PHP nyomkövetők (4. rész)

Sorozatunk előző részeiben áttekintést adtunk néhány PHP nyomkövetőről, a Gubed/Quanta és a Nusphere::PhpED programokról. Ez alkalommal az Xdebug-ra épülő ActiveState::Komodo és a Xored::TruStudio fejlesztőkörnyezeteket vesszük szemügyre, valamint készítünk egy Firefox-bővítményt a nyomkövetés kényelmesebbé tételéhez.

A DBGp

Maga az ***Xdebug a DBGp*** nyomkövetési protokollra épül (***Common DeBuGger Protocol***). Ennek dokumentációja így határozza meg önmagát: ***„A common debugger protocol for languages and debugger UI communication"***, azaz ***„közös nyomkövetési protokoll nyelvek és nyomkövető felhasználói felület közti kommunikációra"***.
A szerzők: ***Shane Caraveo (shanec@ActiveState.com)*** és ***Derick Rethans (derick@derickrethans.nl)***, aki a 2005-ös PHP Konferencián Magyarországon is járt. Az ***Xdebug*** honlapja mintaszerűen egyszerű és igényes ➲ *www.xdebug.org*.
2006 júniusában már régóta az xdebug-2.0.0beta5 a legfrissebb verzió. A holland szerző hasznos tippjei a ➲ ***www.derickrethans.nl/errorhandling/talk.html*** oldalon olvashatóak.
Az URL első részletét nézve a blogból arról értesülünk, hogy igen széles látókörű programozóval van dolgunk, akit messze nem csak a bitek állítgatása érdekel – s ez fontos szempont lehet, amikor dönteni akarunk valamelyik nyomkövető vagy nyomkövetési protokoll mellett.
Az utóbbi időben viták lángoltak fel azzal kapcsolatban, hogy mitől jó/jobb egy nyomkövetési protokoll.
Az ***Xdebug*** a ***DBGp***-t használja, ami ***ASCII*** adatforgalmat jelent, ***XML*** formátumban, szemben pl. a ***Zend Studio*** bináris szerkezetű protokolljával. A nyílt forráskódú programok írásának nyilván az előbbi, átláthatóbb szabvány kedvez – a ma használatos sávszélességek mellett általában elhanyagolható az a kis különbség, amivel több adatforgalmat jelent az ***ASCII*** kódolás a binárissal szemben. Ezzel együtt sajnos meg kell hagyni, hogy a ***Zend Studio*** mint eszköz valóban minden igényt kielégít a Linux-kedvelők háza táján is – nem egyszerű versenyre kelni vele. De erről majd egy későbbi részben.
Az ***Xdebug*** támogatja még a ***GDB***-t (***Gnu DeBugger protocol***) és a ***PHP 3 Debugger protocol***-t is.

Xdebug: a webszerver oldalán

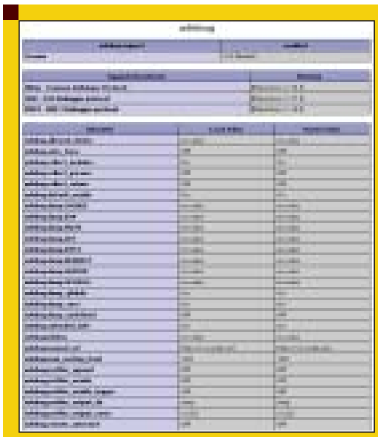
Az ***Xdebug*** a webszerver „oldalán" fut, a kliensprogram tőle függetlenül a helyi gépen. (E kettő meg is egyezhet.)
Ha éles környezetben szeretnénk forrásból lefordítani az ***Xdebug***-ot, akkor (***dpkg*** alapú csomagkezelő jelenléte esetén) fordítás előtt adjunk ki egy

```
apt-get install php-devel
➥automake gcc cpp
```

parancsot, az egész procedúra után pedig majd ezt:

```
apt-get remove  php-devel
➥automake gcc cpp
```

