měly mít minimálně stejné možnosti jako klasické 3D anotace (několik malých výjimek je zmíněno v sekci 2.2.7). Jejich podpora je shrnuta v sekci 5.2.4. Prakticky se ukazuje, že Foxit Reader přestal tento typ anotace podporovat s ukončením podpory Flashe. To je rozhodně zvláštní, protože až na obskurní případy nemá Flash s 3D díly nic společného. Tabulka 5.18 shrnuje podporu 3D Rich Media anotací.

Testování akcí `/RichMediaExecute` nemělo smysl. V kontextu Flashe jsou již nepoužitelné a pro 3D modely také nejsou nikterak zvlášť užitečné.

[2] `https://gitlab.com/agrahn/media9/-/issues/9`

[3] `https://community.adobe.com/t5/acrobat-reader/playing-embedded-video-without-flash/td-p/11266205/page/2`

Tabulka 5.18. Podpora 3D děl zabalených v Rich Media anotacích v PDF prohlížečích

	AR	FR	SU	EV	OK	XP	MU	FF	CR
Základní podpora	✓	✗	✗	✗	✗	✗	✗	✗	✗
PDF 1.7 rozšíření	✓	✗	✗	✗	✗	✗	✗	✗	✗
Ovládání JavaScriptem	✓	✗	✗	✗	✗	✗	✗	✗	✗
Interakce	✓	✗	✗	✗	✗	✗	✗	✗	✗
Tisk	✓	✗	✗	✗	✗	✗	✗	✗	✗
Formát U3D	✓	✗	✗	✗	✗	✗	✗	✗	✗
Formát PRC	✓	✗	✗	✗	✗	✗	✗	✗	✗

Principiální výhoda Rich Media pro 3D díla je, že všechny soubory jsou zabaleny ve specifikacích souboru. Je tedy možné snadné využití jednotlivých souborů pro více Rich Media anotací nebo i pro účely příloh/`//EmbeddedFiles` (viz sekci 2.1.7). Bohužel je to za cenu zbytečně velmi složitých PDF struktur a ani tak není možné specifikace souboru využít ve vší obecnosti – určení souboru cestou nebo pomocí URL není v tomto případě dovoleno.

5.3 Bezpečnostní aspekty

Prohlížeče Acrobat Reader a Foxit Reader se v některých „speciálních situacích“ (například rozkliknutí odkazu do webu, započetí přehrávání videa, otevření přiloženého souboru) nejdříve dotáží uživatele, zda chce daný úkon skutečně provést. Ostatní prohlížeče, pokud podobný úkon vůbec podporují, se neptají a konají ihned.

Na jednu stranu jsou bezpečnostní opatření logická, protože v některých kontextech uživateli nemusí být jasné, že může dojít k vykonání něčeho potenciálně nebezpečného. Na druhou stranu to značně snižuje použitelnost dokumentů využívající pokročilé možnosti interaktivních prvků a multimédií. Například pokud se má video začít automaticky přehrávat, je místo toho uživatel dotázán, zda si přeje s přehráváním začít. Vzhledem k tomu, že tyto prvky jsou využívány hlavně při prezentacích, tak to přirozeně silně kazí dojem, který prezentace vyvolává.

Na druhou stranu může být i přehrávání videa z bezpečnostního ohledu velmi rizikové. Protože se prohlížeče většinou spoléhají na externí programy/knihovny, jsou jim i jejich zranitelnostem dány napospas. Ochrana před kliknutím na škodlivý odkaz je také užitečná, za předpokladu, že PDF dokument nemá důvěryhodný původ (což je většina případů).

Celkově je chování prohlížečů Acrobat Reader a Foxit Reader krok směrem k lepší bezpečnosti a horšímu uživatelskému zážitku, jak to (bohužel) často v případě bezpečnosti bývá.

Při pohledu do zdrojového kódu prohlížeče MuPDF-gl mě zaujala implementace funkce `pdf_execute_action_chain`. Ta se stará o spouštění potenciálně zřetězených akcí. Pro zřetězené akce (`/Next`) používá rekurzi. To znamená, že vhodně sestrojený dokument může zapříčinit pád aplikace přetečením zásobníku. Nejedná se o nijak zajímavý „útok“, ale upozorňuje to na skutečnost, že PDF prohlížeče pracují s nedůvěryhodným vstupem. Malá programátorská zaváhání mohou v širším kontextu vyjevit potenciální zranitelnost. Jinak je ale MuPDF-gl velice tolerantní prohlížeč a dokáže otevřít a v rámci možností zpracovat i dokumenty nesprávné struktury. Při mém testování nikdy nespadl. Podobná věc se nedá říct o prohlížeči Evince, který byl na pády nejnáchylnější. Protože

jsem podobný problém nezaznamenal při testování prohlížeče Okular, který používá stejnou knihovnu poppler, tak se pravděpodobně jedná o „netolerantnost“ samotného prohlížeče Evince. PDF prohlížeče by se měly snažit o co největší odolnost proti špatnému vstupu. Všechen jejich vstup je totiž nedůvěryhodný. To je zvlášť důležité při interpretování uživatelských skriptů napsaných v JavaScriptu.

Právě JavaScript je další velkou hrozbou při prohlížení PDF souborů. Velká část dostupného API [1] je sice relativně neškodná, je ale přirozené, že čím je definovaných funkcí více, tím je větší šance, že se v kódu interpretu nachází chyba, která umožní útočníkovi opustit omezené prostředí. Větším problémem jsou ale některé dobře zamýšlené, avšak snadno zneužitelné funkce v JavaScript API – čtení souborů, možnost připojení k internetu, odesílání e-mailů, atd. Všechny mohou vést ke vzdálené kompromitaci uživatele.

Naštěstí se většina prohlížečů omezuje jen na určitou užitečnou část JavaScript API, takže u nich velké nebezpečí nehrozí.

5.4 Zhodnocení

Z celkového množství interaktivních prvků, které mají smysl v kontextu TEXu (určeny pro primární vkládání, atd.) se ukázalo, že jen minimum funguje ve všech testovaných prohlížečích. Jednalo se pouze o `/Link` anotace s oběma nejčastějšími akcemi – `/GoTo` a `/URI`. Potažmo o čtyři základní pojmenované akce, které nefungovaly jen v prohlížeči Google Chrome, založeném na knihovně pdfium.

Celkově podpora různých druhů akcí ve všech kontextech, kde se mohou vyskytnout je slabá. Výjimku tvoří PDF prohlížeče Acrobat Reader a Foxit Reader, které obstály téměř ve všech interaktivních aspektech. V případě omezení se na určitou podmnožinu interaktivních prvků jsou obstojně použitelné i prohlížeče Evince a Okular. Naopak prohlížeče Xpdf, MuPDF-gl, Firefox a Google Chrome nepodporují funkcionality užitečné pro prezentace, třeba přechody nebo úvodní zobrazení na celou obrazovku.

V oblasti audia/videa jsou k dispozici tři mechanismy, kterými lze dosáhnout podobné výsledky. Nejstarší, jednoduché (ale i omezené), mechanismus Movies nebo Sounds již nejsou rozumně podporovány v žádném prohlížeči na Windows. Nejnovější mechanismus Rich Media potom ještě stále trpí kvůli nedávné ukončení podpory Flashe. Jak vrstvy kompatibility pro přehrávání Flashového audia/videa (Acrobat Reader a Okular), tak i možnosti přehrávání obyčejného audia/videa (Acrobat Reader) jsou dnes velmi omezené.

Nejlépe tak vychází mechanismus „Multimedia“ (Renditions). Ten je podporován jak největším množstvím prohlížečů (Acrobat Reader, Foxit Reader, Evince, Okular), tak také nabízí relativně příjemné množství přizpůsobení a ovládání. Bohužel možnosti přizpůsobení u open source prohlížečů (Evince, Okular) jsou velmi omezené (nefungují akce pomocí kterých by například bylo možné docílit automatické přehrávání, ignorováno je nastavení zvuku, nejsou podporovány úseky, atd.). Přesto se jedná o nejlepší možnost jak dnes audio/video přehrávat.

V případě 3D děl připadají v úvahu jen prohlížeče Acrobat Reader a Foxit Reader na Windows. Podpora prohlížeče Acrobat Reader je kompletní – široké možnosti uzpůsobení, skriptování, možnost zabalení 3D objektů v 3D anotacích i Rich Media anotacích, atd. Foxit Reader naproti tomu zvládá pouze 3D anotace a skriptování v něm nefunguje. Protože ale podporuje nastavení většiny parametrů přes PDF struktury, tak je bez problému použitelný pro „statické“ pohledy.

U PDF prohlížečů jako programů pracujících s nedůvěryhodným vstupem se očekává, že berou bezpečnost vážně. Rozhodně tomu tak je u PDF prohlížečů Acrobat Reader a Foxit Reader, které před si každou potenciálně nebezpečnou činnost nechají interaktivně schválit uživatelem. Open source prohlížeče to většinou nedělají, na druhou stranu většinou nepodporují velké množství potenciálně nebezpečných funkcionalit.

Kapitola 6
Návrh balíčku pro OpTEX

Cílem této kapitoly je popsat návrh balíčku pro TEXový formát OpTEX[1], který byl vytvořen jako součást této práce. Tato praktická kapitola využívá poznatky z předešlých kapitol, které popsaly, jaké možnosti interaktivní prvků a multimédií nabízí norma PDF (kapitola 2), co implementují jiné TEXové balíčky (kapitola 4) a také co funguje v praxi (kapitola 5). Samotná implementace potom staví na informacích z kapitoly o dostupných primitivech LuaTEXu (kapitola 3).

Balíček jsem se pragmaticky rozhodl vytvořit pouze pro OpTEX. Jak vyplývá z kapitoly 3, bylo by nejuniverzálnějším řešením naimplementovat balíček tak, aby se spoléhal jen na primitivy pdfTEXu a případně na další makra ze základního formátu „plainTEX“, která jsou dostupná i ve většině ostatních formátů (včetně OpTEXu). Podle mého názoru dnes není důvod používat pdfTEX – LuaTEX je mnohem lepší volbou. Vychází z pdfTEXu a významně ho rozšiřuje (viz sekci 3.3). Na druhou stranu čistý „plain LuaTEX“ není moc připravený na moderní využití. To a mnoho dalšího napravuje OpTEX. OpTEX je poměrně nový formát pro TEX. I když vychází ze zavedených maker OPmac[2], jde na věci z větší šířky. Především se jedná o plnohodnotný formát, omezující se pouze na LuaTEX. Jeho zaměřením je dvojí jednoduchost – jednoduchost použití a jednoduchost implementace. Související myšlenkou je také neuniverzálnost – očekává se, že pokud uživateli nevyhovuje stávající chování, tak ho předefinuje po svém.

Z důvodu stylu jakým se programují externí balíčky pro OpTEX, toto rozhodnutí bohužel znamená, že balíček nebude přímo použitelný pro plain pdfTEX a bez větších úprav ani pro plain LuaTEX. Na druhou stranu, jak vyjde najevo později, tak podobný balíček vyžaduje existenci hned několika mechanismů (zapisování souborů s pomocnými daty, key-value parametry nebo částečně i správa barev). Pokud by balíček implementující interaktivní prvky a multimédia sám implementoval i ony zmíněné mechanismy, znamenalo by to, že může být špatně integrovatelný do *všech* makro balíků. Omezení na OpTEX přitom znamená možnost využít stávající mechanismy OpTEXu, které jsou navíc jednoduché a adaptovatelné pro jednodušší formáty.

V souladu s výše uvedeným byly pro výsledný balíček byly předem stanoveny následující cíle:

- Implementovat to, co v praxi opravdu *funguje*.
- Žádná část implementace by se neměla rozcházet s tím, co tvrdí *norma* [9]. V takovém případě by se jinak mohlo stát, že balíček bude fungovat dnes, ale může přestat fungovat při nepředvídaných změnách v budoucnosti.
- Přirozená integrace s OpTEXem. Zároveň by se mělo jednat o čistě oddělený externí balíček, který nijak nemění stávající chování. S tím souvisí i konzistentnost uživatelského rozhraní. Příklad – nepovinné parametry a parametry, které se nedostanou do sazby, jsou v hranatých závorkách, kdežto parametry, které se přímo promítnou v sazbě jsou ve složených závorkách.

[1] `http://petr.olsak.net/optex/`
[2] `http://petr.olsak.net/opmac.html`

- Jednoduchost implementace. Z praktických důvodů nemá smysl vyhýbat se využití mechanismů OpTEXu. Ať je potom implementace tak jednoduchá, že ji lze přenést i do jiných systémů.

Volbu výběru implementované funkcionality a poznámky k implementaci samotné (včetně srovnání některých rozhodnutí s ostatními balíčky) obsahují následující sekce.

6.1 Zvolená funkcionalita

Výběr funkcionality pro začlenění do balíčku je silně ovlivněn závěry sekce 5.4. Pro implementaci byly zvoleny následující možnosti formátu PDF:

- nastavení výchozího stavu prohlížeče (zobrazení na celou obrazovku, přednastavení oboustranného tisku, oboustranné zobrazení, vynucení konkrétní lišty);
- přechody, přepínání stránek;
- `/Link` anotace;
- vložené soubory (`/EmbeddedFiles`);
- akce (`/GoTo`, `/GoToR`, `/JavaScript` (včetně inicializačních), `/Named`), úvodní akce, přídavné akce;
- mechanismus multimédií („Renditions“), včetně asociovaných anotací, objektů a akcí;
- Rich Media anotace;
- nastavení 3D pohledů, vkládání „3D JavaScriptu“, `/GoTo3DView` akce.

Pro přehrávání audia/videa byl zvolen mechanismus multimédií (Renditions), který funguje v největším množství PDF prohlížečů.

3D díla budou podporována pouze zabalená v Rich Media anotacích. I když to znamená, že nebudou fungovat v prohlížeči Foxit Reader, má to nezanedbatelné výhody. 3D anotace totiž vyžadují, aby 3D data byla obsažena ve 3D streamech. Mezi parametry streamů jsou ale i nastavení pohledů. Nelze tak jedna 3D data pohodlně využít vícekrát ani je prezentovat v uživatelském rozhraní jako vložený soubor. 3D streamy také dovolují nastavit pouze jeden inicializační JavaScript stream. To se nejprve nemusí zdát jako velké omezení, ale neumožňuje to snadnou kompozici a znovuvyužití jednotlivých souborů se skripty. Obě omezení bohužel znamenají, že 3D anotace nezapadají do „všeobecného“ řešení souborů navrženého pro tento balíček.

U interaktivních prvků je mnohem složitější určit co lze považovat za natolik fungující, že bude nabídnuto koncovým uživatelům. Na druhou stranu není implementačně složité podporovat všeobecně třeba všechny různé akce, pro použití ve všech možných kontextech, ale především pro aktivní oblasti na stránkách (`/Link` anotace).

V prohlížečích Acrobat Reader, Foxit Reader, Evince a Okular fungují relativně dobře možnosti užitečné pro prezentace (přechody a výchozí přepnutí do režimu celé obrazovky). Nezobrazení těchto prvků navíc neznamená úplnou nepoužitelnost dokumentů. Proto je na místě jak implementace, tak i využití uživateli. Podobně lze uvažovat o vynucení zobrazení jedné z lišt (záložky a vložené soubory).

Záložky nebyly vybrány k implementaci, jelikož OpTEX již záložky generovat umí.

6.2 Poznámky k implementaci

V této sekci jsou popsány některé důležité aspekty implementace, které se promítly také do uživatelského rozhraní. Nejedná se o detailní popis kódu (technickou dokumentaci) ani o uživatelský manuál. Ty jsou součástí balíčku samotného (příloha B). Naopak cílem této sekce je také srovnání některých rozhodnutí s ostatními TEXovými balíčky.

6.2.1 Správa souborů

Cílem implementace balíčku byla všeobecnost některých konstrukcí. Například, když už je jeden soubor vložen do PDF (případně odkazován pomocí cesty či URL), lze ho zabalit do PDF specifikace souboru jen jednou a umožnit jeho vícenásobné využití (pomocí uživatelem zvoleného jména). Protože se specifikace souboru objevují nejen v souvislosti s multimédii, ale i s interaktivními prvky, tak má takové rozhodnutí celkem podstatné následky. Někdy totiž všechny tři druhy specifikace souboru nejsou zaměnitelné – přirozeně například pro odkaz do externího PDF dokumentu (akce `/GoToR`) nelze využít specifikaci souboru odkazující na vložený soubor. Ale existují i další omezení, která jsou dána čistě požadavky normy – například Rich Media anotace umí pracovat jen se specifikacemi vložených souborů.

