DIPLOMAMUNKA

Thold Tibor István

Debrecen

2009

Debreceni Egyetem

Informatikai Kar

Interaktív webalkalmazások fejlesztése JavaScript és XML alapokon

Témavezető:

Dr. Adamkó Attila

egyetemi tanársegéd

Készítette:

Thold Tibor István

programtervező matematikus

Debrecen

2009

Tartalomjegyzék

1. Bevezetés

1.1. Diplomamunkám témája

Ez a dolgozat a Laszlo Systems (http://www.laszlosystems.com) által fejlesztett OpenLaszlo (http://www.openlaszlo.org) nevű Web-fejlesztési eszközt mutatja be. Az OpenLaszlo egy nyílt forráskódú, XML szintaktikájú platform, mely kiemelten támogatja a JavaScript-et. Néhány egyszerűbb programon keresztül ismertetem az OpenLaszlo működését, felépítését és néhány egyszerűbb trükköt.

Az összetettebb OpenLaszlo alkalmazások elkészítéséhez nélkülözhetetlen a JavaScript igen erős ismerete. Ebben a dokumentumban az XML-nek és a JavaScript-nek alapszíntű ismertetésére is kitérek, de arra nagyon jó könyvek, jegyzetek, és e-bookok (pl.: http://ebookz.hu, vagy http://javascript.lap.hu) találhatóak mindenfelé.

Az OpenLaszlo elemek (TAG-ek), és ezek attribútumai közül nem fogok sokat ismertetni, de leírásuk, és használatuk megtalálható az Irodalomjegyzékben megadott weboldalakon, és könyvben. Az OL elemek listája pedig szerepel a 8. Függelék fejezetben.

1.2. Témaválasztás indoklása

Manapság egyre inkább elterjednek a Web alapú alkalmazások. Amikkel eddig csak asztali alkalmazásként találkozhattunk, olyat most már az interneten is fellelhetünk igen nagy számban. Ilyenek például a képszerkesztők, dokumentumszerkesztők, táblázat kezelők, a konverterek stb… tartalmú weboldalak. Az OpenLaszlo fejlesztésekor az egyik fő szempont az asztali alkalmazásokhoz hasonló felépítésű, teljesen interaktív weboldalak elkészítése volt. A RIA (Rich Internet Application: Gazdag Internet Alkalmazás) fejlesztés előtt nagy jövő áll, ami az egyre biztonságosabb, gyorsabb, és egyre többek számára elérhető internetnek köszönhető. Mivel Magyarországon az OpenLaszlo még szinte egyáltalán nincs elterjedve, ezért szeretnék ezzel a dokumentummal kedvet hozni, a magyar Web-fejlesztőknek a technológia használatához. És mert magyar nyelvű dokumentáció sem lelhető fel hozzá, ezért az alapok elsajátításában szeretnék minél nagyobb segítséget nyújtani az angol nyelvű dokumentumokat nem igazán kedvelőknek. Bár ez a dokumentum az OpenLaszlo-nak csak alapjait ismerteti a hosszadalmas terjedelme miatt.

2. OpenLaszlo történelme

2.1. OpenLaszlo kezdete

Amikor 2000-ben elkezdte működését a Laszlo Systems, tervbe vették a Web böngészőkben futó, asztali alkalmazás interaktivitásához hasonló, Web alkalmazások létrehozását. Erre már létezett például a Java is, de nekik fő céljuk inkább egy olyan eszköz volt, amivel Flash alapú weboldalak készíthetőek. A Flash-el a fő problémájuk az úgynevezett timeline (idővonal) alapú szemlélet volt, ami nagy alkalmazások készítése esetén sokak számára nehezen kezelhető.

Ezáltal elhatározták, hogy létrehoznak egy keretrendszert a Flash programozás egyszerűsítésére. Így megalkották a saját fordítójukat Flash kimenettel, melynek neve LZX lett. Az OpenLaszlo-ban írt alkalmazásokat ma Flash 7-10-be és Dinamic HTML-be (DHTML)-be lehet fordítani.

2.2. OpenLaszlo fejlődése:

- 2000: Megalakult a Laszlo Systems, és elkezdték tervezni az OpenLaszlo-t, ekkor még Laszlo Presentation Server (LPS) néven.
- 2001: Elkezdik kifejleszteni a rendszert
- 2002: LPS bemutatása; első Laszlo alkalmazás nyilvánosságra hozása: Behr (http://www.behr.com/Behr/home); Flash 5-re való fordítás
- 2003: LPS 1.0, LPS 1.1 közzététele; következő Laszlo alkalmazások nyilvánosságra hozása: Yahoo!, Earthlink
- 2004: LPS 2.0, LPS 2.1, LPS 2.2 bemutatása; LPS nyílt forráskódúvá tétele
- 2005: Névváltoztatás OpenLaszlo-ra; OpenLaszlo 3.0, OpenLaszlo 3.1 nyilvánosságra hozása
- 2006: OpenLaszlo 3.2, OpenLaszlo 3.3 közzététele
- 2007: OpenLaszlo 4.0 szabadon bocsátása
- 2008: OpenLaszlo 4.1, OpenLaszlo 4.2 kibocsátása
- 2009. február: OpenLaszlo 4.2.0.1 kiadása
- 2009. április: OpenLaszlo 4.3 elérhetővé tétele; Flash 9, Flash 10 kompatibilitás

3. XML

3.1. Egy aprócska bevezető

Ez a fejezet az XML szintaktikáját fogja tárgyalni azért, hogy bevezetést tegyen az OpenLaszlo szintaktikájába, mivel a kettőnek a formai szabályai között nincsenek különbségek.

Az XML (eXtensible Markup Language, magyarul: bővíthető leíró nyelv) egy leíró nyelv, melynek fő célja adatok, információk és ezek struktúráinak leírása. Ez egy teljesen általánosított jelölőnyelv. Az XML-t a HTML és az SGML tapasztalataira építve a W3C (World Wide Web Consortium) hozta létre. Felépítésben nagyon hasonlít a HTML-hez, de sokkal rugalmasabb és szintaktikailag szigorúbb is. Létrehozásakor elsődleges szempont volt az egyszerűség, de nem volt szempont a tömör adatábrázolás.

3.2. XML szabályok

Egy XML dokumentumra azt mondjuk, hogy jól formázott, ha teljes egészében megfelel az XML specifikációban meghatározott szabályoknak. Ilyen szabályból nem túl sok létezik, ezért megjegyzésük és betartásuk nem fog senkinek nagyobb gondot okozni. Ezekből a szabályokból most a legfontosabbakat, és az OpenLaszlo használatához szükségeseket most ismertetni fogom.

Az XML-ben az adatokat úgynevezett TAG-ek (innentől: elemek) jelölik. Ezeket az adatokat „<” és „>” jelek közé írjuk.

Például:

```
<Adat>
```

De a HTML-lel ellentétben az XML nem tartalmazhat lezáratlan elemet, szóval az előző adat csak az alábbi formák egyikében lehet jelen:

```
<Adat />
```

```
<Adat>
</Adat>
```

Az XML-ben minden adat saját definiálású, azaz nincsenek kitüntetett, vagy jelentéssel bíró elemek.

Egy másik nagy különbség az XML és a HTML között, hogy még a HTML tartalmazhat ilyen szerű felépítéseket:

```
a<u>b<b>c</u>d</b>e
```

addig az XML-ben a hagyományos zárójelezés szabályai érvényesek, vagyis az ugyanezt eredményező XML kód:

```
a<u>b<b>c</b></u><b>d</b>e
```

3.3. Az XML felépítése

Az XML dokumentumoknak egy vezérlési utasítással kell kezdődniük, aminek legalább a dokumentumunknak azt a tulajdonságát tárolnia kell, hogy melyik XML verzió szerint készült. Ennek megadása nem kötelező, de az XML specifikáció előírja ennek használatát.

Például:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
```

Ez a példa a karakterkódolást is tartalmazza.

(Ugyanezt a sort kell használni az OpenLaszlo-ban is, ha ezeket az adatokat meg szeretnénk adni.)

Minden XML dokumentum fa struktúrában van leírva. Minden dokumentum tartalmaz egy úgynevezett gyökér elemet, ami mindig a legkülső. A dupla elemből álló XML elemek között belső elemek is szerepelhetnek, vagy maga az adat rész, ami nincs „<” és „>” jelek között.

Például:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<alkalmazottak>
   <alkalmazott>
        <név>János</név>
        <telefonszám>12345678</telefonszám>
    </alkalmazott>
    <alkalmazott>
        <név>Sándor</név>
        <telefonszám>87654321</telefonszám>
    </alkalmazott>
</ alkalmazottak>
```

1. XML – példa

A HTML-hez hasonlóan az XML is tartalmazhat úgynevezett tulajdonságokat, amik az elem <, és > jelei közt az elemnév után helyezkednek el, és ennek az értéke aposztrófok, vagy idézőjelek között van.

Például:

```
<alkalmazott név="János" />
```

Most bemutatok 4 olyan XML leírási módszert, amit ugyanúgy értelmezhetünk:

```
<alkalmazott név="János" telefonszám="12345678" />
```

2. XML – 1. módszer

```
<alkalmazott név="János">
    <telefonszám="12345678" />
</alkalmazott>
```

3. XML – 2. módszer

```
<alkalmazott név="János">
    <telefonszám>12345678</telefonszám>
</alkalmazott>
```

4. XML – 3. módszer

```
<alkalmazott>
    <név>János</név>
    <telefonszám>12345678</telefonszám>
</alkalmazott>
```

5. XML – 4. módszer

OpenLaszlo-ban is ezeket a fajta leírási módszereket alkalmazhatjuk. A kódunk szemantikáját mindig nagyon körültekintően kell kiválasztani, hogy mit milyen módszerrel írunk le, mert a későbbi átalakításoknál lehet, hogy teljesen át kell az egészet variálnunk.

3.4. Vigyázat!

A következő táblázat tartalmaz néhány előre definiált karakterhivatkozást, amiknek a használatával meggyűlhet a bajunk. Ezeket a karaktereket nem használhatod elemnévben, vagy szöveg helyén sem.

karakter	karakterhivatkozás	megnevezés
<	&lt	kisebb, mint
>	&gt	nagyobb, mint
"	&quot	idézőjel
'	&apos	aposztróf
&	&amp	és jel

1. Táblázat – XML speciális karakterek

3.5. CDATA

Az előbbi táblázatban szereplő karakterek nem szerepelhetnek bizonyos részeken. De ez bizonyos helyeken áthágható. Pontosabban a CDATA részen, ugyanis az XML értelmező ennek a tartalmát figyelmen kívül hagyja. Szóval ezen belül semmi nem kerül értelmezésre. Ennek a résznek a kezdő része a ”<![CDATA[” karaktersorozat, záró része pedig a ”]]>” karaktersorozat. Ezek közé bármit írhatunk a ”]]>” karaktersorozat kivételével.

Ahol szöveges tartalom szerepelhet, oda ezt a CDATA-t is beszúrhatjuk.