Azaz a fordításhoz szükséges programokat kénytelenek vagyunk feltenni, de utána kár lenne telepítve hagyni őket. A biztonságot szem előtt tartva ne adjunk teret felesleges programok futtatásának a későbbiekben.
Lássuk tehát magát a fordítást. Csomagoljuk ki az ***Xdebug*** programot a `tar -xvzf xdebug-2.x.x.tgz`



1. ábra phpinfo – a lefordított és beüzemelt Xdebug modullal

paranccsal. Váltsunk be (cd-vel) a keletkezett könyvtárba. Bárhová kicsomagolhatjuk a *tar* csomagot, nem kell a *PHP* forrása környékére, hiszen magát a *PHP*-t nem kell újrafordítani (annak ellenére, hogy a forrására szükségünk van, épp a most következő parancshoz). Futtassuk le a

```
phpize
```

parancsot, amely remélhetőleg benne van a *PATH*-unkban. (Ha nincs, adjuk meg abszolút elérési úttal.) Ezután – behelyettesítve a */etc/php.ini* helyére a mi webszerverünk *PHP*-konfigurációs fájlját – jöhet a *make* fájl elkészítése és futtatása:

```
./configure --enable-xdebug
➥-with-php-config=/etc/php.ini
make
```

A kulcsfontosságú lépés, melyben – root-ként – a „végterméket", az *xdebug.so* modult elérhetővé tesszük a webszerver számára:

```
su -c 'cp modules/xdebug.so
➥/usr/lib/php'
```

Értelemszerűen a */usr/lib/php/* helyett a *php.ini*-ben levő *extension_dir* beállításának tartalma értendő. Az *ActiveState* javaslata szerint a nyomkövetéshez érdemes kikapcsolni a *php.ini*-ben az *output_buffering* hatást egy pontosvessző sor elejére írásával (ezzel megjegyzésbe tesszük), vagy *„off"*-ra állításával. A *php.ini* konfigurációs fájl végére írjuk be az alábbi sorokat (a */usr/lib/php/* helyett itt is az *extension_dir* beállításának tartalma értendő, és az *<idekey>* helyett egy azonosító karaktersorozat) :

```
zend_extension="/usr/lib/php/
➥xdebug.so"
xdebug.remote_enable=1
xdebug.remote_handler=dbgp
xdebug.remote_mode=req
xdebug.remote_host=localhost
xdebug.remote_port=9000
xdebug.extended_info=1
xdebug.idekey=<idekey>
xdebug.profiler_enable=1
xdebug.profiler_output_dir="/tmp"
```

Ezekről a kapcsolókról természetesen bőséges információt nyerhetünk az *Xdebug* honlapjáról közvetlenül, vagy az onnan letölthető dokumentációból (*docs.tar.gz*). A *profiler*-re vonatkozó utolsó két sor csak akkor aktuális, ha profilt is szeretnénk készíttetni, azaz teljesítményt elemeztetni. Miért is ne tennénk? Ez az egyik leghasznosabb lehetőség az *Xdebug*-ban.
Ezek után már csak a webszerver újraindítása van hátra. Ha ezek után kiadunk parancssorból egy `php -m` parancsot, akkor a webszerver *PHP* és *Zend* moduljai listázódnak ki – mindkét listában látnunk kell az *Xdebug*-ot. Böngészőnkben hívjunk meg egy ilyen *PHP* oldalt:

```
<?php phpinfo(); ?>
```

Látszania kell az *Xdebug* modulnak (*Zend Technologies with Xdebug v2.0.0beta5, Copyright (c) 2002, 2003, 2004, 2005, by Derick Rethans*) és részleteinek (*1. ábra*)

