Grafikus felület létrehozása a ps parancshoz a Mozilla segítségével

Készítsünk grafikus felületet parancssoros feladatainkhoz a Mozillával.

A ***Linux*** egyik nagyon előnyős tulajdonsága, hogy álnevek, parancsfájlok és egyéb trükkök segítségével könnyen hozhatunk létre új parancsokat. Ezek a módszerek valamennyien a parancssori felületre építenek. Vajon mi a helyzet akkor, ha grafikus felületre van szükségünk?

Kevés olyan eljárás létezik, amely nemcsak kifogástalan külsőt, de könnyű használhatóságot is nyújt. Ebben a cikkben egy olyan ígéretes lehetőséggel ismerkedhetünk meg, amely a ***Mozilla*** felületét használja. Egy meglehetősen bonyolult, de gyakori problémán keresztül vizsgáljuk meg a kérdést: hogyan jeleníthetjük meg minél szemléletesebben a `ps` parancs által szolgáltatott hierarchikus információkat. Ehhez szükségünk lesz a ***Mozilla*** egy friss (legalább 1.4-es) példányára.

Számos grafikus eszközkészletet használhatunk ***Linux*** alatt, kezdve az ***Xt***-től a ***Tcl/Tk***-ig. Ezeknek a készleteknek az oktatóanyaga rendszerint egy gomb létrehozásával kezdődik. S ha ez ennyire bevett szokás, próbáljuk ki mi is, majd haladjunk tovább. A Mozillában a grafikus felületek leírására az ***XML*** parancsformátuma használatos. Egy `button.xul` nevű, gombot meghatározó dokumentum a következőképpen fest:

```
<?xml version="1.0"?>
<window
➥xmlns="http://www.mozilla.org/keymaster/
➥gatekeeper/there.is.only.xul">
  <button label="Press Me"/>
</window>
```

A kicsit hosszúnak tűnő karakterlánc, a `http://www.mozilla.org/keymaster/gatekeeper/stb...` azt jelzi a ***Mozilla*** számára, hogy most nem egy közönséges ***HTML***-fájlról van szó, hanem ***XUL***-fájlról. A ***XUL*** az ***XML*** egy grafikus környezetek leírására használatos, ***Mozilla***-specifikus nyelvjárása. Jelenítsük meg a nyomógomb ablakát a következő paranccsal:

```
mozilla -chrome button.xul
```

Ez a példa nagyon egyszerű, nem is érdemes tovább foglalkoznunk vele, bár még egy egyszerű gomb viselkedésével kapcsolatban is sok minden elmondható lenne. A `ps` parancs kimenetének megjelenítése azonban ennél sokkal nagyratörőbb terv, úgyhogy inkább lépjünk tovább.

1. lista Parancssoros keret a ps számára

```
#!/bin/ksh
export COLUMNS=300
ps h -ew -o ´%p,%P,%C,%x,%z,%G,%n,%U,%a´
➥/tmp/psdata
```

A `<button>` eszközkészlet helyett próbáljuk ki a ***Mozilla*** és a ***XUL*** egyik komolyabb fegyverét, a `<tree>` eszközkészletet. Egy kis kódolásra is szükségünk lesz, no és jóval több ***XML***-re. Most a hangsúly inkább a gyors fejlesztésen van, nem cél a tökéletes program megalkotása. A kódolással kezdjük. Indulásképpen a `ps` elvégzi a kezdeti adatok begyűjtését. Az *1. listán* a 777 jogosultsági módú `psdata.ksh` fájlt látjuk. A kimenet minden minket érdeklő mezőt tartalmaz vesszőkkel elválasztva, a fejlécsor nélkül. A `PID` és a `PPID` kötelező részek, a maradék pedig olyan nem kötelező, de hasznos mezőkből áll, mint például a `COMMAND`. Ez minden, amit a hagyományos ***Linux*** szükségessé tesz.

A kód többi része a ***Mozillától*** függ. Az előre lefordított rendszercsomagok legalább két futtatható fájlt tartalmaznak, a `mozilla`-t, és a `regxpcom`-ot. Mi az `xpcshell` bináris állományt is használni fogjuk. Ez az állománya ***Mozillában*** a Perl parancsértelmező ***JavaScript*** megfelelője és nem rendelkezik grafikus támogatással. Az `xpcshell` olykor a fejlesztés jó kiindulópontja lehet, de soha nem nélkülözhetetlen. Ennek megszerzéséhez a ***Mozilla*** egy teljes lefordított változatára van szükségünk. Ellenőrizzük először is a `www.mozilla.org/build` oldalon az eszközlánccal (toolchain) kapcsolatos előfeltételeket. Ezután töltsük le a forrást az FTP vagy távoli CVS segítségével. Inkább egy fő kiadást, mint a naponta változó legfrissebb változatot ajánlom. A kicsomagolás után kövessük a szabványos fordítási lépéseket:

```
cd mozilla
./configure –disable-debug
```