Példa:

```
<alkalmazottak>
    <alkalmazott>
        <nev>János</nev>
<![CDATA[
Elbocsátva & fizetése megvonva, ha fizetése > 10e Ft, mert a következőt
mondta: ”A főnöknek bejön a felesége”
]]>
    </alkalmazott>
</alkalmazottak>
```

6. XML – CDATA példa

Ez így teljesen szabályos XML kód, pedig látható, hogy szerepel benne &, >, és ” karakter is

3.6. Megjegyzés

Az XML-ben lehetőség van megjegyzés használatára, mely a HTML-ben megszokott módon használatos:

<!-- … -->

Példa:

```
<!--
    Ez egy megjegyzés lesz:
    <megjegyzes szoveg=”Ez most megjegyzés” />
-->
<megjegyzes <!-- Itt NEM lehet megjegyzés, vagyis ez most hibás --> >
    <szoveg>Ez most NEM megjegyzés</szoveg>
    <!-- Ide megint lehet tenni megjegyzést -->
</megjegyzes>
<!-- És természetesen itt is lehet megjegyzés -->
```

7. XML – Megjegyzés

4. JavaScript

Ez a rész a JavaScript nagyon felszínes bemutatásáról fog szólni. Programozási ismeret szükségszerű ennek a résznek a megértéséhez, mivel leginkább csak a nyelv sajátosságait írom le. Nagy hátulütője, hogy nem minden böngésző ugyanúgy értelmezi, és nem minden böngésző ismer minden JavaScript-beli utasítást. A JavaScript és a Sun által kifejlesztett Java programozási nyelv teljesen különböző nyelv.

4.1. Bevezetés

A JavaScript a Netscape által kifejlesztett programozási nyelv. Mint ahogy a neve is mutatja, ez egy script, vagy parancs nyelv. A futtatásához valamilyen böngészőre van szükségünk, amiből látszik is, hogy csak kliens oldalon futó alkalmazásokat írhatunk vele. A böngésző soronként hajtja végre az utasításokat.

Ezt a nyelvet a weblapok interaktivitására fejlesztették ki. A JavaScript reagálhat valamilyen eseményekre. A weboldalunkon végrehajtott valamilyen esemény (például: egérkattintás) hatására mást hajthat végre, mint más eseményre (például: egérmozgatás).

Ez egy kisbetű / nagybetű érzékeny nyelv. Erre nagyon oda kell figyelni programozáskor. A JavaScript blokk jele a C-hez, Java-hoz, és PHP-hez hasonlóan a kapcsos zárójel pár: { }. Az utasításokat pontosvessző zárja, amit csak akkor kötelező kirakni, ha több utasítás követi egymást.

4.2. Változók

A JavaScript típus nélküli nyelv, azaz például a változóinknak nem kell megadnunk, hogy milyen típusú értéket tároljon. Akkor van típusa egy változónak, ha értéket vesz adunk neki, de ez változtatható. Vagyis ha egy változó értéke egy szám, és egy szöveget adunk neki értékül, akkor onnantól kezdve szöveg típusú, amíg más típusú értéket nem kap.

A változó deklaráció kulcsszava a `var`, de ezt nem kötelező megadni. Leginkább azért érdemes ezt megadni, hogy tudjuk, hogy egy változó honnantól kezdve él.

Példák:

```
var newNum;
var positive = true;
newText = toString(newNum);
```

A változónevekre annyi kikötés van, hogy csak betűvel (a-z, A-Z), vagy _ jellel kell kezdődhet, betűvel, számjeggyel (0-9) és _ jellel folytatódhat. Nem egyezhet meg foglalt

szóval (pl.: `if`, `else`, `break`). A változónév hossza nincs korlátozva. Kezdő értékadás nem kötelező.

Egy változót élettartama mindig az őt tartalmazó blokk. Például egy függvény, vagy egy ciklus. Ezek a lokális változók.

Azok a változók, amiket a függvényeken kívül deklarálunk, azok az oldal elhagyásáig élnek. Ezek a globális változók.

4.3. Kifejezések, operátorok

A kifejezés literálok, változók, operátorok és utasítások kombinációja, amelynek egy eredménye van. Ez az érték lehet logikai, szám, vagy szöveg. A kifejezés vagy egy értékadás (pl.: `positive = true`), vagy egy egyszerű érték (pl.: `op`, vagy `operator & 2`).

Operátorok:

Aritmetikai operátorok:

Operátor	**Leírás**
+	összeadás
-	kivonás, vagy negálás
*	szorzás
/	osztás
%	modulus képzés
++	1-el növelés
- -	1-el csökkentés

2. Táblázat – JavaScript aritmetikai operátorok

Logikai operátorok:

Operátor	Leírás
&&	és
||	vagy
!	negáció

3. Táblázat – JavaScript logikai operátorok

Sztring operátor:

Operátor	Leírás
+	konkatenáció

4. Táblázat – JavaScript sztring operátorok

Bitszintű operátorok:

Operátor	Leírás
~	negálás
<<	balra léptetés
>>	jobbra léptetés
>>>	helyiérték nélküli jobbra léptetés
&	és
|	vagy
^	kizáró vagy

5. Táblázat – JavaScript bitszintű operátorok

Értékadó operátorok:

Operátor	ugyanaz, mint
=	x = y
+=	x = x + y
-=	x = x - y
*=	x = x * y
/=	x = x / y
%=	x = x % y
&=	x = x & y
|=	x = x | y
^=	x = x ^ y
<<=	x = x << y
>>=	x = x >> y
>>>=	x = x >>> y

6. Táblázat – JavaScript értékadó operátorok

Összehasonlító operátorok:

Operátor	Leírás
==	egyenlő
!=	nem egyenlő
>	nagyobb
>=	nagyobb egyenlő
<	kisebb
<=	kisebb egyenlő
===	teljesen egyenlő
!==	teljesen egyenlőtlen

7. Táblázat – JavaScript összehasonlító operátorok

4.4. Feltételes utasítások

Felépítése:

```
if (feltétel) {
   utasítások;
} [else {
   utasítások;
} ]
```

A feltétel valamilyen kifejezés, amit a JavaScript képes logikai értékűvé konvertálni. Lehet például numerikus kifejezés is. Az `else` ág megadása nem kötelező.

Példák:

```
if ( (canvas.pressedButtons & 1) != 1 ){
   newNum *= 10;
   newNum += num;
} else {
   canvas.fraction *= 10;
   newNum += (num / canvas.fraction);
}

if ( !positive )
   newNum *= -1;
```

4.5. Elágazások

Felépítése:

```
switch (kifejezés) {
   case érték1: utasítások; break;
   [case érték2: utasítások; break;]
   […]
   [default: utasítások;]
}
```

Ezt akkor használjuk, hogyha egy kifejezésnek több értéke is lehet, és az egyes értékek függvényében szeretnénk befolyásolni programunk futását.

Elágazás szabályai:

- A kifejezés egy tetszőleges kifejezés
- Az érték csak egyetlen konstans lehet
- Minden `case` után csak egy érték állhat
- Ha nincs egy ágon `break;`, akkor az utána lévő ágak is végrehajtódnak, amíg nem talál egy `break;`-et, vagy véget nem és az elágazás
- A `default` ág akkor hajtódik végre, ha nincs a kifejezésnek megfelelő érték

Példa:

```
switch (op){
   case '+': canvas.pressedButtons += 2; break;
   case '-': canvas.pressedButtons += 4; break;
```

```
   case '*': canvas.pressedButtons += 8; break;
   case '/': canvas.pressedButtons += 16; break;
}
```

4.6. Ciklusok

A JavaScript-ben a következő ciklusok vannak:

4.6.1 while ciklus

Felépítése:

```
while (feltétel) {
   utasítások;
}
```

Ezt akkor használjuk, ha az utasításokat egy bizonyos feltétel teljesüléséig szeretnénk végrehajtani.

Példa:

```
var fakt = 1, i = 2;
while (i++ <= n) {
   fakt *= i;
}
```

Ez a példa n faktoriálisát eredményezi.

4.6.2. do … while ciklus

Felépítése:

```
do {
   utasítások;
}while (feltétel)
```

Ezt akkor használjuk, ha az utasításokat egy bizonyos feltétel teljesüléséig szeretnénk végrehajtani, de egyszer mindenféleképpen szeretnénk, hogy lefussanak.

Példa:

```
var fakt = 1, i = 1;
do {
   fakt *= i++;
} while (i <= n)
```

Ez a példa is n faktoriálisát eredményezi.

4.6.3. for ciklus

Felépítése:

```
for (kezdőérték adás; feltétel; növekmény) {
   utasítások;
}
```

Ezt akkor használjuk, ha az utasításokat egy bizonyos számú alkalommal szeretnénk, hogy lefusson.

Példa:

```
var fakt = 1;
for (i = 2; i <= n; i++){
   fakt *= i;
}
```

Ez a példa is `n` faktoriálisát eredményezi.

4.7. Függvény

A JavaScript függvénye is típus nélküli, így nem kell se visszatérési értéket megadni, se a paraméterlistában típust. A függvény kulcsszava a `function`. A nevére ugyanazok a szabályok vannak, mint a változók nevére.

Felépítése:

```
function név ([paraméter1 [,paraméter2 …]]){
   utasítások;
}
```

A függvényünknek alapból nincs visszatérési értéke. Ahhoz, hogy valamilyen értéket visszaadjon a függvényünk, a `return` kulcsszóra van szükség, aminek hatására a vezérlés átadódik az őt hívó programrésznek, és a függvény értékeként azt a kifejezést veszi, ami a `return` után szerepel (pl.: `return 0`).

Példa:

```
function calculate(num1, num2, operator){
   if ( (operator & 2) == 2 ){
      canvas.pressedButtons -= 2;
      return (num1 + num2);
   } else if ( (operator & 4) == 4 ){
      canvas.pressedButtons -= 4;
      return (num1 - num2);
   } else if ( (operator & 8) == 8 ){
      canvas.pressedButtons -= 4;
      return (num1 * num2);
   } else if ( (operator & 16) == 16 ){
      canvas.pressedButtons -= 4;
      return (num1 / num2);
   }

   return 0;
}
```

Ez a példa a számológép programból származó függvény. Az `&` operátor a bitenkénti és-t jelenti. A segítségével egy változót használok 5db logikai változó helyett.

4.8. Speciális karakterek

Lehetőségünk van speciális karakterek használatára a backslash (\) segítségével. Ilyen karakterek:

Karakter	Jelentés
\a	sípoló hang
\b	balra törlés
\f	lap dobás
\n	sortörés
\r	kocsi vissza
\t	tabulátor
\"	idézőjel
\'	aposztróf
\\	backslash

8. Táblázat – JavaScript speciális karakterek

Ezeket a karaktereket használhatjuk sztringekben.