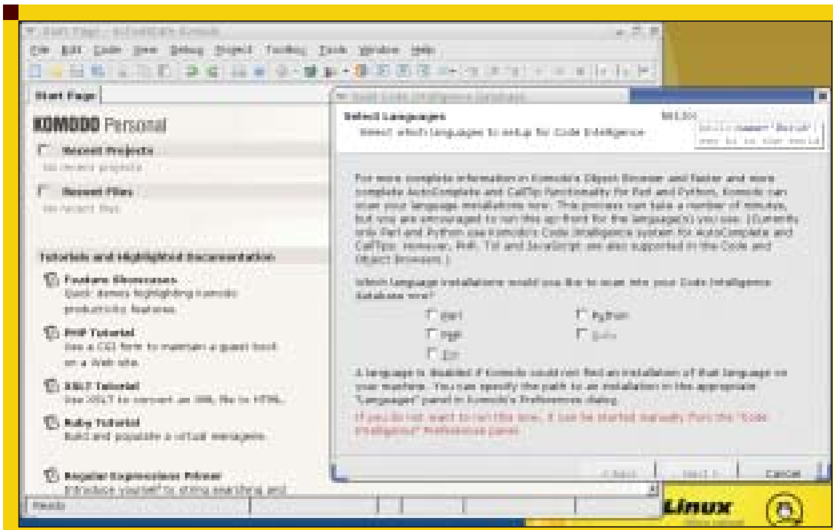
Talán feltűnt a fenti beállítások közt, hogy *localhost*ot adtunk meg *remote_host*-ként, azaz *„távoli gép"*-ként. Ezt a szokásos módon egy *SSH*-alagút nyitásával oldjuk meg – így a legbiztonságosabb:

```
ssh -R 9000:localhost:9000
➥loginnév@gépnév
```

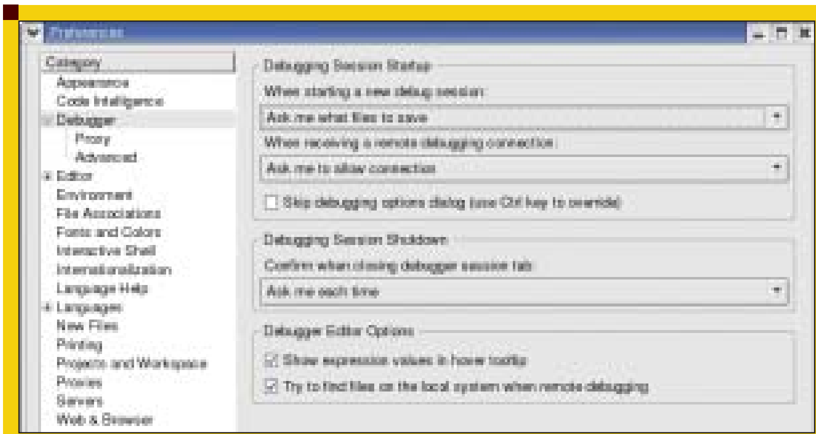
Kliensprogramok Xdebughoz

Többféle kliensprogram is íródott az *Xdebug*-hoz. Magában az *Xdebug* csomagban is van egy *debugclient* nevű parancssori kliens, de ezt elég körülményes használni az *XML* szerkezetek miatt. A szerző fent említett „hasznos tippjei" közt a *Weaverslave* programot nevezi meg egy használható kliensprogram-alternatívaként. Ez Microsoft Windows alá készült, de *wine*-nal is elindul (*2. ábra*).
Az igazán jól használható kliensprogram (sőt, annál több is) az *ActiveState* által kiadott *Komodo*. A 3.5-ös verzióval próbálkoztam 2006 júniusában. Időkorlátos: egy hónapig próbálható ki ingyenesen, de ha lejárt az időszak, akkor lehet kérni, hogy küldjenek újabb licencet – vagy megvásárolható a használati jog 30 dollárért. (A */home/.komodo* törlésével és magának a programnak az újratelepítésével nem lehet meghosszabbítani az időkorlátot, mert a licencfájlba a lejárati dátum van bekódolva.) Angol nyelvű, igazán jól felszerelt fejlesztőkörnyezetet kapunk a kezünkbe, van benne több programnyelvhez is tanfolyam (persze *PHP*-hez is). Alapvetően projekt alapon végzett munkához lett tervezve, de különálló fájlok is szerkeszthetőek.