2. lista Az idegen adatok kötegelt betöltése az xpcshell-be

```
const Cc = Components.classes;
const Ci = Components.interfaces;

var klass = {};
var psdata = null; // last results from ps(1).

klass.file = Cc["@mozilla.org/file/local;1"];
klass.process =
      Cc["@mozilla.org/process/util;1"];
klass.stream
     = Cc["@mozilla.org/network/file-input-
     ➥stream;1"];
klass.jsstream
     = Cc["@mozilla.org/scriptableinputstream;1"];

function execute_ps() {
  // freeze until ps(1) is finished.
  var blocking = true, argv = [], result = {};
  var path =
      "/home/nrm/writing/psviewer/psdata.ksh"

  var file
   = klass.file.createInstance(Ci.nsILocalFile);
  var process
   = klass.process.createInstance(Ci.nsIProcess);

  file.initWithPath(path);
  process.init(file);
  process.run(blocking, argv,argv.length,
        ➥result);
}

function read_raw_data() {
  const path = "/tmp/psdata";
  var mode_mask = 0x01, perm_mask = 0;  //open(2)

  var file
   = klass.file.createInstance(Ci.nsILocalFile);
  file.initWithPath(path);

  var stream = klass.stream.createInstance(
      ➥Ci.nsIFileInputStream);

  stream.init(file, mode_mask, perm_mask, 0);

  var jsstream = klass.jsstream.createInstance(
      ➥Ci.nsIScriptableInputStream);

  jsstream.init(stream);

  var data = jsstream.read(file.fileSize);

  // got the file content. break it down.

  data = data.split("\012");

  for (var i = 0; i < data.length; i++)
  {
    data[i] = data[i].replace(/\s*,\s*/,",");
    data[i] = data[i].replace(/^\s*/,"");
    psdata.push(data[i].split(","));
  }
}

execute_ps();
read_raw_data();
```

```
make
make install
```

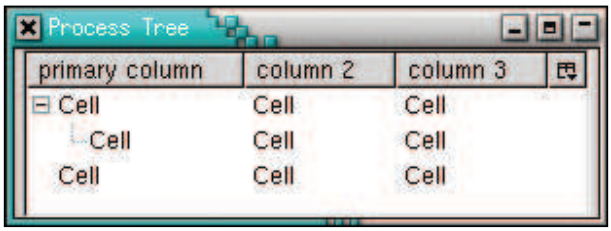
A hibakeresésre használt változatok lassúak és tele vannak hibakereső részletekkel. Igaz ugyan, hogy ezek ártalmatlanok, de most inkább kerüljük őket. Az ilyen kódnak a fordítása is tovább tart egy órával, és akár 1 GB helyet is elfoglalhat.
A kapott bináris állományokat a `mozilla/dist/bin` könyvtárban találjuk majd. Ezeket futtathatjuk ebből a könyvtárból, vagy bármilyen más helyről, amennyiben a `MOZILLA_FIVE_HOME` és `LD_LIBRARY_PATH` változók értéke be van állítva. Most már minden szükséges bináris állomány és héjprogram elérhető.
A *Perl* esetén a `ps` kimenetét valahogy el kell juttatni a programozási környezetbe. Ebben az esetben az eszköz egy *JavaScript* parancsértelmező. Ennek a végrehajtásához nem elég egy nyelvi parancsformátum, az I/O támogatására is szükségünk van. A *Perl*-ben a támogatás a nyelvi függvények szintjén került megvalósításra, ezzel szemben a *JavaScript* nem rendelkezik I/O-függvényekkel.

A *Mozillában* ez az I/O támogatás objektumok használatával érhető el, amelyeket azonban nem szolgáltathat egy könyvtár, mivel a nyelvi alap nem rendelkezik I/O-függvényekkel. Vagyis a *Perl* `use` vagy `require` parancsa ebben az esetben nem fog működni. Olyan áttételes műveleteket sem alkalmazhatunk, mint például az `echo 'pwd'`. Ehelyett a *Mozilla* az *XPCOM*-ot kínálja.
Az *XPCOM* a *Microsoft*-féle *COM* (Component Object Modell, komponens objektum-modell) egy megvalósítása, és használható *Linux*/*UNIX*, *Windows* és *Macintosh* rendszereken egyaránt. Jelenleg a működése egyetlen folyamatra korlátozódik, vagyis nincs lehetőség *DCOM* (Distributed Component Object Modell, elosztott komponens objektum-modell) használatára. Az *XPCOM*/*COM* nyújtja a leggyorsabb megoldást arra, hogy egy parancsnyelvi környezetet új lehetőségekkel bővítsünk. Működése közben egy előre lefordított objektumot (például *C* vagy *C++* objektumot) kapcsol össze a parancsnyelv objektumhivatkozásával.
A legközelebbi *Perl*-megfelelő az *XM*, de míg az *XPCOM* nem igényli a program újrafűzését, az *XM* igen.