Při stejném chování ke všem druhům specifikací souborů není možné odchytit všechny uživatelské chyby. I přesto jsem se pro tento způsob rozhodl. Je to i proto, že ve většině případů chce uživatel použít vložený soubor. Není tak obtížné v místech, kde je očekávána specifikace souboru, zkontrolovat, jestli je daná specifikace souboru definována. V případě, že není, může se kód pokusit interpretovat jako náhradní možnost poskytnuté jméno jako cestu k souboru a automaticky ho vložit (pod jménem, které odpovídá cestě). Další využití stejné cesty, tak využije již vložený soubor. Ve většině případů se tak uživatel může spolehnout na tento náhradní mechanismus. V případě, že chce použít nevložený soubor určený cestou nebo URL, musí ho ručně předem definovat. Vložený soubor se ale může také hodit předdefinovat – například pokud je cesta k souboru dlouhá a uživatel chce soubor použít vícekrát bez jejího vícenásobného zadání.

Jiné řešení vícenásobného využití stejných souborů používá balíček media9. Jeho možnosti na vyšší úrovni popisuje sekce 4.1.4. Ten rozeznává stejné soubory podle MD5 hashe. To má oproti výše zmíněnému řešení výhodu, že rozpozná identické soubory z různých umístění v souborovém systému (což ale není velmi častá situace). Další výhodou je, že to rozpozná stejný soubor *vždy*. To neplatí v případě přidělení jména souborům, protože s předefinováním daného jména je soubor původně asociovaný se jménem zapomenut.

Primitivy pdfTEXu podporují koncept vícenásobného využití PDF objektů a v případě, že není objekt využit ani jednou, tak není vůbec zapsán do výsledného PDF souboru (detailněji popsáno v sekci 3.4). To by teoreticky umožnilo makrům z implementovaného balíčku například nevložení souboru, který nakonec není využit (například když je využit jen v boxu, který se nedostane do finálního výstupu). Kromě toho, že dosáhnutí tohoto výsledku vyžaduje konzistentní použití primitivu `\pdfrefobj` pro zvýšení počtu referencí PDF objektu (což je obtížné obzvlášť v „expansion only“ kontextu, ale řešitelné pomocí Lua funkce), tak to navíc není možné pro formy, protože jejich počet refrencí nelze zvýšit pomocí `\pdfrefobj`. Formy (*form XObject*, viz sekci 2.2.1) jsou v balíčku použity pro vytváření vzhledu anotací (*appearance stream*, viz sekci 2.1.2). Nakonec balíček možnost nevkládání zbytečných PDF objektů nepodporuje tak, jak byl původní záměr.

6.2.2 Správa barev

V kontextu interaktivních prvků a multimédií je obvykle vyžadováno nastavení barvy pomocí trojice čísel v rozsahu 0 až 1 (barevný model RGB). Oproti tomu běžné nastavení barvy grafickými operátory umožňuje tři různá zadání barvy – RGB, CMYK a odstín šedé (opět s čísly v rozsahu 0 až 1). OpTEX obsahuje sofistikovaná makra pro správu barev (definice, míchání, použití, ...). V uživatelském rozhraní barvy reprezentuje kontrolními sekvencemi. Na místě je snaha o podporu zadání barvy jak přímo

(RGB barva, trojice čísel), tak i pomocí OpTEXovských kontrolních sekvencí. Balíček definuje makro `\colortorgbdef`, které přesně tento problém řeší.

6.2.3 PDF akce a anotace

Další oblast, kde s výhodou šla využít všeobecnost, jsou PDF akce. Celkem n druhů akcí jde využít v m kontextech. Navíc lze vytvořit „zřetězené akce“, což je akce, která spustí za sebou několik akcí. Všechny tyto aspekty pokrývá jeden navržený příkaz `\pdfaction`. Ten načte čárkami oddělený seznam „specifikací akcí“ a vytvoří z nich zřetězenou akci. Specifikace akcí sestává ze jména akce a argumentů. Logika pro zpracování jednotlivých akcí ale není v příkazu `\pdfaction`, ale v příkazech odvozených pouze ze jména akce. Obecně je tedy možné libovolně další akce přidávat nebo měnit chování stávajících akcí. Snadno lze také zavést aliasy TEXovým primitivem `\let`. Výsledkem `\pdfaction` je slovník akce, který tak lze použít ve všech kontextech, kde lze akci zadat (`/Link` anotace, záložky, atd.).

Nejčastější využití PDF akcí je ale v rámci `/Link` anotací. Obvykle ve formě `/GoTo` a `/URI` akcí tvoří klikatelné odkazy na jiná umístění. Z TEXového konce se tak typicky očekává akce a materiál pro sazbu (viz sekci 3.4). Pro tento účel byl navržen příkaz `\hlink`, jehož jeden argument je specifikace akce (interně zpracovaná pomocí `\pdfaction`) a materiál pro sazbu. Kromě „zaktivnění“ textu (například URL nebo čísla sekce), lze samozřejmě takto vytvořit i libovolná „tlačítka“. Jen zde lze narazit na limitaci původních pdfTEXových primitivů. Ty totiž rozměry aktivní oblasti dokáží odvodit od rozměrů materiálu k sazbě. V případě běžného odstavce a rozlomení materiálu do dvou (nebo více) řádků je ale mnohem příjemnější, pokud se pro velikost aktivní oblasti řádku nepoužijí rozměry textu v řádku (který může mít proměnlivou a hlavně relativně malou výšku/hloubku), ale velikost podpěry. Podpěra má totiž výšku a hloubku shodnou s maximálními rozměry textu v řádku. To zajistí rovnoměrnou velikost aktivních oblastí v odstavci. Pokud, jako při běžných nastaveních, je velikost podpěry navíc shodná se vzdáleností účaří, je v případě, že je aktivních několik celých řádků za sebou, dosaženo i zaktivnění prokladu. Oba důsledky jsou důležité pro uživatele, který se myší musí strefit do aktivní oblasti. Také to má estetické důsledky v případě, že je aktivní oblast barevně obtažena.

Příkaz `\hlink` lze také vnímat jako zevšeobecnění existujících OpTEXových příkazů `\ilink` a `\ulink`, které byly omezeny na `/GoTo`, respektive `/URL` akce. Příhodně oba příkazy používaly syntaxi, která šla snadno rozšířit i o podporu dalších akcí pomocí `\hlink`. `\ulink` totiž očekává argument uvozený „`url:`“, kdežto `\ilink` pomocí „⟨*typ*⟩`:`“. Přitom „⟨*typ*⟩“ je konkrétní typ interních odkazů (například „`ref`“ nebo „`pg`“). Toto použití zvládá i příkaz `\hlink`, který pokud nerozpozná „⟨*typ*⟩“ jako konkrétní akci, tak ji považuje za interní odkaz pomocí `/GoTo`.

Podobný všeobecný přístup pro akce a „akční tlačítka“ / aktivní oblasti na stránce používá ConTEXt (viz sekci 4.2). Tento přístup je mnohem lepší než roztříštěnost LATEXových balíčků, kdy každý balíček umožňuje vytváření různých „tlačítek“ různými způsoby.

6.2.4 Vložené soubory, inicializační JavaScript

Vložené soubory i tzv. „document level“ (inicializační) JavaScript fungují interně podobně. Jsou určeny zobrazením jméno-objekt vloženým pod katalogem dokumentu. Toto zobrazení interně obsahuje páry objektů jméno-objekt a tyto páry jsou obsaženy obecně ve stromové struktuře. Pro jednoduchost ale implementace tvoří obyčejný seznam. Protože vložených souborů i inicializačních JavaScript skriptů může být neome-

zeně a zároveň se jejich definice mohou objevit kdekoliv v dokumentu, tak to přináší problém volby času, kdy je seznam zapsán do PDF souboru. Story [15] pro LaTeX omezuje možnost definování inicializačních JavaScript skriptů na preambuli a zápis jejich seznamu provádí na konci preambule. V OpTeXu podobný koncept neexistuje. Existuje ale makro `\_byehook`, do kterého lze přidat kód určený pro provedení před koncem zpracování dokumentu. To pro potřeby vložených souborů a inicializačního JavaScriptu stačí a navíc uživateli dovolí oboje definovat kdekoliv.

6.2.5 Multimédia

Makra pro vkládání multimédií jsou poměrně specifická. Mírně totiž jdou proti obvyklým zvyklostem OpTeXu a využívají key-value syntaxi pro změnu parametrů na jiné než výchozí hodnoty. Jednou z filozofií OpTeXu je neuniverzálnost. Místo široké možnosti konfigurace má u OpTeXu uživatel možnost jednoduchý kód adaptovat podle svého. I přesto existuje celkem velké množství parametrů, které umožňují některé věci uzpůsobit (například některé rozměry a barvy). Většina těchto parametrů má charakter nastavení, která jsou pravděpodobně globální – ovlivní třeba některý aspekt vzhledu dokumentu, který je obvyklé dodržet konzistetně. Naproti tomu v oblasti multimédií je obvyklé uzpůsobit každé vložené multimédium jinak – automatické spuštění, úroveň hlasitosti, barva pozadí atd. Důkazem výhodnosti použití key-value syntaxe jsou i ostatní balíčky, které ji také využívají (viz například sekci 4.1.2).

Oba implementované mechanismy multimédií – Renditions (audio, video) a Rich Media (3D, ale i audio a video) jsou řešeny velmi podobně. Jde přeci pouze o vytvoření struktury PDF slovníků tak, jak ji pro daný mechanismus požaduje norma. Cílem je přitom zabalení této struktury slovníků do anotace, která bude mít uživatelem nastavený vzhled – libovolný TeXový materiál, obvykle například obrázek nebo text. Z tohoto materiálu je vytvořena forma, která je odkázána ve slovníkové struktuře. O takovém využití PDF objektu vůbec TeX neví. Je potřeba ho přesvědčit o tom, že objekt je alespoň jednou využit, aby ho zapsal do PDF. Bohužel zvýšit počet referencí objektu formy lze pouze jejím vložením jako vektorové grafiky na stránku. Alternativou je určení objektu pro okamžité zapsání do PDF (pomocí `\immediate`). Tím vzniká problém popsaný v sekci 6.2.1.

Oba mechanismy zároveň umožňují ovládání pomocí akcí. Obecně se může akce ovládající určité multimédium objevit kdykoliv a jakkoliv promíchaná s ostatními vloženými akcemi/multimédii. Proto je potřeba u akce určit, které multimédium ovládá. Pro tento účel je použito stejné jméno, které reprezentuje specifikaci souboru (viz sekci 6.2.1). Jako příjemnou alternativu lze použít prázdné jméno, které je alias pro poslední anotaci daného typu (Rendition / Rich Media). To sebou přináší ale další problém. Jména jsou pro každý vložený soubor unikátní. Tím pádem není možné vložit multimédium například na více stran s různými ovládacími prvky, protože obě instance mají stejné jméno a nelze je rozlišit (akce budou provázány s prvním výskytem). I na toto (i když ne úplně obvyklé) použití balíček myslí. Je totiž možné mezi key-value parametry uvést alternativní jméno unikátní pouze pro tuto instanci.

Rich Media jsou v kontextu balíčku určena především pro 3D díla, díky zavedené všeobecnosti ale není problém místo 3D díla vložit audio či video (typ souboru je automaticky rozeznán).

6.2.6 3D pohledy

Nejzajímavějším aspektem nastavení 3D Rich Media anotace je nastavení 3D pohledu, konkrétně pozice a orientace kamery ve 3D scéně. Ta je určena pomocí matice „C2W“

(viz sekci 2.2.6). Běžný uživatel tuto matici ovšem jednoduše nesestaví (i když balíček podporuje i její přímé zadání). Je proto potřeba ji sestavit, nejlépe za pomoci lépe představitelných a intuitivnějších parametrů.

Jeden z celkem univerzálně známých způsobů sestavení této 3D matice bývá někdy nazýván „look at“. Různé varianty se mírně liší vstupními parametry (avšak vzájemně převoditelnými). Například varianta, kterou popisuje [13, s. 144] přijímá jako parametry pozici kamery, směr jejího pohledu a tzv. „up“ vektor, který reprezentuje globální směr nahoru a slouží k zafixování zbývajícího stupně volnosti „pootočení kamery“.

Matice C2W představuje (afinní) transformaci mezi souřadnicovým systémem kamery a světovým souřadnicovým systémem. Poslední její 3 prvky tak vlastně udávají pozici kamery ve světovém systému souřadnic a jsou zadány. Zbývajících 9 parametrů po trojicích tvoří vektory, které udávají jak se transformuje báze kamerového prostoru. V kontextu PDF má smysl, aby výsledné vektory byly vzájemně ortonormální (stejně jako je původní báze v kamerovém prostoru). V kamerovém prostoru kladný směr osy x směřuje doprava, y nahoru a z dopředu. Kamera samotná je v počátku a hledí kladným směrem osy z. Vstupní parametr směru pohledu kamery tak vlastně (po případné normalizaci) představuje zobrazení kladné osy z a tvoří tak předposlední tři prvky matice zobrazení. Zbývající dva vektory lze dopočítat pomocí vektorových součinů (prvotní hrubý směr „nahoru“, zobrazení kladné osy y, udává libovolně zvolený globální směr nahoru – „up“ vektor).

Při celém výpočtu je potřeba brát v potaz pravo/levotočivost souřadnicového systému a také bázi souřadnicového systému kamer. Právě v tom se PDF liší například od OpenGL a tedy i od [13]. Detaily výpočtu včetně vysvětlení správného pořadí vektorových součinů a vztahů vektorů jsou součástí technické dokumentace balíčku (součást Přílohy B). Výpočet je tam alternativně odvozen ze zadání dvou bodů (polohy kamery a cíle pohledu) a globálního směru nahoru.

Samotný kód výpočet provádí ještě trochu jinak. Přijímá stejné parametry jako balíček movie15 (a další TEXové balíčky, které přejaly stejný systém – media9, rmannot, ConTEXt), viz Obrázek 4.1. Navíc je v tomto systému přidána i automatická detekce případu, kdy výchozí směr nahoru je shodný směru pohledu (nebo opačný). Tento případ totiž produkuje nulový výsledek jejich vektorového součinu a znamená potřebu zvolit jinou hodnotu pro globální směr nahoru.

Přidanou hodnotou balíčku je v tomto ohledu dokumentace tvorby transformační matice a také čitelnost implementace – tato část byla totiž naimplementována v Lue. Podobný výpočet lze sice provést i v TEXu, rozhodně to ale není přímočaré ani čitelné.

6.3 Obsah balíčku, publikace

Výsledný balíček byl pojmenován „pdfextra“ a volně zveřejněn TEXové komunitě na portálu CTAN[3]. Jeho součástí je především soubor `pdfextra.opm`, který obsahuje makra (implementaci samotnou) i technickou dokumentaci. Pro použití balíčku lze tento soubor načíst OpTEXem. Zároveň lze vysázet technickou dokumentaci v něm obsaženou. Přesně to dělá dokumentační soubor `pdfextra-doc.tex`, který navíc obsahuje uživatelskou dokumentaci. Výsledkem je tak jedno PDF obsahující kompletní dokumentaci a zdrojový kód.

[3] V čase odevzdání byl balíček zaslán ke schválení. Dostupný je ale i na `https://github.com/vlasakm/pdfextra`, kde bude probíhat také budoucí vývoj.

Kapitola 7
Závěr

Pro práci byly původně stanoveny následující cíle zaměřené na multimédia a interaktivní prvky v oblasti PDF a TEXu:

- Analýza možností formátu PDF.
- Srovnání existujících TEXových balíčků.
- Otestování PDF prohlížečů a zjištění, co prakticky funguje.
- Vytvoření balíčku využitelného plainTEXovou komunitou.

Všechny stanovené cíle byly *splněny*.

I když se ukázalo, že formát PDF nabízí celou řadu možností v oblasti interaktivních prvků a multimédií, ne všechny dávají smysl v kontextu TEXu a hlavně jsou mnohdy v praxi obtížně využitelné. Zaostává totiž podpora v prohlížečích. Výjimkou je (v podstatě referenční) prohlížeč Acrobat, který zvládá téměř všechny sledované aspekty. Těsně zaostává prohlížeč Foxit, který se ukázal jako nadějná alternativa. Z testovaných open-source PDF prohlížečů je většina v této oblasti nepoužitelná. Výjimku tvoří prohlížeče Evince a Okular, které jsou pro využití v některých oblastech (například prezentace) přijatelně použitelné.

Konec technologie Flash, který nedávno zasáhl obor multimédií ve formátu PDF, je řešitelný, protože lze využít jiný způsob vkládání multimédií, který má dokonce širší podporu mezi PDF prohlížeči.

Funkcionality, které se ukázaly jako v praxi použitelné, byly předmětem implementace balíčku. Tento balíček je určen pro nový formát OpTEX a je veřejně dostupný všem TEXistům na portálu CTAN[1]. Původní myšlenka vytvoření čistě plainTEXového balíčku se ukázala jako neefektivní. Makra pro vkládání interaktivních prvků a multimédií vyžadují přítomnost některých mechanismů, jako je třeba zápis pomocných souborů nebo správa barev. Ty přitom v plainu chybí. Navíc si je každý formát (nebo uživatel stavějící na plainu) implementuje po svém, proto by nebylo efektivní ani přidat do balíčku makra obstrávající tyto mechanismy. Výsledkem jsou tedy makra částečně závislá na mechanismech OpTEXu. Přesto může jednoduchost a obsáhlá dokumentace tohoto balíčku být referencí i pro uživatele plainTEXu či jiných formátů, kteří makra budou chtít převzít. Ani netypické prefixy „_“ či „.“ u kontrolních sekvencí nejsou velkou překážkou – je možné je odstranit jednoduchou textovou náhradou. Z maker si navíc může každý vybrat vhodnou část (například jen vkládání multimédií), což odpovídá plainTEXovskému přístupu.

