Apache Ant – Java-based build tool

Viczián István (viczus@freemail.hu)

Bevezetés

Bármilyen jellegű és bonyolultságú szoftver fejlesztésekor a programozó gyakran találkozik olyan műveletsorokkal, melyeket gyakran kell ismételni és részletesen paraméterezni. Ilyen lehet például egy fordítás, disztribúció készítése, telepítés, tesztelés. Egyszerűbb esetekben erre operációs rendszerhez kötődő scripteket használ, bonyolultabb esetben a de facto szabvány make eszközt.
Java nyelvű fejlesztés során adott a programozási nyelv platformfüggetlensége, amit viszont megkötne az előbbi eszközök operációs rendszerhez való szorosabb kötődése, illetve ezen eszközök konfigurációs fájljai sem a legbarátságosabbak.
Ezek a Java környezetben is használhatóak, de szerencsére létezik egy kvázi szabvánnyá vált egy nyílt forrású eszköz, az Apache Ant (http://ant.apache.org). Írói kitűnően ötvözték a Java nyelv platformfüggetlenségét, az XML gép és ember által is könnyű olvashatóságát, illetve a kibővíthetőséget. Így egy kiváló, tiszta, XML konfigurációs által vezérelhető, Java osztályokkal bővíthető build tool jött létre, melyet egy Java programozó sem nélkülözhet az eszköztárából. Ez a cikk az Ant alapjait mutatja meg, példákkal illusztrálva, javaslatokat téve az optimálisabb felhasználáshoz. Terjedelmi okok miatt a részletes bemutatás nem lehetséges, de a cikk tartalmaz utalásokat az Ant legtöbb képességére, az Olvasó dolga a konkrét megvalósításnak utánajárni.

Az Antről

A cikk írásakor a legutolsó verzió az 1.6.1-es verziószámot viseli, melyet 2004. február 12-én adtak ki. Jelentős különbség van az 1.5.4-es, illetve 1.6.0-ás verziók között. Az egyik legnagyobb változás, hogy az 1.6.0-ás Ant már csak a Java 1.2-vel kompatibilis, míg az előző támogatta az 1.1-es verziót is. Persze ez nem azt jelenti, hogy nem fordíthatjuk saját osztályainkat 1.1-el, de az Ant már csak az 1.2 felett fut. Főbb újdonságok továbbá: az XML névterek támogatása, mely megakadályozza a különböző saját fejlesztésű taszkok névütközését; új osztálybetöltő; adapterek; antlib, mely újrafelhasználható saját taszkok összegyűjtésére alkalmas. Ezeken kívül jelentős mennyiségű új funkció, taszk és hibajavítás is belekerült, melyekről bővebben a WHATSNEW nevű dokumentum tájékoztat, a változásokat egészen az 1.1-es verziótól nyomon követhetjük. A fejlesztők maximálisan ügyeltek arra, hogy az előző verzióval lehetőleg a legnagyobb mértékben kompatibilis maradjon, ne kelljen új build folyamatot, új XML fájlt írni. Ezért a verziószáma sem 2.0, mely jelzi, hogy nem egy teljesen új verzióról van szó. Az 1.6-os nem valószínűleg nem is az utolsó ebből a szériából, de már készül a 2.0-ás (CVS-ben Mutant és Myrmidion modul), mely az Ant teljes átírása, hogy gyorsabb, tisztább legyen. A fejlesztők idejét egyelőre azonban az 1.6.x sorozat fejlesztése köti le.
Kezdetben csak egy eszköz volt, mely a Tomcat-et build-elte platformfüggetlen módon, aztán átkerült a Jakarta projekt alá, majd felkerült az Apache top level projektjei közé. Rengeteg nyílt forráskódú projekt build folyamata épül rá, és szinte minden Java fejlesztőeszköz támogatja (pl. JBuilder, JEdit, JDeveloper, VisualAge, WebSphere Studio, Eclipse, NetBeans, IDEA stb.).
Az Antet letölthetjük a hivatalos honlapról, a http://ant.apache.org/bindownload.cgi címről. Letölthető bináris formában (zip, tar.gz, tar.bz2), illetve ha szükségünk van rá, forráskóddal együtt is. Ha valamilyen okból a legújabb verziót szeretnénk használni (pl. fejlesztés alatt lévő tulajdonság, vagy esetleges hiba), akkor letölthetjük a Nightly Build-et is, mely a legfrissebb változásokat tartalmazó még nem kellően tesztelt változat.