Például a következő szöveget szeretnénk kiíratni:

```
Így szólt Sándor:
„Jó dolog ez a JavaScript”
```

Akkor ez a megoldás helytelen:

```
document.write(”Így szólt Sándor:
„Jó dolog ez a JavaScript””);
```

De ez már helyes:

```
document.write(”Így szólt Sándor:\n\„Jó dolog ez a JavaScript\””);
```

4.9. Megjegyzés

A JavaScript-ben kétféle megjegyzés használható:

1. Egysoros megjegyzés: `//` …
2. Többsoros megjegyzés: `/*` … `*/`

Példa:

```
//Ez most megjegyzés
de ez már nem

/* ez megint megjegyzés
és még ez is */
de ez már megint nem
```

5. OpenLaszlo

Az OpenLaszlo egy teljes mértékben XML szintaktikájú Web-fejlesztési eszköz, JavaScript használattal. Ez annyit takar, hogy egy olyan TAG-eket (innentől elem) tartalmazó nyelv, amelyben minden elemet le kell zárni, a elemekre a hagyományos zárójelezés szabályai érvényesek, és lennie kell egy gyökérelemnek minden esetben. Ezekről bővebben az XML című fejezetben lehet olvasni. JavaScript-et pedig az elemekben tulajdonságként, vagy bizonyos elemek közt adhatunk meg.

Egy OpenLaszlo-ban megírt alkalmazás egy vagy több forrás állományban helyezkedhet el, melyeknek a kiterjesztése .LZX. Ezek mellett természetesen lehetnek különböző más nyelveken megírt fájlok is, például: .JSP, .JAVA, .JS stb…, de ezekkel nem foglalkozunk most.

Az OpenLaszlo egy kis-nagy betűre érzékeny nyelv. Vagyis ha egy elem neve, attribútuma, vagy attribútumának értéke csak abban a formában írható le, ahogy azt a fejlesztők kitalálták. Például a `<text>` elem csak így szerepelhet, és nem szerepelhet `<Text>` vagy `<TEXT>` formában sem. De a `<text height=”100”>` helyett sem írhatunk `<text Height=”100”>` -et, illetve a `<text align=”center”>` helyett nem írhatunk `<text align=”Center”>`-t.

Az elemeket nyugodtan tördelhetjük több sorra, ugyanis a fordító egy elemet „<”-től „>”-ig tekint. Vagyis a következő két kódrészlet ekvivalens egymással:

```
<img src=”pic1.jpg” width=”100” height=”70” />
```

```
<img src=”pic1.jpg”
    width=”100”
    height=”70” />
```

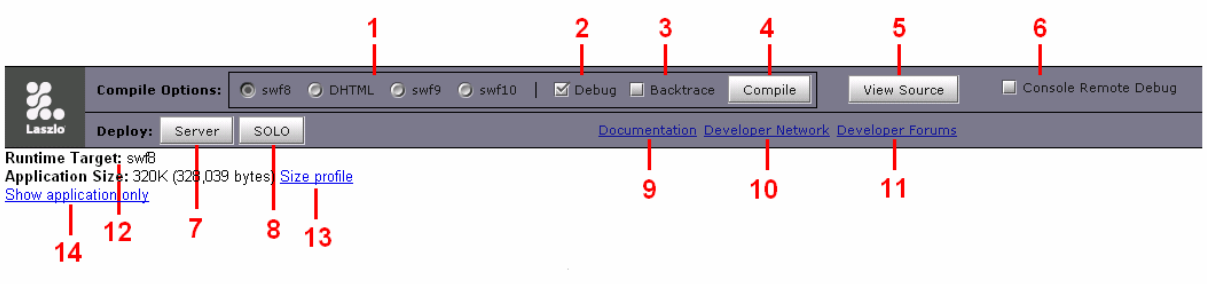
Az OpenLaszlo-ban háromféle kommentet alkalmazhatunk:

- XML komment: `<!-- … -->`
- OpenLaszlo komment: `<?ignore … ?>`
- Script kommentek (csak a JavaScript-ben): egysoros: `//` … vagy többsoros: `/* … */`

5.1. Fordítás, futtatás, nyomkövetés

5.1.1. Fordítás, futtatás

Az OpenLaszlo egy segéd nyelv. Megírjuk benne az alkalmazásunkat egy bizonyos nyelven, majd lefordítjuk egy már régebb óta létező, elterjedt formába. Emiatt nem érdemes a kész alkalmazásunkat OpenLaszlo szerverre telepíteni, mert akkor minden futtatás előtt ellenőrzi, hogy történt-e módosítás, és fordítja az oldalt. Tehát úgy nagyon idő és erőforrás igényesen futna. Ezért csak fejlesztéshez érdemes használnunk az OpenLaszlo szervert. Fejlesztés közben az alkalmazásunk alatt a következő sávot láthatjuk a böngészőnkben:



1. Kép – Fordító sáv

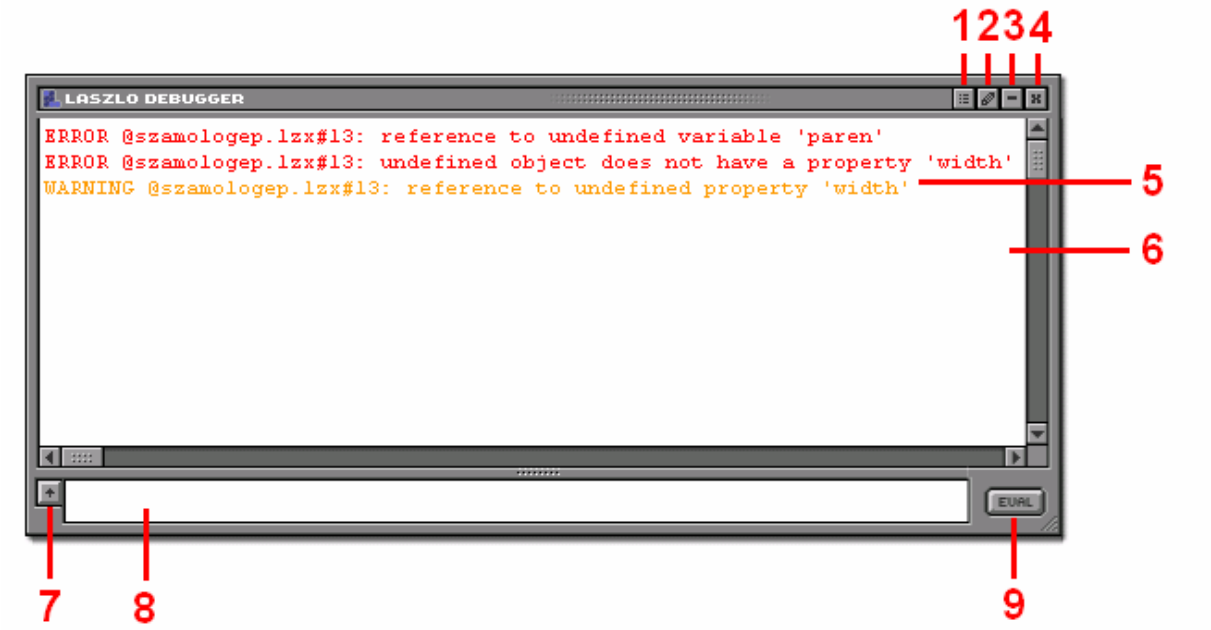
1. Itt választhatjuk ki, hogy mibe szeretnénk fordíttatni a kódunkat.
2. Kiválaszthatjuk, hogy szeretnénk-e nyomkövetést az alkalmazásunkban.
3. Bekapcsolja a Pavé féle backtrace debugger-t.
4. Ezzel a gombbal fordíttathatjuk újra a kódunkat.
5. A böngészőnkbe betölti a weblapunk OpenLaszlo kódját (nem a lefordított, hanem a nyers kódot)
6. Önálló nyomkövetés. Nem lesz része az alkalmazásnak (ugyanis a 2-es hozzá fordítótik az alkalmazásunkhoz).
7. Betölt egy angol nyelvű HTML fájlt, amiben le van írva, hogy hogyan kell telepíteni egy OpenLaszlo alkalmazást, meg néhány példa, hogy milyen HTML kód készül egy OpenLaszlo alkalmazásból. Találhatunk itt olyan linket, ami úgy tölti be alkalmazásunkat a böngészőbe, ahogy az majd telepítés után látszódni fog.
8. Ezzel a gombbal alakíthatjuk át, és menthetjük le gépünkre az általunk választott nyelvre lefordított fájlokat. Például a Flash, illetve DHTML forrásfájlokat. Így tehetjük az alkalmazásunk önálló (OpenLaszlo szerver nélkül) futásra képesnek.

(A 9, 10, 11-es elemek a fejlesztőknek nyújtanak segítséget az OpenLaszlo tanuláshoz)

9. OpenLaszlo Dokumentációt tölt be.
10. A Laszlo Systems fejlesztői oldalát tölti be
11. A Laszlo Systems fejlesztői fórumát tölti be
12. Kiírja, hogy mibe lett fordítva az alkalmazásunk
13. Kiírja az alkalmazásunk pontos méretét, a linkre klikkelve pedig betöltődik egy weboldal, amiben részletezve vannak, hogy mi mennyit foglal az alkalmazásunkban, és általános leírások arról, hogy melyik elem mennyit foglal egy OpenLaszlo alkalmazásban.
14. Betölti az alkalmazásunkat mindenféle plusz adalék nélkül, mint például ez a sáv, amiről most szó van.

5.1.2. Debug

Az OpenLaszlo-ban két módszer létezik a nyomkövetésre. Mindkét esetben a következő ablakot látjuk az alkalmazásunk alatt:



2. Kép – Laszlo Debugger ablak

1. Laszlo Debugger ablak beállítási képernyőjét hozza be
2. Vászon tartalmának törlése
3. Kisebb/nagyobb méret
4. Bezárás

5. Üzenet
6. Vászon, amin az üzenetek vannak
7. Konzol mező mérete
8. Konzol mező
9. Konzol tartalmának végrehajtása

Ebben az ablakban megjelennek a programunk hibái, de mi magunk is írhatunk rá például a `Debug.write()` paranccsal.

Az első módszer, hogy előhívjuk ezt az ablakot, ha a programunkban adjuk meg, hogy legyen nyomkövetés. Ezt úgy tehetjük meg, hogy a programunk gyökér elemében tulajdonságként megadjuk, hogy szeretnénk ezt (`<canvas debug=”true”>`). Ilyenkor az alkalmazás része lesz a Laszlo Debugger ablak, és ha lefordítjuk az alkalmazásunkat Flash-be, vagy DHTML-be, akkor is jelen lesz.

A második módszer az ablak előhívására az, hogyha kipipáljuk a fentebb bemutatott fordítási sávban a Debug jelölőnégyzetet fordítás előtt. Ez nem lesz része a programnak. Ez csak az OpenLaszlo szerveren való futáskor látható.

5.2. Helló, világ!

Elsőként nézzünk meg a legegyszerűbb, legáltalánosabb „Hello, Világ!” szöveget kiíró alkalmazásra két külön forrás kódot.

hello1.lzx:

Helló, világ!

```
1.  <canvas height="100">
2.     <text text="Helló, világ!" />
3.  </canvas>
```

1. OpenLaszlo forrásállomány – hello1.lzx

hello2.lzx:

Helló, világ!

```
1.  <canvas height="100">
2.     <text>
3.        Helló, világ!
4.     </text>
5.  </canvas>
```

2. OpenLaszlo forrásállomány – hello2.lzx

A hello1.lzx programban a kiírandó „Helló, világ!” szöveg a `<text>` elem `text` tulajdonságaként jelenik meg, a hello2.lzx-ben pedig a `<text>` elem adata ként. Nagyon sok

esetben eldönthetjük, hogy melyik módszert használjuk. Én több esetben nem fogok kitérni arra, hogy mikor lehet választani a két eset közt.