3. lista Tiszta XUL kód a <tree> eszköz statikus tartalommal történő megvalósítására

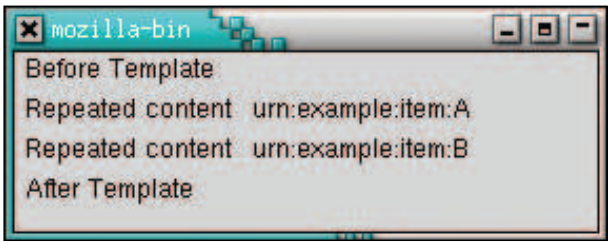
```
<?xml version="1.0"?>
<?xml-stylesheet
  href="chrome://global/skin/global.css"
  type="text/css"?>
<!DOCTYPE window>
<window xmlns="http://www.mozilla.org/keymaster/
            ➥gatekeeper/there.is.only.xul"
            ➥title="Process Tree">

<tree id="t1" flex="1">
  <treecols>
    <treecol flex="1" id="A"
     ➥label="primary column" primary="true"/>
    <treecol flex="1" id="B" label="column 2"/>
    <treecol flex="1" id="B" label="column 3"/>
  </treecols>

  <treechildren id="titems" flex="1">

    <treeitem id="row1" container="true"
                ➥open="true">
      <treerow>
        <treecell label="Cell"/>
        <treecell label="Cell"/>
        <treecell label="Cell"/>
      </treerow>

      <treechildren>
        <treeitem>
          <treerow>
            <treecell label="Cell"/>
            <treecell label="Cell"/>
            <treecell label="Cell"/>
          </treerow>
        </treeitem>
      </treechildren>
    </treeitem>

    <treeitem>
      <treerow>
        <treecell label="Cell"/>
        <treecell label="Cell"/>
        <treecell label="Cell"/>
      </treerow>
    </treeitem>

 </treechildren>
</tree>

</window>
```

Bár a *Mozilla* alapértelmezésben is több ezer *XPCOM* objektumot tartalmaz, az *XPCOM* mégsem egy *Java*-szerű virtuális gép, hanem rendszerint olyan lefordított kód, amely hatékonyságát a hardverközeli futás biztosítja.

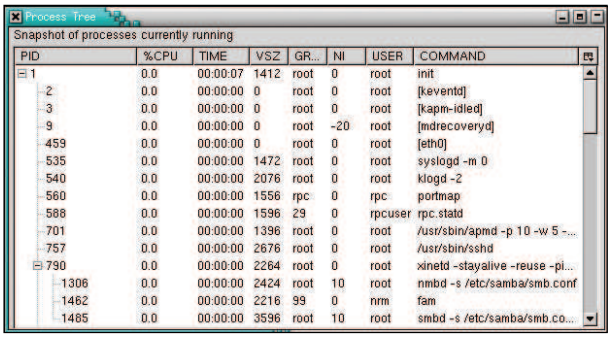


1. kép Egyszerű <tree> eszköz startikus XUL tartalommal

Furcsának tűnhet, hogy *Linuxon* a *Microsoft* elgondolását alkalmazzuk, de az *XPCOM* teljesen nyílt forráskódú és a *UNIX* terén olyan szerepet tölt be, amelyre sokáig nem volt példa: a *Linux*/*UNIX* rendszereken egyszerűen nem létezett ilyen jól használható, közepes méretű komponensmodell. A múltban is volt *CORBA*, és voltak a dinamikus könyvtárak (dll), de az egyik túlságosan nehézsúlyú, a másik túlzottan pehelysúlyú megoldást jelentett. Az *XPCOM* ezzel szemben tökéletesen illeszkedik a közepes méretű feladatokhoz. Rendkívül jól használható azokon a területeken, ahol az alkalmazás viszonylag nagy bináris állományokból áll és a megvalósítás alapvetően teljesítményközpontú.
Az *XPCOM* illetve *COM* használatára a *Windows* `QueryInterface()` metódusának gyakori hívása a jellemző, de most a *Linux*-programozók érzékenysége miatt a cikkben helyette inkább a `createInstance()` és `getService()` metódusokat fogjuk használni. Azért jó tudni, hogy a `QueryInterface()` is rendelkezésre áll.
Térjünk vissza a kódra. Olvassuk be a `ps` kimenetét a *2. listán* látható módon.
A lista első része beállít néhány globális változót.
A `Components` egy már létező objektum, amely valamennyi komponensnek nevezett *XPCOM*-objektum és az általuk támogatott (*Java* vagy *COM* értelemben vett) felületek könyvtára. Egy *XPCOM*-objektum megszerzéséhez keressük meg a megfelelő komponenst (a nevét a `Contract ID` nevű karakterlánc tárolja) és hozzunk létre hozzá egy megjelenési (interface) objektumot (amely szintén egy karakterlánc vagy tulajdonság neve alapján kaphatja a nevét – mi itt az utóbbit használjuk). A komponensek újbóli felhasználása elterjedt gyakorlat, így ha egyszer létrehoztuk, akkor a `klass` objektum egy alkalmas jellemzőjeként kerülnek mentésre – a `class` a *JavaScript* egy fenntartott kulcsszava.
A megadott függvények foglalják el a listában a fennmaradó helyeket. Az `execute_ps()` egyszerűen egy újabb folyamatot futtat: a `ps` keretének parancsfájlját. Ehhez szüksége van egy fájl objektumra (mégpedig egy `nsILocalFile` típusúra) és egy `process` (folyamat) objektumra (`nsIProcess` típus). A `run()` a `fork()` használatával hívja meg a folyamatot. A *Mozillát* úgy fejlesztették, hogy mindezt hordozható formában tegye, de esetünkben most csak a Linux támogatott, mert a futtatható fájl elérési útja konstansként szerepel a kódban. A másik függvény, a `read_raw_data()` az adatokat olvassa be. A *Mozilla* az adatfolyam, átvitel és csatorna elveket használja, vagyis ugyanazokat, amelyeket a *Java* néhány magas szintű szolgáltatása, de mindezt az osztályok létrehozásának bonyodalma nélkül. Nekünk egy fáljobjektumra van szükségünk a `ps`-től jövő adatok tárolá-