Az Ant az Apache License 2.0 hatálya alá esik, azaz tulajdonképpen szabadon felhasználható, csak annyit követelnek meg tőlünk, hogy továbbterjesztéskor csatoljuk a licence-t az alkalmazásunk dokumentációjához.
Az Ant megismeréséhez a hivatalos honlapon jelentős mennyiségű dokumentáció áll rendelkezésünkre, egyrészt egy nagyon részletes dokumentáció, FAQ (gyakran feltett kérdések és válaszok), Wiki (dinamikusan fejlődő tudástár), illetve könyvajánlások, cikkek és prezentációk. Ha segítségre van szükségünk, nyugodtan fordulhatunk a levelezési listákhoz, illetve azok archívumához, valamint a JavaDoc alapú API dokumentációt és a forrást is böngészhetjük.

Alapelemek

Az Antet egy XML fájlal kell konfigurálni, mely egy projektet tartalmaz, illetve hozzá tartozóan legalább egy targetet. A targetek közül egyet ki kell jelölni alapértelmezettként. A target tulajdonképpen több tevékenységet (taszkot) összefoglaló egység. A taszkok a legkisebb egységek, ezek végzik a tulajdonképpeni műveleteket. A taszkok lehetnek beépített, opcionális, illetve akár saját fejlesztésű taszkok is. A projektek, targetek és taszkok mind XML tag-ekként (`project`, `target`, `task`) jelennek meg.

```
<!-- Projekt nevű projekt, melynek az alapértelmezett target-je a dist, és az alap könyvtára
     az aktuális könyvtár. -->
<project name="projekt" default="dist" basedir=".">
    <description>
        Példa XML fájl
    </description>

  <!-- Beállít három property-t, melyek neve src, build, dist, értéke src, build, dist. -->
  <property name="src" location="src"/>
  <property name="build" location="build"/>
  <property name="dist"  location="dist"/>

  <!-- Init nevű target, mely létrehoz egy könyvtárat. -->
  <target name="init" description=”Inicializáció”>
     <!-- Létrehoz a build property-ben megadott névvel egy könyvtárat – mkdir taszk. -->
    <mkdir dir="${build}"/>
  </target>

  <!--Compile nevű target, mely előtt le kell futtatni az init nevű target-et. -->
  <target name="compile" depends="init"
        description="Fordítás " >
    <!--Lefordítja az src property-ben megadott könyvtárban lévő fájlokat, és a class fájlokat
       a build property-ben megadott helyre teszi – javac taszk.-->
    <javac srcdir="${src}" destdir="${build}"/>
  </target>

  <!-- Dist nevű taszk, mely egy jar fájlt készít, és előtte le kell futtatni a compile
target-
       et. -->
  <target name="dist" depends="compile"
        description="JAR" >
     <!-- Létrehoz a build property-ben megadott névvel egy könyvtárat – mkdir taszk. -->
    <mkdir dir="${dist}"/>

    <!-- Létrehoz egy JAR fájlt a dist property-ben megadott könyvtárba, melybe a build
         property-ben megadott könyvtár tartalmát tömöríti be – jar taszk. -->
    <jar jarfile="${dist}/${ant.project.name}.jar" basedir="${build}"/>
  </target>

  <!-- Könyvtárak törlése target. -->
  <target name="clean" description="Törlés" >
     <!--Letörli a build és dist property-ben tárolt könyvtárakat. -->
    <delete dir="${build}"/>
    <delete dir="${dist}"/>
  </target>
</project>
```

1. példa Példa XML fájl

Project

A projektnek három tulajdonsága (XML tag tulajdonság) lehet, a neve (`name`), az alapértelmezett target neve (`default`) és a projekt alap könyvtára (`basedir`). Illetve lehet egy beágyazott `description` tagje, melyben szöveges leírást lehet megadni.

Target

Target lehet például a fordítás, ideiglenes fájlok törlése, fájlok előkészítése, deploy, disztribúció készítése, JavaDoc generálása, használati utasítás kiírása. Egy target függhet másik target-ektől is (pl. a disztribúció készítése előtt fordítani kell), ezek neveit az adott target `depends` tulajdonságának értékeiként kell felsorolni, figyelembe véve a sorrendet. A target lefutását feltételhez is köthetjük, pontosabban egy property-hez. A property tulajdonképpen egy változó, mely vagy üres lehet, vagy szöveges értéket tárolhat. Ha azt akarjuk, hogy egy target csak akkor fusson le, ha egy property be van állítva, használjuk az `if` tulajdonságot, ha azt, hogy csak akkor fusson le, ha a property nincs beállítva, használjuk az `unless` tulajdonságot. Itt is megadhatunk szöveges leírást, de most már egy `description` tulajdonságként. Lehetőleg csak betűkből és számjegyekből álló nevet adjunk meg, kerüljük a szóköz és pont karaktereket. Ha egy target nevét kötőjellel kezdjük, akkor nem hívhatjuk meg a parancssorból.