Zároveň bylo při vypracování této práce objeveno několik chyb nebo nedostatků v PDF prohlížečích a TEXových balíčcích. K některým byl zaslán návrh na opravu[2].

[1] V čase odevzdání je očekáváno schválení umístění balíčku na CTANu. Pravděpodobně bude dostupný na adrese `https://www.ctan.org/pkg/pdfextra`. Jeho budoucí vývoj bude probíhat otevřeně na `https://github.com/vlasakm/pdfextra`.

[2] `https://gitlab.freedesktop.org/poppler/poppler/-/merge_requests/855`, `https://bugs.kde.org/show_bug.cgi?id=436709`, `https://bugs.kde.org/show_bug.cgi?id=436710`.

Také jsem objevil několik více i méně souvisejících oblastí, převážně v OpTEXu a LuaTEXu[3], které lze vylepšit, což bude předmětem mé následující snahy.

K práci je ve formě Přílohy B přiložena také podstatná část implementovaného balíčku ve formě vysázené uživatelské dokumentace a technické dokumentace protkané se zdrojovým kódem (ve stylu „Literate programming“). K tomu jsem se rozhodl z důvodu větší integrity odevzdaného celku práce.

Elektronickou část přílohy tvoří zveřejněná podoba balíčku – zdrojové i ukázkové soubory, seznam obsahu přiloženého média shrnuje příloha A.

Podstatná část dokumentace balíčku opakuje některé koncepty podrobněji rozepsané v této práci. Nepředpokládám totiž, že potenciální uživatel balíčku bude studovat tuto práci. Proto jsou zde některé koncepty vysvětleny ještě jednou, ze spíše praktického pohledu a navíc v anglickém jazyce.

[3] Jedná se například o různé možnosti implementace generování PDF souborů, správy barev a do jisté míry i TEXu samotného.

Literatura

[1] Adobe Inc. *JavaScript for Acrobat API Reference* [online]. 2007 [vid. 2021-03-05]. Dostupné na `https://www.adobe.com/content/dam/acom/en/devnet/acrobat/pdfs/js_api_reference.pdf`.

[2] Adobe Inc. *Adobe Supplement to the ISO 32000*. BaseVersion: 1.7, ExtensionLevel: 3 [online]. 2008 [vid. 2021-03-05]. Dostupné na `https://www.adobe.com/content/dam/acom/en/devnet/pdf/adobe_supplement_iso32000.pdf`.

[3] Adobe Inc. *JavaScript for Acrobat 3D Annotations API Reference* [online]. 2015 [vid. 2021-03-05]. Dostupné na `https://wwwimages.adobe.com/content/dam/acom/en/devnet/acrobat/pdfs/AcrobatDC_js_3d_api_reference.pdf`.

[4] Garcia, Federico. Hypertext capabilities with pdfLaTeX. *TUGboat*. 2007, Vol. 28, No. 1, pp. 129–132 [vid. 2021-03-05]. ISSN 0896-3207. Dostupné na `https://tug.org/TUGboat/tb28-1/tb88garcia.pdf`.

[5] Grahn, Alexander. *The movie15 Package* [online]. 2012 [vid. 2021-03-05]. Dostupné na `https://mirrors.ctan.org/macros/latex/contrib/movie15/doc/movie15.pdf`.

[6] Hagen, Hans. ConTeXt LMTX. *TUGboat*. 2019, Vol. 40, No. 1, pp. 34–37 [vid. 2021-03-05]. ISSN 0896-3207. Dostupné na `https://tug.org/TUGboat/tb40-1/tb124hagen-lmtx.pdf`.

[7] Hagen, Hans. *Interaction* [online]. 2021 [vid. 2021-03-05]. Dostupné na `https://www.pragma-ade.com/general/manuals/interaction.pdf`.

[8] Haralambous, Yannis a Sebastian Rahtz. LaTeX, hypertext and PDF, or the entry of TeX into the world of hypertext. *TUGboat*. 1995, Vol. 16, No. 2, pp. 162–173 [vid. 2021-03-05]. ISSN 0896-3207. Dostupné na `https://tug.org/TUGboat/tb16-2/tb47hara.pdf`.

[9] ISO 32000-1:2008. *Document management — Portable document format — Part 1: PDF 1.7*. ICS 35.240.30. 2008, International Organization for Standardization 2008 [vid. 2021-03-05]. Dostupné na `https://www.adobe.com/content/dam/acom/en/devnet/acrobat/pdfs/PDF32000_2008.pdf`.

[10] ISO 32000-2:2020. *Document management — Portable document format — Part 2: PDF 2.0*. ICS 35.240.30. 2nd ed. 2020, International Organization for Standardization. 2020.

[11] Knuth, Donald E. *Computers & Typesetting Volume B: TeX: The Program*. Reading: Addison-Wesley, 1986. ISBN 0-201-13437-3.

[12] LuaTeX team. *LuaTeX Reference Manual* [online]. 2021 [vid. 2021-03-05]. Dostupné na `https://mirrors.ctan.org/systems/doc/luatex/luatex.pdf`.

[13] Marschner, Steve a Peter Shirley. *Fundamentals of Computer Graphics*. 4 ed. Oakville: CRC Press LLC, 2015. ISBN 9781482229394.

[14] Olšák, Petr. *TeXbook naruby*. Brno: Konvoj, 2001. ISBN 80-7302-007-6. Dostupné na `http://petr.olsak.net/tbn.html`.

[15] Story, D. P. Techniques of Introducing Document-level JavaScript into a PDF file from a LaTeX Source. *TUGboat*. 2001, Vol. 22, No. 3, pp. 161–167 [vid. 2021-03-05]. ISSN 0896-3207. Dostupné na `https://tug.org/TUGboat/tb22-3/tb71story.pdf`.

[16] Story, D. P. Rich media annotations and AcroFleX. *TUGboat*. 2009, Vol. 30, No. 2, pp. 281–284 [vid. 2021-03-05]. ISSN 0896-3207. Dostupné na `https://tug.org/TUGboat/tb30-2/tb95story.pdf`.

[17] Thành, Hàn Thế a kol. *The pdfTeX user manual* [online]. 2021 [vid. 2021-03-05]. Dostupné na `https://tug.org/texlive/Contents/live/texmf-dist/doc/pdftex/manual/pdftex-a.pdf`.

[18] Walden, David. *TUG Interview Corner: David Fuchs* [online]. 2007 [vid. 2021-03-05]. Dostupné na `https://tug.org/interviews/fuchs.html`.

Příloha A
Obsah elektronické přílohy

Následující výčet obsahuje krátké představení obsahu elektronické přílohy:

```
/
├── src ..... adresář s balíčkem
│   ├── pdfextra.opm ..... zdrojový kód balíčku, technická dokumentace
│   ├── pdfextra-doc.tex ..... zdrojový kód uživatelské dokumentace
│   ├── pdfextra-doc.pdf ..... vysázené dokumentace a zdrojový kód
│   └── examples ..... složka s dalšími ukázkami použití balíčku
├── text ..... text práce
│   ├── vlasami6-bp.pdf ..... text práce ve formátu PDF
│   └── vlasami6-bp.tex ..... TEXový zdroj textu práce
├── testing-files ..... adresář s PDF soubory použitými pro testování prohlížečů
└── README.txt ..... popis elektronické přílohy
```

Detailní popis všech souborů a jejich úplný seznam je součástí souboru `README.txt`.

Příloha B
Dokumentace a zdrojový kód balíčku

PDF extra – extra PDF features for OpTEX

Version 0.1

Michal Vlasák, 2021

`PDFextra` is a third party package for OpTEX. It aims to provide access to more advanced PDF features, which are currently not supported in OpTEX – especially interactive and multimedia features. The development is hosted at https://github.com/vlasakm/pdfextra.

In the spirit of OpTEX you may use these macros in any form you want. Either by installing this package and doing `\load[pdfextra]` in OpTEX, or just by copying some useful parts of this package into your documents / packages. OpTEX namespacing is used, but it can be easily stripped, if you wish to incorporate these macros into other macro packages. The code currently depends on LuaTEX, but mostly uses only pdfTEX primitives and a few simple macros from OpTEX.

User documentation (`pdfextra-doc.tex`) and technical documentation interleaved with source code (`pdfextra.opm`) are all typeset in this PDF file. Some examples of usage are in the user documentation, but file `pdfextra-example.tex` contains more examples.

Contents

Chapter 1
User documentation

1.1 Defining files

Many commands provided by this package require you to supply a file ⟨*name*⟩. This is because many commands either work directly (like inserting attachments or multimedia) or can optionally use files (like inserting JavaScript). The "right" way to use ⟨*name*⟩ is to first define the ⟨*name*⟩ with:

`\filedef/`⟨*type*⟩`[`⟨*name*⟩`]{`⟨*path or URL*⟩`}`.

Where ⟨*name*⟩ is the name you will use to refer to this file. It is currently limited to ASCII only (as all "⟨*name*⟩s" required by this package). Interpretation of ⟨*path or URL*⟩ depends on the type, which may be:

- `e`, "embedded file". The file with path ⟨*path*⟩ will be embedded to the PDF file. A file that is embedded this once, can later be used many times in different contexts, e.g. you may use it to attach a video as an attachment but also have it play on page 1 and even other pages. This is the best way, because the resulting PDF file is self contained.
- `x`, "external file". ⟨*path*⟩ can only be a path to the current directory. To refer to the file only ⟨*path*⟩ is used, sort of like a reference. This way the file you want to refer to *has* to be present in the same directory as the PDF file when it is *viewed*!
- `u`, "URL file". ⟨*URL*⟩ is the URL of the file you want to refer to.

All these create the same type of object, which ideally could be used interchangibly everywhere a *file specification* is required in PDF. This is sadly not always true. The limitations to only certain types of `\filedef`'s will be mentioned in due sections. But as a rule of thumb, most of the time you want to embed the files into PDF. The external/URL references are good for refering to external files, although other methods are also possible there.

Because most of the times you want ⟨*name*⟩ to be embedded file, you may omit the prior definition and instead use the ⟨*path*⟩ itself. The file will be autoembedded.

Examples:

```
\filedef/u[doc-internet]{http://petr.olsak.net/ftp/olsak/optex/optex-doc.pdf}
\filedef/x[doc-local]{optex-doc.pdf}
\filedef/e[doc-embedded]{optex-doc.pdf}
```

1.2 Multimedia

It is possible to insert video, audio and 3D files for playback/display inside a PDF file (on a page). There are several different PDF mechanisms for inserting multimedia. For audio/video this package uses the so called "Renditions", which currently have the best support in PDF viewers (fully works in Acrobat and Foxit, partly in Evince and Okular). Rich Media annotations are used for 3D art (works only in Acrobat Reader), although it is possible to also insert audio/video using this mechanism it is very restricting and Renditions are recommended.

Use command `\render[`⟨*name*⟩`][`⟨*optional key-value parameters*⟩`]{`⟨*appearance*⟩`}` for inserting audio/video with Renditions or `\RM[`⟨*name*⟩`][`⟨*optional key-value parameters*⟩`]{`⟨*appearance*⟩`}` for inserting audio/video/3D as Rich Media annnotation. The result of this command is similiar to what `\inspic` produces[1]. Both commands expect ⟨*name*⟩ to be `\filedef`'d name of the file to play/display. As usual, fallback for interpreting ⟨*name*⟩ as path (and embedding it) is in place. It is recommended to only use embedded files with both mechanisms (Rich Media requires it, Renditions with other than embedded files do not work in Acrobat). Optional key-value parameters may be used to customize default values. They may be left out enitrely (including brackets). Last parameter, ⟨*appearance*⟩, defines so called "normal apperance", which is shown before the annotation is activated (audio/video starts playing or 3D scene is displayed). The dimensions of resulting multimedia annotation will be taken from ⟨*appearance*⟩. Most likely you want to use a "poster" picture (inserted with `\inspic`) as appearance – the dimensions will be taken from it and e.g. aspect ratio will be nicely preserved.

[1] But because annotations are involved, transformations using PDF literals will not work as expected.

Customization of Renditions is possible using key-value parameters, but beware that it mostly doesn't work in Evince and Okular (Acrobat and Foxit are fine in this regard). The available parameters are in Table 1.2.1. Most customizations of Rich Media concern 3D art. Available parameters are listed in Table 1.2.2.

Table 1.2.1 Key value parameters available for Renditions (`\render`)

Key	Possible values	Default	Description
`controls`	`true` or `false`	`false`	Whether to display audio/video player controls.
`volume`	decimal betwen 0 and 100	100	Audio volume.
`repeat`	integer ≥ 0	1	Number of repetitions (0 means loop forever).
`background`	OpTEX color	`\White`	Color used for part of the annotation not covered by video player (for wrong aspect ratios).
`opacity`	decimal between 0 and 1	1	Opacity of `background`.
`aactions`	`\renditionautoplay`	(none)	Can be used for autoplay on page open.
`name`	ascii string	⟨*name*⟩	Name for use with actions and scripts.

Table 1.2.2 Key value parameters available for Rich Media (`\RM`)

Key	Possible values	Default	Description
`activation`	`explicit` or `auto`	`explicit`	Whether to automatically activate the annotation on page open or display normal apperance until user clicks.
`deactivation`	`explicit` or `auto`	`explicit`	Whether to automatically deactivate the annotation on page close or require explicit deactivation by user (from right click menu).
`toolbar`	`true` or `false`	`true`	Whether to show 3D toolbar (with view and other options).
`views`	comma separated list of view ⟨*name*⟩s	⟨*name*⟩	List of names of 3D views to be used. The default is to try a view of same ⟨*name*⟩. Beware that unknown views are silently ignored.
`scripts`	comma separated list of script ⟨*name*⟩s	(none)	List of names of JavaScript script file ⟨*name*⟩s to be used.
`name`	ascii string	⟨*name*⟩	Name for use with actions and scripts.

The weird `name` key is only required if one media file is used more than once and control using actions or JavaScript scripts is needed.

Examples of video insertion:

```
% embed file under name "video"
\filedef/e[video]{example-movie.mp4}
% insert video into page using Renditions mechanism with controls and autoplay
\render[video][
  name=bigvideo,
  controls=true,
  aactions=\renditionautoplay,
]{\picwidth=\hsize \inspic{example-image.pdf}}

% render the same file again, but with different dimensions, no controls
% and explicit activation
\render[video]{\inspic{example-image.pdf}}
```

When displaying 3D there are more things involved. First, only U3D and PRC can be included in PDF files. The simplest way to show the 3D scene on page is without any optional parameters:

```
\RM[part.prc]{\picwidth=\hsize \inspic{part.png}}
```

The resulting view will be what is defined in the 3D file. But it is possible to customize it, by creating custom view. Or even more of them – they will be available in the user interface for easy switching, first one is considered default. Parameters not defined in a custom view often take what is in the 3D file as default value. `\DDDview[`⟨*view name*⟩`][`⟨*key-value parameters*⟩`]` is the command for defining 3D views. The brackets surrounding key-value parameters have to be included even if no key-value parameters are used. The available parameters are explained in Table 1.2.3.

Table 1.2.3 Key value parameters available for 3D views (`\DDDview`)

Key	Possible values	Default	Description
`projection`	`perspective` or `ortho`	`perspective`	Projection type to use (perspective distorts the view, e.g. parallel lines are not shown as such, but is more natural to human eye, orthogonal projection is what is generally used with technical parts).
`scale`	decimal number	1	Scaling to use when orthogonal projection is used.
`FOV`	number between 0 and 180	30	Field of view for use with perspective projection.
`background`	OpTEX color	`\White`	Color used as background in the 3D scene.
`rendermode`	`Solid`, `SolidWireframe`, `Transparent`, `TransparentWireframe`, `BoundingBox`, `TransparentBoundingBox`, `Wireframe`, `ShadedWireframe`, `Vertices`, `ShadedVertices`, `Illustration`, `SolidOutline`, `ShadedIllustration`	(taken from 3D file)	Used rendering mode. See PDF standard[i] for more details.
`lighting`	`White`, `Day`, `Night`, `Hard`, `Primary`, `Blue`, `Red`, `Cube`, `CAD`, `Headlamp`	(taken from 3D file)	Used lighting scheme. See PDF standard[ii] for more details.
`method`	`media9`, `manual`, `u3d`	`media9`	Method used for defining 3D camera position and orientation.