Amint azt láthatjuk, mindkét esetben a gyökér elem a `<canvas>` elem. Ez azt jelöli, hogy itt lesz a főprogram. Ezt sosem hagyhatjuk el. Ez olyan, mint például a HTML-ben a `<html>` … `</html>` elem pár, a C-ben a `main(){…}` függvény, a PHP-ban a `<?php` … `?>` pár, vagy a Pascalban a `begin` … `end.` pár.

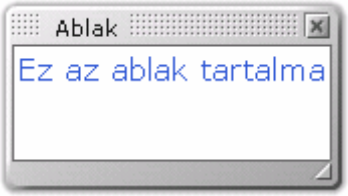
A `<canvas>` elemnek is adhatunk különböző tulajdonságokat, ami esetünkben csak a `height` tulajdonság, ami az oldalunk magasságát jelöli. De megadhatunk szélességet (`width`), LPS (Laszlo Presentation Server) verziót (`version, lpsversion`), dataset-ek gyökér könyvtárát (`dataset`), vagy nagyon sok egyéb mást is.

Minden elemnek léteznek saját tulajdonságaik, és mivel az OpenLaszlo objektum orientált szemléletű, így örökölhetnek más elemektől is tulajdonságokat, de erről majd később lesz szó.

5.3. Ablak

Amint már említettem az OpenLaszlo kifejlesztésekor az egyik fő szempont az asztali alkalmazásokhoz hasonló interaktivitás, és felépítés volt. Erre egy igen jó példa a következő kis alkalmazás:

ablak.lzx:



3. Kép – Ablak alaphelyzetben
Betöltéskor

4. Kép – Ablak elmozgatva
Átméretezés és elmozgatás után

```
1.  <canvas height="300">
2.     <window height="100" resizable="true"
3.             closeable="true" title="Ablak">
4.        <text fontsize="15" fgcolor="#325AD1">
5.           Ez az ablak tartalma
6.        </text>
7.     </window>
8.  </canvas>
```

3. OpenLaszlo forrásállomány – ablak.lzx

Látható, hogy milyen egyszerű dinamikus ablakot létrehozni OpenLaszlo-val Ez egy mozgatható, átméretezhető (`resizable=”true”`), bezárható (`closeable="true"`), „Ablak” feliratú (`title="Ablak"`) ablak. Létrehozásához csak a `<window>` elemre van

szükség, amit nagyon sokféleképpen felruházhatunk különböző tulajdonságokkal. Méretnek csak a magasság (`height="100"`) van megadva. A szélesség ebben az esetben automatikusan igazodik az ablak tartalmának szélességéhez. Ez minden OpenLaszlo-beli elemnél így van, hogy ha nem adunk meg hozzá méretet, akkor a tartalmához igazodik.

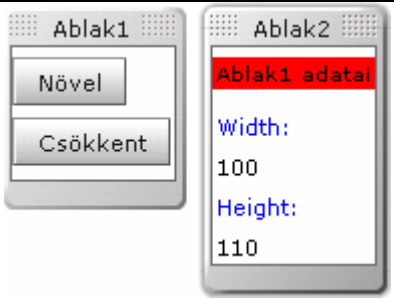
Az `x` tulajdonság az őt tartalmazó tároló (esetünkben `<canvas>`) bal szélétől való távolságot jelenti. A tetejétől vett távolságot az `y` tulajdonsággal adhatjuk meg. Ezeket az értékeket minden megjeleníthető OpenLaszlo elemben megadhatjuk. JavaScripttel megadható más érték is, és így nem csak az őt tartalmazó tárolóhoz képest helyezhetjük el.

Az ablakot mozgatni úgy tudjuk, hogy a keretét valahol megfogva elmozdítjuk azt. Átméretezni a jobb alsó sarokban lévő gombbal lehet, bezárni pedig a már megszokott jobb felső sarokban lévő gombbal lehet.

5.4. JavaScript az OpenLaszlo-ban

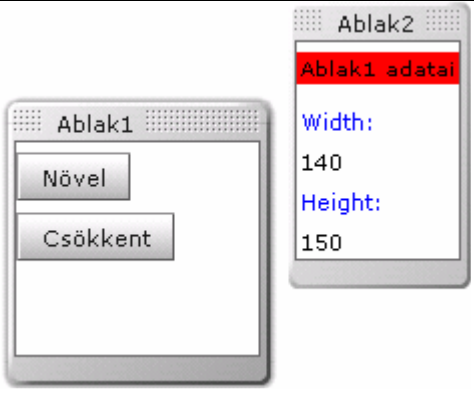
Mint azt már említettem az OpenLaszlo kódba elhelyezhetünk JavaScript kódokat is. Ettől még interaktívabbá, dinamikusabbá tehetőek alkalmazásaink. JavaScript kód csak 3 féle elem között helyezhető el. Ezek a `<method>`, `<handler>` és a `<script>` elemek. Mint azt megszokotuk a JavaScript-től, ebben az esetben is csak a böngészőnk kezeli őket. Lássunk erre egy egyszerű programot.

javascript_method.lzx:



5. Kép – Növelhető ablak

Betöltéskor



6. Kép – Növelt ablak

Ablak1 elmozgatása és 

-re való 4 klikkelés után

```
01.   <canvas height="300">
02.      <window name="ablak1" title="Ablak1" width="100" height="110">
03.         <view height="${parent.height-20}" width="${parent.width-4}">
04.            <simplelayout inset="5" axis="y" spacing="5"/>
05.            <button onclick="canvas.meretez(10);">
06.               Növel
07.            </button>
```

```
08.            <button onclick="canvas.meretez(-10);">
09.               Csökkent
10.            </button>
11.         </view>
12.      </window>
13.
14.      <method name="meretez" args="parameter">
15.         ablak1.setAttribute("width", ablak1.width+parameter);
16.         ablak1.setAttribute("height", ablak1.height+parameter);
17.      </method>
18.
19.      <window name="ablak2" title="Ablak2"
20.              x="${ablak1.x + ablak1.width}">
21.         <simplelayout inset="5" axis="y" spacing="5"/>
22.         <view bgcolor="red">
23.            <text fontsize="10" text="Ablak1 adatai" />
24.         </view>
25.         <view>
26.            <simplelayout inset="5" axis="y" spacing="3"/>
27.            <text fgcolor="blue" text="Szélesség:"/>
28.            <text name="ablak1_width"
29.                  text="${canvas.ablak1.width}"/>
30.            <text fgcolor="blue" text="Magasság:"/>
31.            <text name="ablak1_height"
32.                  text="${canvas.ablak1.height}"/>
33.         </view>
34.      </window>
35.   </canvas>
```

4. OpenLaszlo forrásállomány – javascript_method.lzx

A fenti program annyit csinál, hogy van két ablakunk, melyek közül az „Ablak1”-ben a „Növel” gombra klikkelve megnöveljük az ablakának szélességét, és magasságát 10 pixellel, a „Csökkent” gombbal pedig 10 pixellel csökkentjük. Az „Ablak2” mindig tartalmazza az „Ablak1” magasságát és szélességét pixelben. Az „Ablak2” hozzá van ragasztva az „Ablak1”-hez, tehát ha az „Ablak1”-et elmozdítjuk, vagy átméretezzük, akkor az „Ablak2” mindig annak jobb szélétől fog kezdődni

Az OpenLaszlo elemek attribútumaiban a „$” (dollár) jellel adhatjuk, hogy a kapcsos-zárójelek közti rész egy értelmezendő rész.

Az OpenLaszlo-ban háromféle képpen lehet elérni egy tetszőleges elem attribútumának értékét. Ebben a programkódban mindháromra van példa.

1. A gyökérből kiindulva, például: `${canvas.ablak1.height}`.
   Itt abszolút elérés látható, vagyis mindegy, hogy honnan próbálom elérni azt az adattagot, ez a parancs mindig ugyanazt eredményezi. A `canvas` mindig a gyökér elem azonosítója.
2. Név vagy azonosító alapján, például: `${ablak1.x + ablak1.width}`.

Ebben az esetben is egyfajta abszolút hivatkozás van, de nem a gyökér elemtől, hanem példánkban az „ablak1” nevű objektum-tól tekintve. Ezt vagy a name, vagy az id tulajdonság alapján érhetjük el. Hivatkozáskor nem teszünk különbséget.

3. Az aktuális pozíciónktól tekintve, például: ${parent.height-20}.
   Ilyenkor relatív hivatkozás van. Az aktuális elemhez képest tudunk elérni adattagokat. A „parent” egyel fentebb lép az TAG-fában. Az aktuális elemünk egy adatát a „this”-el tudjuk elérni.

A kódban lévő <view> elem az OpenLaszlo-ban talán a legtöbbet használt elem. Bármi látható elem a vásznunkon egy <view>, vagy egy a <view> osztályból származtatott elem (közvetlen, vagy közvetve). A <view> egy téglalap, amit beállíthatunk láthatóra (visible=”true”), vagy láthatatlanra (visible=”false”). De adhatunk neki színt (bgcolor=”red”), méretet (height=”100%”, width=”50”), beállíthatjuk tartalmának igazítását (align=”center”, valign=”middle”) és rengeteg mást is beállítást végezhetünk rajta.

A <simplelayout> a <layout> elem egy leszármaztatottja. Ez az automatikus elrendezést szolgálja. Minden megjeleníthető elemet el kell rendezni valahogy, vagy különben egymásra csúsznak. A <layout> egy absztrakt osztály, tehát elrendezésre az nem használható közvetlenül. A <simplelayout> vízszintes (axis=”x”) vagy függőleges (axis=”y”) (vagy a kettőt egyszerre használva átlós) elrendezésre szolgál. Használata igen egyszerű. A leírás fában, a vele közös szülővel rendelkező (testvér) elemek elrendezését szabályozza. Az inset tulajdonság az első elemnek, az őt tartalmazó elemtől baloldali, vagy felső szélétől való távolság pixelben. A spacing tulajdonsággal pedig beállíthatjuk az elemek közti távolságot pixelben.