Taszk

Egy taszk is egy XML tag-ként jelenik meg, és felparaméterezni a tag tulajdonságaival lehet. Minden taszkhoz lehet egyedi azonosítót rendelni, és scriptekből ezzel hivatkozni rá. Érdekes tulajdonság, hogy az Ant taszkokat nem szükségképpen csak Antből lehet használni, hanem felhasználhatjuk őket saját programjainkban is, hiszen gyakran használatos funkciókat valósítanak meg, melyekre nekünk is szükségünk lehet.

Property

A property-k név és érték párok, ahol a név nagy- és kisbetű érzékeny. Értéket az XML fájlon belül és kívül is lehet adni. Hivatkozni egy property-re a következőképpen lehet: `${property_neve}`. Az Ant hozzáférést biztosít a rendszerjellemzőkhöz is (system properties) ezen property-ken keresztül, illetve van néhány beépített is (alap könyvtár, az XML fájl helye, Ant verziószáma, projekt neve, Antet futtató Java verziószáma).

Token filters

Az Ant lehetőséget biztosít arra, hogy fájlokban bizonyos szövegeket másokra cseréljünk le. Ez jól használható abban az esetben, ha egy közös fájlban szeretnénk eltárolni a különböző paramétereket, és csak telepítéskor behelyettesíteni az alkalmazás konfigurációs fájljaiba. A kicserélni kívánt szöveget token-nek nevezzük, és `@token_nev@` formátumban kell a fájlban szerepeltetnünk. A név és az érték megfeleltetést a filter taszkban kell megadnunk, és a másolásnál szerepeltetni kell a `filtering="on"` tulajdonságot. Bináris fájlok másolásánál nem alkalmazható.

Path-típusú struktúrák

Abban az esetben, ha path-típusú struktúrákat kell megadnunk, beágyazott tag-eket kell alkalmaznunk. A fő tag lehet pl. a `path` vagy `classpath`, és az alá írhatunk újabb tag-eket, pl. `pathelement`, `dirset`, `fileset` és `filelist`.

A `pathelement` tag szerepelhet `path` tulajdonsággal, ezt akkor érdemes használni, ha az értéket egy property-ből kívánjuk beállítani, illetve szerepelhet a `location` tulajdonsággal, aminek értékeként egy, az alap könyvtárhoz relatív, vagy abszolút path-t kell megadni.
A `dirset` könyvtárak, a `fileset` fájlok csoportja. Mindkettőnek lehet `patternset` altag-je, mellyel azt lehet definiálni, hogy az adott könyvtáron belül szűrünk-e bizonyos könyvtárakra vagy fájlokra. Ez tartalmaz `include` és `exclude` altag-eket, ahol az előbbi a nem szűrt, az utóbbi a szűrt könyvtárakat vagy csoportokat adja meg. Ezeket az altag-eket közvetlenül a `dirset` és `fileset` tag-ek alatt is lehet használni. Ezen egyszerű szűrési lehetőségeken felül lehetőség van úgynevezett szelektorok használatára, melyek különböző tulajdonságok alapján választják ki a könyvtárakat és fájlokat (pl. bizonyos karaktereket tartalmaz, dátum alapján, más fájllal való egyezősége, méret alapján, regexp, típus alapján, módosítás alapján stb.). Szelektorokat magunk is fejleszthetünk.
A `filelist` abban különbözik a `fileset`-től, hogy míg a `fileset` olyan fájlokat ad vissza, amik szerepelnek is a fájlrendszerben, a `filelist`-tel olyanokat is lehet definiálni, melyeknek szerepelniük kellene, de nem szerepelnek.
Abban az esetben, ha egy ilyen struktúrát több helyen is használni akarunk, akkor adjunk meg neki egy `id` tulajdonságot, és így máshonnan hivatkozhatunk rá. Megjegyzendő, hogy ez az összes olyan taszknál használható, amely beágyazott tag-eket is használ.
A következő példa a path-típusú struktúrákra mutat példát. Wildcard karakterek (csillag és kérdőjel) használata lehetséges.

```
<!--Path-típusú struktúra, azonosítóval. -->
<path id=”base”>
  <!—Property-vel meghatározott elem. -->
  <pathelement path="${classpath}"/>
  <!—Relatív elérési úttal meghatározott elem. -->
  <pathelement location="classes"/>
</path>
<!--Path-típusú struktúra. -->
<path>
  <!-- Hivatkozott elem -->
  <path refid="base"/>
  <!-- Könyvtárak -->
  <dirset dir="${build.dir}">
    <patternset>
      <include name="apps/**/classes"/>
      <exclude name="apps/**/*Test*"/>
    </patternset>
  </dirset>
  <!-- Fájlok -->
  <fileset dir="lib">
    <include name="**/*.jar"/>
  </fileset>
  <!-- Fájlok -->
  <fileset dir="lib">
    <include name="**/*.jar"/>
  </fileset>
  <!-- Fájlok -->
  <filelist
    id="docfiles"
    dir="${doc.src}"
    files="foo.xml,bar.xml"/>
</path>
```