[i] https://www.adobe.com/content/dam/acom/en/devnet/pdf/pdfs/PDF32000_2008.pdf#G12.2358303
[ii] https://www.adobe.com/content/dam/acom/en/devnet/pdf/pdfs/PDF32000_2008.pdf#G12.2358356

The value of `method` key influences what other key-value parameters are available. For details about the manual method see the technical documentation (2.8.3). The `u3d` method works only with U3D files (not with PRC files) and requires you to know the internal "path" of the view contained in the file and is hence not always useful (especially when the paths are weirdly constructed by exporting applications). But if you know the path you can use "`method=u3d, u3dpath=`⟨*path*⟩". All Unicode characters are allowed in ⟨*path*⟩.

Table 1.2.4 Key value parameters available for media9 method of 3D views (`\DDDview`)

Key	Possible values	Default	Description
`coo`	three space separated (decimal) numbers	`0 0 0`	"Center of orbit". Coordinates of the point camera is supposed to look at.
`roo`	(decimal) number	`0`	Distance of camera from `coo`.
`c2c`	three space separated (decimal) numbers	`0 -1 0`	"Center of orbit to camera" vector. A directional vector (i.e. length doesn't matter). The direction the camera will be looking at is opposite of this vector. Default is view towards positive y.

The most useful method of specifying the camera position/orientation is `method=media9`. The same method is also used in a number of other packages (movie15, rmannot, ConTEXt). Its input are key-value parameters, listed in Table 1.2.4. For illustration of the parameters check the documentation[2] of media9.

You can construct simple views only by using these few parameters. When talking about meanings/names of different views it is needed to know the real placement of the 3D object in the 3D world. But for a reasonably orientated object in the center (`coo=0 0 0`) constructing simple views is very easy (keep in mind `method=media9` is default):

```
\DDDview[front][
  projection=ortho,
  roo=400,
]
\DDDview[left][
  projection=ortho,
  roo=400,
  c2c=-1 0 0,
]
\DDDview[top][
  projection=ortho,
  roo=400,
  c2c=0 0 1,
]
\DDDview[isometric][
  projection=ortho,
  roo=500,
  c2c=-1 -1 1,
]
```

We can then perhaps use these views in `\RM`:

```
% Auto activated 3D Rich Media annotation. White background just ensures
% the right dimensions. Comma in `views` is escaped using "{}".
\RM[part.prc][
  activation=auto,
  views={front, left},
]{{\White\vrule width\hsize height\vsize}}

\RM[part.prc][ % the same 3D file, now with different views
  name=part2,
  activation=auto,
  views={top, isometric},
]{{\White\vrule width\hsize height\vsize}}
```

[2] https://mirrors.ctan.org/macros/latex/contrib/media9/doc/media9.pdf#figure.8

If you want to deduce the view parameters automatically it is possible. You can just not specify any view, but include the `3Dmenu.js` script from media9 package. It enables you to right click the annotation and select "Get Current View" (or even "Generate Default View" which finds the view all by itself). A window with generated parameters will show up. You can then copy the ones this package understands (`c2c`, `coo`, `roo`) and use them. You don't have to be excessive with precision, because after calculation everything gets rounded to 6 decimal places anyways.

```
% using 3Dmenu.js to generate 3D view parameters automatically
\RM[part.prc][
  name=part3,
  activation=auto,
  scripts=3Dmenu.js,
]{{\White\vrule width\hsize height\vsize}}
```

The use of `scripts` isn't limited to this though, there are many other possibilities. First, you can use as many scripts as you want (`scripts={script1.js, script2.js, ...}`), but be careful that 3D JavaScript is kind of special, and has to come from embedded files (here we were using the auto-embedding, but we could have used e.g. `\filedef/e[3dmenu]{3Dmenu.js}` and then "`3dmenu`" instead). For creating your own scripts check out the Acrobat 3D JavaScript API[3]. It is possible to do different transformations and achieve animations using "time events". The implicit "context" of 3D scripts can be accessed from normal JavaScript actions (see 1.3.5). Vice versa 3D scripts may access the global JavaScript environment using `host` object.

More examples of 3D Rich Media, including usage of 3D JavaScript, are available in the example file `pdfextra-example.tex`. They show how it is possible to port the examples used by media9.

1.3 Actions

Actions are very important aspect of interactivity in the context of PDF. There are a few very useful types of actions, like "goto" actions which jump to other part of document. There are also a few ways how to *execute* actions. The most usual is a clickable area on page, but clickable bookmarks ("document outline") also execute actions behind the scenes. OpTEX supports only basic "goto" and "URI" actions using `\ilink` (used for `\ref`, `\cite`, etc.) and `\ulink` (used for `\url`).

This package offers generalization of this mechanism. The core of it is a way of specifying an action. This syntax is called ⟨*action spec*⟩ and is used for example by `\hlink` command, which can replace both `\ilink` and `\ulink`. ⟨*action spec*⟩ is a comma separated list of ⟨*type*⟩`:`⟨*arguments*⟩. where ⟨*type*⟩ refers to the action type and the syntax of arguments is dependant on ⟨*type*⟩. Leading spaces are ignored, trailing aren't. You probably won't often use the possiblity of specifying multiple actions, but it is for chaining execution of several actions.

`\pdfaction[`⟨*action spec*⟩`]` is available for lower level creation of different actions, but for clickable areas on page you will use `\hlink[`⟨*action spec*⟩`]{`⟨*text*⟩`}` with a very similiar syntax. Although note, that `\hlink`'s interface is also not really high level and wrapping it inside macros like `\ref` or `\url` might be beneficial. ⟨*text*⟩ will be typeset directly and the area it occupies will be clickable. Clicking it executes action defined by ⟨*action spec*⟩. Line breaks inside ⟨*text*⟩ will be possible, in that case several clickable rectangles will be created, one for each line. Normally in text you want the ares to be of the same height and depth (calculated from `\baselineskip`), to achieve sort of a lining, uniform effect. If you want to define big clickable buttons, you may need to turn off the lining effect using `\nolininglinks`. It respects groups, but a counterpart (`\lininglinks`) is also available.

There are a few predefined action types (⟨*type*⟩): `url`, `extref`, `extpgref`, `named`, `transition`, `js`, `goto3dview` and `rendition`. They will be explained in a moment. Any unrecognized ⟨*type*⟩ is understood as an "internal link", where ⟨*type*⟩`:`⟨*link*⟩ is the destination of the link. Hence it is possible to use `\hlink` as `\ilink` for example with OpTEX's normal types of internal links. For example:

```
See section~\hlink[ref:section]{2.2.13} or page~\hlink[pg:5]{5}.
```

`\ulink` may be replaced like this:

```
Visit CTAN's \hlink[url:https://www.ctan.org/]{website}.
```

Before we really get into different types of actions, there is a nicer command for setting the initial ("open") action of PDF document, which is executed when the document is opened. It defaults to

[3] https://wwwimages.adobe.com/content/dam/acom/en/devnet/acrobat/pdfs/AcrobatDC_js_3d_api_reference.pdf

opening the page with the zoom level according to viewing user's preferences. But you can change this using `\openaction[`⟨*action spec*⟩`]`:

```
\openaction[pg:2]
```

1.3.1 External references

There are two actions analogous to internal links that can be used to link to external documents. `[extref:`⟨*name*⟩`:`⟨*named destination*⟩`]` can be used to refer to named locations in a PDF document prepared by `\filedef` with ⟨*name*⟩. `[extpgref:`⟨*name*⟩`:`⟨*page number*⟩`]` is similiar but refers to a page number. Although it would be nice, these actions aren't well supported by all viewers.

Example:

```
\hlink[extref:doc-internet:ref:langphrases]{OpTeX documentation,
        section \"Multilingual phrases and quotation marks".}
\hlink[extpgref:doc-internet:12]{OpTeX documentation, page 12.}
```

(Note that "`:`" in "`ref:langphrase`" is not part of the syntactic rule, it is just the value of ⟨*named destination*⟩ in this case.)

A little bit of customization is possible, see 2.4.4.

Because of the poor support you may find luck with the less universal url action with `#fragment`:

```
\hlink[url:http://petr.olsak.net/ftp/olsak/optex/optex-doc.pdf#ref:langphrases]
      {OpTeX documentation, section \"Multilingual phrases and quotation marks".}
```

1.3.2 Named actions

There are four defined in PDF standard, but viewers may support more. All in examples:

```
\hlink[named:NextPage]{Go to next page,}
\hlink[named:PrevPage]{go to previous page,}
\hlink[named:FirstPage]{go to first page,}
\hlink[named:LastPage]{go to last page.}
```

1.3.3 Transition actions

Transition actions generally make sense only when chained *after* jump actions. The specified transition/animation will occur, before destination is opened, but it will not override a transition defined for the particular page. The syntax is `[transition:`⟨*transition spec*⟩`]`. See 1.4 for more information about ⟨*transition spec*⟩.

Example:

```
\hlink[ref:yellow-slide, transition:Box:3:/M /O]
      {Go to yellow slide with long outward Box transition}
```

1.3.4 JavaScript actions

They allow executing pieces of JavaScript code using syntax: `[js:`⟨*name or script*⟩`]`. If ⟨*name or script*⟩ is a validly `\filedef`'d the script from file ⟨*name*⟩ will be executed (only embedded files are valid). Otherwise ⟨*script*⟩ will be used directly. Apart from reuse in different documents, scripts loaded from files may be encoded in UTF-16BE and hence support Unicode, inline ⟨*script*⟩s current don't.

Examples:

```
\openaction[js:{%
  app.alert("Javascript alert, open action");
  console.println("printing to console from openaction");
}]

\filedef/e[jstest]{test.js}
\hlink[js:jstest]{JavaScript action from file}
```

(Note how braces were used to guard the comma in JavaScript code from being interpreted as a action separator.)

In these actions you may want to use your own functions which should be defined before user has a chance of activating any other JavaScript actions. This is the purpose of "document level"

JavaScript actions. They function exactly the same way as normal JavaScript actions, but have names (although meaningless, they must be unique) and are executed in order of definition. Use `\dljavascript[⟨name⟩]{⟨name or script⟩}` for defining these actions:

```
\filedef/e[preamble]{preamble.js}
\djavascript[preamble]{preamble}

\dljavascript[initialization]{%
  var data = 42;
  function getRandomNumber() {
      return 4; // chosen by fair dice roll, https://xkcd.com/221/
  }
  console.println("initialized with seed " + getRandomNumber());
}
```

The JavaScript API available is not the same as the one in the web browsers. It is instead specified by Adobe: https://wwwimages.adobe.com/content/dam/acom/en/devnet/acrobat/pdfs/js_api_reference.pdf.

1.3.5 3D JavaScript actions

There are no "special" JavaScript actions for dealing with 3D. Normal JavaScript actions are used. You just have to access the 3D context of the annotation you want to control. For annotation with ⟨*name*⟩ the context is made available in `\DDDcontext{⟨name⟩}`. Under this context object you find all global definitions from the respective 3D scripts. For example if a "`turn`" function is defined, it is possible to call it like this:

```
\hlink[js:\DDDcontext{part3}.turn();]{Turn by 90 degrees along $x$ axis.}
```

1.3.6 GoTo3Dview actions

GoTo3Dview actions allow changing the view of the 3D scene to one of the predefined views (those listed by `views` comma separated list of `\RM`). The syntax is `[goto3dview:⟨name⟩:⟨view⟩]`. ⟨*name*⟩ is name of the annotation whose view we want to change and ⟨*view*⟩ is the intended view. You can refer to last inserted Rich Media annotation using empty name. For ⟨*view*⟩ you can either use "`(`⟨*view name*⟩`)`" (e.g. "`(part)`"), index of the view in the view list (zero based, e.g. "`0`" for the first view) or one of the special values: "`/N`" (next), "`/P`" (previous), "`/F`" (first) "`/L`" ("last").

```
% let's define an annotation with a few views
\RM[part.prc][
  activation=auto,
  views={front, left, top, isometric},
]{{\White\vrule width\hsize height\vsize}}

% try the different methods of reffering to views
\hlink[goto3dview::/N]{Next view},
\hlink[goto3dview::(left)]{left view} and
\hlink[goto3dview:part.prc:3]{third view}.
```

1.3.7 Rendition actions

Rendition actions can be used to control playback of "Rendition annotations". They use the syntax `[rendition:⟨name⟩:⟨operation⟩]`, where ⟨*name*⟩ refers to the name of the rendition to control and ⟨*operation*⟩ is one of `play`, `stop`, `pause` or `resume`. As a convenience, you can refer to last inserted rendition using empty name. If a file has been `\rendered` more than once with the same name, the action will influence the first instance.

Beware that currently these actions do not work in Evince and Okular (but do in Acrobat and Foxit).

Examples:

```
% rendition with name=video, that we want to control
\render[video]{\inspic{example-image.pdf}}

% we want the rendition action to have yellow border and red content
```

```
\let\_renditionborder\Yellow
\let\_renditionlinkcolor\Red

To start playing the video, click \hlink[rendition::play]{\"Play"}.
After that you can \hlink[rendition:video]{pause}.
```

1.4 Transitions and other page attributes

In PDF there are a few settings that can be set as *page attributes*. This means that they apply only to said page. For setting these page attributes, there are two options:

- value for "current page" (or rather the page where the command appears),
- default value used if "current page" value is not set.

While this package contains mechanism to handle all page attributes, not that many are useful for end user. The interesting ones remaining are:

- Transitions and page durations. When page has the transition attribute any jump to this page will display the requested animation (customized by the corresponding parameters). Page duration is the time before PDF viewer will auto advance to the next page. `\transition[`⟨*transition spec*⟩`]` sets transition for the "current page", `\transitions` sets the default. ⟨*transition spec*⟩ has three parts:
  ⟨*animation type*⟩`:`⟨*duration*⟩`:`⟨*raw PDF attributes*⟩
  ⟨*animation type*⟩ is one of `Split`, `Blinds`, `Box`, `Wipe`, `Dissolve`, `Glitter`, `Fly`, `Push`, `Cover`, `Uncover` and `Fade`, or the special value `R` which essentially means no animation and instantaneous transition (regardless of the set duration of transition). ⟨*duration*⟩ is the duration of transition in seconds (integer or decimal number). ⟨*raw PDF attributes*⟩ may be used to customize the animations (for example `/M` can set the direction of motion of `Split`, `Box` and `Fly`, e.g. `/M /I` for inward motion). For the raw PDF attributes refer to the standard itself[4]. `:`⟨*raw PDF attributes*⟩ or even `:`⟨*duration*⟩`:`⟨*raw PDF attributes*⟩ may be omitted. Default values specified by PDF standard will be used in that case (1 second duration and default values for all attributes). Page durations can be set either using `\defaultpageduration[`⟨*duration*⟩`]` or `\pageduration[`⟨*duration*⟩`]`, where duration is in seconds (default is no auto advancement, i.e. ∞). Examples:

  ```
  % unless stated otherwise all pages will have Wipe animation
  % with 1 second duration
  \transitions[Wipe:1]

  % but this page has a 1 second Fade animation
  \transition[Fade]

  \pg+ % (if using \slides)

  % and this page has 3 second Split animation
  % with vertical direction and inward motion
  \transition[Split:3:/Dm /V /M /I]
  ```

  Note that transitions are only displayed when in *full-screen mode*. You can use `\fullscreen` to have the document automatically open in full-screen mode.
- Additional actions. It is possible to define actions which will respond to page events: page open (`/O`) and page close (`/C`). They can be set using `\defaultpageactions[`⟨*additional actions spec*⟩`]` (the default for all pages) and `\pageactions[`⟨*additional actions spec*⟩`]` (current page override). ⟨*additional actions spec*⟩ consists of braced pairs of "event" (O or C in this case) and ⟨*action spec*⟩. Example:

  ```
  \pageactions[
    {O} {js:{app.alert("Page open, random = " + getRandomNumber());}}
    {C} {js:{app.alert("Page close!");}}
  ]
  ```

[4] https://wwwimages2.adobe.com/content/dam/acom/en/devnet/pdf/pdfs/PDF32000_2008.pdf#G11.2295795

1.5 Attachments

Every file embedded into a PDF file may optionally be presented in the user interface as an embedded file. This allows readers of the document to save or open the file.

PDF allows two ways of presenting attachments – using annotations (the attachments is represented by a rectangular icon on a page) or global array of attachments (attachments are visible in PDF viewers toolbar). The first method is intended more for document reviewers than for primary insertion. Hence this package supports only the second type.

You may add an embedded file to the global attachments array using `\attach[`⟨*name*⟩`]`. As usual, name is either a ⟨*name*⟩ defined with `\filedef` or alternatively a path to file which will be embedded (and `\filedef`d) automatically. It is not possible to `\attach` files referenced by path or URL.

If you want to automatically display toolbar with embedded files, consider using `\showattached` (see section 1.6).

1.6 Document view

You can choose what is shown when document is opened with commands:

- `\fullscreen` (the document is opened in full-screen mode),
- `\showoutlines` (show bookmarks/outlines toolbar)
- `\showattached` (show attachments in toolbar)

The commands are mutually exclusive and only the first appearing one will be respected.

You can set request two page view (odd pages on the right) using `\duplexdisplay`. It is useful for more natural display of double sided documents. Because it may not be desirable to automatically apply this, it is independent of `\margins`.