2. kép Statikus RDF-tartalomra épülő egyszerű sablonnal támogatott grafikus felület



3. kép A végső <tree> eszköz, amelynek RDF-adatait a JavaScriptből merítettük

sára. A `stream` objektum egy utat nyit meg ennek a tartalomnak a fájl felé. Egy kis trükkre is szükség van: az eredeti `stream` objektumot egy másik különleges `stream`-objektumba kell ágyazni, amelyre parancsok adhatók ki. Egyetlen `read()` hívással a fájl egész tartalmát egy karakterláncban helyezzük el. A következő lépésben néhány ***Perl***-szerű szabályos kifejezés boszorkányos ügyességgel először sorok tömbjeire majd egymásba ágyazott tömbökké alakítja a karakterlánc tartalmát. Minden adatot karakterként kezelünk. Az adatok feldolgozásának helyességét az `xpcshell` által biztosított elemi `print()` paranccsal ellenőrizhetjük. Sajnos a ***Mozilla*** jelenleg nem támogatja a PID azonosítók visszakeresését, így `/tmp/psdata.$$` jellegű fájlokat még nem használhatunk. Ugyanakkor valószínű, hogy ennek a megoldása sem fog sokáig váratni magára.

Ebben a parancsfájlban egy sereg ***XPCOM***-objektumot találhatunk, de vajon hogyan tudjuk a megfelelőket megkeresni? Ahogy bármelyik programozói könyvtár esetén, itt is rendelkezésünkre áll egy referenciaanyag. Keressük meg a ***Mozilla*** forráskódjának .IDL kiterjesztésű fájljait (ezek a `mozilla/dist/idl` könyvtárban helyezkednek el), nézzünk körül a ***Világhálón***, vagy olvassunk el egy könyvet a témában.

Kezdetnek ennyit a parancsfájlokról, a parancsfájlok írása és a táblázatos adatok témája a ***Linux*** amúgy is jól ismert területe. A grafikus felület létrehozásához a ***Mozillának*** az ***XML***-re ezen belül is a ***XUL***-ra van szüksége. Ez a parancssortól nagymértékben eltérő terület és ahhoz, hogy boldoguljunk vele, közelebbről meg kell ismernünk. A folyamatot éppen ezért most könnyen érthető lépésekre osztottam fel. Pillantsunk először is a *3. lista* és az *1. kép* által mutatott ***XUL*** `<tree>` eszközre. A fa megjelenése igen tetszetős, mivel a `<?xml-slylesheet?>` feldolgozási utasítás feltétel nélkül

4. lista Egyszerű tartalomtól függő sablon megvalósítása XUL-kóddal

```
<?xml version="1.0"?>
<?xml-stylesheet
  href="chrome://global/skin/"
  type="text/css"?>
<!DOCTYPE window>
<window xmlns="http://www.mozilla.org/keymaster/
            ➥gatekeeper/there.is.only.xul">
 <description value="Before Template"/>
 <vbox datasources="trivial.rdf"
  ➥ref="urn:example:items"

➥containment="http://www.example.org/
➥TestData#items">

  <template>
   <rule>
    <conditions>
     <content uri="?uri"/>
     <member container="?uri" child="?note"/>
    </conditions>
    <action>
     <hbox uri="?note">
      <description value="Repeated content"/>
      <description value="?note"/>
     </hbox>
    </action>
   </rule>
  </template>
 </vbox>
 <description value="After Template"/>
</window>
```