2. példa Path-típusú struktúrák

Parancssori paraméterek

Parancssori paraméterek esetén az `arg` tag-et kell használni, ahol a `value` tulajdonság egy paramétert ad át, attól függetlenül, hogy van-e benne szóköz karakter. Ha a `line` tulajdonságot használjuk, akkor a szöveget a szóközöknél tagolja, és annyi paramétert ad át a taszk, ahány elem a tulajdonságban szerepel. A perjelek tetszőlegesen használhatók, az Ant automatikusan átfordítja az operációs rendszernek megfelelőre.

Az Ant futtatása

Az Ant a legtöbb elterjedt operációs rendszeren használható, úgymint Linux, üzleti Unix megvalósítások, mint Solaris és HP-UX, Windows 9x és NT, OS/2 Warp, Novell Netware 6 és MacOS X. Futtatásához és fordításához szükség van egy JDK 1.2-re, illetve egy XML feldolgozóra. A disztribúcióban az Apache Xerces2 szerepel, de ez tetszőlegesen kicserélhető. Installáláshoz elegendő kitömöríteni a letöltött tömörített fájlt; a `bin` könyvtárat elhelyezni a `PATH` környezeti változóban; beállítani az `ANT_HOME` környezeti változót, melynek arra a könyvtárra kell mutatnia, ahová az Antet telepítettük; majd beállítani a `JAVA_HOME` környezeti változót, ez pedig arra a könyvtárra mutasson, ahol a JDK található.
Az Antet futtatni egyszerűen az `ant` paranccsal kell. Ekkor az aktuális könyvtárban lévő `build.xml`-t használja, és futtatja az alapértelmezett target-jét. Az XML fájl nevét a `buildfile` kapcsolóval is meg lehet adni. Property-ket átadni a `–Dproperty_neve=ertek` kapcsolóval lehet. A parancs után meg lehet adni egy vagy több target nevet, melyeket le kell futtatni. Bizonyos Ant taszkokat rendszerjellemzőkön át lehet beállítani. Mivel az Ant is Java nyelven készült, közvetlenül a `java` paranccsal is futtatni lehet.

FilterChain és FilterReader

A rövid UNIX parancsok és pipe nagyszerű használhatóságát próbálják az Anten belül a `FilterChain` és `FilterReader` tag-ek biztosítani. Tulajdonképpen a `copy` taszknak lehet megadni egy `FilterChain`-t, mely `FilterReader`-ek egymásutánja. A `FilterReader`-ek a megadott sorrendben feldolgozzák a fájlt, és különböző beépített (pl. UNICODE escape-elés, a fájl első x sorának kiszűrése, grep, reguláris kifejezés, soronkénti prefix, csere, Java megjegyzések szűrése, sortörések szűrése, tab karakterek szóközzé konvertálása, tail, meghatározott karakterek törlése, összefűzés, tokenizálás) transzformációkat végeznek el rajta, de mi is fejleszthetünk ilyen `FilterReader`-t.

InputHandler

Az Ant a target futtatása közben maga is adatokat kérhet be a felhasználótól. Ez nem egyszerű konzolos bekérés, hiszen ekkor nem lehetne különböző IDE-kbe beilleszteni, hanem lehetőség van saját `InputHandler` megírására is. Az alapértelmezett `InputHandler` persze konzolról kéri be a felhasználótól az adatot, melyet Enter billenyűvel kell lezárnia.