Chapter 2

Technical documentation

This is the technical documentation. It is intended for those who want to know how this package works internally. Casual users shouldn't need to read this. But if you would like to customize anything or perhaps just use some part of this package, feel free to copy paste and use anything you want in OpTEX's spirit.

This documentation is interleaved within the source itself, both are contained in a single file, `pdfextra.opm` (according to OpTEX conventions). The user documentation is instead contained in `pdfextra-doc.tex`, which itself `\input`'s the documented source file `pdfextra.opm` so that the user and technical documentation is available in a single PDF file, `pdfextra-doc.pdf`.

pdfextra.opm

```
\_codedecl \RM {Extra PDF features (v\_pdfextra_version)}
```

2.1 Package initialization

We are in the OpTEX package namespace. A couple of shortcuts are defined here: `\.isdefined`, `\.trycs`, `\.cs` `\.slet`, `\.slet`, `\.sdef` and `\.xdef`. They all hard code the package name, because we already have too many levels of indirection.

pdfextra.opm

```
\_namespace{pdfextra}

\_def\.isdefined#1{\_isdefined{_pdfextra_#1}}
\_def\.trycs#1{\_trycs{_pdfextra_#1}}
\_def\.cs#1{\_cs{_pdfextra_#1}}
\_def\.slet#1#2{\_slet{_pdfextra_#1}{_pdfextra_#2}}
\_def\.sdef#1{\_sdef{_pdfextra_#1}}
\_def\.sxdef#1{\_sxdef{_pdfextra_#1}}
```

2.2 Helper macros

The macros here are just helpers for the macros to follow. They are not useful generally, but proved useful in the expandable context of writing to PDF files.

Already the first one limits the use to LuaTEX (but who needs other engines anyways :). `\.emptyor`⟨*possibly empty text*⟩⟨*text to use when first argument is nonempty*⟩ checks whether the first argument is empty, if not it expands the second argument which can use the text from the first argument with `\.nonempty`. `\.attrorempty`⟨*attribute name*⟩⟨*value*⟩ builds upon the first one and is really useful for PDF dictionaries, when we don't want to write an attribute without a value (a default specified by standard will be used instead).

pdfextra.opm

```
\_def\.emptyor#1#2{%
   \_immediateassignment\_edef\.nonempty{#1}%
   \_ifx\.nonempty\_empty\_else #2\_fi
}
\_def\.attrorempty#1#2{\.emptyor{#2}{/#1 \.nonempty}}
```

There is a dillema for handling colors. While typesetting it is possible to use greyscale, CMYK or RGB colors. But there are contexts where it is possible to only use RGB colors. We want to provide the user with two possibilities of specifying colors:

- RGB color using PDF triplet (e.g. `1 0 0`),
- OpTEX color using control sequence (e.g. `\Blue`)

Both are handled by \.colortorgbdef⟨*cs*⟩⟨*color specification*⟩, which defines ⟨*cs*⟩ to the corresponding PDF RGB triplet. The indirection with defining a macro is because we want to use the processed color within expansion only contexts where grouping is not possible.

pdfextra.opm

```
\_def\.colortorgbdef#1#2{\_bgroup
  \_def\_setrgbcolor##1{##1}%
  \_def\_setcmykcolor##1{\_cmyktorgb ##1 ;}%
  \_def\_setgreycolor##1{##1 ##1 ##1}%
  \_xdef#1{#2}%
  \_egroup
}
```

\.xaddto\macro⟨*text*⟩ is a natural extension of OpTEX's \addto that expands ⟨*text*⟩ and is global.

pdfextra.opm

```
\_def\.xaddto#1#2{\_edef\.tmp{#2}%
  \_global\_ea\_addto\_ea#1\_ea{\.tmp}%
}
```

This package defines a few commands in the form \macro[⟨*name*⟩][⟨*optional arguments*⟩]{⟨*text*⟩}. To make it possible to omit the [⟨*optional arguments*⟩] \.secondoptdef is defined.

\.secondoptdef\⟨*macro*⟩⟨*parameters*⟩{⟨*body*⟩} defines \macro with first mandatory argument in brackets (saved to \.name). Second optional argument in brackets is scanned using helper macro defined with \optdef and is saved to _opt token list). Additional ⟨*parameters*⟩ can be specified as with \optdef (numbered from #1).

pdfextra.opm

```
\_def\.secondoptdef#1{%
  \_def#1[##1]{\_def\.name{##1}\.cs{sopt:\_string#1}}%
  \_ea\_optdef\_csname _pdfextra_sopt:\_string#1\_endcsname[]%
}
```

When processing comma separated lists sometimes it is needed to ignore the remaining text. For this we use \.untilend macro which ignores everything up to dummy \.end. This is analogous to OpTEX's _finbody used for the same purpose. Sometimes \.end is used as sentinel and compared in \ifx tests, hence we define it to a unique value.

pdfextra.opm

```
\_def\.untilend#1\.end{}
\_def\.end{_pdfextra_end}
```

For various uses it is necessary to know the number of page where something happens. This has to be handled asynchronously with \write. Here we use OpTEX specific .ref file and associated macros, but this could be replaced as long as the same interface is exposed.

\.setpageof⟨*name*⟩ writes \.Xpageof⟨*name*⟩ to the .ref file. In the next TEX run \.Xpageof finds out the page number (\gpageno) from OpTEX's _currpage and saves it so that \.pageof⟨*name*⟩ can retrieve it. In the first run we can't be sure of the page where the content will end up. As a rough estimate we take the current page – this actually works well for slides where page breaks are manual.

\.pageof is expandable, but we want to let the user know, that the document needs to be processed twice. Therefore we use LuaTEX's \immediateassignment to increment the counter of unresolved references.

When .ref file is read along with the defintion of \.Xpageof this package has not been loaded yet. Hence we can't use namespaced variants of \.isdefined, etc.

pdfextra.opm

```
\_refdecl{%
  \_def\.Xpageof#1{\_isdefined{_pdfextra_pageof:#1}\_iffalse
    \_sxdef{_pdfextra_pageof:#1}{\_ea\_ignoresecond\_currpage}\_fi
  }%
}

\_def\.setpageof#1{\_openref \_ewref\.Xpageof{{#1}}}

\_def\.pageof#1{%
  \.trycs{pageof:#1}{%
    \_immediateassignment\_incr\_unresolvedrefs
    \_the\_numexpr\_gpageno+1\_relax % best effort = current page num
  }%
}
```

2.3 Handling of files

Handling of files is a big topic of this package. Files are everywhere – files containing multimedia, JavaScript script files, attachments, externally referred files... Therefore a more sophisticated mechanism for handling files is needed. The mechanism introduced in this section handles all three cases of a *file specification*:

- files embedded in the PDF ("e", embedded file),
- files determined by path ("x", external file),
- files determined by URL ("u", url file).

Although ideally all three would be interchangible this is not always the case, because e.g. some media files must be embedded and linking to external resources does not work with embedded files.

In most cases there are two many names and other associated values involved:

- Some kind of a "friendly" name. This one is sometimes shown by PDF viewers.
- The real name of the file. Also shown but in different contexts.
- The path or URL used to determine the file.
- MIME type of the file.

For example when talking about OpTEX's documentation we might have a friendly name of "opdoc", file name of "`optex-doc.pdf`", URL of "`http://petr.olsak.net/ftp/olsak/optex/optex-doc.pdf`" and MIME type of "application/pdf". Different subset of them is required in different contexts, but the user should only have to specify the friendly name (by which they will refer to the file) and the path/URL of the file. The rest will be deduced. The friendly name is used as a handle and *is usable* in all places where file specification is required (although it may not produce conforming output, see above).

In this two step process – definition and (re)use – we introduce a command for defining files: `\filedef/`⟨*type*⟩ `[`⟨*friendly name*⟩`]{`⟨*path or URL*⟩`}`. The macro itself does general definitions and dispatches the type dependant work to other macros in the form `_filedef:`⟨*type*⟩.

pdfextra.opm

```
\_def\.filedef/#1#2[#3]#4{%
   \.sxdef{filename:#3}{(\.filename{#4})}%
   \_edef\.tmp{\.exttomime{\.fileext{#4}}}%
   \_ifx\.tmp\_empty
      \_opwarning{MIME type of '#4' unknown, using '\.defaultmimetype'}%
      \_edef\.tmp{\.defaultmimetype}%
   \_fi
   \.sxdef{filemime:#3}{\.tmp}%
   \.cs{filedef:#1}{#3}{#4}%
}
\_nspublic \filedef ;
```

Types "e", "x", "u" are predefined, anything else would essentialy be a variant of these.

External file ("x") is determined only by path.

pdfextra.opm

```
\.sdef{filedef:x}#1#2{%
   \.slet{filespec:#1}{filename:#1}%
}
```

URL file ("u") is determined by URL. Using all sorts of characters is allowed by using `\_detokenize`. This time it is necessary to create full *file specification* – a dictionary, where the "file system" is URL.

pdfextra.opm

```
\.sdef{filedef:u}#1#2{%
   \.sdef{filespec:#1}{<</FS /URL /F (\_detokenize{#2})>>}%
}
```

Embedded files ("e") are the most interesting ones. For further use (e.g. for displaying the embedded files as attachments) MIME type is required. It is saved in the stream as a `\Subtype`, encoded as a PDF name (e.g. `/video#2Fmp4`). The embedded file stream must be wrapped in a full *file specification*, which has the `/EF` ("embedded file") entry. Also the friendly name is used for some purpose by PDF viewers, so it set in `/Desc` (description).

pdfextra.opm

```
\.sdef{filedef:e}#1#2{%
   \_edef\.tmp{\.cs{filemime:#1}}%
   \_isfile{#2}\_iffalse
      \_opwarning{file '#2' not found}%
```

```
  \_fi
  \_pdfobj stream
    attr{/Type /EmbeddedFile /Subtype \_ea\.mimetoname\_ea[\.tmp]}
    file {#2}%
  \_pdfrefobj\_pdflastobj
  \.sxdef{filestream:#1}{\_the\_pdflastobj\_space 0 R}%
  \_pdfobj {<</Type /Filespec
    /F \.cs{filename:#1}
    /Desc (#1)
    /EF << /F \_the\_pdflastobj \_space 0 R >>%
  >>}%
  \_pdfrefobj\_pdflastobj
  \.sxdef{filespec:#1}{\_the\_pdflastobj\_space 0 R}%
}
```

Now the less interesting part – determining the file names from paths and determining MIME types. The file name is the part after the last "/" (if any). The file extension is the part after last "." (if any).

pdfextra.opm

```
\_def\.filename#1{\_ea\.filenameA#1/\.end}
\_def\.filenameA#1/#2{\_ifx\.end#2#1\_else\_afterfi{\.filenameA#2}\_fi}

\_def\.fileext#1{\_ea\.fileextA#1.\.end}
\_def\.fileextA#1.#2{\_ifx\.end#2#1\_else\_afterfi{\.fileextA#2}\_fi}
```

MIME type is determined from file extension (e.g. `mp4` is "video/mp4"). For mapping of file extensions to MIME types we abuse TEX's hash table which gets populated with "known MIME types". This necessarily means that the database is incomplete. Users can define their own additional mappings, or they can contribute generally useful ones to this package.

The default MIME type (used for unknown file extensions) is "application/octet-stream" – binary data.

The uninteresting MIME type database itself is at the very end (2.9).

pdfextra.opm

```
\_def\.mimetoname[#1/#2]{/#1\_csstring\#2F#2}

\_def\.defaultmimetype{application/octet-stream}
\_def\.exttomime#1{\.trycs{mimetype:#1}{}}
```

Here we define an OpTEX style "is-macro" that checks whether the file has already been defined – `\.isfiledefined{`⟨*name*⟩`}\iftrue` (or `\iffalse`). The case where the file has not been defined using `\filedef` can be handled in a lot of ways. As a default we interpret ⟨*name*⟩ as path and try to embed it. Because the path from ⟨*name*⟩ is used as the "friendly name" the file will be embedded only once even when requested more times.

pdfextra.opm

```
\_def\.isfiledefined#1#2{\.isdefined{filespec:#1}\_iftrue\_else
  \_afterfi{\.fileundefined{#1}}\_fi#2%
}

\_def\.fileundefined#1{\_isfile{#1}\_iftrue\.filedef/e[#1]{#1}\_else
  \_opwarning{file '#1' not found, ignored}\_ea\_unless\_fi
}

% strict requirement of preceeding `\filedef` can be set like this:
%\_def\.fileundefined#1{\_opwarning{file '#1' is not defined, ignored}\_unless}
```

2.4 PDF actions

The core of interactivity in PDF are actions. They are all initialy handled by `\pdfaction[`⟨*action spec*⟩`]`. ⟨*action spec*⟩ is a comma separated list of ⟨*type*⟩`:`⟨*arguments*⟩. Leading spaces in the elements of the list are ignored using undelimited-delimited argument pair trick.

An invocation could look like this:

```
\pdfaction[
   js:{app.alert("Yay JavaScript, going to page 5");},
   ilink:pg:5,
   transition:Wipe,
]
```

This is why we have to be very careful when loading the contents between `[]` to arguments. In particular, we can't split immediatly using `[#1:#2]`, because this would discard the braces guarding the comma in the JavaScript code. However we also need to find out the *type* of action which is taken as a type of the first action (`js` in this case). `\.pdfactiontype[`⟨*action spec*⟩`]` does this – we don't mind that there the braces are lost.

`\pdfaction` processes the list, to create a chain of actions using `/Next` field. The handling of each action type is up to macro `\_pdfextra_`⟨*type*⟩`action`, which receives `[`⟨*type*⟩`:`⟨*arguments*⟩`]`. Because of this a single type handler can handle multiple different actions, as is the case with `\.ilinkaction` which is the fallback for unknown action types.

pdfextra.opm

```
\_def\.pdfaction[#1#2]{\.pdfactionA#1#2,\.stop\.end}
\_def\.pdfactionA#1,#2#3\.end{%
   <<%
   \.pdfactionB[#1]%
   % next action
   \_ifx\.stop#3\_else\_space
      /Next \_afterfi{\.pdfactionA#2#3\.end} % intentional space
   \_fi
   >>
}
\_def\.pdfactionB[#1:#2]{\.trycs{#1action}{\_ea\.ilinkaction}[#1:#2]}
\_nspublic \pdfaction ;

\_def\.pdfactiontype[#1:#2]{#1}
```

2.4.1 Additional actions

Some PDF objects, like pages and some annotations, can also have "additional actions". These are actions which will be executed when an event happens – like page getting opened for `/O` action in page's additonal actions or `/PO` in annotation's additional actions. For constructing these additional actions we define a helper macro `\.pdfaactions`. The use is as something follows:

`/AA << \.pdfaactions{ {O} {`⟨*action spec 1*⟩`} {C} {`⟨*action spec 2*⟩`} } >>`

To produce something this:

`/AA << /O <<`⟨*action 1*⟩`>> /C <<`⟨*action 2*⟩`>> >>`

pdfextra.opm

```
\_def\.pdfaactions#1{<<\.pdfaactionsA #1\.end\.end>>}
\_def\.pdfaactionsA#1#2{\_ifx\.end#1\_else /#1 \_ea\.pdfaction\_ea[#2]\_ea\.pdfaactionsA\_fi}
```

2.4.2 Link annotations

The main use of actions – annotations of `/Subtype /Link`. Annotation of this type creates an active rectangular area on the page that executes a PDF action (or chain of them in the general case). `\hlinkactive[`⟨*action spec*⟩`]`⟨*text*⟩ is a natural extension of OpTEX's `\linkactive`, that supports the classic "jump" actions of types `ref`, `bib`, etc. But also other types of actions. No distinction between "internal" (`\ilink`) and external (`\ulink`) links is made. The mechanism is completely generic.

