PŘÍRODOVĚDECKÁ FAKULTA UNIVERZITY PALACKÉHO
KATEDRA INFORMATIKY

BAKALÁŘSKÁ PRÁCE

Průzkum současného návrhu HTML5

2010 Martin Bargl

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval samostatně.

18. květen 2010 Martin Bargl

Anotace

HTML5 je další velkou revizí jazyka HTML, odstraňující jeho nedostatky. Tvůrcům dokumentů dává striktní pravidla pro psaní a nové nástroje. Jejich úlohou je vylepšit současnou úroveň dokumentů publikovaných v rámci sítě Internet. Práce obsahuje úvod do problematiky jazyka HTML a hlavní rozdíly mezi návrhem HTML5 a současným standardem. Naleznete zde popis nových elementů s podporou prohlížečů a jejich aplikačního rozhraní. Důležitou část tvoří canvas element, ke kterému byl vytvořen online tutoriál. Na závěr jsou zde informace o dalších připravovaných změnách. Celá práce je vztažena k návrhu HTML5 z data 2.12.2009.

Rád bych poděkoval Janu Konečnému za jeho cenné rady a vedení při tvorbě bakalářské práce. Dále děkuji Mgr. Josefu Štefanovi za veškerou pomoc a odborné rady, které byly pro dokončení a vznik této práce stěžejní. Také bych chtěl poděkovat své přítelkyni Nele Eršilové a své rodině za jejich podporu a pochopení v průběhu příprav této bakalářské práce.

Obsah

Seznam obrázků

Seznam tabulek

1. Historické pozadí a vývoj HTML

Tato kapitola je úvodem ke značkovacímu jazyku *HTML*. Popisuje podmínky a důvody jeho vzniku. Také se zde zaměřuji na vývoj jazyka od počátků až po současnost. Jelikož tematika HTML a ostatních záležitostí s ním spojených je velmi rozsáhlá, některé důležité pojmy budou vysvětleny v 2. kapitole. Avšak je mimo rozsah této bakalářské práce rozebírat veškeré znalosti jazyka HTML, proto zájemce může najít další informace v [3].

1.1. Internet

V dnešní době je slovo *Internet* trendem, který zachvátil celý svět. Skutečností však je, že většina lidí za tímto pojmem vidí pouze webový prohlížeč, díky kterému si mohou prohlížet webové stránky. Tito lidé však netuší, co přesně se za pojmem Internet skrývá.

Internet je systém vzájemně propojených počítačových sítí. V těchto sítích dochází ke komunikaci počítačů za pomoci rodiny protokolů *TCP/IP*. Jejich účelem je zajištění bezproblémové výměny dat mezi počítači.

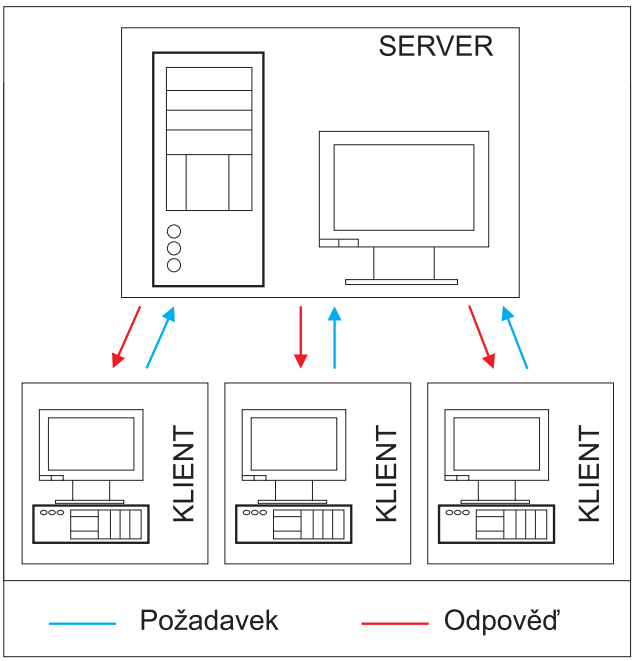
První zmínka o počítačové síti byla již roku 1946 v článku *As We May Think*, jehož autorem je *Vannevar Bush*. Tímto článkem předběhl svou dobu, jelikož první síť zvaná *ARPANET* vzniká až roku 1969. Avšak pojem Internet byl použit teprve roku 1987, kdy síť zahrnovala, v té době ohromných, 27 tisíc počítačů.

Internet sloužil k datové komunikaci, nicméně jeho možnosti měly své nedostatky. Hlavním problémem byla složitá struktura dokumentů určených k publikaci. Nedostatkem byla také nutnost použití hesla pro přístup k dokumentům, které se nacházely na jiném počítači v síti.

1.2. Projekt WWW

V březnu roku 1989 vytvořil *Tim Berners-Lee* návrh, který zmiňoval databázový a softwarový projekt z roku 1980. Jednalo se o systém pro správu informací zvaný *ENQUIRE*. To byl hypertextový program, který umožňoval vědcům uchovat záznamy o lidech, softwaru a projektech, za použití hypertextových odkazů.

Spolu s kolegou, jménem *Robert Caillau*, publikoval Tim Berners-Lee nový formální návrh k vytvoření hypertextového projektu, jenž by sloužil jako síť hypertextových dokumentů. Tento projekt nazval *World Wide Web* a jeho hlavní myšlenkou bylo prohlížení elektronických dokumentů pomocí Internetové sítě. Výhodou měla být skutečnost, že se uživatel nebude nemuset přihlašovat heslem k jinému počítači v síti jen proto, aby si mohl prohlížet hypertextové dokumenty. Za tímto účelem využili architekturu typu klient-server.



Obrázek 1. Architektura klient-server

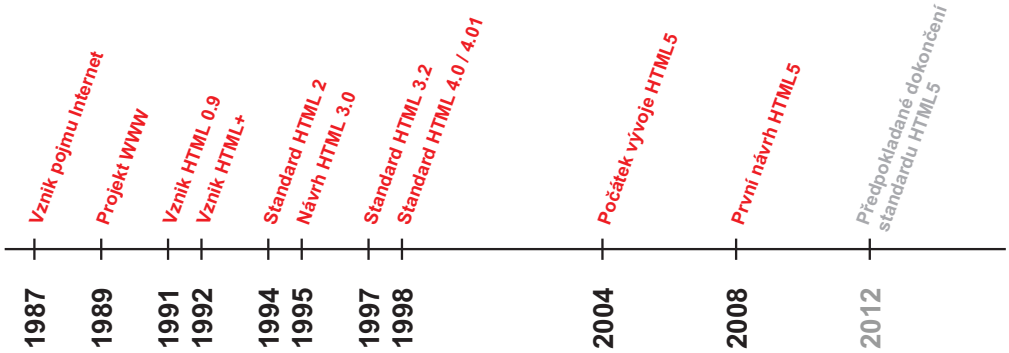
1.3. Vznik HTML

V době okolo roku 1989 byl pro tvorbu hypertextových dokumentů používán *TEX*, *PostScript*, nebo jazyk *SGML (Standard Generalized Markup Language)*. Problém tkvěl ve složitosti tvorby těchto dokumentů.

To vedlo k tomu, že Tim Berners-Lee začal přemýšlet nad jednodušším způsobem jejich tvorby a navrhl jazyk HTML *(HyperText Markup Language)*. Ten vytvořil roku 1991 s pomocí jazyka SGML. Pro přenos jazyka HTML po Internetové síti naprogramoval Tim Berners-Lee protokol *HTTP (HyperText Transfer Protokol)*. Dokumenty vytvořené s využitím jazyka HTML mohly být, díky protokolu HTTP, prohlíženy za pomoci prohlížeče World Wide Web.

1.4. Vývoj jazyka HTML

Nápad s HTML byl převratný a Tim Berners-Lee pracoval pro organizaci *CERN (European Organization of Nuclear Research)*. Není tedy divu, že roku 1991 organizace zprovoznila svůj první web. Stejného roku organizace *NCSA (National Centrum of Supercomputer Applications)* zadala projekt, jehož cílem bylo vytvoření prohlížeče *Mosaic*. Tyto skutečnosti vedly k nutnosti sjednocení možností jazyka HTML a následné tvorbě standardů jazyka.



Obrázek 2. Časové znázornění vývoje jazyka HTML

Verze HTML 0.9 a HTML+

Verze *HTML 0.9* byla první definicí jazyka, kterou Tim Berners-Lee vytvořil okolo roku 1991. Verze nebyla a není považována za standard HTML, nicméně umožňovala sazbu textu, hypertextových odkazů, základní zvýraznění textu a vkládání jednoduchých obrázků. Následně *Dave Radggett*, z laboratoří *Hewlet-Packard*, vytvořil verzi *HTML+*. Ta byla revizí jazyka HTML 0.9 obohacenou o tabulky, seznamy, nadpisy a odstavce.

Standard HTML 2.0

Počátkem roku 1994 bylo HTML tak rozšířené, že většina prohlížečů začala přidávat do jazyka svou vlastní funkcionalitu. To vedlo k tomu, že *Dan Connolly* a jeho kolegové shromáždili nejrozšířenější a nejpoužívanější tagy té doby. Ty následně zahrnuli do návrhu dokumentu, který Tim Berners-Lee nazval *HTML 2.0*. Tento návrh byl schválen a přijat za standard komisí *IETF (Internet Engineering Task Force)*.

Verze HTML 3.0, standard HTML 3.2

Během roku 1995 bylo HTML postupně rozšiřováno o mnoho nových tagů. Mezi novinky patřily například kaskádové styly, pokročilé možnosti práce s tabulkami, formuláře a atribut *class*. Tyto novinky, spolu s dalšími, byly zahrnuty do návrhu *HTML 3.0*. Ten byl komisí IETF odmítnut, a to pro svou robustnost. Až roku 1997 byl vytvořen standard *HTML 3.2*, který sjednotil stávající standard HTML 2.0 s vlastnostmi HTML 0.9, HTML+ a HTML 3.0.

Standard HTML 4.0 a HTML 4.01

Jelikož vývoj šel stále dopředu, není překvapením, že roku 1997 byl vytvořen návrh verze *HTML 4.0*. Ten se zaměřil na odmítnutý návrh verze HTML 3.0. Vzal z něj pouze to nejlepší a soustředil se na nový podpůrný jazyk, kaskádové styly. Dále přibyly nové požadavky na již existující elementy, např. alternativní text pro element *img*. Návrh verze HTML 4.0 se stal standardem roku 1998 a podpora prohlížečů pro tento nový standard byla vytvořena velmi rychle. Chvíli poté, co bylo HTML 4.0 přijato za standard, byl dokument přepracován a opraven. Jednalo se pouze o drobné opravy, a proto byla tato specifikace nazvána standardem *HTML 4.01*.

Návrh verze HTML5

Jelikož i verze HTML 4.01 má nedostatky, např. v sémantice, tak v červnu roku 2004 začala skupina *WHATWG (Web Hypertext Application Technology Working Group)* pracovat na nové specifikaci. Roku 2007 se zapojila organizace *W3C (World Wide Web Consortium)* do vývoje a 22. ledna 2008 vydali první návrh verze *HTML5*. Od té doby byl návrh několikrát přepracován a stále je ve fázi přípravy. Nicméně některé z funkcionalit návrhu HTML5 jsou již dnešními prohlížeči podporovány. Tento návrh je hlavním tématem této bakalářské práce a bude podrobněji prozkoumán v následujících kapitolách. Zájemce o bližší informace odkazuji na [1].

1.5. Další technologie

V současnosti zažívá tvorba webových stránek expanzi a nároky jsou tak vysoké, že bychom si s použitím pouhého HTML nevystačili. Mezi nejpoužívanější technologie při tvorbě webových stránek patří spolu s HTML také *CSS*, *Javascript*, *AJAX*, *PHP*, *ASP.NET* a další.

Kaskádové styly

Každý dokument, ať už psaný, nebo elektronický, má svůj obsah a formu. Forma, neboli formát, nám říká, jak bude stránka vypadat. CSS *(Cascade Style Sheets)* je soubor metod pro grafickou úpravu HTML dokumentů. Pomocí těchto stylů můžeme ovlivnit písmo, obrázky, grafickou podobu dokumentu, odstavce a další elementy jazyka HTML. CSS můžeme psát přímo do tagů pomocí atributu *style*. Také do hlavičky HTML dokumentu za pomoci elementu style, nebo do externího souboru s příponou css *(např. style.css)*.

XML, XHTML

XML (Extensible Markup Language) je obecný značkovací jazyk. Jedná se o zjednodušenou verzi jazyka SGML, která umožňuje snadné vytváření značkovacích jazyků.

Jazyk *XHTML (Extensible HTML)* vznikl proto, aby jazyk HTML vyhovoval podmínkám tvorby XML dokumentů. Důležité je zde zachování zpětné kompatibility s HTML. Dokumenty typu XHTML musí být napsány korektně bez chyb, jinak nebudou správně interpretovány, na rozdíl od čistého HTML. Tato skutečnost zvyšuje kvalitu a přehlednost zdrojového kódu.

Javascript, AJAX

Javascript je multiplatformní objektově orientovaný skriptovací jazyk. Používá se jako interpretovaný programovací jazyk při tvorbě webových stránek. Využívá se především k ovládání uživatelského rozhraní stránky, animacím a různým efektům. Jeho využití je vhodné zejména pro reakce na události vyvolané uživatelem.

AJAX *(Asynchronous Javascript and XML)* je technologie pro vývoj interaktivních webových aplikací, jenž mění svůj obsah bez potřeby opětovného načtení celé stránky. To přináší uživateli větší komfort a rychlejší odezvu na jeho akce.

PHP

PHP *(Hypertext Preprocessor)* je skriptovací programovací jazyk pro tvorbu serverově orientovaných internetových stránek. PHP skripty jsou prováděny na straně serveru, přičemž uživatel může vidět pouze výsledek jejich činnosti. Velké využití má PHP ve spolupráci s databázemi, kdy je používáno pro tvorbu redakčních systémů, Internetových obchodů, atd. Zkratka jazyka vznikla z původního názvu *Personal Home Page*.

ASP.NET

ASP.NET je také skriptovací programovací jazyk pro tvorbu serverově orientovaných internetových stránek, který je součástí *.Net Framework*. Je založen na *CLR (Common Language Runtime)*, čímž umožňuje programátorům vytvářet projekty v jakémkoli jazyce podporujícím CLR. ASP.NET je zpracováno na straně serveru, přičemž ke klientovi se dostane pouze výsledek daného skriptu. Jeho využití je podobné jako u jazyka PHP.

Flash a ActionScript

Flash je program pro jednoduchou tvorbu animací, prezentací a webových stránek. Obrovskou výhodou je využití vektorové grafiky. To ve spojení s *ActionScriptem*, což je programovací jazyk podobný jazyku JavaScript, výrazně zvyšuje interaktivnost animací. Nevýhodou při použití této technologie je velké zatížení procesoru a nutnost mít nainstalovaný plug-in, který umožňuje zobrazení flashové animace. Flash je v dnešní době hojně využíván pro tvorbu online her, aplikací a přehrávání videa na Internetu.

DOM

DOM (Document Object Model) je objektově orientovaná reprezentace XML, nebo HTML dokumentu. Jde o aplikační rozhraní umožňující přístup a modifikaci struktury, obsahu, nebo formátu dokumentu.

1.6. Doplňující literatura

Jelikož není v rozsahu této bakalářské práce probrat podrobně všechny technologie, zájemcům doporučuji literaturu v tab. č. 1.

Téma	**Název knihy**
HTML	HTML: The Definitive Guide, *Ch. Musciano, B. Kennedy*, O'Reilly Media, ISBN: 1-56592-235-2, 2007
CSS	CSS: The Definitive Guide, *E. A. Meyer*, O'Reilly Media, ISBN: 0-596-52733-0, 2007
JavaScript	JavaScript: Programujeme Internetové aplikace, *R. Škultéty*, Computer Press, ISBN: 80-251-0144-4, 2004
Ajax	Ajax Profesionálně, *N. C. Zakas, J. McPeak, J. Fawcett*, Zoner Press, ISBN: 978-80-86815-77-0, 2007
PHP	Mistrovství v PHP5, *A. Gutsman, S. S. Bakken, D. Rethans*, Computer Press, ISBN: 978-80-251-1519-0, 2008
ASP.NET	ASP.NET 3.5, *B. Evjen, S. Hanselman, D. Rader*, Computer Press, ISBN: 978-80-251-2069-9, 2009

Tabulka 1. Přehled doplňující literatury.

2. Seznámení s HTML

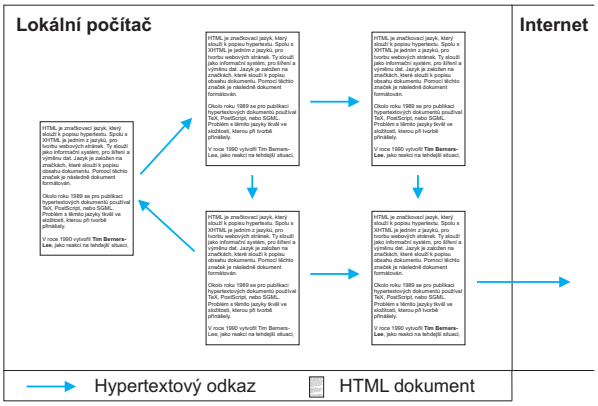
Jazyk HTML a jeho použití je téma rozsáhlé, proto se v této kapitole zabývám pouze nejčastěji používanými pojmy a základní strukturou dokumentu HTML. Pro přesnější informace o práci s jazykem HTML odkazuji zájemce na literaturu uvedenou v tabulce č. 1., popřípadě na [3], [4], nebo [6].

2.1. Pojmy jazyka HTML

V této podkapitole vám přiblížím nejdůležitější pojmy, které jsou v této bakalářské práci použity.

Hypertext

Hypertext je soubor textových dokumentů, které se odkazují na jiné dokumenty, nebo zdroje informací. Hypertextové odkazy určují přesnou adresu odkazovaného materiálu v síti, pomocí které můžeme k těmto dokumentům přímo přistupovat.



Obrázek 3. Propojení dokumentů pomocí hypertextových odkazů

Element a tag jazyka HTML

Jelikož HTML je značkovací jazyk, jedná se o kombinaci značek a textu. Těmto značkám se říká *tagy*. V HTML se tagy dělí na párové a nepárové. Každý tag je zapsán do ostrých závorek, přičemž párové tagy se skládají z otevíracího a uzavíracího tagu. Uzavírací tag obsahuje, kromě ostrých závorek, ještě lomítko za první závorkou. Většina tagů má své atributy a obsah. Za obsah tagu je považován

veškerý text, nebo jiné tagy, které se nachází mezi otevírací a uzavírací značkou tagu.

Element je základním prvkem jazyka HTML. Elementem nazýváme každý prvek jazyka HTML, který je v něm definován. Základním elementem je potom tzv. *HTMLElement* z nějž jsou ostatní elementy odvozeny. Element je v dokumentu zapsán pomocí tagu, který jej reprezentuje. Např. element *HTMLImageElement* je v dokumentu zapsán tagem *<img>*.

Syntaxe 1. *Zápis nepárového tagu: <název_tagu / >*

Např. <br / >

Syntaxe 2. *Zápis párového tagu: <počáteční_tag> obsah < /ukončovací_tag>*

Např. text < /div>

Atribut

Atribut nastavuje vlastnosti elementu a je zapsán vně tagu. Každý atribut má hodnotu, která je uvedena za rovnítkem a uzavřena v uvozovkách. Různým atributům jsou přiřazovány odlišné hodnoty. Těmi mohou být čísla, řetězce, popřípadě funkce skriptovacích jazyků.

Syntaxe 3. *<název_tagu název_atributu="hodnota_atributu">*

Např. <img src="smile.jpg" / >

Definice typu dokumentu

Definice typu dokumentu je množina deklarací, popisující strukturu dokumentu. Informuje o verzi a typu dokumentu, čímž specifikuje tagy a atributy, které může dokument používat. Říká prohlížeči, jakým způsobem obsah dokumentu zpracovat a zobrazit uživateli. Definice typu dokumentu je označována zkratkou *DTD(Document Type Definition)*. Bližší informace pro zájemce naleznete v [5].

Definice je v dokumentu umístěna na začátku, ještě před tagem HTML. Je popsána pomocí *DOCTYPE*, což je instrukce pro prohlížeč, nikoli element jazyka HTML. Více informací o používaných definicích naleznete v [5].

Syntaxe 4. *<!DOCTYPE kořenový_element přístup SYSTEM "[URL]" >*

Např. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

Meta tagy

Metadata jsou strukturovaná data o datech. Jejich použití přináší dodatečné informace o stránkách a lepší organizační strukturu stránek. Stránky ve formátu HTML dokumentu mají metadata uvedena v hlavičce, zapsána pomocí elementu *meta*. Ten vlastní tři atributy, a to spojení atributů *name* a *content*, nebo *http-equiv* a *content*. První kombinace se používá k zadání informací pro vyhledávače. Druhá kombinace pracuje s HTTP hlavičkami, sděluje prohlížeči potřebné informace ke zpracování stránky. Více informací na [4].

Syntaxe 5. *<meta name="název_metadat" content="hodnota">*

Např. <meta name="author" content="Martin Bargl" / >

Syntaxe 6. *<meta http-equiv="název_metadat" content="hodnota" / >*

Např. <meta http-equiv="Content-Type" content="text/html; charset=windows-1250" / >

2.2. Ostatní používané pojmy

Při tvorbě webových stránek se setkáte s pojmy, které s jazykem HTML nepřímo souvisí. Pojmy, které budou zmíněny v této bakalářské práci, uvádím v této podkapitole.

Aplikační rozhraní

API (Application Interface) je soubor funkcí, procedur a tříd, které lze při programování využívat. A to pomocí přídavných knihoven, jádra operačního systému, nebo jádra programu.

HTTP Cache

Cache je vyhrazené místo na disku, složka, kam se stahují a ukládají položky. Těmi jsou obrázky, tlačítka, ikony, bannery, fotografie a i celé webové stránky. Uložené soubory následně slouží ke zrychlení přístupu k webovým stránkám a aplikacím. Tohoto zrychlení je dosaženo tím, že ve chvíli, kdy uživatel otevře webovou stránku, zkontroluje prohlížeč, zda-li data stránky nemá již uložena v paměti cache. Pokud ano, načte data z paměti cache a nemusí je stahovat ze serveru. Pokud data v paměti cache nejsou, dojde k jejich stažení a uložení. Velikost této paměti lze nastavit v prohlížeči, je-li tato paměť plná, dojde k přepsání nejstarších uložených dat.

Události HTML

Od verze HTML 4.0 se v HTML vyskytují události v podobě atributů. Nicméně bez použití skriptovacího jazyka, který by události obsloužil, nemají žádnou funkci. Avšak ve spojení s javascriptem se stávají prostředkem, který slouží k interakci uživatele s webovou stránkou. Kód, který tuto interakci umožňuje je vykonán až tehdy, dojde-li k vyvolání události uživatelem. V současné době jsou v HTML události pro okno prohlížeče, formuláře, klávesnici a myš. Bližší informace pro zájemce na [6].

2.3. Základní struktura HTML dokumentu

Každý zdrojový kód, který má tvořit webovou stránku, musí obsahovat definici typu dokumentu. Poté musí následovat párový tag <*html*>, který vyznačuje HTML dokument. Uvnitř tohoto tagu se musí nacházet další dva tagy, které jsou párové a vymezují hlavičku a tělo stránky.

Obrázek 4. Základní struktura dokumentu

Prvním z nich je tag <*head*>, což je hlavička dokumentu. Ta se v prohlížeči nezobrazuje. Do hlavičky se zapisují meta tagy, titulek stránky, stylopis a jiné. Druhou část tvoří tzv. tělo, tag <*body*>. Ten zahrnuje veškerý obsah, který má být zobrazen prohlížečem. Tento tag má své atributy, nicméně ty jsou v dnešní době nahrazovány pomocí CSS. Do těla se píše text, vkládají obrázky, tabulky, seznamy, formuláře a jiné elementy. Zájemce o přesnější informace odkazuji na [4].

K formátování obsahu se následně používají kaskádové styly, určené pomocí atributů *id*, *class*, nebo samotným názvem tagu. Nejpoužívanějším tagem pro strukturování obsahu do bloků, je tag <*div*>. V dnešní době je používán

k vytvoření grafické podoby stránky. Toho dosáhneme pozicováním div bloků za pomoci CSS. Tyto bloky následně vytváří logické celky stránky. Například část s hlavním obsahem, záhlaví a zápatí, navigační menu stránky atd. Další používanou metodou je pozicování logických celků stránky pomocí tabulek. Tato metoda se nedoporučuje, jelikož tabulky nebyly navrženy pro tento účel. Poslední možností je použití rámů, což je element *frame*. Tato volba se však již delší dobu předními vývojáři nepoužívá, a to kvůli svým nedostatkům. Pro bližší informace k používaným a nepoužívaným tagům odkazuji zájemce na [4].