A következő kód ugyanazt eredményezi, mint az előző. A különbség csak a kódban van.

javascript_handler.lzx:

```
01.  <canvas height="300">
02.     <window name="ablak1" title="Ablak1" width="100" height="110">
03.        <view height="${parent.height-20}" width="${parent.width-4}">
04.           <simplelayout inset="5" axis="y" spacing="5"/>
05.           <button>
06.              Növel
07.              <handler name="onclick">
08.                 parent.parent.setAttribute("width",
```

```
09.                         parent.parent.width+10);
10.                  parent.parent.setAttribute("height",
11.                         parent.parent.height+10);
12.               </handler>
13.            </button>
14.            <button>
15.               Csökkent
16.               <handler name="onclick">
17.                  parent.parent.setAttribute("width",
18.                         parent.parent.width-10);
19.                  parent.parent.setAttribute("height",
20.                         parent.parent.height-10);
21.               </handler>
22.            </button>
23.         </view>
24.      </window>
25.
26.      <window name="ablak2" title="Ablak2"
27.              x="${ablak1.x + ablak1.width}">
28.         <simplelayout inset="5" axis="y" spacing="5"/>
29.         <view bgcolor="red">
30.            <text fontsize="10" text="Ablak1 adatai" />
31.         </view>
32.         <view>
33.            <simplelayout inset="5" axis="y" spacing="3"/>
34.            <text fgcolor="blue" text="Szélesség:"/>
35.            <text name="ablak1_width" text="${canvas.ablak1.width}"/>
36.            <text fgcolor="blue" text="Magasság:"/>
37.            <text name="ablak1_height"
38.                  text="${canvas.ablak1.height}"/>
39.         </view>
40.      </window>
41.   </canvas>
```

5. OpenLaszlo forrásállomány – javascript_handler.lzx

A két alkalmazások közt a legnagyobb különbség a `<method>` és a `<handler>` elemek. Mindkét elem JavaScript kódokat zár körül.

A `<method>` mint a neve is mutatja egy metódus leírására alkalmas. A metódusunk értékkel is térhet vissza. Ez a `return` parancs utáni érték lesz, és akkor egy függvénynek fogható fel. A `<method>`-nak kötelezően rendelkeznie kell névvel. Ez alapján lehet rá hivatkozni, ez alapján lehet futásra bírni valamilyen JavaScript segítségével. Az `args` tulajdonsággal fel tudjuk paraméterezni a metódusunkat. Adhatunk egyszerre több paramétert is neki, amiket vesszővel elválasztva kell felsorolnunk. Például:

osszeadas_method.lzx:

```
11 + 6 = 17
```

```
01.   <canvas>
02.      <text id="szoveg">
03.         <method name="osszeadas" args="a, b">
04.            this.setAttribute('text', (a+' + '+b+' = '));
05.            return a+b;
```

```
06.         </method>
07.      </text>
08.      <script>
09.         szoveg.addText(szoveg.osszeadas(11, 6));
10.      </script>
11.   </canvas>
```

6. OpenLaszlo forrásállomány – osszeadas_method.lzx

A `<script>` elem arra való, hogy JavaScript kódokat írhatunk közé, ami betöltődéskor le is fut. `scr` tulajdonságként megadhatunk benne egy fájlt is, ami ugyanúgy fog lefutni, mintha annak a fájlnak a tartalma a elemek közt lenne. A `<script>` elem csakis és kizárólag a gyökér `<canvas>` elemben tartózkodhat. Vagyis a TAG fa első szintjén, lentebb nem.

A `<handler>` elem akkor hajtódik végre, ha `name` attribútumában megadott esemény kiváltódik az őt tartalmazó elemen. Tehát a `name` tulajdonságának mindenféleképpen definiálva kell lennie, ami mindig egy esemény (pl.: `init`, `onclick`, `onmouseover`, `onmouseout` stb…). Működése teljes mértékben megegyezik a `<method>` működésével, annyi eltéréssel, hogy a `<method>`-ot nekünk kell a program kódban kézzel meghívnunk, amíg a `<handler>`-t egy eseménnyel indíthatunk el futás közben. Ez az elem is rendelkezik a `<method>`-nál megismert `args` attribútummal. De megadható neki például `method` tulajdonság is, ami tartalmazza azt a metódus nevet, amit a `<handler>` meghívásakor meg szeretnénk hívni.

Tekintsük a következő rövidke kódot:

faktor_hibas.lzx:

```
01.   <canvas>
02.      <text id="szoveg" />
03.      <method name="faktor" args="n">
04.         var f = 1;
05.         for (i = 1; i <= n; i++)
06.            f *= i;
07.         return (n+'! = '+f);
08.      </method>
09.      <script>
10.         szoveg.addText(canvas.faktor(5));
11.      </script>
12.   </canvas>
```

7. OpenLaszlo forrásállomány – faktor_hibas.lzx

Ez a kód nem fog lefutni. A fordító azt sem fogja érteni, hogy mit is szeretnénk csinálni. A probléma az OpenLaszlo XML szintaktikájából ered, és az ***5.*** sorban találjuk: `for (i = 1;`

`i <= n; i++)`. A „<” jelet az OpenLaszlo fordító egy elem nyitó jeleként értelmezné, de az nem lehet. Erre a problémára 2 megoldás létezik.

faktor_xml.lzx:

```
5! = 120
01.  <canvas>
02.     <text id="szoveg" />
03.     <method name="faktor" args="n">
04.        var f = 1;
05.        for (i = 1; i &lt;= n; i++)
06.           f *= i;
07.        return (n+'! = '+f);
08.     </method>
09.     <script>
10.        szoveg.addText(canvas.faktor(5));
11.     </script>
12.  </canvas>
```

8. OpenLaszlo forrásállomány – faktor_xml.lzx

Ekkor a „<” jel helyett az XML által ennek a karakternek a használhatóságára definiált karaktersorozatot (`&lt`) használjuk. Ezeket a karaktereket megtalálhatjuk az XML-ről szóló fejezet végén.

De ettől létezik egy elegánsabb, illetve nagyobb kódoknál sokkal praktikusabb megoldás is.

faktor_cdata.lzx:

```
5! = 120
01.  <canvas>
02.     <text id="szoveg" />
03.     <method name="faktor" args="n">
04.        <![CDATA[
05.        var f = 1;
06.        for (i = 1; i <= n; i++)
07.           f *= i;
08.        return (n+'! = '+f);
09.        ]]>
10.     </method>
11.     <script>
12.        szoveg.addText(canvas.faktor(5));
13.     </script>
14.  </canvas>
```

9. OpenLaszlo forrásállomány – faktor_cdata.lzx

A „**`<![CDATA[`**” és „**`]]>`**” közötti JavaScript kódot az OpenLaszlo fordító kihagyja, és nem keres benne semmit, ami számára értelmezhető. Ezt a részt csak a böngészőnk fogja értelmezni, és így már tiszta JavaScript kódot írhatunk.

5.5. Class

Mint azt már említettem, az OpenLaszlo egy objektum orientált nyelv. Definiálhatunk saját osztályokat, amiket származtathatunk és példányosíthatunk is. Az osztályok segítségével nagymértékben csökkenthetjük kódjaink hosszát.

Ez a nyelv igen erősen támogatja a multimédiás elemeket is, többek között a képeket, videókat, hangfájlokat is. A képek kezelésén keresztül fogom bemutatni az objektumok létrehozásának, és példányosításának módszerét.

Az alábbi program úgy működik, hogy minden példányosításkor kirajzolódik egy kép, és ha valamelyik fölé visszük az egeret, akkor a kép címe alatti rész átszíneződik, és a kép lentebb csúszik. Ha levisszük róla az egeret, akkor visszaáll alaphelyzetbe a kép.

class.lzx:

7. Kép – Kép példányok

Betöltődéskor

8. Kép – Kép példányok onmouseover

Ha az egeret a harmadik kép fölé visszük

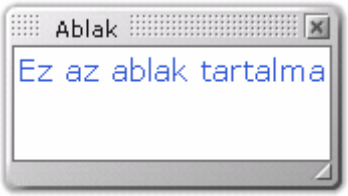
```
01.   <canvas>
02.
03.      <class name="myPicture"
04.             onmouseover="this.move(50)"
05.             onmouseout="this.move(-50)" >
06.         <attribute name="src" type="text"
07.                    value="pictures/default_pic.jpg" />
08.         <attribute name="myTitle" type="string" value="${src}" />
09.         <attribute name="border" value="0" />
10.         <attribute name="borderColor" type="text" value="#DDDDDD" />
```

```
11.         <view name="picBorder"
12.               bgcolor="${parent.borderColor}"
13.               width="${pic.width+2*(parent.border % 10)}"
14.               height="${pic.height+2*(parent.border % 10)}">
15.            <image name="pic"
16.                   src="${parent.parent.src}"
17.                   x="${parent.parent.border % 10}"
18.                   y="${parent.parent.border % 10}"/>
19.
20.            <text name="picName"
21.                  bgcolor="#DDDDDD"
22.                  fgcolor="#000000"
23.                  text="${parent.parent.myTitle}"
24.                  y="${parent.pic.height +
25.                       2*(parent.parent.border % 10 )}"
26.                  width="${parent.pic.width +
27.                       2*(parent.parent.border % 10 )}" />
28.         </view>
29.         <method name="move" args="pix">
30.            picBorder.animate('y', pix, 500, true);
31.            if (pix > 0)
32.               picBorder.picName.setAttribute('bgcolor', '#456789');
33.            else
34.               picBorder.picName.setAttribute('bgcolor', '#DDDDDD');
35.         </method>
36.      </class>
37.
38.      <simplelayout spacing="10" axis="x" />
39.
40.      <myPicture />
41.      <myPicture myTitle=":)" border="15" borderColor="red"/>
42.      <myPicture src="pictures/picture1.jpg" border="5" />
43.      <myPicture src="pictures/picture2.jpg" myTitle="Pic2"/>
44.
45.   </canvas>
```

10. OpenLaszlo forrásállomány – class.lzx

A `<class>` elem a saját osztály létrehozására szolgáló elem. Példánkban az osztályunk neve `myPicture`, és amint ezt elemként használjuk, azzal az objektumunk egy példányát hozzuk létre. Az osztályunkban az `extends` tulajdonsággal adhatjuk meg, hogy milyen már létező OpenLaszlo-beli osztályból származtatjuk. Ezzel átvesszük annak minden tulajdonságát, amiket már eleve be tudunk rajta állítani. Lássuk erre példának a 4.2-es fejezetben lévő ablak.lzx programot objektum orientált szemlélettel.

oo_ablak.lzx:



9. Kép – OO Ablak alaphelyzetben

Betöltéskor

10. Kép – OO Ablak elmozgatva

Átméretezés és elmozgatás után

```
1.  <canvas height="300">
2.     <class name="myWindow" extends="window">
3.        <text fontsize="15" fgcolor="#325AD1">
4.           Ez az ablak tartalma
5.        </text>
6.     </class>
7.     <myWindow height="100" resizable="true"
8.               closeable="true" title="Ablak" />
9.  </canvas>
```

11. OpenLaszlo forrásállomány – oo_ablak.lzx

Az osztályunkban lévő `<attributum>` elemek az osztályunk paraméterei, vagy tulajdonságai. Amikor példányosítjuk az osztályunkat, akkor a példányosító elemünk attribútumaival állíthatjuk be ezeket. Ha valamelyiket nem állítjuk be, akkor annak értéke az osztályleírásunkban megadott alapértelmezett érték lesz. Az `<attributum>` TAG-ben megadhatjuk többek között annak nevét (`name`), amire hivatkozva beállíthatjuk az értékét példányosításkor, illetve típusát (`type`), ami meghatározza, hogy milyen értékeket vehet fel. Ezek a típusok a következők lehetnek: `boolean`, `color`, `expression`, `number`, `size`, `string`, `text`, `html`.