The `\pdfstartlink`/`\pdfendlink` primitives are used to denote the part of the page where ⟨*text*⟩ appears as active. LuaTEX will then handle even the situations where ⟨*text*⟩ gets broken across multiple lines (by creating multiple rectangular annotations to cover all `\hbox`es).

pdfextra.opm

```
\_def\.hlinkactive[#1]#2{\_bgroup\_def\#{\_csstring\#}%
    \_edef\.type{\.pdfactiontype[#1]}%
    \_quitvmode\_pdfstartlink \.linkdimens
       attr{\_pdfborder{\.type}}%
       user{/Subtype /Link /A \.pdfaction[#1]}\_relax
    \_localcolor\.linkcolor{\.type}#2\_pdfendlink\_egroup
}
```

`\hlinkactive` itself is dormant before `\hyperlinks` occurs. Until then, a dummy `\hlink` is used. OpTEX's `\hyperlinks` is extended to hook `\hlink` activation. The colors of internal/external links are remembered for compatibility with OpTEX.

pdfextra.opm

```
\_def\.hlink[#1]#2{\_quitvmode{#2}}
\_nspublic \hlink ;

\_let\.oldhyperlinks\_hyperlinks
\_def\_hyperlinks#1#2{%
   \.oldhyperlinks#1#2
   \_let\.ilinkcolor=#1%
   \_let\.elinkcolor=#2%
   \_let\.hlink=\.hlinkactive
   \_let\hlink=\.hlink
}
\_public \hyperlinks ;
```

Two customizations of `\hlinks` are possible:

- Dimensions of rectangular areas created by `\pdfstartlink`/`\pdfendlink`. This is done using `\.linkdimens` (analogous to OpTEX's `\linkdimens`). Dimensions that are unset are taken from the respective `\hbox`es. `\lininglinks` sets the dimensions for running text – it covers all space of a line using `\baselineskip`. `\nolininglinks` sets no dimensions, this is useful for buttons, that may have larger height/depth than a line.
- The color is determined from the type of link (that is the first action in ⟨*action spec*⟩) by checking `\_`⟨*type*⟩`linkcolor` (compatible with OpTEX) or `\_pdfextra_`⟨*type*⟩`linkcolor`. As a fallback `\.ilinkcolor` is used (set by `\hyperlinks`).

pdfextra.opm

```
\_def\.lininglinks{%
   \_def\.linkdimens{height.75\_baselineskip depth.25\_baselineskip}%
}
\_def\.nolininglinks{\_def\.linkdimens{}}
\.lininglinks

\_nspublic \lininglinks \nolininglinks ;

\_def\.linkcolor#1{\_trycs{_#1linkcolor}{\.trycs{#1linkcolor}{\.ilinkcolor}}}
\_def\.ilinkcolor{}
```

2.4.3 Open action

The document itself has one action defined in the document catalog. It is called `/OpenAction`. We allow the user to set it using the familiar ⟨*action spec*⟩ syntax with the command `\openaction[`⟨*action spec*⟩`]`.

Internally we could directly set it by appending to the catalog using the primitive `\pdfcatalog`, but LuaTEX (pdfTEX really) allows setting the action with special syntax. This has the benefit that it is not allowed to set the action more than once.

pdfextra.opm

```
\_def\.openaction[#1]{\_pdfcatalog{} openaction user{\.pdfaction[#1]}\_relax}
\_nspublic \openaction ;
```

2.4.4 Jump actions

These are the most typical actions. Even LuaTEX itself handles them, although we don't use the possibility for maintaining generality. There are a few types of jump actions:

- `/GoTo` actions are the classic internal links to named destinations in the PDF file (created by `\pdfdest` primitive or OpTEX's `\dest`). The destination names include also the type of internal link (e.g. `ref:section1`). They are handled by `\.ilinkaction[`⟨*type*⟩`:`⟨*name*⟩`]`.
- `/URI` actions which are in most cases used as "goto URL" actions. These are not that useful directly, because special characters should be handled before this actions is used (like with `\url`). The low level use is `\.urlaction[url:`⟨*url*⟩`]`.
- "Goto remote" actions, which can jump to a destination in another PDF file – either determined by name, or by page number. The external files are expected to be defined by `\filedef` (but not the embedded variant). The use is either `\.extrefaction[extref:`⟨*name*⟩`:`⟨*named destination*⟩`]` for links to named destination or `\.extpgrefaction[extpgref:`⟨*name*⟩`:`⟨*page number*⟩`]` for page destinations. Customization is possible with `\.extrefextra`, by default opening in a new windows is requested.

pdfextra.opm

```
\_def\.ilinkaction[#1:#2]{/S /GoTo /D (#1:#2)}

\_def\.urlaction[#1:#2]{/S /URI /URI (#2)}

\_def\.extrefaction[#1:#2:#3]{/S /GoToR
   /F \.cs{filespec:#2}
   /D (#3)
   \.extrefextra
}
\_def\.extpgrefaction[#1:#2:#3]{/S /GoToR
   /F \.cs{filespec:#2}
   /D [\_the\_numexpr#3-1\_relax\_space /Fit]
   \.extrefextra
}

\_def\.extrefextra{/NewWindow true}
```

Transition action is not really a jump action in of itself, but is only useful when chained after jump actions, so we define it here. Transitions (as page attributes) are handled more thoroughly in section 2.5.1.

The use would look something like:
`\.transitionaction[transition:`⟨*animation type*⟩`:`⟨*duration*⟩`:`⟨*raw PDF attributes*⟩`]`, where all fields omitted from right take the default values.

pdfextra.opm

```
\_def\.transitionaction[#1:#2]{/S /Trans \.attrorempty{Trans}{\.maketrans[#2]}}
```

2.4.5 Named actions

User can request arbitrary "named" action with `\.namedaction[named:`⟨*name*⟩`]`. See user documentation for details.

pdfextra.opm

```
\_def\.namedaction[#1:#2]{/S /Named /N /#2}
```

2.4.6 JavaScript actions

JavaScript actions have two forms, either `\.jsaction[js:`⟨*name*⟩`]` or `\.jsaction[js:`⟨*script*⟩`]`. The first variant uses contents of `\filedef`'d ⟨*name*⟩, the second one uses ⟨*script*⟩ directly. There is no special catcode handling.

pdfextra.opm

```
\_def\.jsaction[#1:#2]{/S /JavaScript
   /JS \_ifcsname _pdfextra_filestream:#2\_endcsname \_lastnamedcs \_else
       (#2)
   \_fi
}
```

2.5 Page attributes

PDF represents pages as dictionaries. The dictionaries get generated by LuaTEX, which fills in some attributes *attributes* (like `/Content` with contents of the page and `/Annots` with array of annotations). We can add more using `\pdfpageattr` primitive token list register. While not that many are generally useful, there are a few interesting ones. For example transitions can be set using page attributes, or we might want to set additional actions (`/AA`) to listen for page events.

While the so called "page objects" are in a tree structure (for fast lookup), only the leaves are real "pages". PDF allows some attributes to be inherited from parent page objects, but not all of them and certainly not those we are interested in.

The mechanism introduced in this section is optional, because it takes complete control over `\pdfpageattr`. It gets activated when `\initpageattributes` is first used (which happens automatically for some functionality exposed by this package), but may be activated by the user for any other purpose. Only attributes listed in `\pageattributes` are processed.

We set the attributes anew for each page, by hooking into OpTEX's `\_begoutput`. Because `\pdfpageattr` token list doesn't get expanded before written out to PDF, we expand it using the assignment in `\edef` trick. The token list gets expanded, but the assignment is not made until it reaches main processor when the temporary control sequence gets expanded.

pdfextra.opm

```
% pdfpagattr managament (default for all pages vs current page override)
\_def\.pageattributes{{Trans}{Dur}{Rotate}{AA}}
\_def\.initpageattributes{%
   % add hook for setting primitive \pdfpageattr
   \_addto\_begoutput{\_edef\.tmp{\_pdfpageattr={\.pdfpageattributes}}\.tmp}%
   % no need to do this twice
   \_let\.initpageattributes=\_relax
}
\_nspublic \pageattributes \initpageattributes ;
```

The user interface we want to expose has two parts:

- setting the page attribute for just this one page (\.pdfcurrentpageattr),
- setting the default attribute (used when current page value is not set) (\.pdfdefaultpageattr).

The first one of course brings in the typical TEX problem of knowing the page where something occurs. As always, the page number contained in \gpageno during processing of said content may of course not actually be the number of the page where the content ends up! Hence, we need to note the page number with a delayed write, using \.setpageof and later \.pageof. The different settings of page attributes should have distinct names, we use the \.pageattrcount counter for this.

pdfextra.opm

```
\_newcount\.pageattrcount
\_def\.pdfcurrentpageattr#1#2{\.initpageattributes
   \_incr\.pageattrcount
   \.setpageof{pageattr:\_the\.pageattrcount}%
   \.sxdef{pdfpgattr:\.pageof{pageattr:\_the\.pageattrcount}:#1}{#2}%
}
\_def\.pdfdefaultpageattr#1#2{\.initpageattributes
   \.sxdef{pdfpgattr:#1}{#2}%
}
```

Finally, the macro \pdfpageattributes takes care of setting generating the contents of \pdfpageattr. For each attribute in \pageattributes it first checks its current page value, only then the default value. If neither is set, nothing is added.

pdfextra.opm

```
\_def\.pdfpageattributes{\_ea\.pdfpageattributesA\.pageattributes\.end}
\_def\.pdfpageattributesA#1{\_ifx\.end#1\_else
   % use current page override or "default"
   % don't emit anything if the value is empty
   \.attrorempty{#1}{%
      \.trycs{pdfpgattr:\_the\_gpageno:#1}{\.trycs{pdfpgattr:#1}{}}%
   }%
   \_ea\.pdfpageattributesA\_fi
}
```

Each attributes then has two switches for the respective default and current values. For defining a few of them a helper is introduced:

\.pdfpageattributesetters ⟨*attribute*⟩ \⟨*default setter*⟩ \⟨*current setter*⟩ {⟨*value*⟩},

where ⟨*attribute*⟩ is name of the attribute without the slash (e.g. MediaBox), the two control sequences name the future user setters, which will take single argument in brackets (e.g. \mediabox and \thismediabox) and the ⟨*value*⟩ can use the argument.

pdfextra.opm

```
\_def\.pdfpageattributesetters#1 #2#3#4{%
   \.sdef{\_csstring#2}[##1]{\.pdfdefaultpageattr{#1}{#4}}%
   \.sdef{\_csstring#3}[##1]{\.pdfcurrentpageattr{#1}{#4}}%
   \_nspublic #2 #3 ;
}
```

Some of the useful attributes are /Rotate, which rotates the pages visually (can be set with \defaultpagerotate and \pagerotate), and the additional actions (/AA, see section 2.4.1, set using \defaultpageactions \pageactions).

pdfextra.opm

```
\.pdfpageattributesetters Rotate \defaultpagerotate \pagerotate {#1}

\.pdfpageattributesetters AA \defaultpageactions \pageactions {\.pdfaactions{#1}}
```

2.5.1 Transitions, page durations

There are predefined types of transitions, like `/Wipe`, `/Box`, `/Split`, etc. Most have other customizible attributes – usually directions set in different ways depending on the animation type at hand, but the most important attribute is the duration of the animation. Parsing friendly user notation in the form of `[`⟨*animation type*⟩`:`⟨*duration*⟩`:`⟨*other raw attributes*⟩`]`, where fields from the right may be omitted to produce the default value, is handled by `\.maketrans`. This macro is also used by transition actions (see 2.4.4). The defaults are simply those defined by PDF standard (no transition, 1 second duration and the respective default directions).

pdfextra.opm

```
\_def\.maketrans[#1]{\.maketransA#1:::\.end}
\_def\.maketransA#1:#2:#3:#4\.end{%
   \.emptyor{#1}{<</S /\.nonempty \.attrorempty{D}{#2} #3>>}
}
```

The attribute setters for transitions (`\transitions`, `\transition`) are a simple wrappers. Similiar is the setting of page duration in seconds after which PDF viewer automatically advances to the next page (`\defaultpageduration`, `\pageduration`).

pdfextra.opm

```
\.pdfpageattributesetters Trans \transitions \transition {\.maketrans[#1]}

\.pdfpageattributesetters Dur \defaultpageduration \pageduration {#1}
```

2.6 Attachments and document level JavaScript

These don't have any last place to be in, so they are documented separately, here. Attaching files using `/FileAttachment` annotations:

1. is intended more towards viewers of the document for extra additions and
2. doesn't work in the viewers as well as one would like.

That is why instead embed files using normal `\filedef` and then allow them to be added to the document level `/EmbeddedFiles` entry, which means they will be shown in the user interface by PDF viewers. `/EmbeddedFiles` is a document level name tree (contained inside `/Names` entry of `/Catalog`) that maps names of files to their objects. Although we simplify matters by constructing more of an array.

What works very similiarly is document level JavaScript. It is a name tree within `/JavaScript` field. It maps names of JavaScript actions to their object numbers. The names aren't very useful, but the actions have their purpose. They are executed in turn after the document is opened. Hence they can be used to predefine JavaScript functions in the global context, to be used later within actions explicitly activated by the user.

The user level commands are `\attach[`⟨*name*⟩`]` (to attach a previously `\filedef`'d name with fallback to embedding now if it is a valid path) and `\dljavascript[`⟨*name*⟩`]{`⟨*script*⟩`}` (adds action that executes ⟨*script*⟩ after document is opened, ⟨*name*⟩ is more or less meaningless).

Internally both commands construct lists of what ends up in the resulting name array, i.e. pairs `(`⟨*name*⟩`)␣`⟨*object␣number*⟩`␣0␣R␣`. Intermediate macros `\.embeddedfiles` and `\.dljavascripts` are used for this.

In the case of file attachments, nothing happens if file is defined and not found by the fallback.

pdfextra.opm

```
% file attachment
\_def\.embeddedfiles{}
\_def\.attach[#1]{\.isfiledefined{\.name}\_iftrue
   \.xaddto\.embeddedfiles{(#1) \.cs{filespec:#1} }\_fi
}
\_nspublic \attach ;

\_def\.dljavascripts{}
\_def\.dljavascript[#1]#2{%
   \_immediate\_pdfobj{<< \.jsaction[js:{#2}] >>}%
   \.xaddto\.dljavascripts{(#1) \_the\_pdflastobj \_space 0 R }%
}
\_nspublic \dljavascript ;
```

Object creation, which is common to both, is handled by

`\.makenamearray`⟨*name tree name*⟩⟨*name tree content*⟩.

It doesn't do anything for empty lists, to not bloat PDF files when this mechanism isn't used.

pdfextra.opm

```
\_def\.makenamearray#1#2{\_ifx#2\_empty\_else
   \_immediate\_pdfobj {<< /Names [ #2 ] >>}%
   \_pdfnames{/#1 \_the\_pdflastobj \_space 0 R }\_fi
}
```

The lists themselves can only be written out to the PDF file at the very end of the run. We use OpTEX's `\_byehook`, which is run in `\_bye`. But `\bye` itself may be predefined by the user, for example when using some of the OpTEX tricks. We just hope that user keeps `\_byehook`.

pdfextra.opm

```
\_addto\_byehook{%
   \.makenamearray{EmbeddedFiles}\.embeddedfiles
   \.makenamearray{JavaScript}\.dljavascripts
}
```

2.7 Viewer preferences

There are a few customizations of display (and other preferences of PDF viewers) possible in the document catalog or its subdictionary `/ViewerPreferences`. Most are not that useful. The interesting ones are implemented by `\fullscreen`, `\showoutlines`, `\showattached`. They all set the page mode using `\.setpagemode`. We don't handle respecting the last setting (using `\_byehook`). To prevent invalid PDF files, we set `\.setpagemode` to `\_relax` after use.

pdfextra.opm

```
\_def\.setpagemode#1{\_pdfcatalog{/PageMode /#1}\_glet\.setpagemode=\_relax}

\_def\.fullscreen{\.setpagemode{FullScreen}}
\_def\.showoutlines{\.setpagemode{UseOutlines}}
\_def\.showattached{\.setpagemode{UseAttachments}}

\_nspublic \fullscreen \showoutlines \showattached ;
```

Only the setting of duplex / double sided printing and display is in the nested dictionary. It is handled by `\duplexdisplay`. The simplistic version does not handle more attributes in `/ViewerPreferences`. We also set the meaning to `\_relax` to prevent more (erroneous) uses.

pdfextra.opm

```
\_def\.duplexdisplay{\_pdfcatalog{%
  /PageLayout /TwoPageRight
  /ViewerPreferences <<
    /Duplex /DuplexFlipLongEdge
  >>}%
  \_glet\.duplexdisplay=\_relax
}

\_def\duplexdisplay{\.duplexdisplay}
```

2.8 Multimedia

PDF essentially allows insertion of different types of multimedia:

- images,
- audio/video,
- 3D art.

The first is pretty standard and handled normally by the engine (LuaTEX). Others are possible, but have to be done manually according to one of the mechanisms specified by PDF standard:

- Sounds (audio only),
- Movies (video and/or audio),
- Renditions (video and/or audio),
- 3D annotations (3D art),
- Rich Media (video and/or audio, 3D art)

Sadly all these mechanisms are badly flawed, each in different ways. At least we try to use the one that works in the viewers.

For audio/video "Movies" are the simplest mechanism, but they have been deprecated in PDF 2.0 and no longer work in Acrobat/Foxit (same for "Sounds").

"Renditions" are complicated, partly dependant on JavaScript, but at least supported by Acrobat, Foxit, Evince and Okular.

"Rich media" annotations were designed for Flash. This use case is no longer possible today, but the obscurities remain. They are unnecessiraly complicated, but can be used without Flash too. Although the result is very plain for audio/video – no controls can be displayed and there are no associated actions.

"3D annotations" are reasonably simple, but also flawed. They cannot reuse embedded file as a source for 3D data. Hence it is better and more consistent to use Rich Media for 3D annotations. It even has additional benefits, like the possibility of using multiple initialization scripts.

In the end, this package exposes two user commands corresponding to two mechanisms – first are Renditions (`\render`) for audio/video that works in most browsers and Rich Media (`\RM`) mainly for 3D art, but also for audio/video with limited possibilities.