Zdrojový kód 1. *Základní struktura dokumentu*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>

  <head>
    <title>
      Titulek stránky
    </title>
  </head>

  <body>
    <div id="hlavicka">
            <img src="logo.jpg" alt="Logo stránek" />
    </div>

    <div id="hlavni_menu">
            <a href="index.html" title="Hlavní strana" target="_self">
                    Hlavní strana
            </a>
            <a href="contact.html" title="Kontaktní strana" target="_self">
                    Kontaktní sekce
            </a>
    </div>

    <div id="obsah_stranky">
            Hlavní obsah dokumentu
    </div>

    <div id="paticka">
            Název a rok vzniku webu
    </div>
  </body>

</html>
```

3. Hlavní rozdíly HTML5 oproti současným standardům

Verze 5 je další velkou revizí jazyka HTML, která by měla být novým standardem. Ten má nahradit současné HTML 4.01 a XHTML 1.0. Dále rozšíří možnosti *DOM Level 2 HTML* pomocí nových elementů jazyka a úprav již existujících. V současné době je standard ve stádiu přípravy. Z toho důvodu jsou veškeré texty této bakalářské práce vztaženy k návrhu [1] z 2.12.2009.

3.1. Doctype

Každý tvůrce webových stránek má dnes právo volby. Může zvolit, zda-li dokument bude psát s použitím syntaxe HTML, nebo XHTML. Zadáním definice typu dokumentu následně volí režim zobrazení v prohlížeči. Ten určuje, jak bude (X)HTML dokument prohlížečem interpretován a následně zobrazen. *Doctype* používaný aktuálně pro jazyk HTML můžete najít v tab. č. 2. a pro XHTML v tab. č. 3.

Doctype	**Pravidla pro dokument**
HTML 4.01 Strict	Dokument může obsahovat veškeré tagy a atributy HTML s výjimkou prezentačních tagů, nebo tagů označených jako "deprecated" (*tagy, jejichž používání je nevhodné*). V dokumentu je zakázáno používání rámů.
HTML 4.01 Transitional	Dokument může obsahovat veškeré tagy a atributy HTML. Také prezentační tagy a tagy označené jako "deprecated" (*tagy, jejichž používání je nevhodné*). V dokumentu je zakázáno používání rámů.
HTML 4.01 Frameset	Dokument může obsahovat veškeré tagy a atributy HTML. Také prezentační tagy a tagy označené jako "deprecated" (*tagy, jejichž používání je nevhodné*). V dokumentu je povoleno používat rámy.

Tabulka 2. Možnosti doctype pro HTML dokument

Doctype	Pravidla pro dokument
XHTML 4.01 Strict	Dokument může obsahovat veškeré tagy a atributy HTML s výjimkou prezentačních tagů, nebo tagů označených jako "deprecated" (*tagy, jejichž používání je nevhodné*). V dokumentu je zakázáno používání rámů. V dokumentu musí být správný zápis syntaxe jazyka *XML*.
XHTML 4.01 Transitional	Dokument může obsahovat veškeré tagy a atributy HTML. Také prezentační tagy a tagy označené jako "deprecated" (*tagy, jejichž používání je nevhodné*). V dokumentu je zakázáno používání rámů. V dokumentu musí být správný zápis syntaxe jazyka *XML*.
XHTML 4.01 Frameset	Dokument může obsahovat veškeré tagy a atributy HTML. Také prezentační tagy a tagy označené jako "deprecated" (*tagy, jejichž používání je nevhodné*). V dokumentu je povoleno používat rámy. V dokumentu musí být správný zápis syntaxe jazyka *XML*.

Tabulka 3. Možnosti doctype pro XHTML dokument

Pokud použijete některý z uvedených doctype a dokument nebude dle definované syntaxe, měl by být označen za chybný. Chybně napsaný dokument by neměl být prohlížečem zobrazen. Důvodem je, že by mohl být zobrazen odlišně v závislosti na zvoleném doctype a prohlížeči. Častou příčinou chyb ve zdrojovém kódu je vynechání ukončovacího tagu. Další chybou bývá vynechání jedné z uvozovek, které vymezují hodnotu atributu. Navíc není u současných standardů použití doctype povinné.

Oproti tomuto přístupu požaduje HTML5 kratší formu doctype, která musí být uvedena povinně. Doctype pro HTML5 požaduje pouze upřesnění, jakého typu dokument je. Tedy, je-li dokument typu HTML, nebo XHTML. Ze zápisu je vynechána reference na přesnou definici typu dokumentu. Také zde nenaleznete režim, pod kterým prohlížeč zdrojový kód interpretuje. Důvodem je skutečnost, že všechny dokumenty psané v HTML5 jsou vždy interpretovány striktně.

Syntaxe 7. *<!DOCTYPE kořenový_element >*

Např. <!DOCTYPE html >

3.2. Syntaxe jazyka HTML5

Syntaxe jazyka HTML5 definuje pravidla především pro prohlížeče. Ty musí zachovat zpětnou kompatibilitu s dokumenty typu HTML 4.01, XHTML 1.0 a staršími. Tato skutečnost ruší potřebu označovat některé tagy jako tzv. *"deprecated"*.

V čistém HTML5 nejsou zachovány všechny rysy předchozích verzí. Zápis tagů musí být proveden přesně dle pravidel, např. nesmí být vynechán ukončovací tag. V případě syntaktické chyby nás na ni prohlížeč musí upozornit, a to prostřednictvím chybových zpráv. V čistém HTML5 je zakázáno používání prezentačních tagů a atributů, např. tag *<font>*, atribut *color* a další. Ty musí být nahrazeny pomocí CSS, které by měly sloužit pro veškeré formátování obsahu dokumentu.

3.3. Znaková sada

V dnešní době detekuje prohlížeč znakovou sadu stránky pomocí meta tagu http-equiv. HTML5 umožňuje na rozdíl od předchozích standardů tři způsoby definice znakové sady.

1. Pomocí transportní vrstvy, použitím hlavičky Content-type protokolu HTTP.
2. Použitím tzv. *"Byte Order Mark"*. Jde o znak v *Unicode*, určující pořadí bajtů v textovém souboru. Ten musí být uveden hned na začátku souboru.
3. Použitím meta tagu s novým atributem *charset*, jehož hodnotou je název znakové sady.

I když je možné používat různé znakové sady, HTML5 doporučuje a upřednostňuje použití znakové sady *UTF-8*. Ta je použita defaultně, není-li to v dokumentu specifikováno jinak.

Syntaxe 8. *Nastavení znakové sady pomocí hlavičky protokolu HTTP*

Např. <meta http-equiv="Content-Type" content="text/html; charset=utf-8" / >

Syntaxe 9. *Nastavení znakové sady pomocí Byte Order Mark*

Např. EF BB BF (= Byte Order Mark pro UTF-8)

Syntaxe 10. *Nastavení znakové sady pomocí meta tagu s atributem charset*

Např. <meta charset="UTF-8" / >

3.4. Sémantika

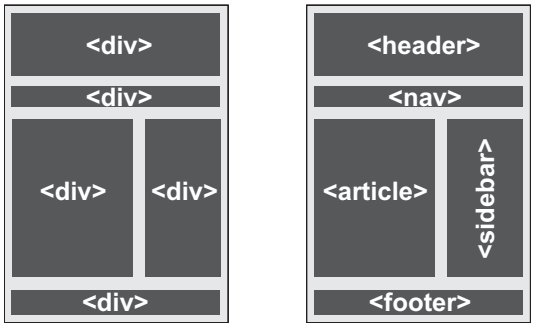
U současných standardů je sémantika velmi laxní. Mnoho prvků jazyka má velmi obecný význam a použití. Hlavním nedostatkem, jak již bylo zmíněno v 2. kapitole, je grafické rozložení stránky. K tomu dnes v ideálním případě využíváme tag . U rozsahově menších (X)HTML dokumentů to nemusí být problém. Nicméně u rozsáhlých projektů se stává zdrojový kód velmi nepřehledným. Jediným, ne však dostatečným, východiskem, je pojmenovat tagy pomocí atributu class, nebo id. Popřípadě používat komentáře. Ty ale nemají žádný význam pro internetové vyhledávače.

Proto přináší HTML5 rozšíření sémantiky o nové elementy. Ty mají reprezentovat logické celky dokumentu. Např. elementy vyznačující hlavičku dokumentu, zápatí dokumentu, či hlavní obsah. Momentálně mezi ně patří:

- *header* - Označuje záhlaví dokumentu.
- *footer* - Označuje zápatí dokumentu.
- *article* - Označuje hlavní obsah dokumentu.
- *sidebar* - Označuje vedlejší obsah dokumentu.
- *section* - Označuje sekci, část obsahu dokumentu.
- *nav* - Označuje hlavní menu dokumentu.
- *aside* - Označuje pomocné *(dodatečné)* menu dokumentu.
- Další by měly být ještě doplněny.

Kromě vylepšení sémantiky mají tyto nové elementy ještě jednu důležitou úlohu. Touto úlohou je ulehčit webovým vyhledávačům. Díky logickému rozdělení obsahu by mohly efektivněji analyzovat hypertextové dokumenty. Oddělením důležitého obsahu od zbytku dokumentu přinesou vyhledávače objektivnější a přesnější ohodnocení webových stránek. Tím běžný uživatel dostane přesnější nástroj k tomu, aby nalezl potřebné informace.

Nástin rozdílu mezi rozvržením dokumentu se současnými možnostmi a s použitím HTML5 najdete na obr. č. 5.



Obrázek 5. Rozvržení dokumentu pomocí bloků div a pomocí HTML5

Zdrojový kód 2. *Dokument v HTML 4.01*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>

  <head>
    <title>
      Příklad dokumentu v HTML 4.01
    </title>

    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  </head>

  <body>
    <div id="hlavicka">
      <img src="url_obrazku" width="300" height="100">
    </div>

    <div id="hlavni_menu">
      <a href="url_odkazu" title="odkaz">Odkaz menu 1</a><br>
      <a href="url_odkazu" title="odkaz">Odkaz menu 2</a><br>
      ...
    </div>

    <div id="obsah_stranky">
      <div id="vedlejsi_obsah_stranky">
        Text na stránce
      </div>

      <div id="vedlejší_menu">
        <a href="url_odkazu" title="odkaz">Odkaz na článek č.1</a><br>
        <a href="url_odkazu" title="odkaz">Odkaz na článek č.2</a><br>
        ...
      </div>
    </div>

    <div id="paticka">
      Copyright 2009 Název společnosti
    </div>
  </body>

</html>
```

Zdrojový kód 3. *Dokument v HTML5*

```
<!DOCTYPE HTML>
<html>

  <head>
    <title>
      Příklad dokumentu v HTML5
    </title>

    <meta charset="utf-8" />
  </head>

  <body>
    <header>
      <img src="url_obrazku" alt="Logo stránek" style="width:300px; height:100px" />
    </header>

    <nav>
      <a href="url_odkazu" title="odkaz">Odkaz menu 1</a><br />
      <a href="url_odkazu" title="odkaz">Odkaz menu 2</a><br />
      ...
    </nav>

    <article>

      <sidebar>
        Text na stránce
      </sidebar>

      <aside>
        <a href="url_odkazu" title="odkaz">Odkaz na článek č.1</a><br />
        <a href="url_odkazu" title="odkaz">Odkaz na článek č.2</a><br />
        ...
      </aside>

    </article>

    <footer>
      Copyright 2009 Název společnosti
    </footer>
  </body>

</html>
```

3.5. Rozšíření API

Příprava standardu HTML5 přináší i rozšíření API jazyka HTML. Rozšíření se týká převážně metod a atributů dvou objektů API. Jedním je objekt *HTMLDocument*, sloužící pro práci s dokumentem. Druhým pak objekt HTMLElement, který slouží pro práci s veškerými elementy jazyka.

Nové API pro HTMLDocument

- metoda getElementByClass
- metoda hasFocus

- metoda getSelection
- atribut activeElement
- atribut innerHTML

Nové API pro HTMLElement

- metoda getElementByClass
- atribut classList
- atribut innerHTML

Metoda getElementByClass

Pokud byste chtěli získat referenci na některý element, musíte ho se současným API označit atributem id a následně použít metodu *getElementById*. Ta náleží objektům HTMLDocument a HTMLElement. Nedostatkem je skutečnost, že identifikátor id je unikátní, tedy může označovat pouze jeden element v dokumentu. Zatímco class označuje skupinu, která má stejné vlastnosti. Neexistuje však žádná metoda, která by uměla získat elementy dle třídy. Takže pokud bychom chtěli vybrat více podobných elementů, museli bychom si vytvořit vlastní funkci. Tvorba takové funkce je složitá a využívá regulární výrazy. Také bychom ji museli vkládat do každého dokumentu, ve kterém bychom jí chtěli využívat. O složitosti této funkce se můžete přesvědčit ve zdrojovém kódu č. 4.

Nyní, s příchodem HTML5, označíme elementy pomocí atributu class a již není potřeba vytvářet vlastní funkci pro popsanou selekci. Pouze zavoláme metodu *getElementByClass* na objektu HTMLDocument, nebo HTMLElement. Té jako argument předáme hodnotu hledaného atributu class, uzavřenou v uvozovkách. Tím získáme pole referencí na požadované objekty, pokud v dokumentu nějaké existují.

Syntaxe 11. *document.getElementByClass("hodnota_atributu_class");*
Tato metoda vrací všechny elementy v dokumentu s daným atributem class.

Např. var blueBoxes = document.getElementByClass("blueBox");

Syntaxe 12. *element.getElementByClass("hodnota_atributu_class");*
Tato metoda vrací všechny elementy s daným atributem class, které se nachází uvnitř elementu, na kterém je metoda volána.

Např.
var header = document.getElementById("header");
var headerLinks = header.getElementByClass("header_link");

Zdrojový kód 4. *Funkce pro získání elementů se zadaným atributem class, bez použití nového rozšíření API HTML5*

```
function getElementByClass(className){

  //Proměnné pro uchování všech elementů a výsledků
  var documentElements = [];
  var requiredResults =[];

  //Vzor (regulární výraz) pro vyhledání třídy
  var pattern = new RegExp("(^| )" + className + "( |$)");


  //Získání všech tagů v dokumentu (s podporou i pro IE)
  if(typeof document.all == "undefined")
  {
    documentElements = document.getElementByTagName("*");
  }
  else
  {
    documentElements = document.all;
  }

  for (var i = 0; i < documentElements.length; i++)
  {
    if(pattern.test(documentElements[i].className))
    {
      requiredResults[requiredResult.length] = documentElements[i];
    }
  }

  // Na závěr vrátíme pole nalezených výsledků
  return requiredResults;
}
```

Metoda hasFocus

Tato metoda náleží objektu HTMLDocument. Její návratovou hodnotou je booleovská hodnota. Ta nám říká, zda-li je dokument, na kterém je metoda volána, aktivní. Tedy má-li tzv. *"focus"*. Metoda nemá žádné argumenty.

Syntaxe 13. *document.hasFocus();*
Návratovou hodnotou je true, má-li dokument focus, jinak false.

Např.

```
var docFocusState = document.hasFocus();
if(docFocusState)
  alert("Tento dokument je aktivní!");
```

Metoda getSelection

Metoda *getSelection* je metodou objektu HTMLDocument. Jejím voláním dostanete objekt, který implementuje rozhraní *Selection*. Obsahem tohoto objektu je výběr, který je označen v dokumentu. Na tomto objektu můžete provádět další operace, ty zatím nejsou ve specifikaci vymezeny. Zájemce o bližší informace k této metodě odkazuji na [1].

Syntaxe 14. *document.getSelection();*
Návratovou hodnotou je označený výběr z obsahu aktuálního dokumentu.

Např.
```
//předpokládejme, že máme v dokumentu označen text
var selection = document.getSelection();
alert("Aktuálně označeným výběrem je: " + selection.ToString());
```

Atribut innerHTML

Atribut *innerHTML* slouží k získání obsahu (X)HTML dokumentu, nebo elementu. Aktuálně je tento atribut hojně využíván pro práci s elementy. Pro práci s celým dokumentem jej však nebylo možné použít.

Zahrnutí tohoto atributu do standardu přichází až nyní, s návrhem HTML5. Atribut vrací, nebo nastavuje obsah formou řetězce. Tento řetězec je parsován a v případě dokumentu XHTML může vrátit chybové hlášení. A to tehdy, nelze-li hodnotu serializovat dle pravidel pro XML, nebo obsahuje-li syntaktické chyby. Nejčastěji je tento atribut používán, chceme-li dynamicky změnit obsah některého z elementů obsažených v dokumentu.

Syntaxe 15. *document.innerHTML;*
Návratovou hodnotou je obsah dokumentu ve formě řetězce.

document.innerHTML = value;
Nastaví obsah dokumentu na zadanou hodnotu.

Např.
```
var docContentUpdate = document.innerHTML;
docContentUpdate.replace("HTML 4.01", "HTML5");
document.innerHTML = docContentUpdate;
```

Syntaxe 16. *element.innerHTML;*
Návratovou hodnotou je obsah elementu ve formě řetězce.

element.innerHTML = value;
Nastaví obsah elementu na zadanou hodnotu.

Např.
```
var article = document.getElementById("article");
var elemContentUpdate = article.innerHTML;
elemContentUpdate.replace("např.", "na příklad: ");
article.innerHTML = elemContentUpdate;
```

Atribut classList

Atribut *classList* vrací seznam tříd *(atribut class)* elementu, na kterém je volán. Objekt, který atribut vrací, implementuje metody has, add, remove a toggle.

Metoda *has* slouží k detekci, obsahuje-li seznam tříd třídu, která je mu předána formou argumentu. Pokud ano, vrací hodnotu *true*, v opačném případě *false*. Implementaci podobného chování bez použití atributů aplikačního rozhraní HTML5 můžete vidět ve zdrojovém kódu č. 5. Metoda *add* slouží pro přidání nové třídy do seznamu tříd elementu. Nová třída je mu opět předána jako argument. Opakem je metoda *remove*, ta odebere ze seznamu třídu, která je argumentem metody. Pokud třída v seznamu tříd není nalezena, nenastane žádná změna. Poslední metodou je *toggle*, přebírající dva argumenty. Najde-li jednu ze tříd v seznamu, provede její záměnu za druhou.

Atribut classList a jeho metody ulehčují programátorům práci, jelikož adekvátní funkcionalita není momentálně defaultně dostupná. Existuje však atribut *className*, který vrací řetězec všech tříd elementu. Ten ale musíme zpracovat sami, a to pomocí vlastních funkcí, nebo přídavných knihoven javascriptu. Ty však musí být importovány do všech dokumentů, kde mají být použity.

Syntaxe 17. *element.classList.nazev_metody(argument1, . . .);*

Např.
```
var article = document.getElementById("article");
if(article.classList.has("odsazeni_zleva"))
  article.classList.remove("odsazeni_zleva");
if(!article.classList.has("justified_text"))
  article.classList.add("justified_text");
article.classList.toogle("article_background_red", "article_background_white");
```

Zdrojový kód 5. *Funkce sloužící k detekci, má-li element danou třídu*

```
// Předpokládejme, že nemůžeme použít API HTML5
// Funkce detekuje, má-li element, předaný jako první argument,
// třídu s názvem dle druhého argumentu

function has(elementReference, clName){

  // Získáme řetězec se všemi třídami elementu
  var allClassesOfElement = elementReference.className;

  //Vzor (regulární výraz) pro vyhledání konkrétní třídy
  var pattern = new RegExp("(^| )" + clName + "( |$)");

  // Na závěr vrátíme informaci, obsahuje-li element požadovanou třídu
  return pattern.test(allClassesOfElement);

}
```

Atribut activeElement

Tento atribut slouží pro získání reference na element, který má tzv. *"focus"*. Jinými slovy atribut vrací element, který je aktivní v rámci daného dokumentu. Pokud v dokumentu není žádný element aktivní, návratovou hodnotou atributu je element *body*. Atribut aktivní element pouze vrací, nelze jej nastavit. Přiřazení aktivního elementu do atributu je otázkou vnitřní implementace prohlížeče.

Syntaxe 18. *document.activeElement;*

Např.
var focusedElem = document.activeArticle;
focusedElem.style.fontSize += 1.0;

4. Současné možnosti HTML5

HTML verze 5 je stále ve fázi příprav, nicméně již dnes nám nabízí nové elementy. Těch můžeme směle využít při vkládání audio a video záznamů, nebo při tvorbě online her a grafických aplikací. Bohužel podpora těchto elementů není stoprocentní a nese s sebou určitá omezení, která jsou spojena s konkrétními prohlížeči.

4.1. Canvas element

Canvas je element, který je sám o sobě bezvýznamný. Z vizuálního hlediska jde o prázdnou bitmapu s předem určenými rozměry. Slovem prázdná se v případě canvas elementu označuje černá průhledná bitmapa. Kdyby canvas neuměl víc než zobrazit průhlednou bitmapu, byl by zbytečný. Mohli bychom jej nahradit elementem img, do kterého bychom vložili průhledný obrázek typu *gif*, nebo *png*. Sílu a možnosti canvas elementu odkrývá až javascript a aplikační rozhraní. Spolu s těmito prostředky slouží bitmapa jako plátno, na které můžeme vykreslovat grafy, text, obrázky, animace a herní grafiku. Ačkoli je canvas párový tag, veškerý obsah napsaný mezi počátečním a ukončovacím tagem je skryt. Tento obsah se nazývá tzv. *Fallback content*. Jde o obsah, který je zobrazen v případě chyby, nebo nepodporuje-li daný prohlížeč canvas. Je doporučením, aby fallback content plnil stejnou, nebo podobnou funkci jako canvas. Důvodem je skutečnost, že by uživatel mohl být ochuzen o důležité informace.

Součástí této bakalářské práce je vytvoření online tutoriálu, který vysvětluje základy práce s elementem canvas. Tento tutoriál se nachází na [2], nebo jej naleznete na přiloženém CD, které je popsáno v příloze B. Obsahuje praktické příklady, na kterých můžete shlédnout práci s metodami aplikačního rozhraní pro canvas element a výsledný efekt jejich použití.

Syntaxe 19. *<canvas> Fallback content </canvas>*

Např.

```
<canvas id="muj_canvas" width="400" height="300">
  Vidíte-li tento text, Váš prohlížeč nepodporuje canvas element jazyka HTML5
</canvas>
```

Současná podpora elementu canvas

V současné době je canvas podporován většinou majoritních prohlížečů. Výjimku tvoří *Internet Explorer* verze *8* a starší. I zde můžeme podporu canvas elementu zprovoznit, avšak pouze s pomocí externí javascriptové knihovny. Ta nám neumožní použít veškeré nástroje, které canvas nabízí.

Přehled majoritních prohlížečů se zabudovanou podporou pro element canvas a jeho aplikační rozhraní následuje:

- Safari 2.0+
- Firefox 3.5+
- Chrome 3.0+
- Opera 9.0+

Atributy elementu canvas

Canvas vlastní pouze 2 atributy, s výjimkou globálních atributů. Jedním z nich je atribut *width*, který slouží pro nastavení šířky canvas bitmapy. Druhým je atribut *height* sloužící k nastavení výšky. Jednotkou obou atributů je *pixel* a je doporučeno jim za hodnotu předávat celá čísla. V případě předání desetinného čísla může dojít k odříznutí desetinné části, v závislosti na vnitřní implementaci prohlížeče. Defaultní hodnotou je *300px* šířka a *150px* výška.

Velikost bitmapy je možné nastavit také pomocí kaskádových stylů. Tato možnost se však nedoporučuje. Důvodem je, že při dynamické změně velikosti bitmapy pomocí atributů se změní pouze velikost bitmapy. Kdežto při změně rozměrů pomocí kaskádových stylů se změní i velikost obsahu bitmapy. Tedy dojde k deformaci obsahu, který je na bitmapě vykreslen.

4.2. Canvas API

Aplikační rozhraní elementu canvas se skládá ze dvou hlavních částí. První částí je rozhraní *HTMLCanvasElement*. Pod toto rozhraní spadají i výše zmíněné atributy. Druhou část tvoří rozhraní *CanvasRenderingContext2D*, které slouží pro práci s bitmapou elementu canvas. Tato rozhraní naleznete v příloze C. Příklady a návod pro práci s tímto aplikačním rozhraním naleznete na [2].

4.2.1. Rozhraní HTMLCanvasElement

Rozhraní HTMLCanvasElement slouží pro přístup k vlastnostem a metodám elementu canvas. Pomocí atributů tohoto rozhraní nastavujeme šířku a výšku bitmapy canvas elementu. Podrobnější informace k atributům canvas elementu byly již uvedeny.

Důležité jsou metody tohoto rozhraní, ty jsou dvě. Abychom s nimi mohli pracovat, musíme nejdříve získat přístup k objektu HTMLCanvasElement. To provedeme pomocí již zmíněné metody getElementById.

Syntaxe 20. *Získání přístupu k objektu HTMLCanvasElement*
var canvas = document.getElementById("id_canvas_elementu");

Např.
```
var canvas = document.getElementById("myCanvas");
alert("Canvas má šířku " + canvas.width + " px");
```

Výše zmíněnými metodami canvasu jsou:

- **toDataURL([image format,...])** - Vrací URL adresu aktuálního obsahu canvas bitmapy.
- **getContext(contextId)** - Vrací kontext dle zadaného argumentu.

Metoda *toDataURL* slouží pro získání *URL* adresy rastrové bitmapy canvas elementu. Tato adresa je návratovou hodnotou. Metoda přebírá žádný, nebo více argumentů. Ty určují formát obrázku, do kterého má být výsledná adresa zakódována. Defaultním nastavením, pokud nezadáme žádný argument, je *image/png*. Současný návrh standardu nařizuje prohlížečům, že musí tento formát podporovat jako výchozí. Některé prohlížeče se snaží implementovat i formát *image/jpeg*, ten však do návrhu standardu zahrnut nebude.

URL adresa obrázku je vrácena ve formě řetězce. Ten má tvar *"data:" + "url obrázku"* v daném formátu. Pokud obrázek na bitmapě nemá žádná data, metoda vrací pouze řetězec *"data:,"*. Tím, že nemá data, je myšleno, má-li nastavenu výšku i šířku na hodnotu *0px*.