Megadhatjuk még az alapértelmezett értékét (`value`), ami például `number` típus esetén szám, `string` esetén szöveg, stb.…

Az `<attributum>` elemet bárhol használhatjuk az alkalmazásunkban, és ezeket lehet használni változók ként, mint azt majd a dokumentum végén lévő szamologep.lzx programban is láthatjuk.

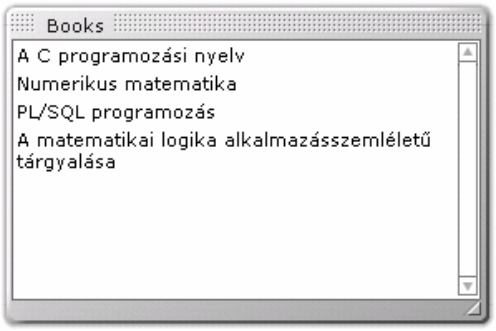
Amint azt láthatjuk is, definiálható a saját osztályunkban `<method>`, illetve `<hander>` elem is. Ezek az osztályunk eljárásai, függvényei.

Az osztályunkban kell leírnunk, hogy az mit jelenítsen meg, mint a class.lzx példánkban a `<view>`, az `<image>` és a `<text>` elemek. Ezeken tetszőlegesen operálhatunk ugyanúgy, minthogyha nem osztályban lennének.

5.6. XML olvasás + dataset

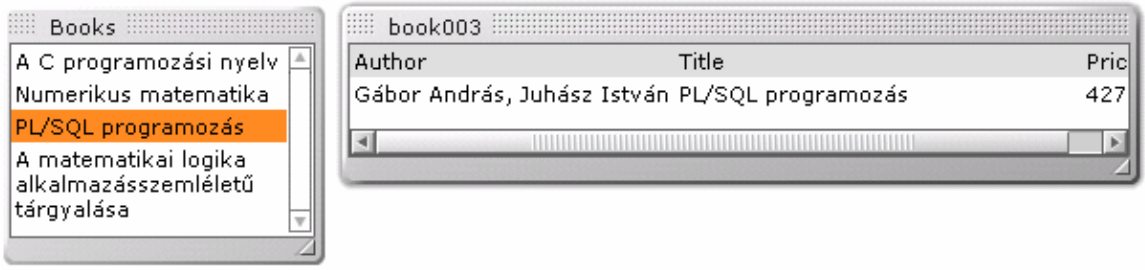
A `<dataset>` egy memóriabeli XML stílusban lévő adat hierarchiát definiál. Például ennek segítségével tudunk egy XML állományt beolvasni, vagy egy XML formátumú HTTP kérést értelmezni. Én a példámban egyszerű XML állományt olvastatok be. Bármilyen XML formátumú adatot ilyen módon kell kezelni.

xml.lzx:



11. Kép - Könyvek

Betöltéskor



12. Kép – Könyv kiválasztva

Elmozgatás, átméretezés és a 3. sorra klikkelés után

```
01.  <canvas>
02.
03.     <dataset name="dsBooks" src="data/books.xml"/>
04.
05.     <window name="main" title="Books" width="300"
06.             height="200" resizable="true">
07.        <view id="showBooks" datapath="" width="${parent.width}">
08.           <text id="tit" width="${parent.width}" multiline="true"
09.                 onmouseover="this.setAttribute('bgcolor', '#FF8822')"
10.                 onmouseout= "this.setAttribute('bgcolor', null)">
11.
12.              <datapath xpath="dsBooks:/books/book/@title"/>
13.              <handler name="onclick">
14.                 bookWindow.datapath.setFromPointer( this.datapath );
15.              </handler>
16.           </text>
17.           <simplelayout/>
18.        </view>
19.     <scrollbar axis="y" />
20.     </window>
21.
22.     <include href="includes/bookWindow.lzx" />
23.
24.  </canvas>
```

12. OpenLaszlo forrásállomány – xml.lzx

A <dataset>-et name attributumával lehet hivatkozni. Az src az adatforrást tartalmazza, ami

- ha URL („http:”-vel kezdődik), akkor a dataset értéke az XML adat, amit megkap az URL-től, amikor az alkalmazás fut.

  Például:

```
<dataset name=”alkalmazottak”
     src=”http://www.peldaoldal.hu/alk.xml” />
```

- ha egy pathname (útvonal név, mint a példánkban is), akkor az adat betöltődik már fordítás közben.
- ha nincs megadva, akkor a <dataset> elemek közti rész.

  Például:

```
<dataset name=”alkalmazotti_lista”>
   <alkalmazottak>
      <alkalmazott>
         <név>Jánosz</név>
      </alkalmazott>
      <alkalmazott>
         <név>Sándor</név>
      </alkalmazott>
   </alkalmazottak>
</dataset>
```

A <dataset> elemnek paraméterként megadható a timeout, aminek az értéke egy szám, és azt határozza meg, hogy mennyi ideig várjon (ezredmásodpercben) az alkalmazás az adat betöltésére.

A cacheable logikai attribútum arra való, hogy megadjuk, hogy elmentheti-e a cache-be az alkalmazásunk az XML adatot. Alapértelmezés szerint ez tiltva van. Ez biztonsági okokból hátrányos, de a gyorsaság szempontjából előnyős.

A <datapath> összekötő az adatok és az XML node-ok (csomópontok) között. Egyesíti az adat hierarchiát az OpenLaszlo alkalmazás hierarchiájával. Ezt lényegében úgy is felfoghatjuk, hogy egy tömbbe rendezi a kívánt adatokat, mint példánkban (<datapath xpath="dsBooks:/books/book/@title"/>) a dsBooks nevű dataset-nek a books elemének a book gyermekének a title attribútumát. Ezekre többek között megadható például az, hogy hogyan legyenek rendezve (sortorder=”ascending” azaz növekvően. A csökkenő: descending).

Amint megvan ez a tömbbe rendezés, és ennek a `<datapath>` tömbnek hivatkozunk valamelyik adattagjára, mint alkalmazásunk következő (include/bookWindow.lzx) fájljában például: `<text width="${parent.width * 0.1}" datapath="@price"/>`, akkor az lefut minden egyes `price` tömbelemre. Ebben a példában csak egy egyelemű tömböt, adunk át, ezért csak egyszer fut le. Ez a tömb heterogén adat kötegeket tartalmaz, azaz egy ilyen tömbben nem csak például a `price` szerepel, hanem az ahhoz a `price`-hoz tartozó `tittle`, `author` és `id` is.

A ***22.*** sorban lévő `<include>` elem arra szolgál, hogy egy másik állomány tartalmát olvassa be. Működésileg úgy néz ez ki, mintha annak az állománynak a tartalma ott lenne a ***22.*** sorban. Ezzel a módszerrel a hosszabb alkalmazásainkat sokkal átláthatóbbá lehet tenni. A külső fájlunk gyökér eleme nem a `<canvas>` lesz, hanem `<library>`. De ha csak egy osztályt tartalmaz a fájl, akkor lehet `<class>` is.

És íme a példánkban hivatkozott

includes/bookWindow.lzx:

```
01. <library>
02.
03.    <window name="bookWindow" y="${ main.y }"
04.            x="${ main.x + main.width }"
05.            resizable="true" closeable="false"
06.            width="500" height="100"
07.            title="$path{ '@id' }">
08.
09.       <datapath />
10.       <view>
11.          <view bgcolor="#DDDDDD" width="${parent.parent.width}">
12.             <text width="${parent.width * 0.4}" text="Author"/>
13.             <text width="${parent.width * 0.5}" text="Title"/>
14.             <text width="${parent.width * 0.1}" text="Price"/>
15.             <simplelayout axis="x"/>
16.          </view>
17.          <view id="showBook" width="${parent.parent.width}" >
18.             <text width="${parent.width * 0.4}"
19.                   datapath="@author"/>
20.             <text width="${parent.width * 0.5}"
21.                   datapath="@title"/>
22.             <text width="${parent.width * 0.1}"
23.                   datapath="@price"/>
24.             <simplelayout axis="x"/>
25.          </view>
26.          <simplelayout axis="y"/>
27.       </view>
28.       <scrollbar axis="x"/>
29.    </window>
30.
31. </library>
```

13. OpenLaszlo forrásállomány – includes/bookWindow.lzx

A program most következő XML fájlt olvassa be, mely könyvek szerzőit, címét, és árát tartalmazzák, valamint egy azonosítót.

data/books.xml:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<books>
   <book id="book001"
         author="Brian W. Kernighan, Dennis M Ritchie"
         title="A C programozási nyelv"
         price="3500" />
   <book id="book002"
         author="Stoyan Gisbert"
         title="Numerikus matematika"
         price="3200" />
   <book id="book003"
         author="Gábor András, Juhász István"
         title="PL/SQL programozás"
         price="4275" />
   <book id="book004"
         author="Pásztorné V. Katalin, Várterész Magda"
         title="A matematikai logika alkalmazásszemléletű tárgyalása"
         price="3900" />
</books>
```

8. XML – data/books.xml

Mint látható, ez az XML fájl az attribútumaiban tárolja az információkat, nem pedig nyitó és záró elemek közt.

5.7. Számológép sok JavaScripttel

Most a végére bemutatok az előzőektől egy kicsit összetettebb alkalmazást, mely egy egyszerű számológép. Ebben a példában szemügyre vehetjük a JavaScript használhatóságát, és hogy milyen könnyedén tud kommunikálni az OpenLaszlo elemekkel.

A következő kódot a sok komment miatt a könnyebb olvashatóság érdekében kissé másabb stílusban írom, mint az eddigieket. A program kódja félkövéren van szedve, a megjegyzések elemei normál betűvel, a megjegyzések szövege pedig dőlt.

szamologep.lzx:

13. Kép - Számológép

```
<canvas height="500">

   <include href="includes/numbers.lzx" />
   <include href="includes/operations.lzx" />

   <attribute name="pressedButtons" value="0" />
<?ignore
A pressedButtons nevű numerikus változó bitjeiben tárolom a lenyomott
gombokat a következő módon:
1. bit (1):  ,
2. bit (2):  +
3. bit (4):  -
4. bit (8):  *
5. bit (16): /
?>
   <attribute name="fraction" value="1" />
<?ignore
A fraction nevű numerikus változóban tárolom, hogy ha tizedes
számjegyet írunk, akkor mennyivel kell elosztani, hogy jó helyi
értékre kerüljön
Például ha 0.0008-at szeretnénk fűzni a számunkhoz, akkor a fraction
értéke: 10000
?>
   <attribute name="lastNumber" value="0" />
<?ignore
A lastNumber nevű numerikus változóban tárolom az előző számot, amin
operálni kell.
Például ha beírtuk már a következőt:
12 + 4
akkor a 12 van eltárolva
?>
   <attribute name="nowPressed"
              type="boolean" value="false" />
<?ignore
A nowPressed nevű logikai változóban jelzem, hogy az utolsó lenyomott
gomb valamilyen operátor volt-e. Mert ha igen, akkor a következő
számjegy lenyomásakor előbb törölni kell a kijelzőt.
?>