5. lista A 4. lista sablonjához illő trivial.rdf fájl

```
<?xml version="1.0"?>
<RDF
 xmlns:TD="http://www.example.org/TestData#"
 xmlns="http://www.w3.org/1999/02/ 22-rdf-
      ➥syntax-ns#">

  <Description about="urn:example:root">
    <T:items>
      <Seq about="urn:example:items">
        <li resource="urn:example:item:A"/>
        <li resource="urn:example:item:B"/>
      </Seq>
    </T:items>

  </Description>
</RDF>
```

6. lista A ps-adatok faszerkezet-nézetének végső XUL-kódja

```
<?xml version="1.0"?>
<?xml-stylesheet
 href="chrome://global/skin/global.css"
 type="text/css"?>
<!DOCTYPE window>
<window xmlns="http://www.mozilla.org/keymaster/
        ➥gatekeeper/there.is.only.xul"
        ➥title="Process Tree" flex="1">
 <script src="tree.js"/>

 <vbox flex="1">
  <description>
   Snapshot of processes currently running
  </description>

 <tree id="proc-tree"
  ➥flex="1"
  ➥datasources="rdf:null"
  ➥ref="http://www.example.org/ProcData#ProcList"
  ➥containment="http://www.example.org/
  ➥ProcData#child">

  <treecols>
   <treecol id="pid" primary="true" label="PID"
            ➥minwidth="75"/>
   <splitter class="tree-splitter"/>
   <treecol id="pcpu" label="%CPU"
            ➥minwidth="40"/>
   <splitter class="tree-splitter"/>
   <treecol id="time" label="TIME"
            ➥minwidth="40"/>
   <splitter class="tree-splitter"/>
   <treecol id="vsz" label="VSZ" minwidth="40"/>
   <splitter class="tree-splitter"/>
   <treecol id="group" label="GROUP"
            ➥minwidth="40"/>
   <splitter class="tree-splitter"/>
   <treecol id="nice" label="NI" minwidth="40"/>
   <splitter class="tree-splitter"/>
   <treecol id="user" label="USER"
            ➥minwidth="40"/>
   <splitter class="tree-splitter"/>
   <treecol flex="1" id="args" label="COMMAND"
            ➥minwidth="40"/>
  </treecols>
  <template>
   <treechildren>
    <treeitem open="true" uri="rdf:*">
     <treerow>
      <treecell label=
        ➥"rdf:http://www.example.org/
        ➥ProcData#pid"/>
      <treecell label=
        ➥"rdf:http://www.example.org/
        ➥ProcData#pcpu"/>
      <treecell label=
        ➥"rdf:http://www.example.org/
        ➥ProcData#time"/>
      <treecell label=
        ➥"rdf:http://www.example.org/
        ➥ProcData#vsz"/>
      <treecell label=
        ➥"rdf:http://www.example.org/
        ➥ProcData#group"/>
      <treecell label=
        ➥"rdf:http://www.example.org/
        ➥ProcData#nice"/>
      <treecell label=
        ➥"rdf:http://www.example.org/
        ➥ProcData#user"/>
      <treecell label=
        ➥"rdf:http://www.example.org/
        ➥ProcData#args"/>
     </treerow>
    </treeitem>
   </treechildren>
  </template>
 </tree>
</vbox>

</window>
```

lemásolja az aktuális *Mozilla*-témát. Ha el szeretnénk hagyni az alapvető vezérlőgombokat és egyéb díszeket, a *Mozilla* futtatható fájlját használhatjuk a `-chrome` kapcsolóval:

```
mozilla -chrome static_tree.xul
```

Az *XML*-tartalom (amire ezentúl kódként hivatkozom), egy kicsit a *HTML* `<table>` címkéjére hasonlít: megadtuk mind az oszlopfejlécek mind pedig az adatsorok tartalmát.
A trükk a `<treeitem>` címkében rejlik, amely tartalmazhat egy `<treechildren>` címkét, amely lehetővé teszi, hogy ne csak egylevelű csomópontokat, hanem további faágakat is elhelyezhessünk a fán. Ahogy az 1. képen is látszik, a faeszköz számos interaktív szolgáltatást is kínál: az ágak ugyanúgy nyithatók/zárhatók, ahogy egy *Nautilus* vagy *Windows Explorer* fájlkezelő programban, oszlopokat hozhatunk létre, vagy törölhetünk az oszlopneveket tartalmazó fejléc jobb szélén lévő oszloprendező gombbal.
Ha akarnánk, tulajdonképpen használhatnánk *JavaScript* parancsfájlokat is a `ps` adatainak ebbe a *XUL*-dokumentumba való dinamikus betöltéséhez. Ez nem is lenne nehéz, és az összes *W3C DOM* csatolófelület elérhető a megvalósításhoz. Kezdjük az `Element`-objektumok hozzáadásával, vagy használjuk inkább az `innerHTML` jellemzőt. Ezzel a cikkel azonban szeretném egy kicsit magasabbra tenni a lécet, és inkább egy teljesen adatvezérelt megközelítést mutatok be, amelyben nem kell egyetlen fát sem kézzel létrehoznunk.