Syntaxe 21. *canvas.toDataURL([image format, ...]);*

Např.
```
var canvas = document.getElementById("myCanvas");
var url = canvas.toDataURL("image/png");
window.location = url;
```

Metoda *getContext* slouží pro získání přístupu k aplikačnímu rozhraní pro kreslení na bitmapu canvas elementu. Typ aplikačního rozhraní je zvolen dle argumentu, který je metodě předán formou řetězce. Ve specifikaci je definována pouze jediná hodnota, tou je řetězec *"2d"*. Je-li tento řetězec předán metodě getContext jako její argument, vytvoří se objekt a vrátí se reference na něj. Tento objekt implementuje rozhraní CanvasRenderingContext2d, dále jen *kontext*. Každý canvas element má právě jeden jediný kontext. Tudíž vícenásobným voláním této metody obdržíte vždy stejný objekt CanvasRenderingContext2d.

Prozatím ve specifikaci neexistuje žádný jiný kontext, než 2d. Avšak některé z prohlížečů definují vlastní, tzv. *3d* kontext. Ten je pouze experimentální a nemá

nic společného s návrhem jazyka HTML5. Pro zajímavost, v současné době s 3d kontextem experimentují prohlížeče Firefox a Opera.

Je-li metodě předán za argument řetězec, který není podporován, nebo je prázdný, vrací metoda hodnotu *null*. Na prohlížeče je také kladen požadavek, aby porovnávání argumentů bylo *case-sensitive*.

Syntaxe 22. *canvas.getContext(contextId);*

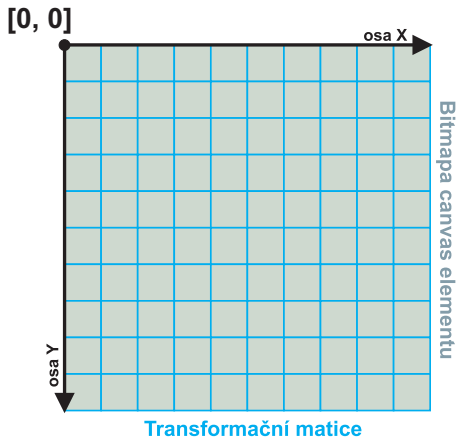
Např.

```
var canvas = document.getElementById("myCanvas");
var context = canvas.getContext("2d");
if(context != null){
  context.fillRect(0, 0, 100, 100);
}
```

4.2.2. Rozhraní CanvasRenderingContext2D

Rozhraní CanvasRenderingContext2D umožňuje přístup k vlastnostem a metodám, které slouží pro kreslení na bitmapu elementu canvas. Referenci na objekt tohoto rozhraní získáte pomocí metody getContext, jejíž použití již bylo uvedeno.

Při kreslení na bitmapu používá kontext souřadnicový systém. Počátek souřadnic je vždy v levém horním rohu. Na tomto souřadnicovém systému závisí také *transformační matice*. Ilustraci souřadnicového systému a transformační matice canvas bitmapy naleznete na obr. č. 6. Toto rozhraní naleznete v příloze C. Metody a vlastnosti budou postupně probrány dle pořadí, jak jsou uvedeny v návrhu standardu.



Obrázek 6. Souřadnicový systém canvas bitmapy

Zpětná reference na canvas

Chceme-li kreslit na bitmapu elementu canvas, musíme pracovat s kontextem tohoto elementu. Z praktického hlediska je však potřeba, abychom v jakémkoli okamžiku mohli přistoupit k objektu canvas, s jehož kontextem právě pracujeme. Proto kontext obsahuje atribut, jehož návratovou hodnotou je reference na objekt daného canvas elementu. Nastavení tohoto atributu je záležitostí vnitřní implementace prohlížeče. Dochází k němu při vytvoření kontextu.

Nás zajímá pouze z hlediska získání reference na element canvas. Tímto atributem je atribut:

- **canvas** - Vrací objekt canvas daného kontextu, na kterém je volán.

Syntaxe 23. *context.canvas;*

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
alert("Šířka canvas elementu je : " + context.canvas.width + " px");
```

Stavy canvas elementu

Kontext elementu canvas obsahuje *stavový zásobník*, viz. obr. č. 7. Stavem je aktuální nastavení kontextu pro kreslení, dále jen *stav kreslení*. Každý takový stav se skládá z aktuálního nastavení transformační matice, závěsného regionu *(clipping-region)* a hodnot kreslících atributů. Těmito atributy jsou strokeStyle, fillStyle, globalAlpha, globalCompositeOperation, lineWidth, lineCap, lineJoin, miterLimit, shadowOffsetX, shadowOffsetY, shadowBlur, shadowColor, font, textAlign a textBaseline.

Pro práci se zásobníkem nám kontext poskytuje dvě metody, obě bez argumentů:

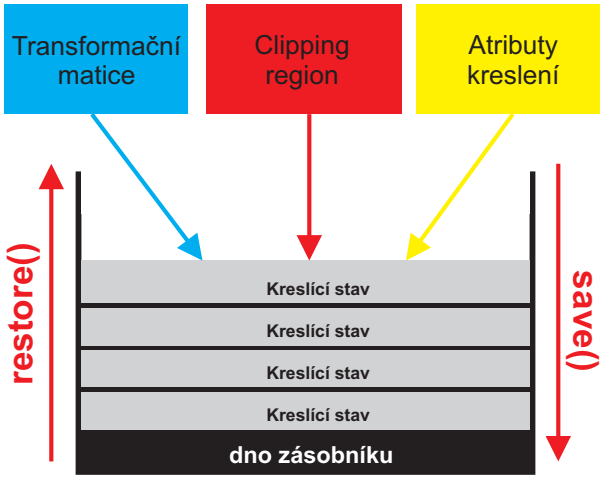
- **save()** - Vloží na zásobník kopii aktuálního stavu kreslení.
- **restore()** - Vyjme z vrcholu zásobníku stav kreslení a nastaví jej jako aktuální stav kreslení pro kontext. Pokud je zásobník prázdný, metoda nic nedělá.

Syntaxe 24. *context.save();*

Např.
```
// uloží aktuální kreslící stav na zásobník (modrá barva výplně)
context.fillStyle = "#99FF00";
context.save();
```

Syntaxe 25. *context.restore();*

Např.
// nastaví kreslící stav na stav z vrcholu zásobníku
context.restore();
context.fillRect(10, 10, 100, 100);



Obrázek 7. Zásobník se stavy kreslení

Transformace

Jak jsem zmínil, kontext má svůj aktuální kreslící stav. Jedním z prvků tohoto stavu je transformační matice. Ta je aplikována na souřadnice tvarů a cest při jejich vytvoření. Tato matice je následně používána při transformacích objektů vykreslených na bitmapě.

Při vytvoření kontextu splňuje transformační matice podmínku *identity*. Kontext tedy přiřazuje transformační matici ji samotnou. Transformace musí být vykonány v opačném pořadí, než jsou aplikovány. Například budeme-li rotovat čtvercem a následně změníme jeho velikost, nejdříve se vykoná operace změny velikosti a následně bude vykonána rotace.

Pro transformaci objektů, které jsou vykreslovány na bitmapu máme pět metod:

- **scale(x, y)** - Změní transformační matici tak, aby došlo ke změně velikosti dle zadaných parametrů.
- **rotate(angle)** - Změní transformační matici tak, aby došlo k otočení o daný úhel. Ten musí být uveden v obloukové míře.

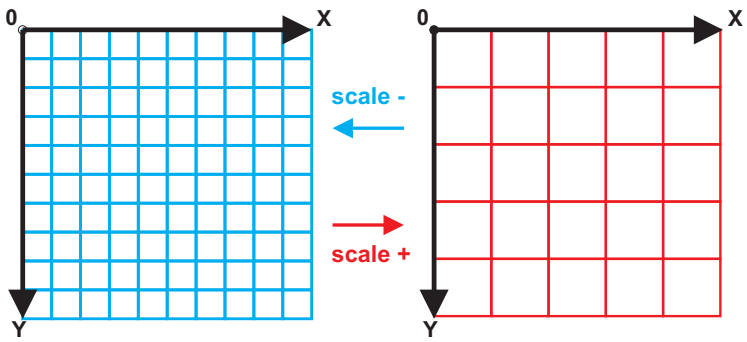
- **translate(x, y)** - Změní transformační matici tak, aby došlo k posunu počátku systému souřadnic.
- **transform(m11, m12, m21, m22, dx, dy)** - Změní transformační matici tak, aby aplikovala matici zadanou argumenty.
- **setTransform(m11, m12, m21, m22, dx, dy)** - Změní transformační matici na matici zadanou argumenty.

Metoda *scale* aplikuje transformaci pro změnu velikosti transformační matice dle zadaných argumentů. Argument x popisuje změnu velikosti v horizontálním směru a argument y ve vertikálním směru. Výchozí hodnotou argumentů je 1.0. Tato hodnota vyjadřuje procentuální velikost vzhledem k aktuální hodnotě. Tedy pokud hodnotu nastavíme na 0.5, dojde ke zmenšení na polovinu. V opačném případě, nastavíme-li hodnotu na 2.0, bude obsah bitmapy jednou tak velký. Při této transformaci dojde ke změně velikosti jak následně vykreslených tvarů a obrázků, tak i šířky linie. Ilustraci změny velikosti naleznete na obr. č. 8.

Syntaxe 26. *context.scale(x, y);*

Např.

```
var context = document.getElementById("myCanvas").getContext("2d");
context.scale(5, 2);
context.fillRect(0, 0, 100, 100);
```

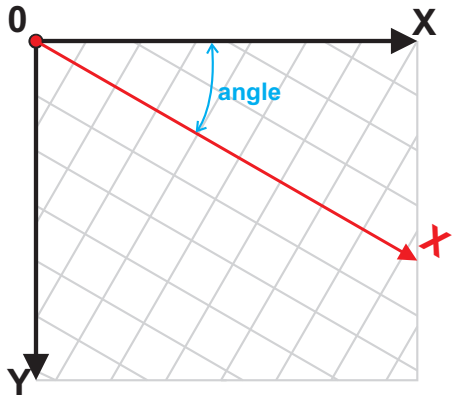


Obrázek 8. Změna velikosti transformační matice

Metoda *rotate* aplikuje transformaci pro otočení transformační matice dle zadaného úhlu. Argument angle představuje úhel rotace v hodinovém směru, vyjádřený v radiánech. Znázornění rotace transformační matice naleznete na obr. č. 9.

Syntaxe 27. *context.rotate(angle);*

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
context.rotate(Math.PI / 6);
context.fillRect(0, 0, 100, 100);
```



Obrázek 9. Rotace transformační matice

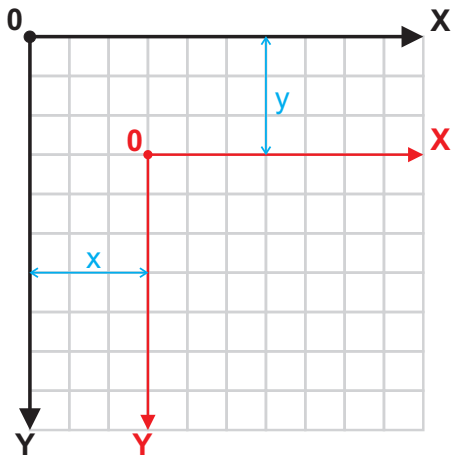
Metoda *translate* aplikuje transformaci pro posun systému souřadnic na transformační matici. Argument x určuje vzdálenost pro posun počátku souřadnic v horizontálním směru. Argument y určuje zase tuto vzdálenost ve vertikálním směru. Argumenty popisující vzdálenost jsou v jednotkách souřadnicového systému. Znázornění tohoto posunu naleznete na obr. č. 10.

Syntaxe 28. *context.translate(x, y);*

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
context.translate(20, 10);
context.fillRect(0, 0, 100, 100);
```

Metoda *transform* násobí aktuální transformační matici s maticí ve tvaru:

m11	m12	dx
m21	m22	dy
0	0	1



Obrázek 10. Posun počátku systému souřadnic transformační matice

Syntaxe 29. *context.transform(m11, m12, m21, m22, dx, dy);*

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
context.transform(1, 0, 0, 1, 1, 1);
context.fillRect(0, 0, 100, 100);
```

Metoda *setTransform* resetuje aktuální transformační matici na matici identity a zavolá metodu transform s argumenty, které jsou jí předány.

Syntaxe 30. *context.setTransform(m11, m12, m21, m22, dx, dy);*

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
context.setTransform(0, 0, 1, 1, 0, 1);
context.fillRect(0, 0, 100, 100);
```

Kompozice

Veškeré kontextové kreslící operace jsou ovlivněny dvojicí kompozičních atributů. Ty ovlivňují průhlednost vykreslovaných objektů a způsob jakým jsou kreslící objekty vrstveny na bitmapu canvas elementu.

Těmito atributy, volanými na objektu kontextu, jsou:

- **globalAlpha** [= **hodnota**] - Vrací, nebo nastavuje aktuální průhlednost objektů.

- **globalCompositeOperation [= hodnota]** - Vrací, nebo nastavuje aktuální způsob vrstvení kreslených objektů na bitmapu.

Atribut *globalAlpha* slouží pro určení hodnoty průhlednosti, která je aplikována na vykreslované objekty. Voláním atributu obdržíme aktuální nastavenou hodnotu průhlednosti. Přiřazením hodnoty můžeme průhlednost změnit. Hodnota se musí nacházet v rozsahu od 0.0 *(plně průhledný)* do 1.0 *(neprůhledný)*. Výchozí hodnotou při vytvoření kontextu je 1.0, tedy neprůhlednost objektů.

Při pokusu o zadání hodnoty mimo povolený rozsah, nekonečného čísla, nebo nečíselné hodnoty, zůstane hodnota atributu nezměněna.

Syntaxe 31. *context.globalAlpha;*
Návratovou hodnotou je aktuální hodnota atributu.

context.globalAlpha = value;
Nastaví hodnotu atributu.

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
if(context.globalAlpha < 1.0)
  context.globalAlpha = 1.0;
```

Atribut *globalCompositeOperation* vrací, nebo nastavuje aktuální způsob vrstvení vykreslovaných objektů vzhledem k obsahu bitmapy. K aplikaci této vlastnosti, při kreslení na bitmapu, dochází až po aplikaci průhlednosti a transformační matice.

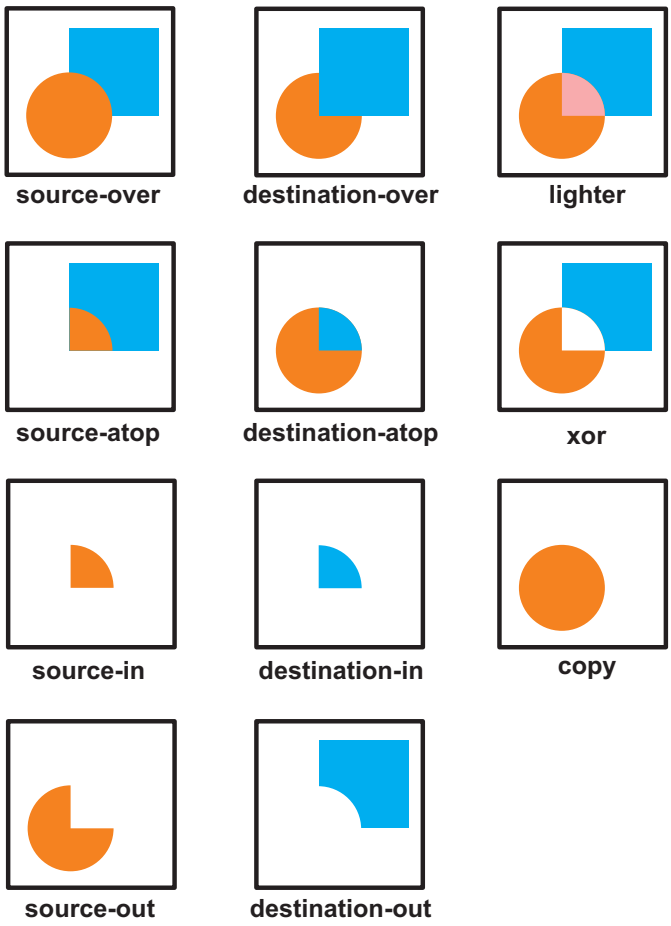
Hodnota tohoto atributu musí být z množiny povolených hodnot. Ty jsou uvedeny v tabulce č. 4. a jejich chování znázorňuje obrázek č. 11. V případě, je-li atributu předána jiná než povolená hodnota, nedojde ke změně. Výchozí hodnotou je řetězec *source-over*. Hodnoty jsou case-sensitive.

Na obrázku i v tabulce jsou použity dva objekty. Modrý čtverec je již vykreslen na bitmapě a je označen jako "destination", v tabulce znakem B. Oranžový kruh je do bitmapy vkládán a ovlivněn nastavením atributu globalCompositeOperation. Je označován názvem "source" a v tabulce znakem A.

Syntaxe 32. *context.globalCompositeOperation;*
Návratovou hodnotou je aktuální hodnota atributu.

context.globalCompositeOperation = value;
Nastaví hodnotu atributu.

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
context.globalCompositeOperation = "source-atop";
```



Obrázek 11. Ilustrace přípustných hodnot atributu globalCompositeOperation

Barvy a stylování

Abychom mohli s kontextem pracovat efektivně, potřebujeme možnost jak nastavit styl výplně a obrysu kreslených objektů. Tímto stylem může být jednoduchá barva, barevný přechod, nebo textura.

Pro nastavení stylu výplně a obrysu slouží dva atributy. Těm můžeme jako hodnotu předat obyčejnou barvu, kterou zadáme ve formátu CSS. Ale také můžeme zadat vytvořenou texturu, či barevný přechod. Těmito atributy jsou:

- **strokeStyle** [= **hodnota**] - Vrací, nebo nastavuje aktuální styl obrysu.
- **fillStyle** [= **hodnota**] - Vrací, nebo nastavuje aktuální styl výplně.

Atribut *strokeStyle* určuje barvu, nebo styl, který je použit pro vykreslení linie ohraničující kreslené objekty a také pro linii jako objekt. Při volání atributu je

Hodnota	**Pořadí**	**Slovní popis**
source-over	A nad B	Zobrazí source nad destination.
destination-over	B nad A	Zobrazí destination nad source.
source-atop	A na vrcholu B	Zobrazí část *source* v místě, kde jsou source i destination viditelné. Source je umístěn nad destination.
destination-atop	B na vrcholu A	Zobrazí část destination v místě, kde jsou source i destination viditelné. Destination je umístěn nad source.
source-in	A vně B	Zobrazí část source v místě, kde jsou source i destination viditelné. Destination nebude zobrazen.
destination-in	B vně A	Zobrazí část destination v místě, kde jsou source i destination viditelné. *Source* nebude zobrazen.
source-out	A mimo B	Zobrazí část source v místě, kde je source viditelný a destination viditelný není. Destination nebude zobrazen.
destination-out	B mimo A	Zobrazí část destination v místě, kde je destination viditelný a source viditelný není. Source nebude zobrazen.
lighter	A plus B	Zobrazí oba obrázky. V místě jejich průniku změní barvu na součet barvy source s barvou destination.
copy	A	Zobrazí source namísto destination. Není-li tato volba podporována, použije se hodnota source-over.
xor	A xor B	Zobrazí oba obrázky s tím, že na ně aplikuje operátor *XOR (exkluzivní OR).*

Tabulka 4. Přehled přípustných hodnot atributu globalCompositeOperation.

vráceno jeho aktuální nastavení. Přiřazením hodnoty dojde ke změně aktuálního nastavení atributu. Povolenou hodnotou může být barva zapsaná ve formátu CSS, objekt CanvasGradient, nebo CanvasPattern.

Pokud dojde k pokusu o změnu atributu na nepovolenou hodnotu, změna bude ignorována. Výchozí hodnotou při vytvoření kontextu je řetězec #000000.

Syntaxe 33. *context.strokeStyle;*
Návratovou hodnotou je aktuální hodnota atributu.

context.strokeStyle = value;
Nastaví hodnotu atributu.

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
context.strokeStyle = "#FF9900";
```

Atribut *fillStyle* určuje barvu, nebo styl, který je použit pro vykreslení výplně kresleného objektu. Voláním atributu získáme jeho aktuální nastavení. Přiřazením hodnoty dojde ke změně aktuálního nastavení atributu. Povolenou hodnotou je pouze barva zapsaná ve formátu CSS, objekt CanvasGradient, nebo CanvasPattern.

Dojde-li k pokusu o změnu atributu na nepovolenou hodnotu, změna bude ignorována. Výchozí hodnotou při vytvoření kontextu je řetězec #000000 *(černá barva)*.

Syntaxe 34. *context.fillStyle;*
Návratovou hodnotou je aktuální hodnota atributu.

context.fillStyle = value;
Nastaví hodnotu atributu.

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
context.fillStyle = "rgba(200, 0, 0, 0.5)";
```

Pro větší komfort poskytuje kontext metody k vytvoření specifické výplně a obrysu. Přesněji nám umožňuje použít za styl vykreslování *barevný přechod*, nebo *texturu*. K tomu slouží tři metody volané na kontextu a jedna volaná na objektu CanvasGradient. Těmi jsou:

- **addColorStop(offset, color)** - Jediná je volána na objektu CanvasGradient, přidává do přechodu barevnou stopu o zadané délce.
- **createLinearGradient(x0, y0, x1, y1)** - Vytvoří objekt CanvasGradient s lineárním přechodem o zadaných souřadnicích.
- **createRadialGradient(x0, y0, r0, x1, y1, r1)** - Vytvoří objekt CanvasGradient s radiálním přechodem o zadaných souřadnicích.
- **createPattern(image, repetition)** - Vytvoří objekt CanvasPattern ze zadaného obrázku a se zadaným stylem opakování.

Metoda *addColorStop* vytvoří v přechodu barevnou stopu o délce, která je zadána argumentem offset. Hodnota může být v rozsahu od 0.0 do 1.0. Tato hodnota vyjadřuje procentuální pozici barevné stopy vzhledem k velikosti celého přechodu.

Pokud je předána hodnota mimo rozsah, nekonečná, nebo nečíselná, dojde k vyvolání výjimky INDEX_SIZE_ERR. V případě, že se nepodaří rozparsovat předanou barvu, vyvolá se výjimka SYNTAX_ERR. Pokud nedojde k vyvolání výjimky, bude vytvořena nová stopa v barevném přechodu objektu CanvasGradient.

Syntaxe 35. *context.addColorStop(offset, color);*

Např.

```
var context = document.getElementById("myCanvas").getContext("2d");
var gradient = context.createLinearGradient(0, 0, 100, 200);
gradient.addColorStop("#FFFFFF", 0);
gradient.addColorStop("#FF6600", 0.4);
gradient.addColorStop("#666666", 1);
context.fillStyle = gradient;
```

Metoda *createLinearGradient* vrací objekt CanvasGradient. Ten reprezentuje lineární barevný přechod o zadaných souřadnicích. Argumenty (x0, y0) určují počátek, levý horní roh, přechodu. Argumenty (x1, y1) určují konec, pravý dolní roh, přechodu.

Je-li metodě předán nečíselný argument, nebo je argumentem nekonečné číslo, dojde k vyvolání výjimky NOT_SUPPORTED_ERR.

Syntaxe 36. *context.createLinearGradient(x0, y0, x1, y1);*

Např.

```
var context = document.getElementById("myCanvas").getContext("2d");
var gradient = context.createLinearGradient(0, 0, 100, 200);
gradient.addColorStop("#FFFFFF", 0);
gradient.addColorStop("#333333", 1);
context.fillStyle = gradient;
```

Metoda *createRadialGradient* vrací objekt CanvasGradient. Ten reprezentuje radiální barevný přechod o zadaných souřadnicích a poloměrech. Argumenty (x0, y0, r0) určují počátek a poloměr vnitřního kruhu. Argumenty (x1, y1, r1) počátek a poloměr vnějšího kruhu. Mezi těmito dvěma kruhy je následně vytvořen barevný přechod. Argumenty jsou v jednotkách souřadnicového systému kontextu.

Je-li metodě předán nečíselný argument, nebo je argumentem nekonečné číslo, dojde k vyvolání výjimky NOT_SUPPORTED_ERR. Je-li zadána záporná hodnota pro některý z poloměrů, dojde k vyvolání výjimky INDEX_SIZE_ERR.

Syntaxe 37. *context.createRadialGradient(x0, y0, r0, x1, y1, r1);*

Např.