2. ábra Weaverslave 4 – kliensprogram



3. ábra Komodo – telepítés. Készül a KódIntelligencia Adatbázis



4. ábra A Komodo nyomkövetési beállításai

5. ábra Komodo felhasználói kézikönyv – mindenre kiterjedő eligazítás

A *Komodo* (vagy bármely más fejlesztői) környezet egyik leghasznosabb támogatása az, hogy már begépeléskor jelzi – hullámos aláhúzással – a szintaktikai hibákat. Ez rengeteg időmegtakarítással jár, mert meg sem kísérli az ember a hibás kód kipróbálását. Az *Edit|Preferences|Debugger|Proxy* menüpont alatt a *„Listen for debug connections on port:”* értékét ugyanarra érdemes állítani, mint amit fent a *php.ini*-ben megadtunk (9000).

Beállításainkat ellenőrizhetjük a *Debug|Listener Status* menüponttal. Ha még nem lenne bekapcsolva, kattintsunk a Debug menü *Listen for Remote Debugger* pontjára (azaz: *Távoli nyomkövető figyelése*). Következik a webböngésző okosítása. A normál *URL* végére írjunk egy *„?XDEBUG_SESSION_START=<idekey>”* kiegészítést, mint *GET* argumentumot. A dokumentáció szerint az *<idekey>*-nek meg kell egyeznie a *Debug|Listener Status-beli Proxy Key* értékével, azonban azt tapasztaltam, hogy ha helyi gépen futtatjuk a webszervert, akkor ez nem kötelező, sőt, még a *php.ini*-beli *idekey*-értékkel sem kellett, hogy megegyezzen a fenti *GET* argumentum. Amikor először kap ilyen „megbízást” a böngésző, hogy keresse fel például ezt az oldalt:

```
http://valami.hu/pelda.php?
➥XDEBUG_SESSION_START=joskapista
```

akkor sütivel (*cookie*) rögzíti az *idekey* információt, és a későbbiekben már arra támaszkodik. Ezekről a böngésző által támogatott munkamenetekről bőven olvashatunk a ➲ *www.xdebug.org/docs-debugger.php#browser_session* oldalon. Ezek után megkezdődik a nyomkövetés a *Komodo*-ban, lehet lépegetni, futtatni az első (akár feltételes) töréspontig stb. Sőt, a *PHP* kódon belül is elhelyezhetünk töréspontot az

```
xdebug_break();
```

használatával (bár ez nem kezdeményez új munkamenetet, és egyébként sem szép belenyúlni a kódba – ez talán inkább a spártaibb kliensprogramoknál segíthet).

A Firefox-bővítmény

Sorozatunk egy korábbi részében vizsgáltuk a *Gubed* nyomkövetőt. Láthattuk, hogy milyen sokat segített egy egészen egyszerű kiegészítés a böngésző menüpontjainál, amiben a nyomkövetést kiváltó *URL*-változtatást lehetett kérni vagy visszavonni. Úgy döntöttem, hogy ezt a lehetőséget az *Xdebug* számára is meg kell teremteni. Az említett *Gubed*-bővítmény nagyon hasonlóan működik, mint amire

az *Xdebug*-gal való nyomkövetéskor van szükség: kis módosítás az *URL*-en, és már megy is a nyomkövetés.
A ➲ *gubed.sf.net/mozilla/gubed.xpi* helyről letölthető *gubed.xpi*-ből indultam tehát ki. Kíváncsi voltam, mi az a minimális erőráfordítás, amivel már el tudom érni célom.
A kiindulási fájlt átneveztem *xdebug.xpi*-nek. Mivel ez (többszörösen) tömörített fájl, először gyártottam egy kibontó és egy összerakó parancsot. Ezzel lehetett kibontani az *xdebug.xpi*-t:

```
rm -Rf ki; mkdir ki; cd ki
➥;unzip ../xdebug.xpi ;cd
➥chrome; mkdir ki; cd ki;
➥unzip ../gubed.jar
```

Ezzel pedig össze lehet csomagolni *xdebug2.xpi* néven, hogy ne írja felül a kiindulási fájlt:

```
cd ki/chrome/ki; zip -9 -r
➥../gubed.jar *; cd .. ; rm -r
➥ki; cd .. ;zip -9 -r
➥../xdebug2.xpi *
```