7. lista Az adatok parancsfájlból RDF-adatforrásba történő átvitele.

```
 // --- globals ---
klass.datasource
 = Cc["@mozilla.org/rdf/datasource;1" +
      "?name=in-memory-datasource"];
klass.rdf
 = Cc["@mozilla.org/rdf/rdf-service;1"];

var schema = "http://www.example.org/ProcData#";
var props =
  [ "pid", "ppid", "pcpu", "time", "vsz",
    "group", "nice", "user", "args" ];

var rdf = klass.rdf.getService(Ci.nsIRDFService);

var root  = rdf.GetResource(schema + "ProcList");
var child = rdf.GetResource(schema + "child");
var preds = [];

for (var p in props)
  preds[p] = rdf.GetResource(schema + props[p]);

// --- mainline ---

window.addEventListener("load",load_handler,true);

// --- functions ---

function update_tree() {
  var tree = document.getElementById
           ➥("proc-tree");

  // get the in-memory ds, not the
rdf:localstore
  var ds   = tree.database.GetDataSources();
      ds   = ( ds.getNext(), ds.getNext() );
      ds   =
       ➥ds.QueryInterface(Ci.nsIRDFDataSource);

  var sub, pred, obj;

  for (var i=0; i < psdata.length; i++)
  {
    if ( psdata[i][1] == "0" ) // a root node
      sub = root;
    else                       // a child node
      sub = rdf.GetResource(
              ➥schema + "process-" +
              ➥psdata[i][1]);

    pred = child;
    obj = rdf.GetResource(
              ➥schema + "process-" +
              ➥psdata[i][0]);

    ds.Assert(sub, pred, obj, true);

    // add all properties for this process

    sub = obj;
    for (var j=0; j < psdata[i].length; j++)
    {
      pred = preds[j];
      obj = rdf.GetLiteral(psdata[i][j]);
      ds.Assert(sub, pred, obj, true);
    }
  }
}

function load_handler() {
  var tree = document.getElementById("proc-
           ➥tree");
  var ds = klass.datasource.createInstance(
          ➥Ci.nsIRDFInMemoryDataSource);
  tree.database.AddDataSource(ds);

  update_tree();
}
```

A *4. lista* és a *2. kép* egy fa nélküli ***XUL*** grafikus felületet mutat. Ebben egy `<template>` címkét alkalmaztunk a cél eléréséhez.

A ***XUL***-sablon nem a ***C++*** sablonokhoz, hanem inkább egy jelentésmintához hasonlítható, amely az ismétlődő adatcsoportok alapjául szolgál. A sablon egy `<vbox>` címkével kezdődik, amely rendelkezik egy `datasources=` tulajdonsággal. A `<template>` szakasz `<action>` része tartalmazza azokat a soronként ismétlődő adattartalmakat, amelyeket a `<conditions>` rész azonosít a ***trivial.rdf*** fájlban. Ha közepes szintű tudással rendelkezünk a `make` vagy ***SQL*** terén, esetleg van némi ismeretünk a ***Lisp***, ***Scheme*** vagy ***Prolog*** nyelvekkel kapcsolatban, nem okozhat gondot a rendszer működésének megértése. Az *5. listán* látjuk a 2. képhez tartozó ***trivial.rdf*** fájl tartalmát.

Ha módosítjuk ezt a fájlt, a 2. kép akkor is megváltozhat, ha a 4. lista közben nem módosult. Ez tehát egy adatvezérelt megoldás. Ez a fájl ***RDF***-típusú, amely a ***W3C*** szigorúbb szabványai közé tartozik. Az alkalmazott logikai szerkezet lényegében egy csomópontokból álló gráf, amelyben minden csomópont három adategységet hordoz. Az egyes egységek nevei `subject` (téma), `predicate` vagy `property` (állítás, jellemző) és `object` (céltárgy). Az egyszerű gráfok fák, így az 5. lista is egy fát ír le. Ha összekapcsoljuk a 4. lista `<hbox>` címkéjét az 5. listában lévő `<li>` címkékkel, a 2. képen látható eredményt kapjuk. Ez valami olyasmi, mint az ***SQL*** táblakapcsolata vagy a `join` parancs. Egyelőre annyit jegyezzünk meg, hogy a 4. lista `ref=` tulajdonsága az 5. lista `<Seq>` címkéjének felel meg. A kettőt így kapcsolja össze a ***Mozilla*** sablonfeldolgozó logikája. A ***Mozilla*** RDF-

támogatása nem túl szigorú, így majdnem az összes URI és URL helyben feldolgozható, mintha változóról vagy konstansról lenne szó. Ebben a cikkben is végig kihasználtuk ezt a tulajdonságot. Írjunk be az 5. listába még egy `<li>` címkét, indítsuk újra a *Mozilla*-t és jelenítsük meg ismételten az oldalt.