```
var context = document.getElementById("myCanvas").getContext("2d");
var gradient = context.createRadialGradient(50, 50, 100, 30, 60, 120);
gradient.addColorStop("#000000", 0);
gradient.addColorStop("#99CC00", 1);
context.fillStyle = gradient;
```

Metoda *createPattern* vrací objekt CanvasPattern. Ten slouží jako textura pro výplň, nebo obrys. Metoda přebírá dva argumenty. Prvním je obrázek, který bude pro výplň použit. Zdrojem tohoto obrázku může být HTMLImageElement, HTMLCanvasElement, nebo HTMLVideoElement. Tedy obrázek, obsah bitmapy jiného canvas elementu, nebo video element, který bude probrán v této kapitole.

Je-li zadán neplatný typ obrázku, nebo hodnotou je *null*, je vyvolána výjimka TYPE_MISMATCH_ERR. V případě, že data obrázku jsou porušena, vrací metoda hodnotu *null*. Pokud by jeden z rozměrů obrázku byl nulový, dojde k vyvolání výjimky INVALID_STATE_ERR.

Druhým argumentem je styl opakování. Tj. způsob, jakým má být textura opakována v rámci kresleného objektu. Nařízením pro prohlížeče je, že musí podporovat hodnoty, které jsou uvedeny v tab. č. 5. Nicméně tato podpora prozatím není stoprocentní.

V případě, kdy je metodě předána za obrázek animace, musí prohlížeč animaci zpracovat. Jako texturu pak použije náhled animace, kterým je první snímek. Je-li argumentem video element, je za texturu považován snímek z aktuální pozice přehrávání videa. Rozměry tohoto snímku vychází z rozměrů zdroje.

Hodnota	**Slovní popis**	**Prohlížeče s podporou**
repeat	neopakovat	Safari, Firefox, Chrome
no-repeat	opakovat všemi směry	Safari, Firefox
repeat-x	opakovat dle osy x	Safari, Chrome
repeat-y	opakovat dle osy y	Safari, Chrome

Tabulka 5. Hodnoty argumentu repetition metody createPattern.

Syntaxe 38. *context.createPattern(image, repetition);*

Např.

```
var context = document.getElementById("myCanvas").getContext("2d");
var image = new Image();
image.src = "http://www.pisuweb.cz/images/canvas/pattern1.png";
var pattern = context.createPattern(image, "repeat-x");
context.fillStyle = pattern;
context.strokeStyle = pattern;
context.fillRect(0, 0, context.canvas.width, 100);
context.strokeRect(0, 0, context.canvas.width, 100);
```

Stylování linie

Kontext umožňuje změnit styl pro vykreslování linií. Vlastní čtyři atributy, které ovlivňují výsledný tvar linie. Ovlivnění se týká spojů a zakončení. Atributy slouží jak k nastavení, tak získání aktuální hodnoty. Všechny náleží kontextu.

- **lineWidth** [= **hodnota**] - Vrací, nebo nastavuje šířku linie.
- **lineCap** [= **hodnota**] - Vrací, nebo nastavuje styl zakončení linie.
- **lineJoin** [= **hodnota**] - Vrací, nebo nastavuje styl spojování linií.
- **mitterLimit** [= **hodnota**] - Vrací, nebo nastavuje velikost spoje na pokos.

Atribut *lineWidth* reprezentuje šířku linie v jednotkách systému souřadnic. Při vytvoření kontextu je výchozí hodnotou 1.0. Pokud se pokusíte nastavit jako šířku nulovou, nekonečnou, zápornou, nebo nečíselnou hodnotu, bude ignorována. Tedy hodnota atributu se nezmění.

Syntaxe 39. *context.lineWidth;*
Návratovou hodnotou je aktuální hodnota atributu.

context.lineWidth = value;
Nastaví hodnotu atributu.

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
if(context.lineWidth > 3)
  context.lineWidth = 1.4;
context.strokeRect(10, 10, 10, 10);
```

Atribut *lineCap* určuje styl zakončení linie. Požadavkem na prohlížeče je podpora tří hodnot, uvedených v tab. č. 6. Výchozí hodnotou je řetězec butt. V případě, je-li atributu předána jiná hodnota, ke změně aktuální hodnoty atributu nedojde.

Syntaxe 40. *context.lineCap;*
Návratovou hodnotou je aktuální hodnota atributu.

context.lineCap = value;
Nastaví hodnotu atributu.

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
if(context.lineCap == "butt")
  context.lineCap = "round";
...
```

Hodnota	Slovní popis
butt	Zakončení všech linií vytvoří rovnou hranu vzhledem k směru linie.
round	Zakončení všech linií bude provedeno pomocí půlkruhu. Jeho průměr bude roven šířce linie.
square	Zakončením všech linií bude čtverec o délce hrany, která je rovna šířce linie. Bude umístěn na konec linie zarovnán oproti hraně ve směru linie.

Tabulka 6. Hodnoty atributu lineCap.

Atribut *lineJoin* určuje styl pro spojení linií. Spoj bude vytvořen v místě, kde se dvě linie setkají. Opět máme tři přípustné hodnoty, uvedené v tab. č. 7. Defaultní nastavenou hodnotou je řetězec miter. Zadáte-li atributu hodnotu, která není uvedena v tab. č. 7., bude ignorována.

Hodnota	Slovní popis
bevel	Spoj linií bude vytvořen tak, jak na sebe linie navazují. Tedy bude vytvořena hrana, kdy vnitřní úhel bude závislý na směru obou linií.
round	Spoj linií bude vyplněn kruhem spojujícím dvě hrany.
miter	Spoj linií, zvaný spoj na pokos. Bližší informace pro zájemce naleznete na [7].

Tabulka 7. Hodnoty atributu lineJoin.

Syntaxe 41. *context.lineJoin;*
Návratovou hodnotou je aktuální hodnota atributu.

context.lineJoin = value;
Nastaví hodnotu atributu.

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
if(context.lineJoin == "miter")
   context.lineJoin = "round";
...
```

Atribut *miterLimit* reprezentuje maximální hodnotu, která určuje vzdálenost od průniku linií. Délka tohoto atributu může být rovna maximálně polovině šířky

linie. Při překročení této hodnoty nebude spoj vykreslen dle očekávání. Výchozí hodnotou je 10.0. Pokusíte-li se zadat nečíselnou, nekonečnou, zápornou, nebo nulovou hodnotu, bude ignorována.

Jelikož spoj na pokos není předmětem této bakalářské práce, zájemce o podrobnější informace odkazuji na [7].

Syntaxe 42. *context.miterLimit;*
Návratovou hodnotou je aktuální hodnota atributu.

context.miterLimit = value;
Nastaví hodnotu atributu.

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
if(context.miterLimit < 10)
  context.miterLimit = 10;
...
```

Stínování

Veškeré objekty vykreslované pomocí kontextu mohou mít stín. Ten je vykreslen pouze tehdy, když atributy globalAlpha, shadowBlur, shadowOffsetX a shadowOffsetY nejsou nulové. Stíny také nebudou vykresleny v případě nastavení kompozičního atributu globalCompositeOperation na hodnotu copy.

Pro vytvoření stínu má kontext čtyři atributy, které určují jeho vlastnosti. Těmito atributy jsou:

- **shadowColor** [= **hodnota**] - Vrací, nebo nastavuje barvu stínu.
- **shadowOffsetX** [= **hodnota**] - Vrací, nebo nastavuje posun stínu od stínovaného objektu na ose X.
- **shadowOffsetY** [= **hodnota**] - Vrací, nebo nastavuje posun stínu od stínovaného objektu na ose Y.
- **shadowBlur** [= **hodnota**] - Vrací, nebo nastavuje hodnotu rozostření stínu.

Atribut *shadowColor* udržuje informaci o barvě, která je použita pro vykreslení stínu. Nastavení tohoto atributu provedeme přiřazením barvy ve formátu zadávání barev dle CSS. Výchozí hodnotou tohoto atributu je černá plně průhledná barva.

Syntaxe 43. *context.shadowColor;*
Návratovou hodnotou je aktuální hodnota atributu.

context.shadowColor = value;
Nastaví hodnotu atributu.

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
context.shadowColor = "rgb(120, 30, 0)";
context.fillRect(10, 10, 100, 100);
```

Atribut *shadowColorX* určuje velikost horizontálního posunu stínu od stínovaného objektu. Hodnota je v jednotkách souřadnicového systému.

Syntaxe 44. *context.shadowColorX;*
Návratovou hodnotou je aktuální hodnota atributu.

context.shadowColorX = value;
Nastaví hodnotu atributu.

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
context.shadowColorX = 10;
...
```

Atribut *shadowColorY* určuje velikost vertikálního posunu stínu od stínovaného objektu. Hodnota je v jednotkách souřadnicového systému.

Syntaxe 45. *context.shadowColorY;*
Návratovou hodnotou je aktuální hodnota atributu.

context.shadowColorY = value;
Nastaví hodnotu atributu.

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
context.shadowColorY = 2.2;
...
```

Atribut *shadowBlur* určuje míru rozostření stínu. Nastavení probíhá přiřazením číselné hodnoty. Ta se nevztahuje k souřadnicovému systému, jedná se o číslo od hodnoty 0 do neurčeno *(maximální číselná hodnota)*.

Syntaxe 46. *context.shadowBlur;*
Návratovou hodnotou je aktuální hodnota atributu.

context.shadowBlur = value;
Nastaví hodnotu atributu.

Např.
var context = document.getElementById("myCanvas").getContext("2d");
context.shadowBlur = 3.6;
...

Ani jeden z atributů není ovlivněn transformační maticí. Výchozí hodnotou atributů je 0, s výjimkou atributu shadowColor. Při pokusu o nastavení nečíselné, nekonečné, nebo záporné hodnoty posledním třem atributům, bude změna ignorována.

Jednoduché tvary

Abychom mohli s kontextem kreslit, potřebujeme nějaký nástroj. Nejjednodušším nástrojem pro kreslení jsou tři základní metody. Ty slouží pro vykreslení základního tvaru, kterým je čtyřúhelník.

- **clearRect(x, y, w, h)** - Vymaže pixely v zadaném čtyřúhelníku.
- **fillRect(x, y, w, h)** - Vykreslí vyplněný čtyřúhelník.
- **strokeRect(x, y, w, h)** - Vykreslí obrys čtyřúhelníku.

Všechny tři metody přebírají 4 argumenty. A to x a y určující počátek vykreslovaného čtyřúhelníku, tím je jeho levý horní roh. Nastavením souřadnic určíte polohu počátku čtyřúhelníku vzhledem k počátku systému souřadnic. Dalšími atributy jsou w a h, určující šířku a výšku vykreslovaného čtyřúhelníku.

Při vykreslení čtyřúhelníku dochází k aplikaci transformační matice s předanými argumenty. Ta vytvoří uzavřenou cestu ze 4 bodů. Ty jsou na souřadnicích (x, y), (x + w, y), (x + w, y + h) a (x, y + h). Při vykreslení není ovlivněna aktuální cesta. Pokud metodám předáme za šířku, nebo výšku hodnotu 0, nebudou provedeny žádné akce.

Metoda *clearRect* vymaže veškeré pixely uvnitř čtyřúhelníku, který je vytvořen ze zadaných argumentů. Veškeré tyto pixely dostanou opět černou plně průhlednou barvu.

Syntaxe 47. *context.clearRect(x, y, w, h);*

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
context.clearRect(0, 0, context.canvas.width, 100);
```

Metoda *fillRect* vykreslí čtyřúhelník o zadaných souřadnicích a rozměru. Tento čtyřúhelník je plochou, která je vyplněna dle nastavení atributu fillStyle. Obrys není vykreslen.

Syntaxe 48. *context.fillRect(x, y, w, h);*

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
context.fillRect(0, 0, 100, 100);
```

Metoda *strokeRect* vykreslí čtyřúhelník o zadaných souřadnicích a rozměru. Tento čtyřúhelník je obrysem, tedy linií. Je ovlivněn aktuálním nastavením atributů strokeStyle, lineWidth, lineJoin a v případě potřeby i miterLimit.

U této metody je výjimka. Předáme-li jí za jeden z argumentů určujících rozměr hodnotu 0 a druhý bude nenulový, bude vykreslena linie. Jsou-li oba nulové, nebude vykresleno nic.

Syntaxe 49. *context.strokeRect(x, y, w, h);*

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
context.strokeRect(10, 10, context.canvas.width, context.canvas.height);
```

Složité tvary

Pro vykreslování složitějších grafických objektů nabízí kontext pestré spektrum metod. Ty pracují s aktuální *cestou*. Každý kontext má vždy právě jednu aktuální cestu, ta však není součástí stavů kreslení, které jsou ukládány na zásobník. Aktuální cesta je složením více *podřízených cest*, které se skládají z bodů. Těch může být 0 až teoreticky nekonečno. Podřízené cesty si s sebou nesou i informaci o tom, zda-li jsou uzavřené, nebo ne. Uzavřením je myšleno, zda z posledního bodu v podřízené cestě vede přímá linie do prvního bodu. Platí pravidlo, že všechny cesty musí mít minimálně 2 body. Pokud tomu tak není, je podřízená

cesta ignorována. Ve výchozím nastavení má kontext nulový počet podřízených cest.

Pro práci s cestou máme k dispozici metody, které ji vytváří, testují, nebo vykreslují. Veškeré body vložené pomocí následujících metod jsou transformovány dle aktuální transformační matice.

- **beginPath()** - Vytvoří novou aktuální cestu.
- **moveTo(x, y)** - Vytvoří novou podřízenou cestu a vloží do ní bod.
- **closePath()** - Označí aktuální podřízenou cestu jako uzavřenou a vytvoří novou podřízenou cestu.
- **lineTo(x, y)** - Přidá do cesty bod a s předchozím ho spojí přímkou.
- **quadraticCurveTo(cpx, cpy, x, y)** - Přidá do cesty bod a spojí ho s předchozím kvadratickou Bézierovou křivkou.
- **bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)** - Přidá do cesty bod a spojí ho s předchozím kubickou Bézierovou křivkou.
- **arcTo(x1, y1, x2, y2, radius)** - Přidá do cesty bod a spojí ho s předchozím přímou linií. Následně přidá druhý bod spojený s předchozím obloukem o zadaném poloměru.
- **arc(x, y, radius, startAngle, endAngle, anticlockwise)** - Přidá do cesty bod a s předchozím jej spojí obloukem dle zadaných argumentů.
- **rect(x, y, w, h)** - Přidá do aktuální cesty novou podřízenou cestu ve tvaru čtyřúhelníku.
- **fill()** - Vyplní podřízené cesty dle nastavení.
- **stroke()** - Nakreslí obrys podél podřízených cest.
- **clip()** - Vynutí ořez závěsného regionu dle aktuální cesty kontextu.
- **isPointInPath(x, y)** - Otestuje, leží-li daný bod na aktuální cestě.

Metoda *beginPath* smaže seznam podřízených cest. Tímto seznamem je aktuální cesta. Kontext bude mít opět nula podřízených cest a můžeme začít s vytvářením nových. To se hodí k tomu, abychom neovlivnili části grafiky, které mají mít různé vlastnosti. Veškeré podřízené cesty v aktuální cestě jsou vykreslovány se stejným nastavením.

Syntaxe 50. *context.beginPath();*

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
context.beginPath();
...;
```

Metoda *moveTo* zajišťuje vytvoření nové podřízené cesty. Při jejím vytvoření dojde k vložení prvního bodu na souřadnice x a y dle argumentů. Tento bod je jediným v dané podřízené cestě při jejím vytvoření.

Syntaxe 51. *context.moveTo(x, y);*

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
context.beginPath();
context.moveTo(15, 25);
...
```

Metoda *closedPath* označí aktuální podřízenou cestu jako uzavřenou a ukončí ji. Tedy poslední bod této podřízené cesty je spojen přímou linií s prvním bodem. Následně vytvoří novou podřízenou cestu s počátečním bodem, který bude v místě spoje prvního a posledního bodu uzavřené podřízené cesty. Pokud kontext nemá žádnou podřízenou cestu, metoda nic nedělá.

Syntaxe 52. *context.closedPath();*

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
context.beginPath();
context.moveTo(15, 25);
...
context.closedPath();
...
```

Metoda *lineTo* ověří, zda-li má kontext nějakou podřízenou cestu. Pokud ano, tak do této cesty přidá bod o zadaných souřadnicích. Spojí poslední bod podřízené cesty a nově vytvořený bod přímkou. Pokud neexistuje podřízená cesta, metoda nic nedělá.

Syntaxe 53. *context.lineTo(x, y);*

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
context.beginPath();
context.moveTo(15, 25);
context.lineTo(40, 30);
context.lineTo(20, 100);
...
```

Metoda *quadraticCurveTo* vloží do podřízené cesty bod na zadané souřadnice. Ten bude s předchozím bodem spojen kvadratickou Bézierovou křivkou sestrojenou dle kontrolního bodu *(o souřadnicích [cpx, cpy])*. Pokud v kontextu neexistuje podřízená cesta, metoda nic nedělá. Pro zájemce o bližší informace k této problematice doporučuji prozkoumat reference v návrhu standardu [1].

Syntaxe 54. *context.quadraticCurveTo(cpx, cpy, x, y);*

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
context.beginPath();
context.moveTo(15, 25);
context.lineTo(40, 30);
context.quadraticCurveTo(100, 0, 150, 45);
...
```

Metoda *bezierCurveTo* vloží do podřízené cesty bod na zadané souřadnice. Ten bude s předchozím bodem spojen kubickou Bézierovou křivkou sestrojenou dle dvou kontrolních bodů *(o souřadnicích [cp1x, cp1y] a [cp2x, cp2y])*. Pokud v kontextu neexistuje podřízená cesta, metoda nic neudělá.

Syntaxe 55. *context.bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y);*

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
context.beginPath();
context.moveTo(15, 25);
context.lineTo(40, 30);
context.bezierCurveTo(50, 25, 90, 10, 100, 40);
...
```

Metoda *arcTo* vloží do podřízené cesty bod a s předchozím jej spojí přímkou. Následně vloží druhý bod, který bude s předchozím spojen obloukem. Poloměr tohoto oblouku je předán argumentem radius a jeho hodnota je v obloukové míře.

Metodu lze použít pouze tehdy, pokud existuje nějaká podřízená cesta. Když neexistuje, metoda nic nedělá.

Zadáme-li pro poloměr zápornou hodnotu, dojde k vyvolání výjimky INDEX_SIZE_ERR.

Syntaxe 56. *context.arcTo(x1, y1, x2, y2, radius);*

Např.

```
var context = document.getElementById("myCanvas").getContext("2d");
context.beginPath();
context.moveTo(15, 25);
context.arcTo(30, 40, 100, 60, Math.PI / 6);
...
```

Metoda *arc* vytvoří oblouk mezi předchozím bodem a nově vkládaným bodem. Tento oblouk je určen poloměrem radius. Dále úhlem startAngle, který je přisouzen zadanému bodu. Druhým úhlem je endAngle, pomocí něj se vypočítá pozice koncového bodu vzhledem k počátečnímu. Posledním argumentem je booleovská hodnota, která určuje, má-li být oblouk vykreslen proti směru hodinových ručiček.

Pokud předáte záporný poloměr, dojde k vyvolání výjimky INDEX_SIZE_ERR.

Syntaxe 57. *context.arc(x, y, radius, startAngle, endAngle, anticlockwise);*

Např.

```
var context = document.getElementById("myCanvas").getContext("2d");
context.beginPath();
context.moveTo(15, 25);
context.arc(50, 100, 30, 0, Math.PI * 2, true);
...
```

Metoda *rect* vytvoří novou uzavřenou podřízenou cestu. Ta má tvar čtyřúhelníku o zadaných souřadnicích a rozměrech. Argumenty jsou zpracovány obdobně metodám fillRect a strokeRect.

Syntaxe 58. *context.rect(x, y, w, h);*

Např.

```
var context = document.getElementById("myCanvas").getContext("2d");
context.beginPath();
context.rect(10, 20, 100, 200);
...
```

Metoda *fill* vykreslí výplň všem podřízeným cestám, které jsou obsaženy v aktuální cestě. Výplň je závislá na nastavení atributu fillStyle.

Syntaxe 59. *context.fill();*

Např.

```
var context = document.getElementById("myCanvas").getContext("2d");
context.beginPath();
context.rect(10, 20, 100, 200);
context.fill();
```

Metoda *stroke* vykreslí obrys veškerých podřízených cest, nacházejících se v aktuální cestě. Obrysová linie je závislá na nastavení atributů strokeStyle, lineWidth, lineCap, lineJoin a popřípadě miterLimit.

Syntaxe 60. *context.stroke();*

Např.

```
var context = document.getElementById("myCanvas").getContext("2d");
context.beginPath();
context.rect(10, 20, 100, 200);
context.stroke();
```

Metoda *clip* vytvoří omezení, tzv.*masku*, vzniklou ze závěsného regionu. Veškeré cesty, které budou následně vytvořeny mimo tuto masku, nebudou vykresleny. Cesty vně masky vykresleny budou.

Syntaxe 61. *context.clip();*

Např.

```
var context = document.getElementById("myCanvas").getContext("2d");
context.beginPath();
context.moveTo(50, 50);
context.arc(100, 100, 50, 0, Math.PI / 4, false);
context.fill();
context.clip();
context.rect(20, 30, 200, 200);
context.fill();
```

Metoda *isPointInPath* slouží pro testování, zda-li se bod na předaných souřadnicích nachází uvnitř aktuální cesty. Návratovou hodnotou je true, pokud cesta bod obsahuje, v opačném případě false.

Syntaxe 62. *context.isPointInPath(x, y);*

Např.

```
var context = document.getElementById("myCanvas").getContext("2d");
context.beginPath();
context.rect(10, 20, 100, 200);
if(context.isPointInPath(10, 50))
  alert("Bod {x, y} náleží aktuální cestě.");
```

Text

Jelikož text je důležitou složkou informací, ani canvas element o něj není ochuzen. Nicméně by canvas neměl být používán za účelem psaní textu, který by obsahoval důležité informace. Díky omezené podpoře můžeme narazit na problém s kódováním, kterým by mělo být UTF-8. Avšak některé prohlížeče mají problém s vykreslením speciálních znaků národních abeced. Ty jsou pak vykresleny bez diakritiky, nebo ignorovány.

Pro práci s textem je použit objekt kontextu a objekt *TextMetric*. Ten je vytvořen metodou kontextu, sloužící pro měření šířky textu. Kontext obsahuje 3 metody a 3 atributy, kterými jsou:

- **font** [= **hodnota**] - Nastavuje, nebo vrací font pro vykreslení textu.
- **textAlign** [= **hodnota**] - Nastavuje, nebo vrací zarovnání textu.
- **textBaseline** [= **hodnota**] - Nastavuje, nebo vrací zarovnání vzhledem k vodící lince.
- **fillText(text, x, y** [**,maxWidth**]) - Vykreslí výplň textu na zadaných souřadnicích s nastaveným fontem.
- **strokeText(text, x, y** [**,maxWidth**]) - Vykreslí obrys textu na zadaných souřadnicích s nastaveným fontem.
- **measureText(text)** - Z předaného argumentu vypočte velikost textu a vrátí ji v objektu TextMetric.

Objekt TextMetric, vrácený metodou measureText, má jeden atribut, kterým je:

- **width** - Vrací šířku textu, která byla vypočítána metodou measureText.

Atribut *font* slouží pro nastavení fontu písma. Při volání vrací své aktuální nastavení. Výchozím fontem při vytvoření kontextu je ”10px sans-serif”. Pokud atributu hodnotu přiřadíte, bude parsována dle formátu CSS. Pokud nebude hodnota v souladu s pravidly, nedojde ke změně aktuální hodnoty. Ke změně nedojde ani při použití CSS hodnot *inherit*, nebo *initial*. Použijeme-li jako typ jednotky *em*, nebo *ex*, bude velikost písma vypočítána vzhledem k velikosti bitmapy.

Syntaxe 63. *context.font;*
Návratovou hodnotou je aktuální hodnota atributu.

context.font = value;
Nastaví hodnotu atributu.

Např.
```
var context = document.getElementById(”myCanvas”).getContext(”2d”);
context.font = ”2em Arial”;
```

Atribut *textAlign* určuje zarovnání textu vzhledem k maximální šířce a počátečnímu bodu. Při volání vrací svou aktuální hodnotu. Pro nastavení mu hodnotu přiřadíme. Tou může být pouze některá z povolených hodnot, které jsou uvedeny v tab. č. 8. Výchozí hodnotou je start. Pokud dojde k pokusu o přiřazení odlišné hodnoty, změna se neprojeví.

Hodnota	**Slovní popis**
start	Text bude zarovnán k počátku vykreslovaného textu. Zda-li napravo, či nalevo, závisí na vlastnosti *directionality*. Ta určuje směr vypisování textu.
end	Text bude zarovnán ke konci vykreslovaného textu. Zda-li napravo, či nalevo, závisí na vlastnosti directionality. Ta určuje směr vypisování textu.
left	Text bude zarovnán k levému okraji boxu, v němž je text vykreslen. *(Tento box není viditelný.)*
right	Text bude zarovnán k pravému okraji boxu, v němž je text vykreslen. *(Tento box není viditelný.)*
center	Text bude zarovnán na střed boxu, v němž je text vykreslen. *(Tento box není viditelný.)*

Tabulka 8. Hodnoty atributu textAlign.

Syntaxe 64. *context.textAlign;*
Návratovou hodnotou je aktuální hodnota atributu.

context.textAlign = value;
Nastaví hodnotu atributu.

Např.
var context = document.getElementById("myCanvas").getContext("2d");
context.textAlign = "end";

Atribut *textBaseline* určuje zarovnání textu vzhledem k vodící lince. Ta je reprezentována boxem do nějž je vykreslován znak, nebo posloupnost znaků. Box má velikost v jednotkách typu em.

Voláním získáme aktuální hodnotu. Pro nastavení atributu přiřadíme novou hodnotu. Výběr je omezený na hodnoty uvedené v tab. č. 9. V okamžiku vytvoření kontextu je hodnota nastavena na volbu alphabetic. Na obr. č. 12. můžete shlédnout ilustraci použití jednotlivých nastavení. Přiřadíte-li atributu hodnotu, která není v tabulce uvedena, ke změně aktuální hodnoty nedojde.

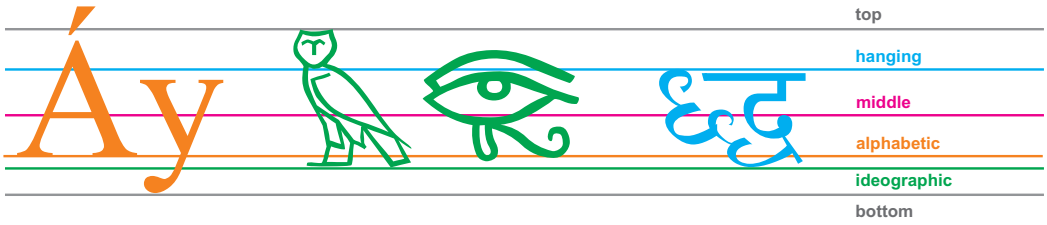
Hodnota	**Slovní popis**
top	Zarovná text k horní hranici boxu.
hanging	Zarovná text k závěsné linii prvního dostupného fontu vně boxu. Používá se pro hindštinu.
middle	Zarovná text doprostřed, mezi zarovnání top a bottom.
alphabetic	Zarovná text k linii písma prvního dostupného fontu vně boxu.
ideographic	Zarovná text k ideografické linii prvního dostupného fontu vně boxu. Tato linie je používána pro hieroglyfy, petroglyfy.
bottom	Zarovná text k dolní hranici boxu.

Tabulka 9. Hodnoty atributu textBaseline.

Syntaxe 65. *context.textBaseline;*
Návratovou hodnotou je aktuální hodnota atributu.

context.textBaseline = value;
Nastaví hodnotu atributu.

Např.
var context = document.getElementById("myCanvas").getContext("2d");
context.textBaseline = "ideographic";



Obrázek 12. Ilustrace přípustných hodnot atributu textBaseline

Metoda *fillText* vykreslí text, který je předán argumentem text na souřadnice x a y. Tato metoda vykreslí pouze výplň. Argument maxWidth je zadáván v jednotkách souřadnicového systému bitmapy. Určuje maximální šířku, na kterou bude text upraven při překročení limitu. Výsledný vzhled textu je při vykreslení ovlivněn nastavením dříve uvedených atributů.

Syntaxe 66. *context.fillText(text, x, y [, maxWidth]);*

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
context.fillText("Ahoj", 0, 0, 200);
```

Metoda *strokeText* vykreslí obrys textu, který je předán v argumentu text na souřadnice x a y. Argument maxWidth je zadáván v jednotkách souřadnicového systému bitmapy. Určuje maximální šířku, na kterou bude text upraven při překročení limitu. Výsledný vzhled textu je při vykreslení ovlivněn nastavením dříve uvedených atributů. Opět se jedná o vykreslení obrysu, výsledek je tedy ovlivněn i atributy, které určují nastavení linie.