Listener-ek és loggerek

Az Ant futásának figyelésére két módszer is lehetséges, egyrészt listener-ekkel, melyek a bizonyos esetekben (build indítása, build befejezése, target indítása, target befejezése, taszk indítása, taszk befejezése, üzenet loggolása) kerülnek meghívásra, másrészt logger-ekkel, melyek képesek elkapni a konzolra írt üzeneteket, és szűrni, formázni, továbbítani azokat. Persze saját logger elkészítésére is lehetőségünk van.

Adapterek

Az adapterek Java osztályok, melyek olyan Java osztályokat alakítanak (pontosabban hívnak meg) Ant taszkká vagy típussá, amelyek nem terjesztik ki a megszabott interfészeket. Ezek tulajdonképpen olyan meta-taszkok, melyekkel dinamikusan lehet új taszkokat létrehozni.

Alap, opcionális és 3rd party taszkok

Az Ant ereje legfőképpen a szinte megszámlálhatatlan alap és opcionális taszkokban rejlik, illetve abban, hogy ilyeneket magunk is fejleszthetünk. Az előbbiek közül sorolok fel néhányat, a teljesség igénye nélkül.
Tömörítő taszkok alkalmasak Gzip, BZip, Cab, Zip (akár Jar, War, Ear), Rpm típusú fájlok előállítására és kitömörítésére, valamint Jar fájl aláírására.
Az Audit/Coverage taszkok alkalmasak a forrás fájlok felderítésére, analizálására, különböző mérőszámok kiszámolására.
A fordítást segítő taszkok alkalmasak Java forrás fájlok lefordítására, függőségek vizsgálatára, JSP fordításra, RMI csonkok generálására.
Egy deploy taszk segíti a J2EE környezetben szükséges alkalmazásszerverre történő telepítést.
Egy dokumentációs taszk a JavaDoc generálására alkalmas.
Külön EJB taszkok vannak, melyek különböző alkalmazásszerverek specifikus telepítésleíró fájljait képesek generálni. A következő alkalmazásszerverek támogatottak: Borland Application Server 4.5, iPlanet Application Server 6.0, JBoss 2.1 és későbbi verziók, Weblogic 4.5.1-től egészen az 7.0 EJB szerverekig, JOnAS 2.4.x és 2.5 nyílt forrású EJB szerver, valamint az IBM WebSphere 4.0.
A futtató taszkok alkalmasak Ant futtatására, operációs rendszertől függő rendszerparancs hívására, JVM futtatására mind egymás után, mind párhuzamosan, ez utóbbi esetekben lehetőség van szinkronizálásra.
Külön taszkok vannak a fájlokon végzett műveletekre: attribútum állítása, checksum generálása és ellenőrzése, tulajdonos és jogosultságok megváltoztatása, másolása, törlése, fájl vége jelek konvertálása, mozgatás, könyvtár létrehozása, touch és patch parancsoknak megfelelő műveletek futtatása.
Ezen kívül lehetőség van loggoló beállítására, levél küldésére, üzenet kiírására, kulcs generálásra, hangfájl lejátszására, felhasználói interakcióra, SQL parancsok futtatására JDBC-n keresztül, XML validálásra.
Számos preprocesszor taszk is rendelkezésünkre áll: pl. ANTLR, JavaCC, Javah, JJDoc, JJTree, XSLT transzformáció, valamint operációs rendszertől függő karakterek kicserélése escape-elt UNICODE karakterekre.
Lehetőség van buildnumber kezelésére, mely egy build-elésenként automatikusan növelt szám, property-k fájlból való betöltésére (properties vagy XML fájl).
Lehetőség van különböző hálózati protokollok és eszközök használatára: ftp, rexec, scp, ssh és telnet.
Támogat több változáskezelő rendszert is, úgymint CVS, Continuus/Synergy, Microsoft Visual SourceSafe, Perforce, Pvcs, SourceOffSite, StarTeam.
JUnit-tal való tesztelést is biztosítja.
Az Ant elterjedésével egyre több ingyenes, illetve kereskedelmi terméket illesztettek, valósítottak meg Ant taszként is. Ezek a dokumentációban az „External Tools and Task„ részben megtalálhatóak. Néhány elterjedtebb: AspectJ, Anakia, Checkstyle, Cocoon, Doxygen, J2ME, Jalopy, Java2Html, JMX, JNI, PMD, XDoclet stb.