A *„ki”* (azaz *kibontott*) könyvtárban végrehajtottam az alábbi (kis-nagybetű érzékeny!) cserét:

```
perl -pi -w -e 's/Gubed/
➥Xdebug/g' `tree -fi`
```

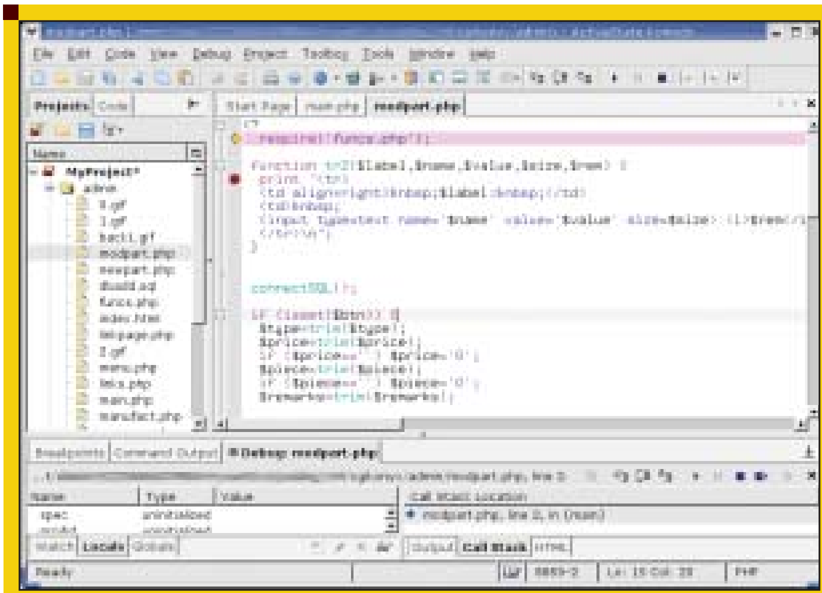
Ezek után már bele lehetett nyúlni a *JavaScript* programocskába és a menürendszerbe (ez utóbbi nem volt létszükséglet, de ha már hozzányúltam, miért ne magyarítsam azt a 4 menüpontot).
A *gubedOverlay.js* fájlban megjegyzéssel (azaz // jellel) láttam el a

```
getURL="?gbdScript=";
```

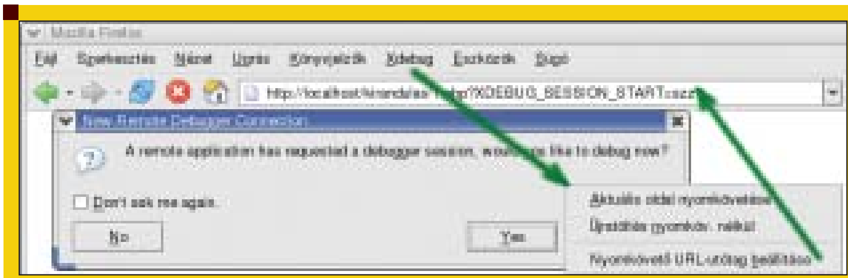
sort, és alatta elhelyeztem a számomra hasznosabb

```
getURL="?XDEBUG_SESSION_START=
➥kulcsom";
```