Syntaxe 67. *context.strokeText(text, x, y [, maxWidth]);*

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
context.strokeText("Canvas on UPOL", 0, 0, 200);
```

Metoda *measureText* má jako návratovou hodnotu objekt TextMetric. Při volání metody je vytvořen nekonečný řádek o šířce určené dle vlastností CSS. Tento řádek však ve skutečnosti neexistuje. Metoda hypoteticky zjistí délku tohoto řádku, kdybychom do něj vložili text z argumentu, a to bez zalomení řádků. Následně interpret vytvoří objekt TextMetric s atributem width. Do něj uloží informaci o tom, jak široký by předaný text byl. Atribut width pouze vrací svou hodnotu, nastavit jej lze pouze nepřímo metodou measureText.

Syntaxe 68. *context.measureText(text);*

Např.
var context = document.getElementById(”myCanvas”).getContext(”2d”);
var metric = context.measureText(”Canvas text width”);

Syntaxe 69. *metric.width;*

Např.
var context = document.getElementById(”myCanvas”).getContext(”2d”);
var metric = context.measureText(”Canvas text width”);
alert(”Šířka textu je: ” + metric.width + ” px”);

Obrázky

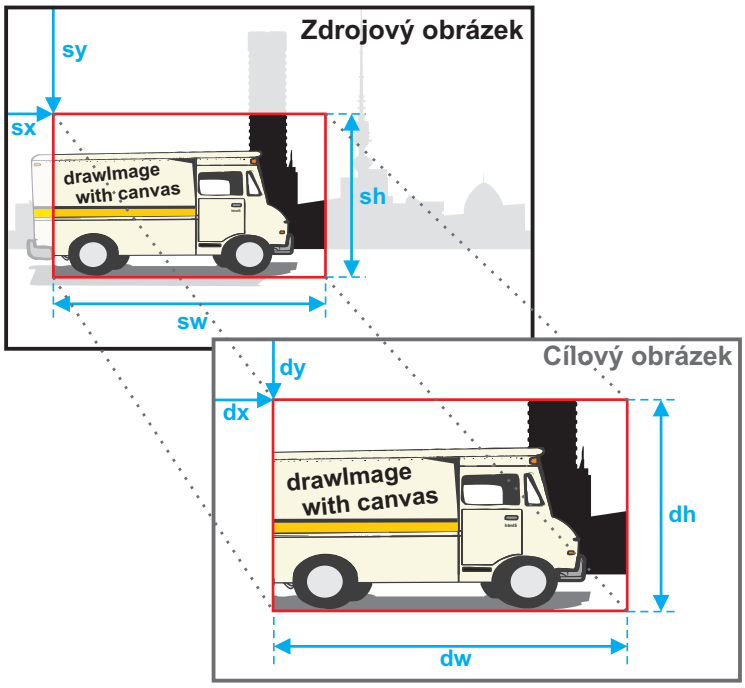
Pomocí kontextu máme možnost vykreslovat i obrázky. K tomu slouží jediná metoda, která je 3x přetížená.

- **drawImage(image, dx, dy)** - Vykreslí obrázek na dané souřadnice.
- **drawImage(image, dx, dy, dw, dh)** - Vykreslí obrázek na dané souřadnice se zadanými rozměry.
- **drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh)** - Vykreslí výřez obrázku o dané velikosti na souřadnice dx a dy.

Metoda *drawImage* vykreslí daný obrázek. Díky přetížení metody je možné obrázek před vykreslením ještě upravit. Význam a efekt jednotlivých argumentů určujících souřadnice a rozměry naleznete na obr. č. 13.

Zdrojem pro vykreslovaný obrázek může být HTMLImageElement, HTMLCanvasElement, nebo HTMLVideoElement. Pokusíme-li se použít jiný zdroj, dostaneme výjimku TYPE_MISMATCH_ERR. Pokud by předaný obrázek neobsahoval žádná data, došlo by k vyvolání výjimky INVALID_STATE_ERR. Pokud nejsou data zdroje kompletní, metoda nic neudělá.

Předáme-li metodě za zdroj HTMLCanvasElement, bude vždy použit aktuální obsah bitmapy, zobrazovaný v čase aplikace metody drawImage. Předáme-li metodě za zdroj HTMLVideoElement, bude vždy vzat rámec aktuální pozice přehrávání.



Obrázek 13. Ilustrace významu argumentů metody drawImage

Syntaxe 70. *context.drawImage(image, dx, dy);*

Např.

```
var context = document.getElementById("myCanvas").getContext("2d");
var image = new Image();
image.src = "url_zdroje";
context.drawImage(image, 0, 0);
```

Syntaxe 71. *context.drawImage(image, dx, dy, dw, dh);*

Např.

```
var context = document.getElementById("myCanvas").getContext("2d");
var image = new Image();
image.src = "url_zdroje";
context.drawImage(image, 0, 0, image.width - 50, 200);
```

Syntaxe 72. *context.drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh);*

Např.

```
var context = document.getElementById("myCanvas").getContext("2d");
var image = new Image();
image.src = "url_zdroje";
context.drawImage(image, image.width / 2, image.height / 2, 200, 100, 0, 0, 200, 300);
```

Manipulace s pixely

Kontext nám také umožňuje pracovat s jednotlivými pixely obrázku. To je vhodné zejména tehdy, chceme-li pro obrázek vytvářet filtry. Příkladem může být záměr převést barvy na odstíny šedi. K tomu stačí vypočítat průměr všech tří barevných složek a následně toto číslo nastavit jako hodnotu každé složky. Tento příklad naleznete ve zdrojovém kódu č. 6. na konci kapitoly.

K dispozici máme 4 metody kontextu pracující s objektem *ImageData*. Na tomto objektu můžeme pracovat se třemi atributy. Ty obsahují informace o obrázku, který je objektem ImageData reprezentován. Metodami kontextu pro manipulaci s pixely jsou:

- **createImageData(sw, sh)** - Vytvoří prázdný objekt ImageData o zadaných rozměrech.
- **createImageData(imagedata)** - Vytvoří prázdný objekt ImageData o stejných rozměrech, jaké má předaný objekt.
- **getImageData(sx, sy, sw, sh)** - Vytvoří objekt ImageData z obsahu vykresleného na bitmapě canvasu.
- **putImageData(imagedata, dx, dy[, dirtyX, dirtyY, dirtyW, dirtyH])** - Vloží na bitmapu obrázek s daty objektu ImageData.

Metoda *createImageData* slouží pro vytvoření nového objektu ImageData. Při jeho vytvoření jsou data nastavena tak, aby reprezentovala černou průhlednou výplň. Metoda je dvakrát přetížená. Jednou vytváří objekt o zadaných rozměrech pomocí argumentů sw a sh. Tyto argumenty jsou v jednotkách souřadnicového systému. Druhá verze si rozměry vypočítá z již předaných dat.

Je-li některým z argumentů nekonečno, nečíselná hodnota, nebo má-li metoda jeden argument a tím je *null*, dojde k vyvolání výjimky NOT_SUPPORTED_ERR. Je-li za argument šířky, nebo výšky předána nula, bude vyvolána výjimka INDEX_SIZE_ERR.

Musím však konstatovat, že podpora druhé verze je stoprocentní pouze v prohlížeči Safari. V ostatních prohlížečích její implementace není prozatím funkční, nebo je nepřesná a data nemusíte voláním metody získat.

Syntaxe 73. *context.createImageData(sw, sh);*

Např.

```
var context = document.getElementById("myCanvas").getContext("2d");
var imagedata = context.createImageData(200, 200);
```

Syntaxe 74. *context.createImageData(imagedata);*

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
var blankImageData = context.createImageData(250, context.canvas.height);
var imagedata = context.createImageData(blankImageData);
```

Metoda *getImageData* slouží k získání objektu ImageData. Jeho data reprezentují obrázek vykreslený na bitmapě canvasu, který se nachází vně zadaného čtyřúhelníku. Argumenty jsou v jednotkách souřadnicového systému.

Je-li hodnotou některého z argumentů nekonečno, nebo nečíselná hodnota, je vyvolána výjimka NOT_SUPPORTED_ERR. Je-li za argument šířky, nebo výšky předána nula, vyvolá prohlížeč výjimku INDEX_SIZE_ERR.

Syntaxe 75. *context.getImageData(sx, sy, sw, sh);*

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
var canvasImageData = context.getImageData(0, 0, context.canvas.width, context.canvas.height);
```

Metoda *putImageData* slouží k zapsání dat z objektu ImageData zpět na bitmapu. Data jsou vykreslena dle zadaných souřadnic, ty reprezentují levý horní roh bitmapy.

Volitelné argumenty, s předponou dirty, nám umožňují ze zadaných dat udělat pouze výřez. Tímto výřezem jsou data vně předaného čtyřúhelníku. Pokud je neuvedeme, jsou automaticky nastaveny na hodnotu nula.

Číselné hodnoty jsou opět v jednotkách souřadnicového systému. Je-li předána nečíselná, nebo nekonečná hodnota, dojde k vyvolání výjimky NOT_SUPPORTED_ERR. Pokud je za data obrázku předána hodnota *null*, nebo cokoli jiného než objekt ImageData, vyvolá prohlížeč výjimku TYPE_MISMATCH_ERR.

Syntaxe 76. *context.putImageData(imagedata, sx, sy);*

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
var canvasImageData = context.getImageData(0, 0, context.canvas.width, context.canvas.height);
context.canvas.width += 100;
context.canvas.height += 50;
context.putImageData(canvasImageData, 50, 25);
```

Syntaxe 77. *context.putImageData(imagedata, sx, sy, dirtyX, dirtyY, dirtyW, dirtyH);*

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
var canvasImageData = context.getImageData(0, 0, context.canvas.width, context.canvas.height);
context.putImageData(canvasImageData, 0, 0, 25, 25, 200, 100);
```

Objekt *ImageData* má při inicializaci nastavenu šířku a výšku dle zadání. Dále obsahuje data obrázku, která jsou reprezentována objektem *CanvasPixelArray*. Pokud již máme objekt ImageData, tedy referenci na něj, můžeme s ním pracovat s pomocí následujících atributů:

- **width** - Vrací aktuální šířku obrázku reprezentovaného daty objektu ImageData.
- **height** - Vrací aktuální výšku obrázku reprezentovaného daty objektu ImageData.
- **data** - Jednorozměrné pole obsahující data objektu ImageData.

Atributy *width* a *height* vrací hodnotu šířky a výšky obrázku. Atributy slouží pouze k získání hodnot, pokud se je pokusíte nastavit, nic se nestane. Hodnoty jsou nastaveny při vytvoření objektu ImageData.

Syntaxe 78. *ImageData.width;*

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
var canvasImageData = context.getImageData(0, 0, context.canvas.width ,context.canvas.height);
alert("Šířka obrázku je: " + canvasImageData.width) + " px";
```

Syntaxe 79. *ImageData.height;*

Např.
```
var context = document.getElementById("myCanvas").getContext("2d");
var canvasImageData = context.getImageData(0, 0, context.canvas.width ,context.canvas.height);
context.canvas.height += canvasImageData.height;
```

Atribut *data* slouží pro přístup k datům objektu ImageData. Voláním atributu získáme objekt CanvasPixelArray. Jedná se o jednorozměrné pole obsahující data jednotlivých pixelů. Každý pixel v poli zabírá 4 bajty. Ty reprezentují jednotlivé barevné složky spektra barev *rgba*. Každý bajt se skládá z 8-bitové hodnoty v rozsahu 0 - 255.

Pokud tedy chceme přistupovat ke složkám jednotlivých pixelů, vždy bereme 4 za sebou následující indexy. Těchto indexů je celkem *(výška * šířka * 4)*. Hodnotu na dané pozici můžeme získat voláním atributu s daným indexem. Přiřazením hodnoty můžeme tu původní změnit.

Syntaxe 80. *ImageData.data[index];*
Návratovou hodnotou je aktuální hodnota atributu na daném indexu.

ImageData.data[index] = value;
Nastaví hodnotu atributu na daném indexu.

Např.
var context = document.getElementById(”myCanvas”).getContext(”2d”);
var idata = context.getImageData(0, 0, context.canvas.width ,context.canvas.height);
alert(”Průhlednost posledního pixelu: ” + idata.data[idata.width * idata.height * 4]);

Zdrojový kód 6. *Převedení barevného obrázku do stupňů šedi*

```
// Tento příklad je plně funkční pouze v prohlížeči Safari
// Opera nemá podporu pro metodu getImageData
// Firefox a Chrome mohou mít problémy se čtením dat, kvůli
// vnitřnímu zabezpečení prohlížeče, proto data nemusí získat

// Pomocí knihovny jQuery vykoná funkci init() po načtení celé stránky
$(document).ready(function(){init();});


// Globální proměnné
var canvas;
var context;
var sourceImage;


// Základní nastavení po kompletním načtení dokumentu
function init(){
  canvas = document.getElementById("canvas");
  context = canvas.getContext("2d");

  // Vytvoříme nový objekt obrázku a vložíme do něj zdrojovou url
  sourceImage = new Image();
  sourceImage.src = "http://www.pisuweb.cz/images/canvas/desaturation.jpg";
}
```

```
// Funkce smaže canvas a vloží do něj originální obrázek
function reset(){
  canvas.width = canvas.width;
  context.drawImage(sourceImage, 0, 0);
}


// Funkce odbarví obrázek na stupně šedi
function grayscale(){

  // Načteme obrázek do bitmapy canvasu
  reset();

  // Získáme data aktuálního obrázku bitmapy
  var sourceData = context.getImageData(0, 0, canvas.width, canvas.height);

  // Vytvoříme prázdný obrázek o velikosti originálu
  var outputImage = context.createImageData(canvas.width, canvas.height);

  // Projdeme všechny pixely a vypočítáme průměr barev,
  // ten následně vložíme na stejnou pozici v prázdném
  // obrázku
  for(i = 0; i < sourceData.height; i++){
    for(j = 0; j < sourceData.width; j++){

      // Získáme posun v poli pro daný pixel
      var offset = (i * (sourceData.width * 4)) + (j * 4);

      // Získáme barevné složky pixelu a průhlednost
      var red = sourceData.data[offset + 0];
      var green = sourceData.data[offset + 1];
      var blue = sourceData.data[offset + 2];
      var alpha = sourceData.data[offset + 3];

      // Vypočítáme průměr barevných složek
      var desaturation = (red + green + blue) / 3;

      // Nastavíme odstín šedé pro pixel ve výsledném obrázku
      for(var k = 0; k < 3; k++){
        outputImage.data[offset + k] = desaturation;
      }

      // Průhlednost zachováme beze změny
      outputImage.data[offset + 3] = alpha;
    }
  }

  // Vložíme obrázek na bitmapu
  canvas.width = canvas.width;
  context.putImageData(outputImage, 0, 0);
}
```

4.3. Audio element

Element *audio* je určen pro vkládání zvukových záznamů do dokumentu. V současné době se za tímto účelem používají pluginově závislé technologie, nebo javascriptové frameworky. Tento element implementuje rozhraní *HTMLAudioElement*, které naleznete v příloze C. Přehrávání pak zajišťuje přehrávač, který je implementován prohlížečem, nebo autorem dokumentu.

Audio element je v dokumentu reprezentován párovým tagem *<audio>*. Veškerý obsah vepsaný mezi počáteční a ukončovací tag je skryt a zobrazen pouze tehdy, není-li element prohlížečem podporován. Tudíž jej opět můžeme označit za fallback content. Vyjímkou je element *source*, který bude vysvětlen dále v této kapitole.

Zvukový záznam elementu přiřadíme pomocí atributu src, jehož hodnotou je URL tohoto záznamu, nebo využitím source elementu. Při přehrávání záznamu musí být data synchronizována dle aktuální pozice přehrávání na časové ose přehrávače. Také musí dojít k modulaci hlasitosti v závislosti na nastavení v uživatelském rozhraní přehrávače.

Podpora prohlížečů pro audio element není úplná, proto zde uvádím seznam těch, které jej podporují alespoň částečně:

- Safari 2.0+
- Firefox 3.6+
- Chrome 3.0+
- Opera 10.0+

Syntaxe 81. *<audio> Fallback content </audio>*

Např.

```
<audio controls loop>
  <source src="audiostream.wma" type="audio/x-ms-wma">
</audio>
```

Audio element má jednou přetížený konstruktor, pomocí nějž můžeme vytvořit nový objekt typu HTMLAudioElement a s ním následně pracovat. Pokud konstruktor zavoláme bez argumentu, dojde k vytvoření nového objektu s prázdným zdrojem. Pokud jej zavoláme s argumentem, kterým je atribut src, vytvoříme nový element se zdrojem zvukového záznamu. Při vytvoření nového objektu HTMLAudioElement je atribut autobuffer nastaven na stejnojmennou hodnotu.

Syntaxe 82. *Audio([src]);*

Např.

```
var audio1 = new Audio();
audio1.src = "audiosource.mp3";
...
var audio2 = new Audio(audio1.src);
```

Musím zdůraznit, že audio element nemá vlastní atributy. Přebírá pouze globální atributy elementu *HTMLMediaElement*. Z toho dědí atributy src, autobuffer, autoplay, loop a controls. HTMLMediaElement, reprezentující všechny mediální elementy, bude spolu se svými atributy a metodami probrán v této kapitole.

Audio element má mediální charakter a je reprezentován pouze zvukovým záznamem bez obrazu. Je doporučením pro tvůrce HTML dokumentů, aby zvukový záznam byl dostupný i tehdy, není-li element prohlížečem podporován. Tedy formou odkazu na zvukový záznam, nebo použitím některé ze současných pluginově závislých technologií.

4.4. Video element

Element *video* slouží pro vkládání video záznamů do dokumentu. Přehrání tohoto záznamu je zajištěno přehrávačem implementovaným v prohlížeči. Aktuálně jsou k přehrávání používány pluginově závislé technologie a javascriptové frameworky. Prohlížeče implementují pro video element rozhraní HTMLVideoElement, které vychází ze zmíněného globálního rozhraní mediálních elementů. Rozhraní video elementu naleznete v příloze C.

Video element je reprezentován párovým tagem *<video>*. Veškerý obsah vepsaný mezi počáteční a ukončovací tag je zobrazen pouze v případě, není-li element prohlížečem podporován. Opět je zde vyjímka u elementu source, který je zpracován dle implementace prohlížeče.

Syntaxe 83. *<video> Fallback content </video>*

Např.

```
<video controls autoplay>
  <source src="videostream.mp4" type="video/mp4">
</video>
```

Podpora prohlížečů pro video element není úplná, proto uvádím seznam prohlížečů s alespoň částečnou podporou:

- Safari 2.0+
- Firefox 3.6+
- Chrome 3.0+
- Opera 10.0+

Rozhraní přebírá globální atributy pro média, kterými jsou src, autobuffer, autoplay, loop a controls. Navíc element vlastní pět atributů, které slouží k základnímu nastavení, těmi jsou:

- **width** - Určuje šířku přehrávače v dokumentu.
- **height** - Určuje výšku přehrávače v dokumentu.
- **videoWidth** - Obsahuje šířku přehrávaného obrazového záznamu.
- **videoHeight** - Obsahuje výšku přehrávaného obrazového záznamu.
- **poster** - Obsahuje URL obrázku, který je vykreslen jako úvodní snímek.

Atributy *width* a *height* slouží k nastavení rozměrů přehrávače video záznamů. Hodnotou obou atributů je nezáporné číslo, jehož jednotkou je pixel. Výchozí hodnotou je rozměr přehrávaného záznamu, je-li nějaký uveden. Pokud není, použije se rozměr obrázku z atributu poster. Není-li uveden ani ten, je výchozím nastavením 300px šířka a 150px výška.

Syntaxe 84. *HTMLVideoElement.width;*
Návratovou hodnotou je aktuální hodnota atributu.

HTMLVideoElement.width = value;
Nastaví hodnotu atributu.

Např.
```
var video = document.getElementById("myVideo");
if(video.width < 300)
  video.width = 300;
```

Syntaxe 85. *HTMLVideoElement.height;*
Návratovou hodnotou je aktuální hodnota atributu.

HTMLVideoElement.height = value;
Nastaví hodnotu atributu.

Např.
```
var video = document.getElementById("myVideo");
if(video.height < 150)
  video.height = 150;
```

Atributy *videoWidth* a *videoHeight* slouží pouze ke čtení a obsahují skutečný rozměr video záznamu v pixelech. Tyto hodnoty jsou využívány prohlížeči pro výpočet velikosti přehrávací plochy, do které je vykreslen přehrávač se záznamem.

Syntaxe 86. *HTMLVideoElement.videoWidth;*

Např.
```
var video = document.getElementById("myVideo");
alert("Šířka záznamu je: " + video.videoWidth);
```

Syntaxe 87. *HTMLVideoElement.videoHeight;*

Např.
```
var video = document.getElementById("myVideo");
alert("Výška záznamu je: " + video.videoHeight);
```

Atribut *poster* obsahuje, nebo nastavuje URL adresu obrázku. Ten je vykreslen do přehrávače nejsou-li k dispozici data video záznamu. Pokud je atribut poster nastaven, je přehrávání rozšířeno o tzv. *poster frame*. Tento rámec je přidán na začátek video záznamu jako jeho první snímek. Po začátku přehrávání videa, tedy kompletním načtení dat, nebude již poster frame vícekrát zobrazen. To ani v případě, že video zastavíme, nebo přehrávací pozici posuneme na začátek.

Syntaxe 88. *HTMLVideoElement.poster;*
Návratovou hodnotou je aktuální hodnota atributu.

HTMLVideoElement.poster = value;
Nastaví hodnotu atributu.

Např.
```
var video = document.getElementById("myVideo");
video.poster = "http://www.pisuweb.cz/images/html5.jpg";
```

Při přehrávání záznamu musí docházet k synchronizaci obrazového i zvukového záznamu spolu s aktuální pozicí přehrávání na časové ose. Pokud je aktuální pozicí první snímek záznamu, je na prohlížeči zda-li použije poster frame, nebo první snímek video záznamu. Je však doporučením, aby byl použit poster frame, a to pouze při prvním přehrávání záznamu. Při pozastavení přehrávání je vykreslen snímek záznamu dle aktuální pozice přehrávání. Není-li dostupný, například při postupném načítání dat záznamu, je vykreslen poslední načtený snímek.

Vzhled přehrávače není pevně stanoven a je na tvůrcích prohlížečů, aby zvážili rozložení ovládacích prvků. Nicméně GUI přehrávače může autor dokumentu vytvořit i sám.

4.5. Source element

Element *source* slouží pro vložení více záznamů do audio a video elementu. Sám o sobě se v dokumentu nijak neprojeví. Avšak vepíšeme-li jej do těla mediálních elementů *(audio a video)*, každý jeden tag bude reprezentovat jeden zdroj záznamu. Zobrazení těchto záznamů je otázkou implementace přehrávače.

Rozhraním pro implementaci tohoto elementu je *HTMLSourceElement*. Naleznete jej v příloze C. Do dokumentu je element vkládán nepárovým tagem *<source>*.