   <window name="win" title="Calculator"
           width="162" height="200"
```

```
42.             resizable="false" closeable="false">
43.       <simplelayout axis="y" />
44.       <edittext name="number_e" text="0"
45.                 width="${parent.width-18}"
46.                 maxlength="18" pattern="[0-9]*"/>
47.       <view width="${parent.width-18}"
48.             height="${parent.height}" bgcolor="gray" >
49.
50.          <wrappinglayout axis="x"/>
51.  <?ignore
52.  A <wrappinglayout> folyamatos elrendezést eredményez, vagyis a
53.  tengely irányban teszi sorba/oszlopba az elemeket a szülő elem
54.  végéig, majd új sorban/oszlopban folytatja
55.  ?>
56.
57.          <number_b name="_1" num="1"/>
58.          <number_b name="_2" num="2"/>
59.          <number_b name="_3" num="3"/>
60.          <operator_b name="_plu" op="+"/>
61.          <number_b name="_4" num="4"/>
62.          <number_b name="_5" num="5"/>
63.          <number_b name="_6" num="6"/>
64.          <operator_b name="_min" op="-"/>
65.          <number_b name="_7" num="7"/>
66.          <number_b name="_8" num="8"/>
67.          <number_b name="_9" num="9"/>
68.          <operator_b name="_mul" op="*"/>
69.          <operator_b name="_cha" op="+/-"/>
70.          <number_b name="_0" num="0"/>
71.          <operator_b name="_fra" op=","/>
72.          <operator_b name="_div" op="/"/>
73.          <operator_b name="_c" op="C"
74.                      myWidth="${(parent.width/2)-1}" />
75.          <operator_b name="_ent" op="="
76.                      myWidth="${(parent.width/2)-1}" />
77.  <?ignore
78.  A <number_b> TAG-ek a számokat ábrázoló gombok osztályának példányai
79.  Az <operator_b> TAG-ek az összes többi gomb osztályának példányai
80.  ?>
81.       </view>
82.    </window>
83.
84.  </canvas>
```

14. OpenLaszlo forrásállomány – szamologep.lzx

includes/numbers.lzx:

```
01.  <library>
02.
03.     <class name="number_b" extends="button" width="35">
04.        <attribute name="num" value="0" />
05.  <?ignore
06.  A num nevű attribútum-mal adhatjuk meg, hogy melyik számjegyet
07.  jelentse az adott gomb
08.  ?>
09.        <text fgcolor="red" text="${parent.num}"
10.              align="center" valign="middle" />
11.
12.        <handler name="onclick">
```

```
13.          <![CDATA[
14.
15.          var newNum;
16.  <?ignore
17.  A newNum változóban tároljuk a kijelző értékét numerikusan
18.  ?>
19.          var numText = canvas.win.number_e.text;
20.  <?ignore
21.  A numText változóban tároljuk a kijelző értékét karakteresen
22.  ?>
23.          var positive = true;
24.  <?ignore
25.  A positive változóban tároljuk, hogy az eddig beírt szám negatív vagy
26.  pozitív-e
27.  ?>
28.
29.          if (canvas.nowPressed){
30.  <?ignore Ha az előző lenyomott billentyű operátor billentyű volt ?>
31.             numText = '0';
32.             canvas.nowPressed = false;
33.          }
34.          newNum = parseFloat(numText);
35.  <?ignore Itt alakítjuk numerikussá a kijelző értékét ?>
36.
37.          if ( newNum < 0 ){
38.  <?ignore
39.   Ha az eddigi számunk negatív, akkor pozitívvá alakítom a gyorsabb és
40.  könyebb számolás érdekében
41.   ?>
42.             positive = false;
43.             newNum *= -1;
44.          }
45.
46.          if ( (canvas.pressedButtons & 1) != 1 ){
47.  <?ignore
48.  Ha az eddigi számunk egész, akkor megszorozzuk azt 10el, és
49.  hozzáadjuk az új számjegyet
50.  ?>
51.             newNum *= 10;
52.             newNum += num;
53.          } else {
54.  <?ignore
55.  Ha az eddigi számunk NEM egész, akkor az új számjegyet a megfelelő
56.  helyiértékre helyezzük, és hozzáadjuk az eredeti számhoz
57.  ?>
58.             canvas.fraction *= 10;
59.             newNum += (num / canvas.fraction);
60.          }
61.
62.          if ( !positive )
63.  <?ignore Ha az eredeti számunk negatív volt, akkor visszaalakítjuk ?>
64.             newNum *= -1;
65.
66.          canvas.win.number_e.setAttribute('text', newNum);
67.  <?ignore Kiírjuk a kijelzőre az új számot ?>
68.
69.          ]]>
70.       </handler>
71.    </class>
```

```
72.
73.  </library>
```

15. OpenLaszlo forrásállomány – includes/numbers.lzx

includes/operations.lzx:

```
01.  <library>
02.
03.     <class name="operator_b" extends="button"
04.            text="${op}" width="${myWidth}">
05.        <attribute name="op" type="string" value="=" />
06.  <?ignore
07.  Az op attribútumban adhatjuk meg, hogy milyen gombot reprezentáljon
08.  az adott objektum
09.  ?>
10.        <attribute name="myWidth" value="35" />
11.  <?ignore A myWidth attribútummal megadhatjuk a gombunk szélességét ?>
12.
13.        <handler name="onclick">
14.           <![CDATA[
15.
16.  <?ignore
17.  A következő calculate függvény az eltárolt és az új értékkel
18.  kiszámolja az operátor függvényében az új értéket.
19.  ?>
20.           function calculate(num1, num2, operator){
21.              if ( (operator & 2) == 2 ){
22.                 canvas.pressedButtons -= 2;
23.                 return (num1 + num2);
24.              } else if ( (operator & 4) == 4 ){
25.                 canvas.pressedButtons -= 4;
26.                 return (num1 - num2);
27.              } else if ( (operator & 8) == 8 ){
28.                 canvas.pressedButtons -= 8;
29.                 return (num1 * num2);
30.              } else if ( (operator & 16) == 16 ){
31.                 canvas.pressedButtons -= 16;
32.                 return (num1 / num2);
33.              }
34.
35.              return 0;
36.           }
37.
38.           var numText = canvas.win.number_e.text;
39.           var newNum;
40.           if (numText != '')
41.              newNum = parseFloat(numText);
42.           else
43.              newNum = 0;
44.           numText = toString(newNum);
45.
46.  <?ignore Kiválasztjuk, hogy melyik gomb lett lenyomva ?>
47.           switch (op){
48.              case 'C':{
49.  <?ignore
50.  Ha a 'C' feliratú törlés (clear) gomb lett lenyomva, akkor minden
51.  értéket alaphelyzetbe állítunk
52.  ?>
53.                 canvas.win.number_e.setAttribute( 'text', '0');
```

```
54.                canvas.pressedButtons = 0;
55.                canvas.fraction = 1;
56.                break;
57.             }
58.             case '+/-':{
59.    <?ignore
60.    A '+/-', azaz előjelváltó gombra klikkelve az eddigi számunkat
61.    megszorozzuk -1-el, hogy előjelet váltson
62.    ?>
63.                if (newNum != 0)
64.                   canvas.win.number_e.setAttribute( 'text',
65.                          newNum * -1);
66.                break;
67.             }
68.             case ',':{
69.    <?ignore
70.    A ',' gombot lenyomva eltároljuk a pressedButtons attribútumban, hogy
71.    innentől kezdve nem egész résszel bővítjük a számot, hanem tört
72.    számmal
73.    ?>
74.                if ((canvas.pressedButtons & 1) != 1)
75.                   canvas.pressedButtons += 1;
76.                break;
77.             }
78.             case '=':{
79.    <?ignore
80.    Az '=' gomb hatására meghatározzuk az új értéket a calculate függvény
81.    segítségéve, ha volt lenyomott művelet az utolsó számolás óta
82.    ?>
83.                canvas.nowPressed = true;
84.                canvas.fraction = 1;
85.                if (canvas.pressedButtons > 1){
86.                   newNum = calculate(canvas.lastNumber, newNum,
87.                                      canvas.pressedButtons);
88.                   canvas.win.number_e.setAttribute( 'text', newNum);
89.                   canvas.lastNumber = newNum;
90.                }
91.
92.                if ((canvas.pressedButtons & 1) == 1)
93.                   canvas.pressedButtons -= 1;
94.                break;
95.             }
96.             default :{
97.    <?ignore
98.    A default ágban a 4 alapművelet eltárolása, és elvégzése van
99.    ?>
100.               canvas.nowPressed = true;
101.               canvas.fraction = 1;
102.               if (canvas.pressedButtons > 1){
103.   <?ignore
104.   Ha van már eltárolva alapművelet, akkor mostmár számolni kell
105.   ?>
106.                  newNum = calculate(canvas.lastNumber, newNum,
107.                                     canvas.pressedButtons);
108.                  canvas.win.number_e.setAttribute( 'text', newNum);
109.                  canvas.lastNumber = newNum;
110.               }
111.               switch (op){
112.                  case '+': canvas.pressedButtons += 2; break;
```