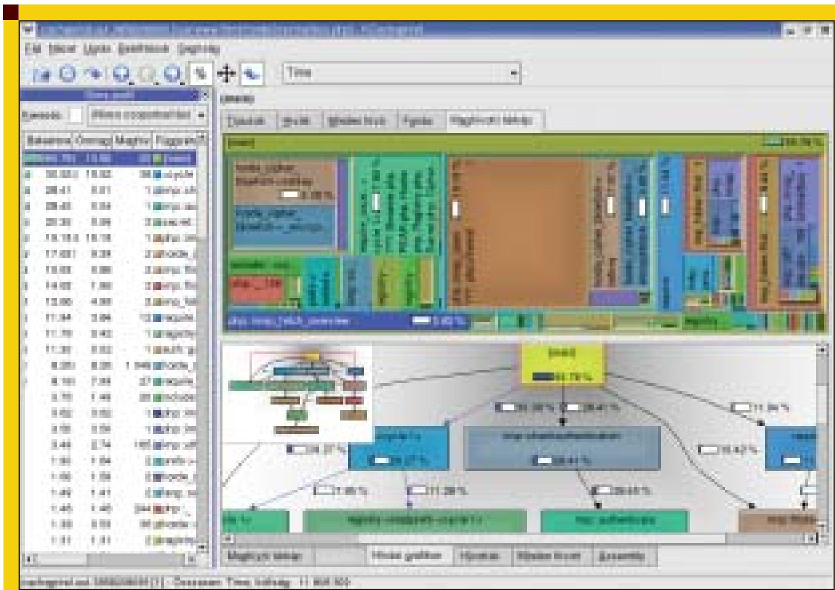
parancsot. Itt a *„kulcsom”* egy olyan *„IdeKey”*, azaz *Fejlesztő-környezetiKulcs*, amivel azonosítani tudja a webszerver, hogy melyik fejlesztő környezet kéri a nyomkövetést. Ez benne van a *php.ini* fájlban is a megfelelő részletnél, mint fentebb említettük. Arra szolgál, hogy lehetővé tegye az azonosítást, ha netán egyszerre többen is szeretnének nyomot követni, és segít kivédeni, hogy illetéktelen kezdjen el munkálkodni. Nem kell lelkiismereti problémát csinálni abból, hogy ezt belehuzalozzuk a kódba, mert van egy beállítási lehetőség



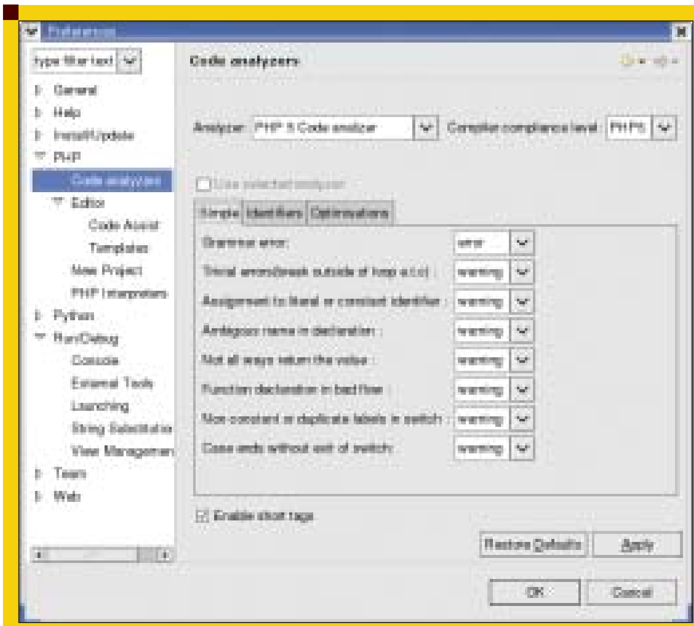
6. ábra A Komodo teljes díszben, beállított törésponttal



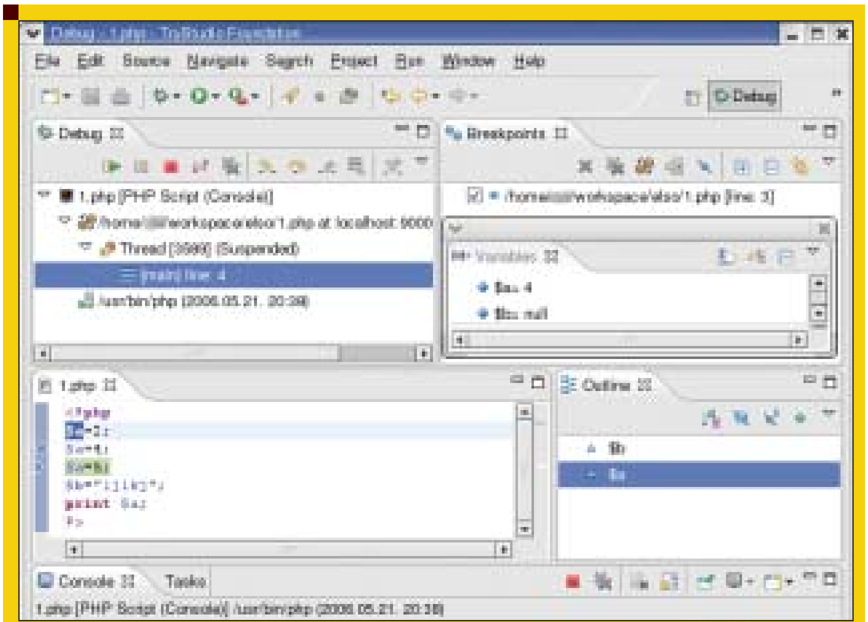
7. ábra A saját gyártású Xdebug bővítmény