Both mechanisms have an annotation at their core. Annotations is essentially a rectangular area on page. The area corresponds to where the multimedium will show up. After activating the area somehow (by user click, or action) the multimedium will start playing. Before annotations the rectangular area will show something that is called "normal appearance". This appearance is of type form XObject. Those are really similiar to pages – they have dimensions, contents made up of PDF graphics operators, ..., but they are reusable. Not that useful for annotations where we will need the form only once, but nice anyways. pdfTEX has primitives for creating them – `\pdfxform` and friends. They essentially do the same code like `\shipout` does, but instead of page, they make this reusable object. One can then either use this reusable object in another page/form, but we will indirectly refer to it for the appearance.

Important aspect of annotations is that they are really only rectangular areas on the page, but they are not really part of the page. They sort of sit on another level and are not influenced by PDF graphic operators which make the page. In pdfTEX annotations are handled by *whatsit* nodes. While most nodes map to known primitive TEX concepts (like typeset characters, boxes, rules, etc.) Whatsits are essentially commands for TEX that are delayed until page is being shipped out (written to PDF file). `\write`, `\special`, and most pdfTEX commands create whatsits. For annotationos this is important, because this means that the engine only stores the information about annotation that we specify, but creates it at due time, when it should be written to PDF.

Because whatsits are essentially dimensionless and we want it to be a part of normal TEX typesetting material we create the annotation (whatsit) in `\hbox`. This box will be otherwise empty, because the apperance of the rectangular area is determined by the normal appearance field (`/N` in `/AP`). We set the dimensions of the box to the dimensions of normal appearance. Everything will line up nicely, because when processed, the annotation will take dimensions from the box.

All of these concepts are implemented in:

`\.boxedannot[`⟨*type*⟩`:`⟨*name*⟩`]{`⟨*appearance*⟩`}{`⟨*special text*⟩`}{`⟨*annotation attributes*⟩`}`

⟨*type*⟩ is used to determine the annotation border (same principle as with Link annotations, section 2.4.2), ⟨*name*⟩ will be used as the annotation name (`/NM`), ⟨*special text*⟩ is used for influencing the `\pdfannot` primitive, and ⟨*annotation attributes*⟩ will become the body of the annotation.

pdfextra.opm

```
\_def\.boxedannot[#1:#2]#3#4#5{%
   \_setbox0=\_hbox{#3}\_setbox2=\_null
   \_ht2=\_ht0 \_wd2=\_wd0 \_dp2=\_dp0
   \_immediate\_pdfxform0
   % box with annotation both stretching to dimensions of appearance
   \_hbox{\.setpageof{#1:#2}%
      \_pdfannot #4 {#5
         /AP <</N \_the\_pdflastxform \_space 0 R>>
         \_pdfborder{#1}
         /NM (#2)
         /Contents (#1 '#2')
      }%
      \_copy2
   }%
}
```

There is another weird thing common to both multimedia mechanisms – the redefinition of `\.name`. It is initially set by `\.secondoptdef` to ⟨*name*⟩, but may be redefined by user supplied `name` key-value

parameter. This should be used when there are multiple uses of the same content. Otherwise samely named annotations would be indistinguishable both for PDF viewer and our handling of actions (which would all refer only to the first instance).

To somewhat overcome this, trying to use the same ⟨*name*⟩ (within the same type of annotaiton) will use dummy name from `\.unnamedannotcount` (for uniqueness). This means that ⟨*name*⟩ will always refer to the first instance. `\.redefinename` handles this.

pdfextra.opm

```
\_newcount\.unnamedannotcount
\_def\.redefinename#1{%
   \.isdefined{#1:\.name}\_iftrue
      \_incr\.unnamedannotcount
      \_edef\.name{\_the\.unnamedannotcount}%
   \_else
      \_edef\.name{\_kv{name}}%
   \_fi
}
```

2.8.1 Renditions (audio/video)

There are three main types of PDF objects involved in the Renditions ("Multimedia") mechanism:

- Screen annotations define the area for playing multimedia.
- Rendition objects define the multimedia to play.
- Rendition actions associate Rendition objects with Screen annotations.

You can theoretically arbitrarily mix and match rendition objects and screen annotations by invoking different actions. In practice Evince and Okular do really simplistic parsing and don't fully support the actions fully. But by keeping it simple it is possible to make it work almost the same in all viewers that support renditions.

Different sources of audio/video should be possible. In fact all three file specifications (embedded files, files specified by URL/path) could work. Again in practice embedded file is the safest bet, that works in all viewers that support renditions.

The user facing command is:

`\render[`⟨*name*⟩`][`⟨*optional key-value paramers*⟩`]{`⟨*horizontal material*⟩`}`

⟨*name*⟩ is the friendly name set using `\filedef` or file path if ⟨*name*⟩ isn't `\filedef`d and is to be embedded. The key-value parameters in brackets can be entirely omitted. They can influence the playback (except for `controls` most are not well supported). Default values are taken from `\.renderdefaults`.

`\render` doesn't do anything (except print warning) if file ⟨*name*⟩ isn't defined and ⟨*name*⟩ isn't a path to file that can be embedded.

The first PDF object it defines is Rendition, which specifies information about the multimedium (name, file specification, MIME type and options from key-value parameters). Some of the fields are in `/BE` ("best effort") dictionaries. This is due to the very general design of Renditions, which theoretically allows the PDF viewer to choose from multiple Renditions if they know they can't support some of the requested features. But that is not much useful in practice, so we just don't complicate it.

Next defined object is Screen annotation, which complicates thing by requiring (`/P`) reference to the page where the annotation is (handled by `\setpageof` and `\pageof` pair). Important field is `/A` which specifies actions that shall be executed when the screen area is clicked. We let the user change the action, but the sensible default of starting to play the multimedium is used (and this is the only thing that works in some viewers anyways). Additional actions `/AA` may be used to react to events like mouse over or page open/close – the most probable use case is autoplay on page open, for which shortcut of `\renditionautoplay` is defined.

The code is slightly complicated by the fact, that actions need to reference the Rendition and Screen objects. In the case of the action contained in Screen annotation this essentialy involves a self reference. Hence it is needed to first reserve an object number and later use it for the annotation. Because the object numbers may also be needed by actions defined later, we need to save them to `\_pdfextra_rendition:`⟨*name*⟩ and `\_pdfextra_screen:`⟨*name*⟩ respectively, but also define aliases with empty names, so users can easily reference the latest rendition.

pdfextra.opm

```
\.secondoptdef\.render#1{\.isfiledefined{\.name}\_iftrue\_bgroup
   \_ea\_readkv\_ea{\_ea\.renderdefaults\_ea,\_the\_opt}%
   \.colortorgbdef\.bgcolor{\_kv{background}}%
   % rendition object ("media specifaction")
```

```
  \_pdfobj {<</Type /Rendition
     /S /MR
     /N \.cs{filename:\.name}
     /C <<%/Type /MediaClip
       /S /MCD % subtype MediaClipData
       /D \.cs{filespec:\.name}
       /CT (\.cs{filemime:\.name})
       /P << /TF (TEMPALWAYS) >> % allow creating temporary files
     >>
     /P <<%/Type /MediaPlayParams
       /BE << /C \_kv{controls} /V \_kv{volume} /RC \_kv{repeat} >>
     >>
     /SP <<%/Type /MediaScreenParams
       /BE << /O \_kv{opacity} /B [\.bgcolor] >>
     >>
  >>}\_pdfrefobj\_pdflastobj
  \.redefinename{rendition}%
  \.sxdef{rendition:\.name}{\_the\_pdflastobj}%
  % screen annotation ("screen space allocation")
  \_pdfannot reserveobjnum% "self" reference will be needed inside screen annot.
  \.sxdef{screen:\.name}{\_the\_pdflastannot}%
  % aliases to latest rendition/screen with empty name
  \_global\.slet{rendition:}{rendition:\.name}%
  \_global\.slet{screen:}{screen:\.name}%
  \_edef\.action{\_kv{action}}\_edef\.aactions{\_kv{aactions}}%
  \.boxedannot[rendition:\.name]{#1}{useobjnum\_the\_pdflastannot}{%
     /Subtype /Screen
     % reference to page of the rendition (\setpageof done by \.boxedannot)
     % the spaces are weird, but \pdfpageref eats them
     /P \_pdfpageref\.pageof{rendition:\.name} \_space 0 R
     /A \_ea\.pdfaction\_ea[\.action]
     /AA \_ea\.pdfaactions\_ea{\.aactions}
  }%
  \_egroup\_fi
}
\_nspublic \render ;
```

Here are the defaults used for `\render` – `\.renderdefaults`. Users can redefine them all together or override as needed with key-value parameters. The defaults correspond to values specified by PDF standard. Other values may not be respected by all viewers.

pdfextra.opm

```
\_def\.renderdefaults{%
  name=\.name,
  controls=false,
  volume=100,
  repeat=1,
  opacity=1.0,
  background=1 1 1,
  action=rendition::play,
  aactions={},
}
```

Most probable use of additional actions is to start auto-start playing of the multimedium. For this purpose `\renditionautoplay` is defined as a shorthand for action to play the lastly defined rendition on page visible event.

pdfextra.opm

```
\_def\.renditionautoplay{{PV}{rendition::play}}
\_nspublic \renditionautoplay ;
```

Rendition actions

Rendition actions unfortunately use cryptic symbolic numbers (`0`, `1`, `2` and `3`) for actions that could be called `play`, `stop`, `pause` and `resume` respectively. Except for these predefined actions (that use `/OP`) running of JavaScript is possible using `/JS` (⟨*script*⟩) with potential fallback to `/OP`. This is dangerous teritory, because support of the right API in the viewer is very low. Although it is possible to define such action type by:

```
\.sdef{renditionaction:myaction}{/JS (app.alert("something useful");) /OP 0}
```

The use of rendition action is: `\.renditionaction[rendition:⟨name⟩:⟨action type⟩]`. Empty name refers to last rendition, so e.g.`\.renditionaction[rendition::pause]` is possible.

pdfextra.opm

```
\.sdef{renditionaction:play}{/OP 0}
\.sdef{renditionaction:stop}{/OP 1}
\.sdef{renditionaction:pause}{/OP 2}
\.sdef{renditionaction:resume}{/OP 3}
\_def\.renditionaction[#1:#2:#3]{/S /Rendition
   \.cs{renditionaction:#3}
   /R \.cs{rendition:#2} 0 R
   /AN \.cs{screen:#2} 0 R%
}
```

2.8.2 Rich Media (3D/audio/video)

Some principles seen with Renditions (section 2.8.1) apply here too. But additionally we deal with 3D specifics and unfortunate Flash leftovers.

Unlike Renditions both page area and multimedium specifaction are handled in a single annotation – the Rich Media annotation. The code is unfortunately obscured due to the weird requirements, but this is essentially what we are trying to create with `\RM`:

```
/Type /Annot
/Subtype /RichMedia
/RichMediaSettings <<
  /Activation <<
    /Condition /PV
    /Scripts [ 14 0 R ]
  >>
  /Deactivation << /Condition /XD >>
>>
/RichMediaContent <<
  /Assets << /Names [ (kladka.prc) 2 0 R (wireframe.js) 14 0 R ] >>
  /Configurations [ <<
      /Type /RichMediaConfiguration
      /Subtype /3D
      /Instances [ <<
          /Type /RichMediaInstance
          /Subtype /3D
          /Asset 2 0 R
        >> ]
    >> ]
>>
```

The activation/deactivation will be dealt with later. But we see that to insert a simple 3D file, we have to pack it inside a file specification (indirect reference to object `2 0 R`), then in "instance", inside a "configuration" inside "content". As if it wasn't enough the names (normally contained in the file specification) have to be specified again in `Assets` name tree that uselessly maps names to file specifications. Because this is a 3D Rich Media annotation there are other files at play – initialization scripts. These are specified in `/Scripts` and are executed in turn when the annotation is activated. Not shown is, that some "configurations" and "instances" actually have to be specified indirectly.

If it wasn't for Flash we could do with something like:

```
/Type /Annot
/Subtype /RichMedia
/Activation /PV
/Scripts [ 14 0 R ]
/Deactivation /XD
/Content 2 0 R
```

Which contains equivalent information. But unfortunately here we are...

pdfextra.opm

```
\.secondoptdef\.RM#1{\.isfiledefined{\.name}\_iftrue
      \_edef\.tmp{\.cs{filemime:\.name}}%
      \_edef\.subtype{\_ea\.mimetormsubtype\_ea[\.tmp]\_space}%
      \_ifx\.subtype\_space
         \_opwarning{unknown rich media type for '\.name', ignored}\_else
   \_bgroup
   \_ea\_readkv\_ea{\_ea\.RMdefaults\_ea,\_the\_opt}%
   % Instance that has the media file as an asset
   \_pdfobj {<</Type /RichMediaInstance
     /Subtype /\.subtype
     /Asset \.cs{filespec:\.name}
   >>}\_pdfrefobj\_pdflastobj
   % Configuration with one single instance (the above)
   \_pdfobj {<</Type /RichMediaConfiguration
     /Subtype /\.subtype
     /Instances [ \_the\_pdflastobj \_space 0 R ]
   >>}\_pdfrefobj\_pdflastobj \_edef\.configuration{\_the\_pdflastobj}%
   \_edef\.names{\.cs{filename:\.name} \.cs{filespec:\.name} }% initial asset
   \.redefinename{rm}%
   \_def\.scriptfilespecs{}%
   \_edef\.views{\_kv{views}}\_edef\.scripts{\_kv{scripts}}%
   \_ifx\.views\_empty \_edef\.views{\.name}\_fi
   \_ea\.DDDscripts\_ea{\.scripts}%
   % annotation in hbox
   \.boxedannot[rm:\.name]{#1}{}{%
      /Subtype /RichMedia
      /RichMediaSettings <<
        /Activation <<
          /Condition \.cs{activation:\_kv{activation}}
          \.emptyor{\.scriptfilespecs}{/Scripts [ \.nonempty ]}
          /Presentation << /Toolbar \_kv{toolbar} \.RMpresentationextra >>
        >>
        /Deactivation << /Condition \.cs{deactivation:\_kv{deactivation}} >>
      >>
      /RichMediaContent <<
        /Assets << /Names [ \.names ] >>
        /Configurations [ \.configuration \_space 0 R ]
        \.emptyor{\_ea\.DDDviews\_ea{\.views}}{/Views [ \.nonempty ]}
      >>
   }%
   \.sxdef{rm:\.name}{\_the\_pdflastannot}%
   \_global\.slet{rm:}{rm:\.name}%
   \_egroup\_fi\_fi
}
\_nspublic \RM ;
```

The code is similiar to `\render`, but we also ignore everything if we don't recognize the type of media (`Video`, `Sound` or `3D`). For that we use a simple mapping from MIME types with `\.mimetormsubtype`. This means that although we aim Rich Media mostly for 3D art it may also be used for Video and Sound.

pdfextra.opm

```
\_def\.mimetormsubtype[#1/#2]{\.cs{rmtype:#1}}

\.sdef{rmtype:model}{3D}
\.sdef{rmtype:video}{Video}
\.sdef{rmtype:audio}{Sound}
```

Then we also need to construct the weird name "tree" (essentialy an array in our case) and script array. `\.DDDscripts` and `\.DDDviews` do this. Name tree is accumulated in `\.names`, and starts with the media file. After that each script is added to `\.names` and `\.scriptfilespecs`. The scripts are passed as a comma separated array. Ignoring initial spaces is done using undelimited-delimited argument pair trick.

pdfextra.opm

```
\_def\.DDDscripts#1{\.DDDscriptsA#1,,,\.end}
\_def\.DDDscriptsA#1#2,{\_ifx,#1\_ea\.untilend\_else
   \.isfiledefined{#1#2}\_iftrue%
     \_addto\.scriptfilespecs{\.cs{filespec:#1#2} }%
     \_addto\.names{\.cs{filename:#1#2} \.cs{filespec:#1#2} }%
   \_fi
   \_ea\.DDDscriptsA\_fi
}
```

For 3D views we need to process yet another comma separated list, this time with \.DDDviews. The result has to be separated by spaces and we also don't want to emit something if the specified view was invalid. Unfortunately this is expansion only context, so we can't issue a warning.