Syntaxe 89. *<source>*

Např.
<source src="videosource.mp4">

Rozhraní pro element source obsahuje tři atributy, které popisují vkládaný záznam. Těmito atributy, vkládanými do tagu, jsou:

- **src** - Slouží k vložení URL adresy záznamu.
- **type** - Slouží k vložení MIME-type daného záznamu.
- **media** - Obsahuje tabulku médií, pro která je zdroj určen.

Atribut *src* je jediným povinným atributem, který musí být vždy uveden. Zapisuje se přímo do tagu a musí obsahovat platnou URL adresu záznamu. Míním tím to, že musí jít o záznam ve formátu pro zvuk, či video.

Syntaxe 90. *<source src="value">*

Např.
<source src="videosource.mp4">

Atribut *type* upřesňuje formát vkládaného záznamu. Je nezbytný pro rozhodnutí, umí-li daný prohlížeč formát zpracovat. Hodnotou musí být validní *MIME type* pro média, viz. RFC 2045, RFC 2046, RFC 2047, RFC 2049, RFC 4288 a RFC 4289. Vně hodnoty je možné specifikovat kodeky pro přehrávání pomocí atributu *codecs*. Ten určuje způsob šifrování záznamu. Podpora kodeků se v jednotlivých prohlížečích liší, proto zde hodnoty neuvádím. Zájemci některé z nich naleznou v sekci *The source element* návrhu [1].

Syntaxe 91. *<source src="..." type="MIME-type; codecs='value'">*

Např.
<source src="videosource.mp4" type="video/mp4; codecs='avcl.42E01E, mp4.a.40.2'">

Atribut *media* slouží k zadání tabulky médií, která určuje, do jakých elementů může být záznam vložen. Defaultní hodnotou je řetězec "all". Nicméně atribut nemá v současné době podporu a jeho popis je nekompletní, tudíž jeho použití neuvádím.

4.6. Media element

Element *media* je jednou z nejdůležitějších novinek v HTML5. Ačkoli se sám v dokumentu nevyskytuje, můžeme na něj narazit při konfrontaci s elementy audio a video. Ty jsou vytvořeny pomocí rozhraní HTMLMediaElement, dále jen *media*, které naleznete v příloze C. Z něj následně audio i video dědí jak atributy, tak i metody určené pro přehrávání.

V dokumentu je media element reprezentován pomocí elementů audio a video. Tudíž je zapisován párovými tagy <audio> a <video>, které již byly popsány. Pro přístup k metodám a atributům media elementu stačí získat referenci na audio, či video objekt.

Syntaxe 92. *Získání přístupu k objektu HTMLMediaElement*
var media = document.getElementById("id_audio_nebo_video_elementu");

Např.
var media = document.getElementById("videoPlayer");
alert("Chyba načtení č. " + media.error.code);

Chybové kódy

Elementy mediálního typu si udržují číselný kód chyby, která nastala jako poslední při výběru zdroje záznamu. K identifikaci této chyby slouží jeden atribut volaný na objektu media.

- **error** - Vrací objekt MediaError.

Atribut *error* obsahuje chybový kód, uzavřený v objektu *MediaError*. Nastavení atributu zajistí prohlížeč při načtení záznamu. Ten je vybrán pomocí algoritmu pro výběr zdroje záznamu *(resource selection alghoritm)*, který bude vysvětlen později.

Při volání atributu je návratovou hodnotou, nastane-li chyba, objekt MediaError, který naleznete v příloze C. V opačném případě vrací atribut hodnotu null.

Syntaxe 93. *media.error;*

Např.
```
var audio = document.getElementById("audioRecord");
var mediaError = audio.error;
```

Objekt MediaError obsahuje čtyři konstanty a jeden atribut. Konstanty určují typ chyby a atribut obsahuje číslo poslední zaznamenané chyby.

- **code** - Atribut obsahující aktuální chybu.

Atribut *code* při volání vrací číslo aktuální chyby. Toto číslo obsahuje vždy, jelikož pokud chyba nenastala, nedošlo ani k vytvoření objektu MediaError. Tímto číslem může být jedna z konstant uvedených v tab. č. 10.

Název	**Hodnota**	**Slovní popis**
MEDIA_ERROR_ABORTED	1	Proces přenášení záznamu byl přerušen požadavkem uživatele.
MEDIA_ERROR_NETWORK	2	Nějaký, blíže nespecifikovaný, problém se sítí přerušil přenášení dat záznamu poté, co došlo k ověření zdroje.
MEDIA_ERROR_DECODE	2	Chyba při dekódování záznamu poté, co došlo k ověření zdroje.
MEDIA_ERROR_SRC_NOT_SUPPORTED	3	Záznam na dané URL adrese není vhodný. Může jít o blokovaný obsah, nebo špatný formát záznamu.

Tabulka 10. Přehled přípustných hodnot atributu code.

Syntaxe 94. *MediaError.code;*

Např.
```
var audio = document.getElementById("audioRecord");
if(audio.error != null)
  alert("Chyba č. " + audio.error.code);
```

Umístění zdroje záznamu

Atribut src mediálních elementů určuje URL adresu záznamu, který má být přehrán. Jak sem již zmínil, musí obsahovat validní URL. Při nastavení i změně URL záznamu dojde k vyvolání algoritmu pro nahrání záznamu *(media element load alghoritm)*. Ten bude vysvětlen později v této kapitole. Tento algoritmus není vyvolán při smazání src atributu.

Jelikož přehrávač může obsahovat více záznamů, s pomocí elementu source, mají elementy mediálního typu atribut obsahující aktuálně vybraný záznam. Avšak má-li media element zadán atribut src, je source element zastíněn.

- **currentSrc** - Vrací URL adresu aktuálního záznamu.

Atribut *currentSrc* obsahuje URL adresu aktuálně vybraného záznamu pro přehrávání. Atribut slouží pouze ke čtení, jeho nastavení provádí prohlížeč, a to při změně přehrávaného záznamu. Přesněji po zpracování algoritmu pro výběr záznamu.

Pokud není do mediálních elementů vložen žádný záznam, návratovou hodnotou při volání atributu je prázdný řetězec. V opačném případě atribut vrací URL adresu aktuálně vybraného záznamu formou řetězce.

Syntaxe 95. *media.currentSrc;*

Např.
```
var audio = document.getElementById("audioRecord");
if(audio.currentSrc != "")
  audio.play();
```

MIME type

Zdroj záznamu je popsán svým typem *(MIME type)* a volitelně doplněn specifikací kodeků dle internetového standardu RFC 4281.

Tento popis však není dostatečný k tomu, aby prohlížeč s jistotou určil, zda-li umí daný záznam přehrát. A to z důvodu, že MIME type a kodeky neobsahují veškeré nezbytné informace. Tudíž prohlížeč dokáže určit pouze to, že by měl být schopen daný záznam přehrát. Pro testování, dokáže-li prohlížeč záznam zpracovat a následně zobrazit, obsahuje objekt media jednu metodu.

- **canPlayType(type)** - Zjistí, zda-li je prohlížeč schopen přehrát záznam s daným MIME type.

Metoda *canPlayType* detekuje, zda-li prohlížeč dokáže přehrát záznam v uvedeném formátu. Ten je metodě předán jako argument formou řetězce. Návratovou hodnotou je řetězec, který může nabývat tří hodnot. Je-li prázdný, tak si prohlížeč s daným formátem záznamu neporadí. K této odpovědi dospěje tehdy, pokud s jistotou ví, že daný formát nepodporuje. Další možností je řetězec *probably*, určující, že daný formát by prohlížeč měl umět přehrát. Tedy je schopen záznam zpracovat pomocí audio, nebo video elementu. Tato možnost by měla být dostupná pouze tehdy, jsou-li zadány i kodeky. Poslední hodnotou je *maybe*. Ta je vrácena, pokud nenastanou předchozí 2 případy.

Syntaxe 96. *media.canPlayType(type);*

Např.
```
var video = document.getElementById("videoRecord");
var canPlay = video.canPlayType("vide/non-existing-format; codecs='none'");
if((canPlay != "") && (canPlay != "maybe"))
   video.play();
else{
   // Vložíme záznam do externího přehrávače
   ...
}
```

Stavy sítě

Důležitým faktorem při práci se zdrojem záznamu je připojení k síti. To z důvodu, že záznamy se nemusí vždy nacházet na stejném počítači, na kterém se je snažíme spustit. Media element má pro detekci stavu jeden atribut.

- **networkState** - Obsahuje informaci o stavu sítě.

Atribut *networkState* udržuje informaci o aktuálním stavu sítě. Při volání je návratovou hodnotou jedna z konstant uvedených v tab. č. 11. Nastavení této hodnoty provádí prohlížeč v průběhu algoritmu pro výběr zdroje záznamu.

Syntaxe 97. *media.networkState;*

Např.
```
var video = document.getElementById("videoRecord");
if(video.networkState == 1)
   video.play();
```

Název	Hodnota	Slovní popis
NETWORK_EMPTY	0	Media element ještě nebyl nastaven a všechny atributu mají výchozí nastavení.
NETWORK_IDLE	1	Algoritmus pro výběr zdroje záznamu je aktivní a vybral zdroj, ale aktuálně nevyužívá připojení k síti.
NETWORK_LOADING	2	Prohlížeč se snaží stáhnout data.
NETWORK_NO_SOURCE	3	Algoritmus pro výběr zdroje záznamu je aktivní, ale nepodařilo se nalézt zdroj.

Tabulka 11. Přehled přípustných hodnot atributu networkState.

Načtení zdroje záznamu

Načítání zdroje záznamu pro element media se skládá z několika fází. Ty jsou tvořeny třemi algoritmy, které se spustí při každém načtení a vložení nového zdroje pro záznam. Zmíněné algoritmy zajistí načtení media elementu, výběr záznamu a jeho získání. Tyto algoritmy naleznete v příloze A.

Media elementy mají příznak autoplay, jehož výchozí hodnotou je true. Dále mají příznak pro událost zpožděného načtení, ten je zpočátku nastaven na false. Pokud je nastaven na true, dojde ke zpožděnému načtení celého dokumentu, v němž je element obsažen. Důvody pro toto chování budou vysvětleny. K manuálnímu vyvolání nového načtení zdroje záznamu má objekt media jednu metodu.

- **load()** - Provede reset elementu a spustí nový výběr a načtení zdroje záznamu.

Metoda *load* při svém volání resetuje element media, na kterém je volána. Následně spustí algoritmus pro načtení elementu.

Syntaxe 98. *media.load();*

Např.
```
var video = document.getElementById("videoRecord");
video.load();
```

Kromě metody load mají media elementy 2 atributy ovlivňující načítání záznamu. Jedním je atribut media elementu vepisovaný přímo do tagu. Druhý vyjadřuje to samé, ale je volán na objektu media. Těmito atributy jsou:

- **autobuffered** - Určuje předpoklad, že bude záznam použit.
- **buffered** - Vytváří statický objekt TimeRanges.

Atribut *autobuffered* se vkládá do tagu, přičemž neobsahuje žádnou hodnotu. Pokud je v tagu přítomen, informuje prohlížeč o předpokladu autora dokumentu, že daný záznam bude použit. A to i tehdy, nemá-li přehrávač zapnuté automatické přehrávání pomocí atributu autoplay. V důsledku toho bude záznam načítán do vyrovnávací paměti. Pokud je automatické přehrání povoleno, atribut autobuffered je ignorován, jelikož je záznam načítán automaticky.

Syntaxe 99. *<media_element autobuffered>...</media_element>*

Např.
```
<video src="myVideo.mp4" autobuffered>
  ...
</video>
```

Atribut *buffered* při svém volání vrací nový objekt typu TimeRanges. Ten reprezentuje rozsah dat, která jsou přednačtena ve vyrovnávací paměti. Nastavení tohoto objektu zajišťuje prohlížeč v okamžiku volání atributu. Vysvětlení významu objektu TimeRanges bude následovat na konci kapitoly.

Syntaxe 100. *media.buffered;*

Např.
```
var video = document.getElementById("videoRecord");
var bufferedSize = video.buffered.length;
```

Časové údaje

Media element obsahuje také několik atributů, které slouží k reprezentaci důležitých časových údajů. Ty prohlížeč využívá k vykreslování grafického uživatelského rozhraní přehrávače. Těmito atributy jsou:

- **duration** - Vrací celkovou délku záznamu.

- **currentTime** [= **value**] - Vrací, nebo nastavuje, aktuální pozici přehrávání.
- **startTime** - Vrací časový údaj začátku záznamu.
- **loop** - Určuje opětovné přehrání záznamu.

Atribut *duration* při svém volání může vrátit 3 různé hodnoty. Pokud došlo k získání záznamu pomocí dříve popsaných algoritmů, je návratovou hodnotou časový údaj o délce záznamu v sekundách. Pokud by se jednalo o nekonečný datový proud, je hodnotou plus nekonečno, přesněji řetězec *Infinity*. Pokud není délka záznamu známa, návratovou hodnotou je řetězec *NaN*. Ten popisuje nečíselnou hodnotu. Délka záznamu je nezbytná k jeho přehrání. Pokud by nebyla známa, záznam nelze přehrát. Při změně délky záznamu je vyvolána událost *durationchange*.

Syntaxe 101. *media.duration;*

Např.
```
var audio = document.getElementById("audioRecord");
alert("Celková délka záznamu je: " + audio.duration + "s.");
```

Atribut *currentTime* vrací při volání čas aktuálně přehrávaného snímku v sekundách. Při nastavení hodnoty dojde ke změně pozice přehrávání na časové ose. Pokud je zadaný časový údaj platný v rámci záznamu, bude vytvořen náhled snímku na dané pozici. Pak můžeme pokračovat v přehrávání záznamu od dané pozice. Je-li zdrojem záznamu nekonečný proud dat, čas je brán v rámci úseku uloženého ve vyrovnávací paměti.

Pokud není vybrán záznam, dojde při volání atributu k vyjímce INVALID_STATE_ERR. V případě, kdy nově zadaný časový údaj není v rozsahu délky záznamu, je vyvolána vyjímka INDEX_SIZE_ERR.

Syntaxe 102. *media.currentTime;*
Návratovou hodnotou je aktuální hodnota atributu.

media.currentTime = value;
Nastaví hodnotu atributu.

Např.
```
var video = document.getElementById("myVideo");
if(video.currentTime > 3600)
  video.currentTime = 0;
```

Atribut *startTime* při svém volání vrací časový údaj o pozici záznamu, která je dostupná jako první v rámci celého záznamu. Tento údaj je v sekundách. Většinou se bude jednat o hodnotu 1, jsou-li však některá data porušena, ale ne závažně, může být začátek posunut například o 5 sekund. Hodnotou bude 0 pouze v případě, kdy záznam nemá žádná data.

Pokud je pozice přehrávání na časové ose před počátkem přehrávání, tedy prvním použitelným snímkem, dojde k vyvolání události *timeupdate*. Ta nastaví čas přehrávání na hodnotu atributu startTime.

Syntaxe 103. *media.startTime;*

Např.
```
var audio = document.getElementById("audioRecord");
alert("Začátek záznamu je v čase: " + audio.startTime + "s.");
```

Atribut *loop* je vepsán přímo do tagu daného media elementu. Jeho zadáním autor určuje, že jakmile záznam dosáhne konce, bude opětovně přehrán. Tedy atribut currentTime se nastaví na hodnotu atributu startTime.

Syntaxe 104. *<media_element loop>...</media_element>*

Např.
```
<audio src="mySong.mp3" loop>
  ...
</audio>
```

Připravenost záznamu

V popisu předchozích algoritmů jsem zmínil atribut, který specifikoval, je-li záznam připraven k přehrání. Tento atribut náleží objektu media. Tagy mediálních elementů navíc obsahují atribut určující automatické přehrání záznamu po načtení zdroje.

- **readyState**
- **autoplay**

Atribut *readyState* určuje aktuální stav media elementu. Přesněji, je-li element připraven přehrát vybraný záznam. Hodnotu atributu lze pouze získat, nastavení se provede v průběhu vykonávání předchozích algoritmů. Veškeré hodnoty tohoto atributu jsou uvedeny v tab. č 12. Upozorňuji, že návratovou hodnotou je číslo. Pokud je stav připravenosti media elementu změněn a zároveň stav sítě nemá hodnotu NETWORK_EMPTY, dojde k provedení následujících kroků:

- Pokud předchozím stavem byla hodnota HAVE_NOTHING a novým stavem je HAVE_METADATA, bude vyvolána událost loadedmetadata. Její vyvolání je součástí algoritmů spouštěných metodou load.
- Pokud předchozím stavem byla hodnota HAVE_METADATA a novým stavem je některá z vyšších hodnot, provede se následující:

    1. Došlo-li k této situaci poprvé od vyvolání algoritmů metodou load, prohlížeč je nucen zařadit do fronty úkolů vyvolání události loadeddata.
    2. Je-li novým stavem HAVE_FUTURE_DATA, nebo vyšší hodnota, dojde také k vykonání následujících kroků. V opačném případě vykonány nejsou.

- Byla-li předchozím stavem hodnota HAVE_FUTURE_DATA a nyní se změní na hodnotu HAVE_CURRENT_DATA, nebo nižší, může dojít k vyvolání události waiting. To záleží na aktuální pozici přehrávání, jelikož pokud záznam přehráván není, k této události nedojde.
- Byla-li předchozím stavem hodnota HAVE_CURRENT_DATA, nebo nižší, tak je-li nyní hodnotu HAVE_FUTURE_DATA, prohlížeč provede následující:

    1. Zařadí do fronty úkolů vyvolání události *canplay*.
    2. Pokud je záznam přehráván, zařadí se mezi úkoly vyvolání události *playing*.

- Je-li novým stavem HAVE_ENOUGH_DATA, vykonají se následující kroky:

    1. Pokud byl předchozím stavem HAVE_CURRENT_DATA, nebo nižší, dojde k vyvolání události canplay a pokud je záznam přehráván, tak i k události playing.
    2. Má-li media element nastaven příznak automatického přehrání, kdy atribut paused má hodnotu true a tag obsahuje atribut autoplay, může být atribut paused nastaven na false. Následně jsou vyvolány události play a playing. Automatické přehrání záznamu při dosažení tohoto stavu je volbou jednotlivých prohlížečů.
    3. Nakonec je vyvolána událost *canplaythrough*.

Název	Hodnota	Slovní popis
HAVE_NOTHING	0	Nejsou k dispozici žádná data záznamu, od jeho délky, až po aktuální pozici přehrávání.
HAVE_METADATA	1	K dispozici je dostatek informací pro určení délky záznamu, popřípadě jeho rozměrů. Avšak data záznamu pro aktuální pozici přehrávání nejsou dostupná.
HAVE_CURRENT_DATA	2	Data záznamu jsou pro aktuální pozici přehrávání dostupná. Nicméně data následující této pozici dostupná nejsou.
HAVE_FUTURE_DATA	3	Data záznamu jsou pro aktuální pozici přehrávání dostupná. Data následující této pozici dostupná jsou, ale nemusí jít o kompletní data pro celou zbývající část záznamu. Po skončení záznamu nemůže tento stav nastat.
HAVE_ENOUGH_DATA	4	Jsou splněny veškeré podmínky, které platí pro předchozí hodnotu. Navíc prohlížeč odhadl vykonání dat v čase o dané rychlosti přehrávání, pokud došlo k přenastavení atributu *defaultPlaybackRate*.

Tabulka 12. Přehled přípustných hodnot atributu readyState.

Syntaxe 105. *media.readyState;*

Např.
```
var audio = document.getElementById("audioRecord");
if(audio.readyState > 2)
  audio.play();
```

Atribut *autoplay* je zadáván přímo do tagu media elementu. Jde o booleovskou proměnnou, která svou přítomností detekuje, má-li být záznam automaticky spuštěn, a to co nejdříve je to možné.

Syntaxe 106. *<media_element autoplay>...</media_element>*

Např.
<audio src="mySong.mp3" autoplay>
...
</audio>

Přehrávání záznamu

K tomu, aby měl tvůrce dokumentu kontrolu nad vkládaným záznamem, má k dispozici několik nástrojů. Těmi jsou metody a atributy ovlivňující přehrávání záznamu. Metodami pro přehrávání záznamu, které se vztahují k objektu media, jsou:

- **play()** - Zruší stav pozastavení a po nahrání záznamu jej přehraje.
- **pause()** - Zastaví přehrávání a pokud je potřeba, tak i načítání záznamu.

Metoda *play* při svém volání spustí vykonání posloupnosti následujících kroků. Ty zajistí přehrávání záznamu, je-li to možné.

1. Má-li stav sítě hodnotu NETWORK_EMPTY, je spuštěn algoritmus pro výběr záznamu.
2. Pokud došlo k ukončení přehrávání, z důvodu dosažení konce záznamu, posune se pozice přehrávání na začátek záznamu. Dojde k vyvolání události *timeupdate*.
3. Pokud je atribut paused nastaven na hodnotu true, provedou se následující kroky:
   (a) Hodnota atributu se změní na false.
   (b) Vyvolání události play se zařadí mezi úkoly pro zpracování.
   (c) Pokud má atribut readyState hodnotu HAVE_NOTHING, HAVE_METADATA, nebo HAVE_CURRENT_DATA, zařadí se mezi úkoly pro zpracování vyvolání události waiting.
   (d) Pokud má atribut readyState hodnotu HAVE_FUTURE_DATA, nebo HAVE_ENOUGH_DATA, zařadí se mezi úkoly pro zpracování vyvolání události playing.
4. Příznak automatického přehrávání se nastaví na false.

Syntaxe 107. *media.play();*

Např.
```
var audio = document.getElementById("audioRecord");
if(audio.paused)
  audio.play();
```

Metoda *pause* při svém volání vykoná posloupnost následujících kroků, které zajistí zastavení přehrávání záznamu.

1. Má-li stav sítě hodnotu NETWORK_EMPTY, je spuštěn algoritmus pro výběr záznamu.
2. Příznak automatického přehrávání se nastaví na false.
3. Je-li hodnotou atributu paused nepravda, čili hodnota false, vykonají se následující kroky:
   (a) Hodnota atributu se změní na true.
   (b) Do fronty úkolů čekajících na vykonání se zařadí vyvolání události timeupdate.
   (c) Do fronty úkolů čekajících na vykonání se zařadí vyvolání události pause.

Syntaxe 108. *media.pause();*

Např.
```
var audio = document.getElementById("audioRecord");
if(audio.duration == 3600)
  audio.pause();
```

Kromě metod máme k dispozici i atributy, které ovlivní výsledné přehrávání, nebo nás informují o současném stavu. Těmito atributu jsou:

- **played** - Obsahuje informace o přehraném úseku záznamu.
- **paused** - Obsahuje informaci, je-li záznam pozastaven.
- **ended** - Obsahuje informaci, zda-li záznam skončil.
- **defaultPlaybackRate** [= **value**] - Určuje výchozí nastavení rychlosti pro přehrání záznamu. Defaultní hodnotu lze i nastavit.

- **playbackRate** [= **value**] - Vrací, nebo nastavuje, aktuální hodnotu rychlosti pro přehrávání záznamu.

Atribut *played* při svém volání vrací nově vytvořený objekt typu TimeRanges. Ten obsahuje informace o úseku záznamu, který již byl přehrán. Tento úsek se liší v závislosti na čase, kdy je volán. Jde vždy o novou hodnotu, vytvořenou od počátku přehrávání, až po dobu volání atributu played. Díky tomu můžeme přehrát pouze tento úsek, přičemž přesnější informace budou uvedeny na konci této kapitoly.

Syntaxe 109. *media.played;*

Např.
```
var video = document.getElementById("videoRecord");
var timeRange = video.played;
```

Atribut *paused* je booleovská proměnná určující, zda-li je přehrávání záznamu pozastaveno, nebo ne. Je-li záznam pozastaven, hodnotou je true. V opačném případě má atribut hodnotu false. Je-li hodnotou atributu true, záznam nemůže být přehráván. Taková situace by byla v rozporu s návrhem standardu.

Syntaxe 110. *media.paused;*

Např.
```
var video = document.getElementById("videoRecord");
alert("Záznam " + (video.paused == false ? "je" : "není") + " momentálně spuštěn.");
```

Atribut *ended* při svém volání vrací hodnotu true, pokud přehrávání dosáhlo konce záznamu. V opačném případě je návratovou hodnotou volání hodnota false.

Syntaxe 111. *media.ended;*

Např.
```
var video = document.getElementById("videoRecord");
if(video.ended){
  alert("Děkujeme, že jste shlédli náš dokument");
}
```

Atribut *defaultPlaybackRate* při svém volání vrací aktuálně nastavenou hodnotu, kterou lze změnit přiřazením hodnoty nové. Ta určuje rychlost, jakou bude záznam přehráván. Při změně tohoto atributu je vyvolána událost *ratechange*.

Tento atribut není prohlížečem běžně využíván. Jeho uplatnění přijde pouze v situaci, kdy uživatel záznam přehrává zpomaleně, nebo zrychleně. Pokud záznam dosáhne počátku, nebo konce, je rychlost přehrávání nastavena na hodnotu tohoto atributu.

Syntaxe 112. *media.defaultPlaybackRate;*
Návratovou hodnotou je aktuální hodnota atributu.

media.defaultPlaybackRate = value;
Nastaví hodnotu atributu.