A faszerkezet jó megoldást nyújt a folyamatok hierarchikus listájának megjelenítésére és a `<template>` is megfelelő eszköz a fa megjelenésének adatokból történő vezérlésére. Bár nincs olyan RDF-dokumentumunk, amit használni tudnánk, ehelyett rendelkezünk a rekordok egy *JavaScrtipt*-tömbjével. A megoldás az, hogy összekapcsoljuk a `<tree>` és a `<template>` címkéket, és az RDF-fájlnak az `rdf:null` értéket adjuk, ami azt jelenti, hogy nincs ilyen fájl megadva. Egy parancsfájlt használunk arra, hogy az RDF-tartalmat közvetlenül az adatokból állítsuk elő. Az RDF különleges felépítésének köszönhetően a tartalom minden gond nélkül áttölthető a sablonba, és így a működés egészen egyszerű. Ez egy sokkal tisztább, bár kétségkívül kényesebb megoldás, mint a *XUL*-fa kézzel történő felépítése a *JavaScript*-ből. Az RDF és a sablonok használatának egy másik előnyös tulajdonsága, hogy a fa bármikor egyszerűen frissíthető. Ez azt jelenti, hogy az ablakban dinamikusan megjeleníthetőek a ps(1) parancsból származó adatok, mintha a `watch ps H` egy grafikus megfelelője futna. Ennek bemutatása azonban már meghaladja e cikk kereteit.

A `<tree>` és `<template>` címkék együttes használatával létrehozott XUL-dokumentum eredménye a *3. képen* és a *6. listán* látható. Ebben is megtalálhatjuk a `datasource=` és `ref=` tulajdonságokat, valamint a `<template>` címkét. Az URL-ek az `rdf:` karakterlánccal kezdődnek és azokat a pontokat mutatják, ahol a sablonokba RDF-adatokat kell beillesztenünk. A korábbi példában a változók kérdőjellel kezdődtek, ezek megadására kétféle parancsformátum áll rendelkezésre, és természetesen minden sorhoz és minden oszlophoz egy ilyen adat tartozik.

A `<splitter>` címke egyszerűen egy felhasználóbarát kiegészítés, amely lehetővé teszi az oszlopok átméretezését. Ez javítja az olvashatóságot, ahogy a `minwith=` és `flex=` attribútumok is. A 3. képen látható, ahogy a folyamatok hierarchikusan megjelenített rendszere természetes módon kitölti a faszerkezetet.

A 6. lista felső részén lévő `<script>` címke beemeli a 2. lista teljes kódját egy kis kiegészítéssel. Az ilyen parancsrészek beemelésekor rögtön adódik egy biztonsági probléma mégpedig az, hogy a *Mozilla* eljárásának kell biztosítania a távoli adatok és parancsfájlok biztonságos megjelenítését, éppen úgy, mint a *HTML*-oldalakét. Ez olyan, mint a *Java*-kiszolgáló eredet-szabálya. Az `xpcshell` nélkülöz minden biztonsági beállítást, a *Mozilla* bináris állománya azonban rendelkezik a szokásos biztonsággal. A beállítások alapos átdolgozásával felülkerekedhetünk ezeken a megszorításokon, de egyszerűbb a parancsfájlt egy csomagként regisztrálni. Ehhez minden fájlt át kell helyeznünk a Mozilla telepítőjének `chrome` könyvtárába, ahol ezek a biztonsági korlátok feloldódnak. Ennek módjáról rövidesen szó lesz, de először befejezzük a programunkat egy olyan parancsfájllal, ami a `ps` adatait az egyszerű

8. lista A psviewer csomag chrome-regisztrációjához szükséges információk.

```
<?xml version="1.0"?>
<RDF xmlns=
"http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:chrome="http://www.mozilla.org/rdf/chrome#
">

 <Seq about="urn:mozilla:package:root">
  <li resource="urn:mozilla:package:psviewer"/>
 </Seq>

 <Description
➥about="urn:mozilla:package:psviewer"
        ➥chrome:displayName="PSViewer"
        ➥chrome:author="Nigel McFarlane"
        ➥chrome:name="psviewer">
 </Description>
</RDF>
```