Taszk fejlesztése

Abban az esetben, ha a rendelkezésre álló taszkok közül nem találunk megfelelőt, vagy saját eszközünket szeretnénk Ant taszként használni, akkor lehetőség van saját fejlesztésére. Ebben az esetben az Ant saját API-ját kell felhasználnunk. Egy egyszerű osztályt kell írnunk, mely az `org.apache.tools.ant.Task` osztályt vagy egy, speciális leszármazottját terjeszti ki. A saját taszkunk is egy XML tag-ként fog megjelenni, melynek ugyanúgy tulajdonságokat lehet megadni. Ezen tulajdonságokhoz kell „setter” metódusokat deklarálni, melyet az Ant

automatikusan felderít és meghív. Abban az esetben, ha a taszkunk egy másik taszkot is magába foglalhat, akkor implementálnia kell az `org.apache.tools.ant.TaskContainer` interfészt. Abban az esetben, ha a taszkunk belső XML tageket is tartalmazhat, akkor definiálnunk kell hozzájuk `create`, `add` vagy `addConfigured` metódusokat. Az `execute` metódusban kell implementálnunk a taszk működését. A következő példa egy olyan taszk forrását mutatja, mely egy üzenetet ír ki, ahol az üzenet egy tulajdonságként adható meg. A példa eredetileg az Ant dokumentációjában szerepel.

```
package com.mydomain;

import org.apache.tools.ant.BuildException;
import org.apache.tools.ant.Task;

public class MyVeryOwnTask extends Task {
    private String msg;

    // The method executing the task
    public void execute() throws BuildException {
        System.out.println(msg);
    }

    // The setter for the "message" attribute
    public void setMessage(String msg) {
        this.msg = msg;
    }
}
```

Az XML fájlban a taszkunkat előbb definiálni kell, majd használni:

```
<taskdef name="mytask"
        classname="com.mydomain.MyVeryOwnTask"
        classpath="build"/>
<mytask message="Hello World! MyVeryOwnTask works!"/>
```

Scriptelés

Az Ant egy különleges lehetősége a scriptelés, mely a BSF-en (Bean Scripting Framework) alapszik. A BSF Java osztályok gyűjteménye, melyek lehetővé teszik Java objektumok és metódusok elérését különböző script nyelvekből. Az Ant a JavaScript-et támogatja, melyhez a Rhino ECMAScript-et használja a Mozilla projektből. A JavaScript-et egy `CDATA`-ként kell megadni, és az összes Ant elem elérhető (pl. project, target, task stb.) a nevével vagy egyedi azonosítójával (ID) hivatkozva.

Konklúzió

E cikk terjedelmi okok miatt csak egy rövid bepillantást nyújtott az Ant rejtelmeibe, felvillantva a lehetőségeket, hogy mit is nyújthat egy fejlesztés kapcsán ez a kvázi szabvánnyá vált eszköz. Tömörsége miatt lehetséges, hogy több hasznos szolgáltatás, illetve a taszkok pontos leírása is kimaradt. Ennek ellenére ezek megismerése nem komoly feladat, hiszen az Ant elterjedése miatt rengeteg helyen lehet vele és róla szóló leírásokkal találkozni, és a hivatalos honlapon, illetve a letöltött disztribúcióban is jelentős mennyiségű dokumentáció lelhető fel.

A cikk szerzője külön köszönettel tartozik Tóth Ferencnek (http://www.etus.hu/) a cikk megírásában nyújtott segítségéért.

Hivatkozások

The Apache Software Foundation http://www.apache.org/
The Apache Ant Project http://ant.apache.org/

A szerzőről

Viczián István a Debreceni Egyetem programtervező matematikus szakán végzett 2001 nyarán, most ugyanott levelező PhD hallgató az elosztott rendszerek, middleware-ek témakörében. Jelenleg vezető fejlesztőként dolgozik Budapesten, melynek keretében csoportmunkát segítő eszközökkel, Java nyelvvel, webes technológiákkal, middleware szoftverekkel és alkalmazásintegrációval foglalkozik. Szabadidejében optimalizációs alkalmazást fejleszt, új Java technológiákkal ismerkedik, valamint Java blog-ot ír (JTechLog).
E-mail: viczus@freemail.hu
Honlap: http://dragon.unideb.hu/~vicziani