Např.
```
var video = document.getElementById("myVideo");
if(video.defaultPlaybackRate > 1.5)
   video.defaultPlaybackRate = 1.0;
```

Atribut *playbackRate* určuje rychlost přehrávání záznamu. Při volání vrací svou aktuální hodnotu, kterou lze nastavit přiřazením. Není-li nastaven jinak, hodnotou je 1.0. Pokud se nová hodnota liší od výchozí, dojde k vyvolání události *ratechange*. Je-li hodnotou tohoto atributu 0.0, záznam je sice přehráván, ale vykreslen je pouze snímek na aktuální pozici přehrávání.

Syntaxe 113. *media.playbackRate;*
Návratovou hodnotou je aktuální hodnota atributu.

media.playbackRate = value;
Nastaví hodnotu atributu.

Např.
```
var video = document.getElementById("myVideo");
if(video.paused){
   video.playbackRate = 0.0;
   video.play();
}
```

Prohledávání

Pokud dojde ke změně aktuální pozice přehrávání na časové ose, musí být záznam prohledán. Následně prohlížeč testuje, zda-li je možné záznam v daném čase přehrát, tedy přejít k požadovanému času přehrávání. K tomu má objekt media následující atributy.

- **seeking** - Vrací informaci o tom, zda-li prohlížeč prohledává záznam.
- **seekable** - Vrací objekt TimeRanges s úsekem, který lze prohledat.

Atribut *seeking* obsahuje informaci o tom, zda-li prohlížeč aktuálně prohledává záznam. Výchozím nastavením je hodnota false. Pokud prohlížeč potřebuje prohledat nově zvolenou pozici přehrávání záznamu, pokračuje následujícími kroky. Během těch se nastaví i hodnota tohoto atributu.

1. Je-li hodnotou atributu readyState HAVE_NOTHING, je vyvolána vyjímka INVALID_STATE_ERR a další kroky už se nevykonají.
2. Je-li nová pozice přehrávání větší, než koncová pozice záznamu, nastaví se tato nová pozice na koncovou.
3. Je-li nová pozice přehrávání menší, než počátek záznamu, nastaví se na tento počátek.
4. Pokud nová pozice není v rozsahu udaném atributem *seekable*, je vyvolána vyjímka INDEX_SIZE_ERR a další kroky už se nevykonají.
5. Aktuální pozice přehrávání se nastaví na nově zvolenou.
6. Hodnota atributu seeking se nastaví na true.
7. K úkolům se přidá vyvolání události timeupdate.
8. Pokud je záznam přehráván a dojde ke změně pozice přehrávání a je-li readyState nastaven na HAVE_FUTURE_DATA, nebo nižší hodnotu, je vyvolána událost waiting.
9. Pokud jsme v tomto kroku, prohlížeč stále data záznamu prohledává, jelikož neví, zda-li jsou dostupná. Pokud prohlížeč prohledal dostatečné množství dat k přehrání, vyvolá se událost *seeking*.
10. Pokud pátrání bylo vyvoláno pomocí metod DOM, nebo nastavením atributu seeking, provedou se následující kroky. Ty jsou vykonány asynchronně.
    (a) Prohlížeč sečká, dokud nezjistí, že má dostatek dat pro přehrání záznamu na nové pozici a jsou dekódována.
    (b) Atribut seeking se nastaví na false.
    (c) Prohlížeč vloží do fronty úkolů pro zpracování vyvolání události *seeked*. To vede k tomu, že je tato událost vyvolána jakmile je to možné.

Syntaxe 114. *media.seeking;*

Např.
```
var video = document.getElementById("videoRecord");
alert("Záznam " + (video.seeking != false ? "je" : "není") + " momentálně prohledáván.");
```

Atribut *seekable* při svém volání vrací nový statický objekt typu TimeRanges. Ten u tohoto atributu představuje část záznamu, kterou je již možné prohledat. Pokud je za pozici přehrávání zvolena některá, která se shoduje s intervalem

připraveným k prohledání, záznam může automaticky pokračovat v přehrávání. V opačném případě se provedou kroky uvedené u atributu seeking s tím, že dojde k čekání na dostupnost dat záznamu. Při vytvoření objektu TimeRanges se použijí data dostupná v okamžiku volání. Je tedy pravděpodobné, že při vícenásobném volání se data budou lišit.

Syntaxe 115. *media.seekable;*

Např.
var video = document.getElementById("videoRecord");
var seekInterval = video.seekable;

Uživatelské rozhraní

K ovlivnění uživatelského rozhraní přehrávače má media element několik atributů. Konkrétně dva volané na tomto objektu a jeden vkládaný do těla tagu. Těmito atributy jsou:

- **controls** - Povolí ovládání přehrávače přes rozhraní prohlížeče.
- **volume** [= **value**] - Vrací, nebo nastavuje, hlasitost záznamu.
- **muted** [= **value**] - Vrací, nebo nastavuje, informaci, zda-li má záznam vypnutý zvuk.

Atribut *controls* je vkládán přímo do tagu media elementu. Jde o booleovskou hodnotu, která je nastavena přítomností v tagu. Pokud je atribut v tagu obsažen, říká prohlížeči, že má pro přehrávání zobrazit defaultní uživatelské rozhraní definované prohlížečem. Jestli v tagu uveden není, prohlížeč předpokládá, že autor vytvořil své vlastní uživatelské rozhraní pro ovládání přehrávače.

Pokud grafické uživatelské rozhraní vytváří prohlížeč, měl by přehrávač obsahovat prvky obsluhující alespoň následující úkoly:

- Spuštění a zastavení záznamu.
- Změna přehrávací pozice.
- Regulace hlasitosti.
- Zvětšení obrazu přes celou obrazovku.

Syntaxe 116. *<media_element controls>...</media_element>*

Např.
```
<audio src="mySong.mp3" controls>
  ...
</audio>
```

Atribut *volume* při svém volání vrátí aktuální hlasitost reprezentovanou číslem v rozsahu od 0.0 do 1.0. Nejnižší hodnota reprezentuje maximální ztlumení, intuitivně potom nejvyšší hodnota reprezentuje maximální zesílení. Tuto hodnotu je možné změnit pomocí grafického uživatelského rozhraní, nebo přiřazením nové hodnoty. Pokud je nově přiřazená hodnota mimo zmíněný rozsah, je vyvolána vyjímka INDEX_SIZE_ERR. V takovém případě se změna hodnoty neprojeví. Při změně hlasitosti pomocí tohoto atributu je do fronty úkolů, které čekají na vykonání, zařazeno vyvolání události *volumechange*.

Syntaxe 117. *media.volume;*
Návratovou hodnotou je aktuální hodnota atributu.

media.volume = value;
Nastaví hodnotu atributu.

Např.
```
var video = document.getElementById("videoRecord");
if(video.volume == 1.0){
  video.volume -= 0.4;
}
```

Atribut *muted* při svém volání vrací booleovskou hodnotu. Tou je true v případě, že zvuk je vypnutý. V opačném případě je hodnotou false. Výchozím nastavením je hodnota false, nicméně je volbou prohlížeče, zda-li použije poslední známé nastavení této hodnoty. Přiřazením nové hodnoty můžeme tu původní změnit. Učiníme-li tak, dojde k vyvolání události volumechange, na kterou musí prohlížeč zareagovat.

Syntaxe 118. *media.muted;*
Návratovou hodnotou je aktuální hodnota atributu.

media.muted = value;
Nastaví hodnotu atributu.

Např.
```
var video = document.getElementById("videoRecord");
if(video.muted){
  video.muted = false;
}
```

Časové úseky

Media element rozděluje přehrávaný záznam na jednotlivé úseky dle dostupných dat. K tomu používá rozhraní *TimeRanges*, které naleznete v příloze C. Objekt implementující toto rozhraní reprezentuje seznam jednotlivých dostupných časových úseků. Přístup k tomuto objektu získáme pomocí dříve zmíněných atributů played, buffered a seekable. Následně se získaným seznamem časových úseků můžeme pracovat pomocí jednoho atributu a dvou metod, které jsou uvedeny níže.

Během průzkumu návrhu standardu jsem narazil na několik článků, které zmiňovaly jednu nepravdivou informaci. Tou byla skutečnost, že bychom pouze s pomocí následujících metod a atributu mohli vytvořit z původního záznamu záznam nový. A to s předpokladem, že nový záznam by byl tvořen jedním z dostupných časových úseků. Pravdou je, že takovou funkcionalitu HTML5 momentálně neposkytuje a pravděpodobně ani poskytovat nebude. Toto rozhraní slouží převážně pro prohlížeč, který pomocí něj zjišťuje, zda-li jde danou část záznamu přehrát, přesunout se na ni, nebo jsou-li její dat ve vyrovnávací paměti. Pro autora představuje nástroj, kterým může přeskakovat chybějící, nebo doposud nestažená data záznamu.

Pro všechny úseky v objektu jsou definovány dvě základní pravidla, která musí být vždy dodržena.

1. Počátek úseku musí být větší než konec předchozího úseku.
2. Počátek úseku musí být menší než konec tohoto úseku.

Následující metody a atribut jsou momentálně implementovány pouze prohlížečem Safari. Ostatní prohlížeče je nepodporují, nebo je podporují jen částečně. Z toho plyne i současná použitelnost atributů, které objekt TimeRanges vytváří.

- **start(index)** - Metoda vrací počáteční čas úseku na dané pozici seznamu.
- **end(index)** - Metoda vrací koncový čas úseku na dané pozici seznamu.
- **length** - Tento atribut vrací počet úseků obsažených v objektu.

Metoda *start* při svém volání vrací pozici na časové ose, reprezentující počátek úseku záznamu na pozici zadané argumentem index. Je volána na objektu media, z nějž dostaneme objekt TimeRanges. Jednotkou návratové hodnoty jsou sekundy. Tento vrácený časový údaj je vztažen vzhledem k celé časové ose záznamu. Pokud je hodnotou argumentu index číslo větší než pozice posledního záznamu, je vyvolána vyjímka INDEX_SIZE_ERR.

Syntaxe 119. *media.start(index);*

Např.
```
var video = document.getElementById("videoRecord");
video.currentTime = video.buffered.start(video.buffered.length - 1);
video.play();
```

Metoda *end* při svém volání vrací pozici na časové ose, reprezentující konec úseku záznamu na zadané pozici. Je volána na objektu media, z nějž dostaneme objekt TimeRanges. Návratovou hodnotou je číselný údaj v sekundách. Ten je vztažen vzhledem k celé časové ose záznamu. Pokud by hodnotou argumentu index bylo číslo větší než pozice posledního záznamu, bude vyvolána vyjímka INDEX_SIZE_ERR.

Syntaxe 120. *media.end(index);*

Např.
```
var audio = document.getElementById("audioRecord");
audio.currentTime = audio.buffered.end(audio.buffered.length - 1);
audio.pause();
```

Atribut *length* při svém volání, na objektu media, vrací počet úseků v získaném objektu TimeRanges. Pomocí něj získáme rozsah přípustných hodnot argumentu index pro předchozí metody.

Syntaxe 121. *media.length;*

Např.
```
var audio = document.getElementById("audioRecord");
var range = audio.buffered;
if(range != 0){
   var index = Math.floor(range.length / 2);
   audio.currentTime = range.start(index);
   audio.play();
}
```

5. Co dalšího HTML5 ještě nabídne

Kdyby zmíněné novinky měly být tím jediným, co HTML5 přinese, mohl by být návrh předložen ke schválení. Avšak jeho robustnost je markantní, a tak není divu, že můžeme očekávat více změn, které nám ulehčí tvorbu HTML dokumentů ještě více. Některé z nich jsou již kompletně navrženy, ale ještě nemají podporu současných prohlížečů. U jiných zase prohlížeče čekají, až budou kompletně dodefinovány.

5.1. Ukládání obsahu pro režim offline

Převratným nápadem je možnost ukládat část obsahu webových aplikací na lokální počítač klienta. Jedná se o uložení celých dokumentů i s daty z databází. Pokud se klient pokouší připojit na stránku v režimu offline, je možné načíst data, která byla přenesena z databáze na serveru, do databáze ve vyrovnávací paměti prohlížeče. To umožní klientovi např. výběr zboží v internetovém obchodě i tehdy, nemá-li připojení k Internetu. Jelikož je možné informace ukládat do databáze, může si klient připravit objednávku se všemi údaji předem a po přechodu do režimu online objednávku odeslat.

Tvůrce dokumentu potom určuje, která data budou ze serveru stažena a uložena ve vyrovnávací paměti. To definuje pomocí tzv. *manifestu*, který vkládá do HTML dokumentu. Ten je umístěn za tagem <html>. Části, které nejsou v manifestu povoleny pro režim offline budou dostupné pouze online. V současné době není tato možnost podporována, s vyjímkou prohlížeče Safari. Ten ji implementuje dle původního návrhu ukládání offline aplikací, který byl navržen skupinou WHATWG. Od té doby byl však návrh již několikrát přepracován a dokumentace pro ukládání dat do databáze v offline režimu ještě není kompletní. Tato funkcionalita byla již přesunuta do vlastní specifikace, na kterou se HTML5 pouze odkazuje.

5.2. API pro geolokaci

Dalším obohacením má být nové API pro geolokaci, které vychází z původního návrhu od společnosti Google, Inc. Pomocí něj bychom měli mít možnost zjistit svou aktuální pozici. Ta bude vrácena jako řetězec obsahující zeměpisnou šířku a výšku. Pomocí těchto údajů budeme mít možnost zobrazit získanou pozici na mapě.

Aktuální pozice bude získávána pomocí GPS, Wifi zařízení, IP paketů a signálu rádiových stanic. Jelikož např. IP paket při své cestě prochází množstvím zařízení, která mohou aktuální polohu zkreslit, nemusíme vždy získat přesnou polohu. V současné době je tato funkcionalita dostupná jako experiment společnosti Google, Inc.

5.3. Editace částí dokumentu

Editace částí dokumentu, tedy možnost přepsat část dokumentu přímo v prohlížeči, je další novinkou. Toho dosáhneme pomocí nového atributu *contenteditable*, jehož hodnotou je true, nebo false. Pokud editaci povolíme, je možné přepsat text zobrazený v daném elementu. Nicméně tyto změny nejsou trvalé, pokud autor dokumentu nesestrojí mechanizmus, který by provedené změny uložil.

Doposud je tato možnost použitelná pouze v prohlížečích Firefox a Safari, a to u elementů div a p. V návrhu standardu ještě není tato funkcionalita dodefinována. Uvažuje se o rozšíření zápisu atributu na celý dokument a veškeré elementy, v nichž se nachází text.

5.4. Drag a drop

Na mnoha profesionálních webových stránkách se dnes setkáváme s boxy, které můžeme libovolně uspořádat. Velmi příjemným způsobem je potom uchopení takového boxu myší a jeho následné přetažení na požadované místo. V současnosti se k docílení tohoto efektu využívá AJAX a velmi rozsáhlé algoritmy pro výpočet přesunu.

Do budoucna HTML5 počítá s definováním událostí a atributů, které by nám umožnily algoritmy zjednodušit. Prozatím však nabízí pouze velmi malou ukázku toho, co můžeme očekávat. Ta zahrnuje atributy definující, může-li být daný element přesunut a události pro zpracování tohoto přesunu. Některé z možností jsou experimentálně implementovány v prohlížečích Safari a Firefox.

5.5. Webové formuláře

Již nějakou dobu se vytváří nová specifikace webových formulářů označovaná jako *Web Forms 2.0*. Ta vnáší do světa ponurých formulářových prvků pestrost, která jim momentálně chybí. Jelikož se jedná o velký krok kupředu, je specifikace formulářů zahrnuta do HTML5 a ještě vylepšena.

K současným možnostem přibudou prvky pro zadávání emailu a telefonního čísla s vestavěnou validací. Dále paleta pro výběr barvy, číselný posuvník pro výběr čísla v určitém rozmezí a kalendář pro výběr data a času. Některé z těchto prvků jsou již v dnešní době experimentálně implementovány prohlížeči Opera, Safari, Firefox a Chrome. Avšak jelikož standard ještě není dokončen, není vyloučeno, že jejich chování bude poupraveno. Je možné, že se mezi těmito novými prvky objeví i další, ostatně veškeré nápady na vylepšení jsou editory návrhu HTML5 vítány.

Závěr

Cílem této bakalářské práce byl průzkum současných možností připravovaného návrhu HTML5, jakožto nového standardu pro tvorbu webových aplikací. Mým záměrem bylo zjistit, se kterými vlastnostmi jazyka můžeme již dnes experimentovat. Postupně jsem odhalil podporu současných prohlížečů pro nové prvky a způsob práce s nimi. Především jsem se zaměřil na nové elementy jazyka a jejich přínos vzhledem k současnému standardu. Podstatnou část práce pak tvoří canvas element a jeho aplikační rozhraní.

Druhým cílem mé práce bylo vytvoření online webového tutoriálu k elementu canvas. Ten naleznete na [2], nebo offline verzi na přiloženém CD. Popisuji zde způsob práce s canvasem, jeho výhody a současná omezení. Tutoriál obsahuje velké množství příkladů a také interaktivní ukázky kódu, které lze dle libosti upravit a pozorovat výsledek jejich aplikace.

HTML5 je stále ve fázi příprav a v průběhu mé práce byl návrh již několikrát pozměněn. Například aplikační rozhraní canvas elementu bylo vloženo do vlastní specifikace a návrh HTML5 se na tuto specifikaci pouze odkazuje. Podobných změn můžeme očekávat ještě mnoho, jelikož předpokládaný rok pro dokončení návrhu je 2012. Nicméně W3C předpokládá, že HTML5 se stane doporučením až v roce 2022. Do té doby může každý jednotlivec vývoj ovlivnit, jelikož editoři uvítají veškeré konstruktivní myšlenky, nebo informace o nedostatcích a chybách.

Osobně patřím mezi část populace, která je této specifikaci nakloněna. Myslím si, že současný stav HTML dokumentů na Internetu je chaotický. Pokud se tedy do budoucna nechceme probírat množstvím špatně přístupných dokumentů, je potřeba zpřísnit pravidla jejich tvorby. Věřím, že současný návrh HTML5 je prvním krokem k této nápravě.

Conclusions

The aim of this BSc. thesis was a survey of currently upcoming HTML5 draft as a new standard for web applications. My intentions were to find out which features of the draft we can use nowadays. I have revealed current support of web browsers for a new features and a way how to work with them correctly. First of all I focused on new elements and their contribution to current standard. The biggest part belongs to the canvas element and its application interface.

The second aim was to create an online web tutorial for the canvas element. This tutorial is situated on the [2] or you can use its offline version which the enclosed CD contains. I have described there the way how to work with the canvas, its advatages and current limits. Tutorial contains many examples and also interactive samples of source code which could be edited at will. When you change the code of this samples you can see the result immediately.

Formation of this draft is still in progress and during my work on this thesis the draft has been altered several times. For example the canvas application interface was dislocated to independent specification and the current draft possesses only link label to this specification. We can expect further revisions of the draft since this draft is supposed to be finalized in 2012. However W3C consortium expects that HTML5 will have became a recommendation by the year 2022. Until then everybody can affect development of this draft because editors are glad for any constructive thoughts or data about deficiencies and errors.

I personally belong to the part of the population who wish well towards this specification. I think that current state of HTML documents on Internet is chaotic. If we do not want to go through a big amount of not well formed documents in the future, it is necessary to tighten up the rules of their creation. I bealive that current HTML5 draft is the first step to this retrieval.

Reference

[1] HTML5 [Internet], W3C, *22.leden 2008 [citována verze 2.prosinec 2009]* http://www.w3.org/TR/html5/

[2] Píšu web: Canvas tutoriál [Internet], Martin Bargl, 2009 http://www.pisuweb.cz/canvas/

[3] HTML: Tvorba dokonalých WWW stránek, Jiří Kosek, Grada Publishing, 1998

[4] Jak psát web: o tvorbě Internetových stránek [Internet], Dušan Janovský http://www.jakpsatweb.cz/

[5] W3C XML Specification DTD [Internet], W3C, *7.dubna 1998 [citována verze 10.října 1998]* http://www.w3.org/XML/1998/06/xmlspec-report-19980910/

[6] Full Web Building Tutorials [Internet], Refsnes Data, *1999* http://www.w3schools.com/

[7] Wikipedia: The free encyklopedia [Internet], Wikipedia Foundation, Inc., *článek Miter Joint* http://en.wikipedia.org/wiki/Miter_joint

A. Algoritmy pro zpracování mediálních záznamů

Pro svou práci používá media element tři algoritmy, které zajistí načtení media elementu, výběr záznamu pro přehrávání a jeho získání. Těmito algoritmy jsou:

- **Algoritmus načtení elementu**
- **Algoritmus výběru zdroje**
- **Algoritmus získání zdroje**

Algoritmus *načtení elementu* je vyvolán automaticky při načtení dokumentu, nebo volání metody load. V průběhu vykonávání algoritmu může být přerušen, a to opětovným voláním metody load. Vykonávání algoritmu probíhá v několika následujících krocích:

1. Přeruší se veškeré běžící instance algoritmu pro výběr zdroje, které jsou spuštěny na daném elementu.
2. Jsou-li spuštěny nějaké úkoly v rámci daného media elementu, nebo jsou-li ve frontě úloh čekajících na zpracování, dojde k jejich zrušení. Většinou je tak učiněno při načítání nového zdroje, kdy jsou přerušena veškerá zpětná volání a nedokončené obsluhy událostí.
3. Pokud je stavem sítě hodnota NETWORK_IDLE *(hodnota 1)*, nebo NETWORK_LOADING *(hodnota 2)*, dojde k zařazení nového úkolu do fronty úkolů. Tím je vyvolání události zvané *abort*, jež je vyvolána na daném media elementu.
4. Je-li hodnotou stavu sítě NETWORK_EMPTY *(hodnota 0)*, algoritmus neúspěšně končí. V opačném případě je spuštěn proces získání zdroje. Pokud jeho instance už běží, je přerušena. Atribut networkState je následně nastaven na hodnotu NETWORK_EMPTY a atribut readyState na hodnotu HAVE_NOTHING, pokud ji již neobsahuje. Je-li nastavením atributu *paused* hodnota false, bude změněna na true. Podobně je-li atribut *seeking* nastaven na true, je změněn na false. Aktuální pozice přehrávání, zobrazená na časové ose, je nastavena na hodnotu 0. Nakonec je do fronty úloh zařazeno vyvolání události *emptied*.
5. Atribut *playbackRate* je nastaven na hodnotu atributu *defaultPlaybackRate*.

6. Atributu error je přiřazena hodnota null. A příznak automatického přehrávání je nastaven na true.

7. Provede se volání algoritmu pro výběr zdroje záznamu.

8. Přehrávání předchozích záznamů v daném elementu je zastaveno.

Úkolem algoritmu pro *výběr zdroje záznamu* je nalézt mezi možnými zdroji jeden, který bude následně přehrán. Část algoritmu je vykonávána v synchronním módu a část v asynchronním. Ta běží paralelně na pozadí spolu s dalšími skripty. Algoritmus komunikuje také se *smyčkou událostí*, která řídí a vykonává události, zpětná volání, obsluhuje uživatelské rozhraní atp. Tato smyčka spouští jednotlivé synchronní sekce algoritmu, ten je vykonáván v několika následujících krocích.

1. Atribut networkState se nastaví na NETWORK_NO_SOURCE *(hodnota 3)*.

2. Následně je v asynchronním módu očekáván adekvátní stav atributu networkState. Když takový stav nastane, algoritmus pokračuje synchronně, dokud nedojde k ukončení synchronní sekce.

3. Synchronně se zjistí mód pro zpracování. Má-li media element atribut src nastaven, bude mód nastaven na hodnotu *attribute*. Pokud media element nemá src atribut, ale má ve stromové struktuře DOM za následníka element source, je mód nastaven na hodnotu *children*. Jde o následníka elementu, kdy kandidátem pro zdroj se stává první source element ve stromové struktuře. Pokud nemá media element atribut src a ani následníka jímž by byl source element, nastaví se atribut networkState na hodnotu NETWORK_EMPTY. Dále dojde k ukončení synchronní sekce vykonávání algoritmu.

4. Příznak události zpožděného načítání se nastaví na true a atribut networkState na hodnotu NETWORK_LOADING.

5. Do fronty úkolů čekajících na zpracování se zařadí vyvolání události *loadstarted*.

6. Je-li módem pro zpracování hodnota attribute, jsou vykonány tyto kroky:

   (a) Nastaví se absolutní URL adresa, ta je získána analýzou URL adresy specifikované pomocí atributu src. Tento krok je vykonán pouze tehdy, došlo-li ke změně adresy v rámci daného media elementu.

   (b) Pokud byla absolutní URL adresa získána, stane se hodnotou atributu currentSrc.

(c) Dojde k ukončení synchronní sekce vykonávání algoritmu. Následující kroky jsou prováděny asynchronně.

(d) Byla-li absolutní URL adresa získána úspěšně, je vyvolán algoritmus získání zdroje záznamu. Tomu je tato absolutní URL předána. Pokud algoritmus získání zdroje přeruší tento algoritmus, vše je v pořádku. V opačném případě došlo k chybě při stahování zdroje.

(e) Pokud algoritmus dojde k tomuto kroku, nastala chyba při načtení zdroje, nebo předaná URL adresa nebyla korektní. V takovém případě je vytvořen objekt MediaError s atributem code nastaveným na hodnotu MEDIA_ERROR_SRC_NOT_SUPPORTED.

(f) Atribut networkState media elementu je nastaven na hodnotu NETWORK_NO_SOURCE.

(g) Do fronty úkolů pro zpracování je zařazeno vyvolání události *error*.

(h) Příznak události zpožděného načítání je nastaven na false, čímž je událost zpožděného načítání zablokována.