As a user convenience, before \.DDDview is executed, view with the name of \.name is tried instead of empty view array. This means that for simple 3D art with one view, one can create view with the same name as the 3D object and not have to specify anything. We also take the name only after it is redefined from optional key-value parameters – this is so we can support even the case of e.g. screw 3D model used twice, once with name=screw1, another time with name=screw2 (with the corresponding screw1 and screw2 views). This is probably less useful, but...

pdfextra.opm

```
\_def\.DDDviews#1{\.DDDviewsA#1,,,\.end}
\_def\.DDDviewsA#1#2,{\_ifx,#1\_ea\.untilend\_else
   \.isdefined{3dview:#1#2}\_iftrue
      \_lastnamedcs\_space \_fi
   \_ea\.DDDviewsA\_fi
}
```

The activation/deactivation names are kind of cryptic, so we give them descriptive names. Default is explicit (de)activation. Instead of /PV (page visible) and /PI (page invisible) it would be possible to use "page open" and "page close". These are slightly different in cases when more pages are shown on screen at once, because only one page is "open", while multiple are "visible".

pdfextra.opm

```
\.sdef{activation:explicit}{/XA}
\.sdef{activation:auto}{/PV}
\.sdef{deactivation:explicit}{/XD}
\.sdef{deactivation:auto}{/PI}
```

Additional means of customization are here. \.RMdefaults contains the default key-value parameters. \.RMpresentationextra can be used to set more attributes in /RichMediaPresentation dictionary (although those are more specific and not generally useful).

pdfextra.opm

```
\_def\.RMdefaults{%
   name=\.name,
   activation=explicit,
   deactivation=explicit,
   toolbar=true,
   views=,
   scripts=,
}
\_def\.RMpresentationextra{}
```

For scripting using JavaScript actions one needs to access the 3D context of the 3D / Rich Media annotation. This requires the page number. We can't use this.pageNum from [TODO], because the script strictly doesn't have to be on the same page. We use \.pageof (\.setpageof was done in \.boxedannot) to retrieve the page number in next run. Also PDF indexes page numbers from 0. \DDDannot{⟨*name*⟩}. and \DDDcontext{⟨*name*⟩} allow this.

pdfextra.opm

```
\_def\.DDDannot#1{%
   this.getAnnotRichMedia(\_the\_numexpr\.pageof{rm:#1}-1\_relax, '#1')%
}
\_def\.DDDcontext#1{\.DDDannot{#1}.context3D}

\_nspublic \DDDannot \DDDcontext ;
```

2.8.3 3D views

This is the interesting part about 3D art. They can have a set of predefined views – although a user may start from one, they can interactively change all the aspects by dragging with mouse or messing with the settings shown by right click menu.

There are several transformations that have to be done before it is possible to display 3D scene on a computer screen:

1. 3D transformation from the coordinate system of 3D artwork ("model") to the "world coordinate system".
2. 3D transformation from the world coordinate system to camera coordinate system.
3. projection to 2D (3D to 2D transformation).

When talking about PDF, positive x goes to the right, positive y up, and positive z "away" from us ("into the page"). This means we are working with a left handed coordinate system. In camera space, the camera sits at $(0, 0, 0)$ facing towards positive z with positive x and y going right and up respectively. Projection (one way or another) discards the z coordinate.

Although the transformations are not strictly linear, they are essentially done using multiplication by *transformation matrices*. The matrix for "model to world" (or "model") transformation is part of the 3D art file and can't be changed. However, we can make it up, because we can fully control the second transformation ("world to camera" or "view" transformation) – although we don't specify the "world to camera" matrix but rather its inverse, the "camera to world" matrix (`/C2W`). This matrix has the 4×4 form, which also allows *linear transformation* and *translation*:

$$M_{c2w} = \begin{pmatrix} a & d & g & t_x \\ b & e & h & t_y \\ c & f & i & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Here we use the column major convention, which is also the order how we would write the matrix to PDF file, where it is an array of 12 elements:

`/C2W [`$a\ b\ c\ d\ e\ f\ g\ h\ i\ t_x\ t_y\ t_z$`]`

In the rendering pipeline everything is transformed from world coordinates to camera space coordinates. We can think about the process also in the other way. Using M_{c2w} we specify camera's position and orientation in the world coordinate system. Due to how transformation using matrix multiplication works, the first column in the M_{c2w} matrix (vector $(a, b, c)^T$) specifies how "positive x direcetion" ("right") from camera space ends up in world coordinate system. Similiarly for $(d, e, f)^T$ being the image of positive y ("up") and $(g, h, i)^T$ being the image of positive z ("forward"). The last column, $(t_x, t_y, t_z)^T$ represents translation from camera space to world. Translation of origin (camera position) will leave it in the point with coordinates (t_x, t_y, t_z). Because of these associations with the intuitive meanings of x, y, z in camera space we also sometimes call the vectors in the first three columns of M_{c2w} "right", "up" and "forward" and the last one "eye":

$$\vec{R} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}, \quad \vec{U} = \begin{pmatrix} d \\ e \\ f \end{pmatrix}, \quad \vec{F} = \begin{pmatrix} g \\ h \\ i \end{pmatrix}, \quad \vec{E} = \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix}.$$

We usually want $\vec{R}$, $\vec{U}$ and $\vec{F}$ to form an orthonormal set of vectors, i.e. all of unit length and each pair is orthogonal. The orthoganility will come from the way we calculate them, but the normality has to be ensured by normalizing the vectors after computing them, which will not be explicitly written out in the following text. $\vec{E}$ is a positional, not directional, vector and it's length will be preserved.

Now we only need a convenient way to calculate all four vectors. A wide spread method is sometimes called "look at". It essentially involves having two points: "eye" (E, position of the camera) and "target" (T, the point where the camera is pointing at). The camera position is already provided:

$$\vec{E} = E$$

From these two points alone we can easily calculate the forward vector, which corresponds to the direction of the camera:

$$\vec{F} = \overrightarrow{ET} = T - E$$

When we now imagine the point E and vector $\vec{F}$ looking towards T we can see that there is a degree of freedom – the camera can rotate about the forward vector. There is no other way than to arbitrarily choose either up or right vector. Usually we choose an arbitrary "global up" vector U_G, which will influence the general direction of the final up vector. This is because we use it to calculate the right vector:

$$\vec{R} = \vec{U}_G \times \vec{F}$$

The cross product makes it so that:

1. $\vec{R}$ is perpendicular to $\vec{F}$
2. it is also perpendicular to global up vector ($\vec{U}_G$) which we used to get rid of remaining degree of freedom.

Now that we have two orthonormal vectors (with the normalization not being explicit) we can calculate the remaining up vector:

$$\vec{U} = \vec{F} \times \vec{R}$$

The last mysterious part about the calculation are the cross products. They are of course not commutitative, so why e.g. $\vec{U}_G \times \vec{F}$ and not the other way around? This is because we have to preserve the relations these vectors had as directions of positive axes of the original camera space. There we had positive x going right, positive y up and positive z forward in a left handed coordinate system. This means that following holds (according the left hand rule):

$$\vec{R} = \vec{U} \times \vec{F}$$
$$\vec{U} = \vec{F} \times \vec{R}$$
$$\vec{F} = \vec{R} \times \vec{U}$$

The scheme has one flaw though. When the directions of global up vector $\vec{U}_G$ and forward vector $\vec{F}$ are linearly dependent the computed right vector will be $(0, 0, 0)$. Hence some handling of this special case is needed.

The "look at" method is essentially what is used in Alexander Grahn's package movie15[1]. Although the input aren't two points, but rather a "center of orbit" point (COO, our "target"), "center of orbit to camera vector" ($\overrightarrow{C2C}$, default is $(0, -1, 0)$) and distance of camera from the center of orbit (ROO). The value used for the arbitrary "global up" vector is $(0, 0, 1)$. When forward vector is $(0, 0, z)$, then global up is chosen to be $(0, -1, 0)$ or $(0, 1, 0)$ to handle the "0 right vector" issue.

Because the movie15 method of providing the parameters is used in essentially all packages that handle PDF 3D art (movie15, media9, rmannot, ConTEXt) we also follow the suite.

\DDDview[⟨*view name*⟩][⟨*key-value parameters*⟩] is the command for defining 3D views. These have to be saved into separate PDF objects anyways, using this interface we allow their reuse. If ⟨*view name*⟩ is same as ⟨*name*⟩ of \RM argument and no other views are specified ⟨*view name*⟩ view is automatically used (see \RM for details).

Key-value parameters are not optional this time, because rarely one suffices with default values – different 3D views are about customization. Handling of them is not as straightforward as before. We initially read the key-value parameters only to determine the `method` used for calculating the `/C2W` matrix. Then we reread key-value parameters again, this time with also with the default values for this particular method. Not that the general 3d views details are changed, but the methods themselves have key-value parameters of their own, and we support specifying them in this "flat" way.

Additionally we allow the different methods used to compute `/C2W` to not be expandable. Hence they are executed outside of expansion only context and are fully processed – the text they add to 3D view PDF object is temporarily stored in \.viewparams.

The rest is simply setting sensible defaults (or user overrides) for internal/external name of the view (`/IN` and `/XN`, one is used for scripting, one is shown by the PDF viewer), background color, rendering mode, and lighting. Cross sections and nodes are currently not supported, although users can hook in their own code using `\.DDDviewextra`, `\.DDDrendermodeextra` or `\.DDDprojectionextra`.

We have to be careful about setting rendering mode and lighting scheme, because they normally fall back to the values specified in 3D art file, which we can't access, so better not set them to anything if they are empty.

pdfextra.opm

```
\_def\.DDDview[#1][#2]{\_bgroup
  \_readkv{\.DDDviewdefaults,#2}%
  \_edef\.tmp{\.DDDviewdefaults,\.cs{3dview:\_kv{method}:defaults}}%
  \_readkv{\.tmp,#2}%
  \.colortorgbdef\.bgcolor{\_kv{background}}%
  \.cs{3dview:\_kv{method}}% sets \.viewparams (/MS, /C2W, /CO)
  \_pdfobj {<</Type /3DView
    /XN (#1)
    /IN (#1)
    \.viewparams % /MS, /C2W, /CO
    /P <<
      \.cs{3dprojection:\_kv{projection}}
      \.DDDprojectionextra
```

[1] https://www.ctan.org/pkg/movie15

```
      >>
      /BG <<%/Type /3DBG
        /Subtype /SC
        /C [\.bgcolor]
      >>
      \.emptyor{\_kv{rendermode}}{%
        /RM <<%/Type /3DRenderMode
          /Subtype /\.nonempty
          \.DDDrendermodeextra >> }%
      \.emptyor{\_kv{lighting}}{%
        /LS <<%/Type /3DLightingScheme
          /Subtype /\.nonempty >> }%
    >>}%
    \_pdfrefobj\_pdflastobj
    \.sxdef{3dview:#1}{\_the\_pdflastobj \_space 0 R}%
    \_egroup
}

\_nspublic \DDDview ;
```

\.DDDviewdefaults stores default key-value parameters for 3D views. They are mostly the PDF standard defaults or what movie15/media9 uses (for compatibility).\.DDDviewextra, \.DDDrendermodeextra or \.DDDprojectionextra can be used by the users to hook themself into 3D view object creation.

pdfextra.opm

```
\_def\.DDDviewdefaults{
  projection=perspective,
  scale=1,
  ps=Min,
  FOV=30,
  background=1 1 1,
  rendermode=,
  lighting=,
  method=media9,
}
\_def\.DDDprojectionextra{}
\_def\.DDDrendermodeextra{}
\_def\.DDDviewextra{}
```

There are two different projection methods:

- Orthographic: z coordinate is simply thrown away, `scale` is used for scaling the result. For technical parts where we want lines that are parallel stay parallel in the view.
- Perspective: is the way human eye sees. `FOV` can be used to set field of view. (`ps` parameter for additonal scaling to fit width/height is also available, but the default is fine for casual users).

pdfextra.opm

```
\.sdef{3dprojection:ortho}{/Subtype /O /OS \_kv{scale}}
\.sdef{3dprojection:perspective}{/Subtype /P /FOV \_kv{FOV} /PS /\_kv{ps}}
```

We offer the possibility of setting `/C2W` matrix and `/CO` (distance from camera to center of orbit) directly using `method=manual`.

pdfextra.opm

```
\.sdef{3dview:manual:defaults}{
  matrix=1 0 0 0 1 0 0 0 1 0 0 0 ,
  centeroforbit=0,
}
\.sdef{3dview:manual}{\_edef\.viewparams{
  /MS /M
  /C2W [\_kv{matrix}]
  /CO \_kv{centeroforbit}
}}
```

Another simple way of specifying camera position/orientation is to use a named setting of U3D file using a U3D path with `method=u3d`.

pdfextra.opm

```
\.sdef{3dview:u3d:defaults}{
  u3dpath=,
}
\.sdef{3dview:u3d}{%
```

```
  \_pdfunidef\.tmp{\_kv{u3dpath}}%
  \_edef\.viewparams{
  /MS /U3D
  /U3DPath \.tmp \_space
}}
```

The most advanced method of setting `/C2W` matrix and `/CO` is `method=media9`. It is thoroughly explained above, the few differences are because the input values are not two points. Also for conciseness "x", "y" and "z" are used instead of right, up and forward. The calculations are done in Lua, for simplicity.

We expect the user to supply the numbers in the form "`1 2 3`", but in Lua we need them comma separated ("1, 2, 3"). `\.luatriplet` does this. Just in case the code is somehow adapted without ensuring that x and z are orthonormal, we normalize also y after the second cross product.

pdfextra.opm

```
\_def\.luatriplet#1 #2 #3 {#1, #2, #3}

\.sdef{3dview:media9:defaults}{%
  roo=0,
  coo=0 0 0,
  c2c=0 -1 0,
}
\.sdef{3dview:media9}{\_edef\.coo{\_kv{coo}}\_edef\.c2c{\_kv{c2c}}\_edef\.viewparams{
  /MS /M
  /C2W [\_directlua{
  local function normalize(x, y, z)
      local len = math.sqrt(x*x + y*y + z*z)
      if len ~= 0 then return x/len, y/len, z/len else return 0, 0, 0 end
  end
  local function cross(ux, uy, uz, vx, vy, vz)
      return uy*vz - uz*vy, uz*vx - ux*vz, ux*vy - uy*vx
  end
  local function printmat(...)
      local arr = table.pack(...)
      for k, v in ipairs(arr) do
          arr[k] = string.format("\_pcent.6f", v)
      end
      tex.print(table.concat(arr, " "))
  end

  local roo = \_kv{roo}
  local coo_x, coo_y, coo_z = \_ea\.luatriplet\_expanded{\_kv{coo}}
  local c2c_x, c2c_y, c2c_z = normalize(\_ea\.luatriplet\_expanded{\_kv{c2c}})

  local eye_x, eye_y, eye_z = coo_x + c2c_x*roo, coo_y + c2c_y*roo, coo_z + c2c_z*roo

  local z_x, z_y, z_z = -c2c_x, -c2c_y, -c2c_z

  local up_x, up_y, up_z = 0, 0, 1
  if math.abs(z_x) + math.abs(z_y) < 0.0000001 then % z_x == 0 and z_y == 0
      if z_z < 0.0000001 then % z_z <= 0
          up_x, up_y, up_z = 0, 1, 0
      else
          up_x, up_y, up_z = 0, -1, 0
      end
  end

  local x_x, x_y, x_z = normalize(cross(up_x, up_y, up_z, z_x, z_y, z_z))
  local y_x, y_y, y_z = normalize(cross(z_x, z_y, z_z, x_x, x_y, x_z))

  local eye_x, eye_y, eye_z = coo_x - z_x*roo, coo_y - z_y*roo, coo_z - z_z*roo

  printmat(x_x, x_y, x_z, y_x, y_y, y_z, z_x, z_y, z_z, eye_x, eye_y, eye_z)
  }]
  /CO \_kv{roo}
}}
```

Last, but not least, is an action for setting the 3D view of a 3D/RM annotation using an action. `\.goto3dviewaction[goto3dview:`⟨*name*⟩`:`⟨*view*⟩`]`. ⟨*name*⟩ is name of the annotation which will be influenced. ⟨*view*⟩ is passed directly to PDF. Therefore it can be either an index to the view array (starting at 0) or name of view in parentheses – "`(`⟨*view name*⟩`)`".

pdfextra.opm

```
\.sdef{goto3dviewaction}[#1:#2:#3]{/S /GoTo3DView
   /TA \.cs{rm:#2} 0 R
   /V #3
}
```

2.9 MIME type database

This is the uninteresting MIME type database teased in section 2.3. Ideally this would only be a subset of what IANA defines at https://www.iana.org/assignments/media-types/media-types.xhtml. But there are additions like `model/u3d` and `model/prc`, which don't seem to be official, yet. Other "unofficial" MIME types are taken from Mozilla's "common" lists:

- https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types.
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types/Common_types

`\.mimetype{`⟨*extension*⟩`}{`⟨*MIME type*⟩`}` is a shortcut of mapping ⟨*extension*⟩ to ⟨*MIME type*⟩.

pdfextra.opm

```
\_def\.mimetype#1#2{\_sdef{_pdfextra_mimetype:#1}{#2}}

\.mimetype{js}{application/javascript}
\.mimetype{pdf}{application/pdf}

\.mimetype{prc}{model/prc}
\.mimetype{u3d}{model/u3d}

\.mimetype{wav}{audio/x-wav}
\.mimetype{mp3}{audio/mpeg}
\.mimetype{opus}{audio/opus}

\.mimetype{avi}{video/x-msvideo}
\.mimetype{mp4}{video/mp4}
\.mimetype{webm}{video/webm}

\_endnamespace
```