JavaScript adatszerkezetből egy RDF-adatforrásba mozgatja át. Ez a parancsfájl fogja helyettesíteni a korábban használt statikus RDF-fájlt (lásd a *7. listát*).
A 7. listán azt a parancsfájlt láthatjuk, amely a statikus RDF-fájlt fogja helyettesíteni. A faszerkezet sablonja által használt RDF-fájlhoz egy többlépcsős folyamat során tudunk *JavaScript* adatokat adni. A *Mozilla* egy adatforrásnak (datasource) nevezett objektumba olvassa be az RDF-adatokat. Mivel korábban az `rdf:null` meghatározást használtuk, nem létezik adatforrás objektumunk, ezért létre kell hoznunk egyet és hozzákapcsolnunk a sablonhoz. Ezt a dokumentum biztonságos betöltése után a `load_handler()` eljárás végzi el. A betöltés közben működő kezelőeljárás egy megszokott *HTML*-trükk, ami ugyanilyen jól alkalmazható a *XUL* esetében is. Ezt az adatforrást ezután az `update_tree()` függvény tölti fel az RDF-tartalommal a sablon számára. Ez igen egyszerűen történik, két egymásba ágyazott ciklus lépked végig a *JavaScript*-tömb adatain. A `ps(1)` minden egyes folyamatánál meghívódik az `Assert()` függvény, amely létrehoz egy RDF adatcsomópontot (három elemből álló adathármast), amely meghatározza, hogy a `PPID X` rendelkezik egy `PID Y` gyerekkel és további RDF-csomópontokat, amelyek szerint a `PID X`-hez a `USER` A tartozik vagy a `GROUP B`. A `<template>` és a `<tree>` címkék közösen dolgoznak azon, hogy ezek az RDF-csomópontok önműködően faszerkezetbe rendeződjenek, éppen úgy, ahogy a `make` értékeli ki egy adott `Makefile`-ban lévő összes fájl függőségi-fáját. Ezzel a statikus RDF-fájl helyére lépő parancsfájllal el is készültünk az egyszerű folyamatfigyelő programunkkal. Végül az alábbi lépéseket kell tennünk a biztonsági intézkedések feloldásához szükséges kódregisztráció érdekében:

```
M5H = $MOZILLA_FIVE_HOME
mkdir -p $M5H/chrome/psviewer/content
cp *   $M5H/chrome/psviewer/content
vi     $M5H/chrome/psviewer/content/contents.rdf
vi     $M5H/chrome/installed-chrome.txt
```

A `vi` első futtatásával létrehozzuk a `contents.rdf` fájlt. Ennek pontosan a *8. listán* látható kódot kell tartalmaznia. A következő lépésben hozzáadjuk ehhez a fájlhoz az ***installed-chrome.txt*** fájl tartalmát, amely egyetlen sor:

```
content,install,url,resource:/chrome/psviewer/
➥content/
```

A *Mozilla* az elindításkor megvizsgálja ez utóbbi fájlt, és ha változást észlel benne, akkor a felsorolt könyvtárakban megkeresi a ***contents.rdf*** fájlokat. Ezeket beolvassa, és a `make`-hez hasonlóan megjegyzi az összes létező csomagot. Minden ismert csomag teljes biztonsági hozzáféréssel rendelkezik, a 8. lista pedig hozzáadja ezekhez a `psviewer` csomagot is. A biztonságos fájlok most már egyetlen URL segítségével megjeleníthetők és futtathatók:

```
mozilla -chrome
chrome://psviewer/content/tree.xul
```

ahelyett, hogy azt kellene írnunk, hogy:

```
mozilla -chrome
file:///home/nrm/psviewer/tree.xul
```

A `pswiewer` a *Mozilla* telepítőjén belül első osztályú helyet foglal el. Ha szükséges, más programokkal is összeépíthetjük, így a *Firefox*/*Firebird* böngészővel vagy a *Thunderbird* levelezőügyféllel. Hozzáadhatjuk a Tools (Eszközök) menühöz is új menüpontként.
Bár nagyon sok módszertani lehetőséget érintettünk ebben a cikkben, nem szabad elkövetnünk azt a hibát, hogy rögtön mindet alkalmazni akarjuk. Mivel az *XML* használata a *Mozillában* nem túl jól dokumentált dolog, könnyen belebonyolódhatunk. Jobban jár az, aki valamilyen egyszerű feladattal kezdi, és csak fokozatosan közelít az itt bemutatott érdekes megoldások felé. Habár a `ps` kimenete dinamikus *HTML*-oldalként is megjeleníthető lenne, a *XUL* végső soron nagyobb teljesítményű és szebb, a munkaasztalba teljesen beilleszthető grafikus megjelenítést tesz lehetővé.
A *Mozilla* egy még felfedezésre váró hatékony grafikus környezet. Komoly az esélye, hogy *Linux* alatt olyan szerepet töltsön be, mint amilyet a *Visual Basic* a *Windows* alatt. Mi több, a *Mozilla* hordozható, operációs rendszertől független megoldásokat tesz lehetővé. Az elkészült alkalmazásunk működhet *BSD*, *HP-UX*, *SunOS*, *AIX* és *Mac OS X* operációs rendszereken is a *Linux* mellett.

Linux Journal 2004. július, 123. szám

Nigel McFarlane kiterjedt programozói tapasztalattal rendelkező, a tudomány és technika területén tevékenykedő szabadúszó író. Legfrissebb könyve a Rapid Application Developement with Mozilla (Gyors alkalmazásfejlesztés a Mozilla segítségével), ISBN 0131423436.
Az nrm@kingtide.com.au címen léphetünk vele kapcsolatba.