(i) Vykonávání algoritmu je přerušeno do doby, kdy bude změněn atribut src, nebo dojde k manuálnímu vyvolání algoritmu metodou load.

7. Je-li hodnotou módu children, jsou vykonány tyto kroky:

(a) Je nastaven ukazatel, který je definován pro dva sousední uzly stromové struktury vně media elementu. Ukazatel se nachází mezi uzly, kdy jeden uzel je označován jako uzel předcházející ukazateli *(je před ním)* a druhý uzel je uzlem následujícím ukazatel *(je za ním)*. Na počátku je ukazatel za uzlem kandidáta a před následujícím uzlem, pokud takový existuje. Při odebírání a vkládání uzlů do media elementu dochází ke změně pozice ukazatele. Při vložení nového uzlu mezi dva uzly kterými je definován ukazatel, dojde ke změně. Uzel předcházející ukazateli je zachován. Uzlem následujícím ukazatel se stane nově vložený uzel. Je-li uzel předcházející ukazateli odstraněn, dojde k posunu ukazatele. A to tak, že předcházejícím uzlem se stane uzel následující ukazatel a jeho náhradníkem se stane uzel, který jej následuje. Je-li uzel následující ukazatel odstraněn, ukazatel zůstává na svém místě. Novým následujícím uzlem se stane uzel, který je za odstraněným uzlem. Jakékoli jiné změny ukazatel neovlivní. Znázornění zmíněných změn naleznete na obr. č. 14.

(b) **Zpracování kandidáta:** V tomto kroku je zpracován předaný kandidát. Pokud nemá atribut src, dojde k ukončení synchronní sekce. Algoritmus pokračuje krokem (j), kterým je chybový stav.

(c) Nastaví se absolutní URL adresa na adresu získanou analýzou src atributu kandidáta. A to pouze tehdy, došlo-li ke změně hodnoty src atributu, nebo kandidáta.

(d) Pokud se absolutní URL adresu nepodařilo získat, synchronní sekce je ukončena a algoritmus pokračuje krokem (j), což je opět chybový stav.

(e) Má-li kandidát nastaven atribut type na hodnotu, u které prohlížeč ví, že ji neumí zpracovat, je synchronní sekce ukončena. Následně algoritmus přejde do chybového stavu, kterým je krok (j).

(f) Pokud se media atribut kandidáta neshoduje s prostředím defaultního zobrazení, je synchronní sekce ukončena a přejdeme ke kroku (j).

(g) Atribut currentSrc je nastaven na absolutní URL adresu získanou z kandidáta.

(h) Synchronní sekce je ukončena, další kroky jsou prováděny asynchronně.

(i) Je spuštěn algoritmus pro získání zdroje s absolutní URL adresou. Pokud skončí a tento algoritmus není přerušen, došlo k chybě při načítání. V takovém případě pokračujeme následujícími kroky.

(j) **Chybový stav:** Do fronty úkolů čekajících na zpracování se zařadí požadavek na vyvolání události *error* pro element zvoleného kandidáta.

(k) Asynchronně je očekáván stabilní stav sítě. Poté synchronně vykonává následující kroky, dokud není synchronní sekce opět ukončena.

(l) **Hledání dalšího kandidáta:** Kandidáta nastaví na hodnotu null.

(m) **Vyhledávací smyčka:** Je-li uzel následující ukazatel koncový, přejde na krok (q). Ten je označen jako čekání.

(n) Jestliže uzlem následující ukazatel je element source, je zvolen novým kandidátem pro zdroj záznamu.

(o) Ukazatel je posunut. Uzlem předcházejícím ukazatel je nyní uzel, který ukazatel následoval. Novým následujícím uzlem je potom uzel, jenž byl za původním následujícím uzlem, existuje-li takový.

(p) Je-li kandidátem hodnota null, přejde algoritmus opět do vyhledávací smyčky, tedy krok (m). V opačném případě algoritmus přejde do kroku zpracování kandidáta, což je krok (b).

(q) **Čekání:** Nastaví atribut networkState na hodnotu NETWORK_NO_SOURCE.

(r) Nastaví příznak události zpožděného načítání na false, čímž zamezí vyvolání události zpožděného načítání.

(s) Ukončí synchronní sekci a následující kroky provádí asynchronně.

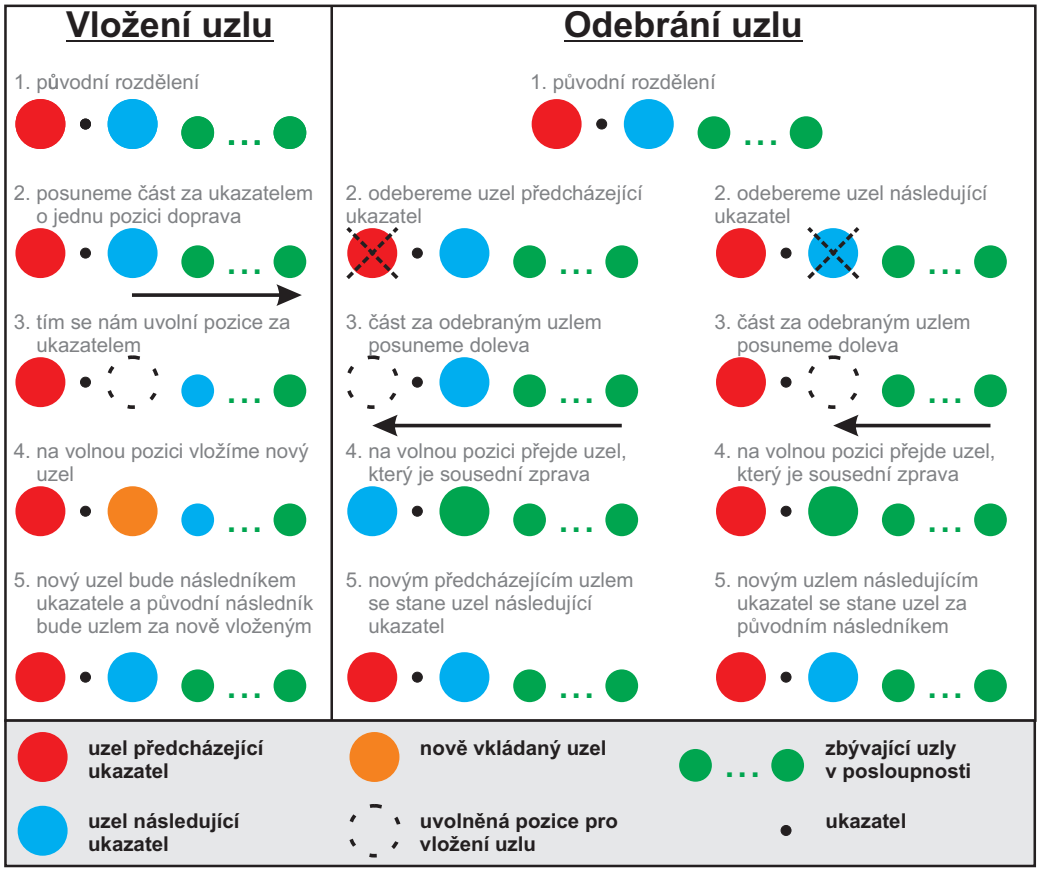
(t) Algoritmus čeká dokud uzel následující ukazatel není koncovým uzlem v seznamu uzlů. Tento krok nemusí nikdy skončit.

(u) Asynchronně očekává stabilní stav. Zbývající kroky pak provede v synchronním režimu, dokud není ukončena synchronní sekce.

(v) Nastaví příznak události zpožděného načítání na true, to vyvolá událost zpožděného načítání, pokud již nebyla vyvolána.

(w) Nastaví atribut networkState na hodnotu NETWORK_LOADING.

(x) Přejde do kroku hledání dalšího kandidáta, jde o krok (l).



Obrázek 14. Algoritmus pro výběr zdroje - modifikace uzlů

Algoritmus *získání zdroje* je spouštěn pro daný element, na kterém je vyvolán algoritmem pro výběr zdroje. Jedinou hodnotou, která je mu k činnosti předána, je absolutní URL adresa získaná pomocí předešlého algoritmu. Získání zdroje probíhá v následujících krocích.

1. Aktuálním zdrojem záznamu je zvolen soubor, který se nachází na zmíněné URL adrese.

2. V tomto kroku dojde k získání zdroje záznamu. Při každém přijatém bajtu, nebo každých 350ms (±200ms), dojde k vyvolání události *progress*. Ta indikuje to, že algoritmus stále probíhá a nedostal se do stavu, že by nic nedělal. V opačném případě, pokud by nepřijímal žádná data v rozmezí 3s (±200ms), dojde k vyvolání události *stalled*. Uživatel má možnost proces získání zdroje zpomalit, nebo dokonce zablokovat. Snížení rychlosti pro stahování zdroje může vyvolat i prohlížeč, usoudí-li, že je to vhodné. To má význam při získávání více záznamů najednou, aby nedošlo k přehlcení přenosové šířky pásma. Prohlížeč se v libovolném čase stahování záznamu může rozhodnout stahování odložit a čekat na uživatele, dokud nespustí přehrávání. To například, uloží-li si do vyrovnávací paměti 20% z celého záznamu. Poté očekává reakci uživatele, na jejímž základě se rozhodne, zda-li ve stahování pokračovat. Pokud se rozhodne stahování odložit, nastaví atribut networkState na NETWORK_IDLE a vyvolá událost *susspend*. Když následně ve stahování záznamu pokračuje, nastaví networkState na NETWORK_LOADING. Jestliže prohlížeč stahování zastaví úplně a čeká, než uživatel záznam spustí, nastaví se příznak události zpožděného načítání na false. K získání zdroje záznamu může prohlížeč využít veškerých možných alternativ *(resetovat připojení k síti, použít HTTP žádosti, nebo změnit směrovací protokol)*. Pokud se předchozí neprojeví a prohlížeč může pokračovat v procesu stahování záznamu, vykonají se následující kroky.

   - Nelze-li získat záznam kvůli problémům v síti, čímž prohlížeč přeruší proces získávání dat. Nebo prohlížeč zjistí, že záznam s daným typem a kodeky neumí zpracovat. Nebo pokud formát zdroje záznamu není podporován, či jej nelze zpracovat, provede se následující:
     (a) Prohlížeč zruší stahování záznamu.
     (b) Tento algoritmus bude přerušen a řízení bude předáno zpět algoritmu pro výběr zdroje.
   - Je-li získáno dostatečné množství dat záznamu, určí se délka záznamu, metadata a rozměr v případě video elementu. Získáním těchto dat prohlížeč ví, že záznam je možné použít. Potom pokračuje následujícími kroky:
     (a) Aktuální pozice přehrávání na časové ose přehrávače se nastaví na nejnižší možnou hodnotu. Tou by měl být časový úsek *00:00*. V případě video elementu jde o poster frame, nebo první snímek záznamu.
     (b) Atribut readyState se nastaví na hodnotu HAVE_METADATA.
     (c) Je-li media elementem video, nastaví se hodnota atributů videoWidth a videoHeight tak, jak bylo uvedeno v kapitole 4.4.
     (d) Atribut *duration* se nastaví na délku záznamu. Dojde k vyvolání události *durationchanged*.

(e) Do fronty úkolů se zařadí vyvolání události *loadedmetadata* pro daný media element. Před vyvoláním události bude u video elementu provedena příslušná změna rozměru v rámci dokumentu.

(f) Pokud zdroj záznamu určí svůj počáteční čas pro spuštění, prohlížeč vyhledá data v daném časovém umístění.

(g) Když bude mít readyState hodnotu HAVE_CURRENT_DATA, získanou vyvoláním události *loadeddata*, příznak události zpožděného načítání se nastaví na false.

- Jakmile jsou data získána kompletně, je vyvolána událost progress.
- Je-li spojení přerušeno, následkem závažných chyb v síti, prohlížeč může přestat se stahováním dalších dat záznamu. To i v případě, kdy data byla již označena za použitelná. Potom se vykonají následující kroky:
  (a) Proces stahování záznamu je přerušen.
  (b) Atribut error se nastaví na novou instanci objektu MediaError. Jeho atribut code získá hodnotu MEDIA_ERR_NETWORK.
  (c) Dojde k vyvolání události error.
  (d) Atribut networkState se nastaví na hodnotu NETWORK_EMPTY a do fronty úkolů se zařadí vyvolání události emptied.
  (e) Příznak události zpožděného načítání se nastaví na false.
  (f) Algoritmus výběru zdroje záznamu je kompletně přerušen.
- Jsou-li data záznamu porušena, následují tyto kroky:
  (a) Proces stahování záznamu je přerušen.
  (b) Atribut error se nastaví na novou instanci objektu MediaError. Jeho atribut code získá hodnotu MEDIA_ERR_DECODE.
  (c) Dojde k vyvolání události error.
  (d) Atribut networkState se nastaví na hodnotu NETWORK_EMPTY a do fronty úkolů se zařadí vyvolání události emptied.
  (e) Příznak události zpožděného načítání se nastaví na false.
  (f) Algoritmus výběru zdroje záznamu je kompletně přerušen.
- Pokud je proces stahování dat přerušen uživatelem, při přechodu na jiný dokument, vykonají se následující kroky. Upozorňuji, že jsou vykonány pouze při přechodu na jinou stránku, ne však při volání metody load.
  (a) Proces stahování záznamu je přerušen.
  (b) Atribut error se nastaví na novou instanci objektu MediaError. Jeho atribut code získá hodnotu MEDIA_ERR_ABORT.

(c) Dojde k vyvolání události error.

(d) Má-li readyState hodnotu HAVE_NOTHING, potom se atribut networkState nastaví na NETWORK_EMPTY. V ostatních případech bude mít hodnotu NETWORK_IDLE.

(e) Příznak události zpožděného načítání se nastaví na false.

(f) Algoritmus výběru zdroje záznamu je kompletně přerušen.

- Získaná data mohou obsahovat nezávažné chyby, které neovlivní spuštění záznamu. Podmínkou však je, že chybné bity záznamu budou ignorovány, což může zkreslit záznam. Pokud je záznam kompletně získán bez závažných chyb a připojení k síti již není potřebné, pokračujeme krokem č. 3. Tato situace však nemusí nastat, například u nekonečných proudů záznamu, jakými jsou internetová rádia.

3. Pokud se prohlížeč dostane do tohoto kroku, kompletně přeruší algoritmus pro výběr zdroje záznamu.

B. Popis přiloženého CD

Přiložené CD obsahuje offline verzi webového tutoriálu pro canvas element. Původní verze je situována na webových stránkách [2]. Offline verze obsahuje veškeré aspekty online verze tutoriálu k datu *22.4.2010*. Na CD jsou navíc přiloženy instalátory prohlížečů s podporou canvas elementu a původní návrh jazyka HTML5, ke kterému se tato bakalářská práce vztahuje.

B.1. Spuštění tutoriálu

Tutoriál je vytvořen formou (X)HTML dokumentu, jehož hlavní soubor naleznete v kořenovém adresáři CD. Tento spouštěcí soubor má název *index.html*. Ke spuštění tutoriálu je nezbytné mít nainstalovaný libovolný prohlížeč s podporou pro canvas. Instalátory těchto prohlížečů naleznete v adresáři *data/install*. Pro přesnější informace otevřete textový soubor *readme.txt*, který se nachází v kořenovém adresáři CD.

B.2. Popis tutoriálu

Po otevření souboru index.html se v prohlížeči objeví úvodní strana tutoriálu. Zde naleznete potřebné informace pro navigaci v tutoriálu. Odkazy na jednotlivé články jsou umístěny v levé části pod úvodním textem. Napravo se nachází odkazy na instalátory prohlížečů a odkaz na specifikaci HTML5 z data *2.12.2009*.

Každý tutoriál obsahuje kromě popisu i ukázky kódů a jejich aplikace. Na pravé straně se potom nachází odkaz pro zobrazení jednoduchého editoru javascriptového kódu. Vně se nachází editovatelné textové pole s ukázkovým kódem pro daný článek. Tento kód můžete dle libosti změnit a následně jej aplikovat na canvas, který se nachází na levé straně editoru. Aplikaci kódu, jeho smazání, obnovení původního kódu a zavření editoru provedete pomocí tlačítek, která se nachází pod canvasem a polem pro editaci. Budete-li chtít editor zavřít, můžete také použít klávesu *Esc*, nebo jednoduše klikněte myší mimo oblast editoru.

B.3. Upozornění

Jelikož se návrh HTML5 stále vyvíjí, je pravděpodobné, že online verze tutoriálu bude do budoucna rozšířena. Pro účely této bakalářské práce tedy slouží tutoriály č. *1. (Začínáme s canvasem)* až *11. (Transformace)*. Ostatní tutoriály, které se na webových stránkách vyskytnou, jsou již mimo rozsah této bakalářské práce.

C. Rozhraní zmíněná v bakalářské práci

Zde naleznete rozhraní, která jsou použita pro implementaci elementů jazyka HTML5. Upozorňuji, že rozhraní jsou definována návrhem standardu HTML5, více na [1]. Jejich konkrétní implementace závisí na daném prohlížeče.

C.1. Rozhraní HTMLCanvasElement

Zdrojový kód 7. *Rozhraní HTMLCanvasElement*

```
interface HTMLCanvasElement{
   attribute unsigned long width;
   attribute unsigned long height;

   DOMString toDataURL(in optional DOMString type, in any... args);
   Object getContext(in DOMString contextId);
};
```

C.2. Rozhraní CanvasRenderingContext2D

Zdrojový kód 8. *Rozhraní CanvasRenderingContext2D*

```
interface CanvasRenderingContext2D{

   // back-reference to the canvas
   readonly attribute HTMLCanvasElement canvas;

   //state
   void save(); // push state on state stack
   void restore(); // pop state stack and restore state

   // transformations (default transform is the identity matrix)
   void scale(in float x, in float y);
   void rotate(in float angle);
   void translate(in float x, in float y);
   void transform(in float m11, in float m12, in float m21, in float m22, in float dx,
    in float dy);
   void setTransform(in float m11, in float m12, in float m21, in float m22, in float dx,
    in float dy);

   // compositing
   attribute float globalAlpha; // (default 1.0)
   attribute DOMString globalCompositeOperation; // (default source-over)

   // colors and styles
   attribute any strokeStyle; // (default black)
   attribute any fillStyle; // (default black)
```

```
CanvasGradient createLinearGradient(in float x0, in float y0, in float x1, in float y1);
CanvasGradient createRadialGradient(in float x0, in float y0, in float r0, in float x1,
 in float y1, in float r1);
CanvasPattern createPattern(in HTMLImageElement image, in DOMString repetition);
CanvasPattern createPattern(in HTMLCanvasElement image, in DOMString repetition);
CanvasPattern createPattern(in HTMLVideoElement image, in DOMString repetition);

// line caps/joins
attribute float lineWidth; // (default 1)
attribute DOMString lineCap; // "butt", "round", "square" (default "butt")
attribute DOMString lineJoin; // "round", "bevel", "miter" (default "miter")
attribute float miterLimit; // (default 10)

// shadows
attribute float shadowOffsetX; // (default 0)
attribute float shadowOffsetY; // (default 0)
attribute float shadowBlur; // (default 0)
attribute DOMString shadowColor; // (default transparent black)

// rects
void clearRect(in float x, in float y, in float w, in float h);
void fillRect(in float x, in float y, in float w, in float h);
void strokeRect(in float x, in float y, in float w, in float h);

// path API
void beginPath();
void closePath();
void moveTo(in float x, in float y);
void lineTo(in float x, in float y);
void quadraticCurveTo(in float cpx, in float cpy, in float x, in float y);
void bezierCurveTo(in float cp1x, in float cp1y, in float cp2x, in float cp2y,
 in float x, in float y);
void arcTo(in float x1, in float y1, in float x2, in float y2, in float radius);
void rect(in float x, in float y, in float w, in float h);
void arc(in float x, in float y, in float radius, in float startAngle,
 in float endAngle, in boolean anticlockwise);
void fill();
void stroke();
void clip();
boolean isPointInPath(in float x, in float y);

// text
attribute DOMString font; // (default 10px sans-serif)
attribute DOMString textAlign; // "start", "end", "left", "right", "center"
                               (default: "start")
attribute DOMString textBaseline; // "top", "hanging", "middle", "alphabetic",
                                     "ideographic", "bottom" (default: "alphabetic")
void fillText(in DOMString text, in float x, in float y, in optional float maxWidth);
void strokeText(in DOMString text, in float x, in float y, in optional float maxWidth);
TextMetrics measureText(in DOMString text);

// drawing images
void drawImage(in HTMLImageElement image, in float dx, in float dy, in optional float dw,
 in float dh);
void drawImage(in HTMLImageElement image, in float sx, in float sy, in float sw,
 in float sh, in float dx, in float dy, in float dw, in float dh);
```

```
   void drawImage(in HTMLCanvasElement image, in float dx, in float dy, in optional float dw,
    in float dh);
   void drawImage(in HTMLCanvasElement image, in float sx, in float sy, in float sw,
    in float sh, in float dx, in float dy, in float dw, in float dh);
   void drawImage(in HTMLVideoElement image, in float dx, in float dy, in optional float dw,
    in float dh);
   void drawImage(in HTMLVideoElement image, in float sx, in float sy, in float sw,
    in float sh, in float dx, in float dy, in float dw, in float dh);

   // pixel manipulation
   ImageData createImageData(in float sw, in float sh);
   ImageData createImageData(in ImageData imagedata);
   ImageData getImageData(in float sx, in float sy, in float sw, in float sh);
   void putImageData(in ImageData imagedata, in float dx, in float dy,
   in optional float dirtyX, in float dirtyY, in float dirtyWidth, in float dirtyHeight);
};
```

C.3. Pomocná rozhraní pro CanvasRenderingContext2D

Zdrojový kód 9. *Pomocná rozhraní pro CanvasRenderingContext2D*

```
interface CanvasGradient{
   // opaque object
   void addColorStop(in float offset, in DOMString color);
};

interface CanvasPattern{
   // opaque object
};

interface TextMetrics{
   readonly attribute float width;
};

interface ImageData{
   readonly attribute unsigned long width;
   readonly attribute unsigned long height;
   readonly attribute CanvasPixelArray data;
};

interface CanvasPixelArray{
   readonly attribute unsigned long length;
   getter octet (in unsigned long index);
   setter void (in unsigned long index, in octet value);
};
```

C.4. Rozhraní HTMLAudioElement

Zdrojový kód 10. *Rozhraní HTMLCanvasElement*

```
[NamedConstructor=Audio(),
 NamedConstructor=Audie(in DOMString src)]
interface HTMLAudioElement : HTMLMediaElement {};
```

C.5. Rozhraní HTMLVideoElement

Zdrojový kód 11. *Rozhraní HTMLVideoElement*

```
interface HTMLVideoElement : HTMLMediaElement {
  attribute DOMString width;
  attribute DOMString height;
  readonly attribute unsigned long videoWidth;
  readonly attribute unsigned long videoHeight;
  attribute DOMString poster;
};
```

C.6. Rozhraní HTMLSourceElement

Zdrojový kód 12. *Rozhraní HTMLSourceElement*

```
interface HTMLSourceElement : HTMLElement {
  attribute DOMString src;
  attribute DOMString type;
  attribute DOMString media;
};
```

C.7. Rozhraní HTMLMediaElement

Zdrojový kód 13. *Rozhraní HTMLMediaElement*

```
interface HTMLMediaElement : HTMLElement {

  // error state
  readonly attribute MediaError error;

  // network state
  attribute DOMString src;
  readonly attribute DOMString currentSrc;

  const unsigned short NETWORK_EMPTY = 0;
  const unsigned short NETWORK_IDLE = 1;
  const unsigned short NETWORK_LOADING = 2;
  const unsigned short NETWORK_NO_SOURCE = 3;

  readonly attribute unsigned short networkState;
  readonly attribute TimeRanges buffered;

  void load();
  DOMString canPlayType(DOMString type);

  // ready state
  const unsigned short HAVE_NOTHING = 0;
  const unsigned short HAVE_METADATA = 1;
  const unsigned short HAVE_CURRENT_DATA = 2;
  const unsigned short HAVE_FUTURE_DATA = 3;
  const unsigned short HAVE_ENOUGH_DATA = 4;
```

```
  readonly attribute unsigned short readyState;
  readonly attribute boolean seeking;

  // playback state
  attribute float currentTime;
  readonly attribute float startTime;
  readonly attribute float duration;
  readonly attribute boolean paused;
  attribute float defaultPlaybackRate;
  attribute float playbackRate;
  readonly attribute TimeRanges played;
  readonly attribute TimeRanges seekable;
  readonly attribute boolean ended;
  attribute boolean autoplay;
  attribute boolean loop;

  void play();
  void stop();

  // controls
  attribute boolean controls;
  attribute float volume;
  attribute boolean muted;

};
```

C.8. Rozhraní MediaError

Zdrojový kód 14. *Rozhraní MediaError*

```
interface MediaError{
  const unsigned short MEDIA_ERR_ABORTED = 1;
  const unsigned short MEDIA_ERR_NETWORK = 2;
  const unsigned short MEDIA_ERR_DECODE = 3;
  const unsigned short MEDIA_ERR_SRC_NOT_SUPPORTED = 4;
  readonly attribute unsigned short code;
};
```

C.9. Rozhraní TimeRanges

Zdrojový kód 15. *Rozhraní TimeRanges*

```
interface TimeRanges{
  readonly attribute unsigned long length;
  float start (in unsigned long index);
  float end (in unsigned long index);
};
```