8. ábra Teljesítményelemzés az Xdebug kimenetéből, KcacheGrind-del



9. ábra Kódelemző a Xored::TruStudio-ban



10. ábra Teljes díszben a Xored::TruStudio – nyomkövetés törésponttal

a bővítmény menüpontjai közt, ami éppen ennek átírására szolgál. Majd néhány sorral lejjebb:

```
//  var url = scriptURL +
➥getURL + current_url;
  var url = current_url +
  ➥getURL ;
```

Végül egy apróság (ami különös módon nem a menüpontok szövegeinél van, hanem bele lett drótozva a programba) – ezt *iso-8859-2*-es kódolással fogadta el a program:

```
//  var tmp = prompt( "Enter the
➥Script URL: ", scriptURL );
  var tmp = prompt( "Írja be
  ➥a nyomkövetési utótagot: ",
  ➥getURL );
```

Lejjebb a *ScriptURL* értékadása helyett:

```
  getURL = tmp;
```

Ezen kívül még a *locale* könyvtár alatti *en-US*-t másoltam át *hu-HU* nevűként, és írtam át azt a néhány menüpontot, ami ott felsorakozott, *UTF-8*-as kódolással. (Vagy ékezet nélkül is lehet, ha biztosra akarunk menni, és nem lényeges az akkurátus ékezetek látványa).
Ezek után összecsomagoltam az *.xpi* fájlt (ami leszedhető a ➲ *www.osb.hu/z/xdebug.xpi* helyről), és húzd-és-ejtsd módszerrel ráhúztam a *Firefox*-ra. Következő indításkor már meg is jelent és használhatóvá vált a megfelelő *Xdebug* menüpont.
(Ezt azonban nem lehetett egyszerre használni a *Gubed* menüponttal, az ütköző azonosító- és változónevek miatt; vállalkozó kedvűek kijavíthatják ezt a szépséghibát.)
Megfelelően beállított *Xdebug*-gal teljesítményelemzést is végezhetünk (lásd az `xdebug.profiler...` beállításokat a *php.ini* fájlban).
A megadott könyvtárban szépen gyülekeznek a *KCacheGrind*-nek közvetlenül átadható fájlok.

A Xored::TruStudio

Maga a program a ➲ *www.xored.com/trustudio/download* oldalról tölthető le. Az *Eclipse*-re épül, de ezt nem kell (sőt nem szabad) külön telepíteni előtte! Bár nagyon komoly, *Java* alapú fejlesztőkörnyezetként működik, a *Xored::TruStudio* (tapasztalataim szerint) sajnos csak a helyi gépen, *CGI PHP*-környezetben tud futni, és ráadásul időnként egy licenc után érdeklődő (de aztán eltüntethető) dialógusablak zavarja meg jó közérzetünket.
Ennek ellenére – különösen a kódelemző része miatt – megér egy próbát, hátha egy hamarosan megjelenő következő verzióban már távoli nyomkövetésre is képes lesz.
Sorozatunk befejező részében a *java* alapú *Zend Studio* kliensprogram és a hozzá tartozó (*Apache*-modulként működő, *.so* fájlokat tartalmazó) szerver kerül terítékre.

Szabó Zoltán
(szz@freemail.hu)
Négy gyermekével és feleségével Pannonhalmán él. Tíz éve kísérletezik a Linuxszal. Matematikát és informatikát tanít, diákotthonban keseríti a rábízottak életét. Szívügye a PHP, a PostgreSQL és a Moodle.