```
113.                    case '-': canvas.pressedButtons += 4; break;
114.                    case '*': canvas.pressedButtons += 8; break;
115.                    case '/': canvas.pressedButtons += 16; break;
116.                }
117.                canvas.lastNumber = newNum;
118.
119.                if ((canvas.pressedButtons & 1) == 1)
120.  <?ignore
121.  Műveleti billentyű lenyomása esetén új szám következik, és az nem
121.  kezdődhet törtként ként
123.  ?>
124.                    canvas.pressedButtons -= 1;
125.
126.                break;
127.             }
128.          }
129.
130.          ]]>
131.       </handler>
132.
133.    </class>
134.
135   </library>
```

16. OpenLaszlo forrásállomány – includes/operations.lzx

6. Összefoglalás

A dolgozatban a Laszlo Systems által kifejlesztett, OpenLaszlo nevű webfejlesztési eszköz alapszintű bemutatását tűztem ki célul. A tárgyalását nehezítette, hogy magyar nyelvű dokumentációkat még nem tettek közzé, csak angol nyelvű dokumentumok találhatóak ezzel kapcsolatosan. Így több helyen nehéz volt megérteni bizonyos eszközök működését. A másik nehezítés az volt, hogy körültekintően kellett eljárnom a kiválasztott eszközökkel kapcsolatban a rövid tárgyalási rész, és a dokumentum statikussága miatt.

Remélhetőleg felkeltettem több webfejlesztő érdeklődését a nyelv iránt, és ez a könnyen használható, igen látványos, interaktív, és gyorsan fejlődő eszköz sokkal jobban el fog terjedni.

A téma terjedelme miatt nem lehet minden alapnak számító eszközt bemutatni az OpenLaszlo világában, de az elinduláshoz, és az alapvető ismeretek elsajátításához ezeket az eszközöket találtam a legcélravezetőbbnek. A mélyebb megismeréshez az Irodalomjegyzékben feltűntettem néhány weboldalt, és egy nagyon jól megszerkesztett könyvet, melyek angol nyelvezetük mellett is igen érthetőek.

A szerző az anyaggal kapcsolatosan bármilyen észrevételt szívesen fogad a tholdt@gmail.com e-mail címen.

7. Irodalomjegyzék

Könyvek:

- Norman Klein, Max Carlson with GlennMacEwen: Laszlo in Action

  Weboldal: http://www.manning.com/klein

Weboldalak:

- Laszlo System weboldala: http://www.laszlosystem.com
- OpenLaszlo weboldala: http://www.openlaszlo.org
- JavaScript és XML e-bookok: http://ebookz.hu
- Wikipedia: http://en.wikipedia.org/wiki/Openlaszlo

Videó tutorialok:

- OpenLaszlo 4 Programming Tutorial:

  http://blip.tv/file/get/Laszlo-OpenLaszlo4ProgrammingTutorial213.flv

  vagy

  http://icharleston.multiply.com/video/item/2/Laszlo-OpenLaszlo4ProgrammingTutorial213.flv

  (Ez a két link ugyanaz a videó, csak a neten történő sok oldalmegszüntetés miatt raktam ki kétszer)

8. Függelékek

8.1. OpenLaszlo telepítése

8.1.1. Szerver telepítési útmutató Windows-hoz

1. Töltsd le az OpenLaszlo szervert a Windows-hoz a következő helyről: http://www.openlaszlo.org/download
   Ez tartalmazza a fordítót, a futásidejű könyvtárakat (runtime libraries), magát az Laszlo LZX nyelvet, sok LZX példát, egy Tomcat szervert és a JDK-t is.
2. Futtasd a telepítőt, amiben csak el kell fogadni a szerződést, és ki kell választani a helyet, ahova telepíteni szeretnénk.
3. A szervered gyökérkönyvtára a 4.3.0-ás verziónál a következő lesz:
   „*kiválasztott könyvtár*”/Server/lps-4.3.0/

A szerver a 8080-as portot fogja használni, tehát azt hagyd szabadon. Ebben akadály leginkább a Skype, vagy egy Apache szerver lehet.

8.1.2. Szerver telepítési útmutató Unix/Linux-hoz

1. Töltsd le az OpenLaszlo szervert a Linux-hoz a következő helyről: http://www.openlaszlo.org/download
2. Csomagold ki az /usr/local –ba (vagy bárhova).
3. NE csak másold a Tomcat könyvtárat az /user/local-ba. Hagyd egyben az egészet, és valami hasonlót fogsz látni:
   /usr/local/lps-4.3.0/
   Ezen belül lennie kell egy Server/Tomcat-5.0.24 könyvtárnak.
4. Biztosítsd a JAVA_HOME helyes beállítását
5. Biztosítsd egy legalább 5-ös Flash playert
6. Majd indítsd el az
   /lps-4.3.0/Server/tomcat-5.0.24/bin/sh
   scriptet. A script beállítja a következő környezeti változókat:
   CATALINA_BASE
   CATALINA_HOME
   CATALINA_TMDIR

JAVA_HOME

7. Most már készen állsz a használatra

8.1.3 Szerver telepítési útmutató MAC OS X-re

1. Töltsd le az lps-4.3.0-macosx.dmg fájlt a következő helyről: http://www.openlaszlo.org/download
2. Telepítsd azt
3. És készen is állsz a használatára

Ha ez hibát eredményez, megtalálod a kézi telepítési útmutatót angol nyelven a http://www.openlaszlo.org/lps4.1/docs/installation/install-instructions.html weboldalon

8.1.4. IDE telepítési útmutató

Az OpenLaszlo-hoz készítettek IDE-t is, ami egy Eclipse kiegészítő. Ezt a következő képpen kell feltelepíteni:

1. Töltsd le és telepítsd az **all in one** eclipse-t, ami tartalmazza a WTP-t (WebTools Platform) a következő helyről: http://www.eclipse.org/webtools/
2. Töltsd le az OpenLaszlo IDE plug-in-t az Eclipse-hez. Elvileg innen lehet, de én nem találtam: http://www.eclipse.org/laszlo/
   Én innen töltöttem le: http://www.riftware.com/laszlodev/laszloIDE.zip
3. Indítsd el az Eclipse-t; a defaultWorkspace tökéletes lesz
4. Válaszd ki a következő menüt: „Help → Software Updates → Find and Install…”.
5. Válaszd a „Search for new features to install”-t.
6. Hozz létre egy „New Archived Site…”-ot, ami mutasson a letöltött ide4laszlo.zip-re.
7. Klikkelj a Finish-re
8. A következő ablakban legyen kipipálva minden eleme a laszloIDE.zip-nek, majd klikkelj a Next-re.
9. Fogadd el a szerződést (I accept…), és klikkelj a Next-re
10. Majd klikkelhetsz a Finishre, aminek hatására telepítődnek az összetevők
11. Indítsd újra az Eclipse-t, és kész is vagy.

Emellett szükségünk lehet a futtatáshoz egy Flash Player-re:
http://www.adobe.com/products/flashplayer/

8.1.5. Minimum követelmény:

8.1.5.1. Kliens gép

Az OpenLaszlo 4.2 a következő platform és böngésző kombinációkon lett tesztelve:

- Windows XP/Internet Explorer 7 -- Flash 8, Flash 9, és DHTML
- Windows XP/Internet Firefox 3 -- Flash 8, Flash 9, és DHTML
- Linux 2.6 kernel Firefox 3 -- Flash 8, Flash 9, és DHTML
- Mac OS X Firefox 3 -- Flash 8, Flash 9, és DHTML
- Mac OS X Safari 3 -- Flash 8, Flash 9, és DHTML

Az OpenLaszlo 4.2 Windows Vistán is lett tesztelve, de az nem volt teljesen tökéletes

8.1.5.2. Szerver

Ajánlatos a szervert csak fejlesztésre használni a lassú fordítási idő miatt, és amikor kész az alkalmazásunk, akkor SOLO módban lefordítani, amivel létrejön a DHTML vagy a Flash kód, és azt telepíteni valamilyen szerver kofigurációra.

Az OpenLaszlo teljes mértékben Java alapú, ezért a fejlesztők a következőket ajánlják:

- Java Runtime Environment (JRE) 1.5 vagy újabb.

  Az OpenLaszlo Developer Kit-nek szüksége van a Java SDK-ra.
- Az OpenLaszlo Core-hoz szükség van a Java Servlet által támogatott 2.2-es Java Servlet Specification-ra.

  A Tomcat 5.0.24-be beintegrálták az OpenLaszlo Developer Kit-et.

8.2. XML-ek

8.3. Táblázatok

8.4. Képek

8.5. OpenLaszlo program fájlok

8.6. OpenLaszlo elemeinek (TAG-jeinek) listája

<a>	<class>	<layout>
<alert>	<columnchart>	<lazyreplicator>
<animator>	<combobox>	<legend>
<animatorgroup>	<command>	<library>
<attribute>	<constantboundslayout>	<linechart>
<audio>	<constantlayout>	<linestyle>
<axis>	<contextmenu>	<list>
<axisstyle>	<contextmenuitem>	<listitem>
<b>	<datacolumn>	<LzTextFormat>
<barchar>	<datacombobox>	<mediadevice>
<basebutton>	<datalabel>	<mediastream>
<basebuttonrepeater>	<datamarker>	<menu>
<basecombobox>	<datapath>	<menuarrow>
<basecomponent>	<datapointer>	<menubar>
<basedatacombobox>	<dataselectionmanager>	<menubutton>
<basedatepicker>	<dataseries>	<menuitem>
<basedatepickerday>	<dataset>	<menuseparator>
<basedatepickerweek>	<datastyle>	<menutrackgroup>
<basefloatinglist>	<datastylelist>	<method>
<basefocusview>	<datatip>	<microphone>
<baseform>	<datepicker>	<modaldialog>
<baseformitem>	<datepickerday>	<multistatebutton>
<basegrid>	<datepickerweek>	<node>
<basegridcolumn>	<debug>	<p>
<baselist>	<DebugObject>	<param>
<baselistitem>	<dragstate>	<params>
<basescrollarrow>	<drawview>	<piechart>
<basescrollbar>	<edittext>	<piechartplotarea>
<basescrollthumb>	<event>	<plainfloatinglist>
<basescrolltrack>	<face>	<plotstyle>
<baseslider>	<floatinglist>	<pointstyle>
<basesliderthumb>	<focusoverlay>	<pre>
<baseslidertrack>	<font>	<radiobutton>
<basestyle>	<form>	<radiogroup>
<basetab>	<frame>	<rectangularchart>
<basetabelement>	<grid>	<regionstyle>
<basetabpane>	<gridcolumn>	<remotecall>
<basetabs>	<gridtext>	<replicator>
<basetabsbar>	<handler>	<resizelayout>
<basetabscontent>	<hbox>	<resizestate>
<basetabslider>	<horitontalaxis>	<resizestatemin>
<basetrackgroup>	<hscrollbar>	<resource>
<basetree>	<html>	<reverselayout>
<basevaluecomponent>	<httpdataprovider>	<richinputtext>
<basewindow>	<httpdatarequest>	<rpc>
<basezoomarea>	<i>	<rtmpconnection>
 	<image>	<rtmstatus>
<button>	<img>	<script>
<camera>	<import>	<scrollbar>
<canvas>	<include>	<security>
<chart>	<inputtext>	<selectionmanager>
<chartbgstyle>	<javarpc>	<sessionrpc>
<chartstyle>	<label>	<settler>
<checkbox>	<labelstyle>	<simpleboundslayout>

<simpleinputtext>	<tabscontent>	<videolibraryicon>
<simplelayout>	<tabslider>	<videolibrarypopup>
<slider>	<Test>	<videoplayer>
<sliderthumb>	<TestCase>	<videoscreen>
<slidertrack>	<TestResult>	<videoslider>
<soap>	<TestSuite>	<videothumbnail>
<splash (swf8)>	<text>	<videotogglebutton>
<splash view (swf8)>	<textformat>	<videoview>
<stableborderlayout>	<textlistitem>	<videovolumebutton>
<state>	<tickstyle>	<view>
<statictext>	<timetext>	<viewslist>
<style>	<tree>	<virtualdrawview>
<submit>	<u>	<vscrollbar>
<swatchview>	<valueline>	<webapprpc>
<switch>	<valuelinestyle>	<window>
<SyncTester>	<valuepoints>	<windowpanel>
<tab>	<valuepointstyle>	<wrappinglayout>
<tabelement>	<valueregion>	<XMLHttpRequest>
<tabpane>	<valueregionstyle>	<xmlrpc>
<tabs>	<vbox>	<zoomarea>
<tabsbar>	<verticalaxis>	

9. Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani a diplomadolgozat létrejöttéhez nyújtott segítségért, szakmai felkészítésemért, és a dolgozat formai helyességéért témavezetőmnek:
Dr. Adamkó Attila egyetemi tanársegédnek.