United Business Media

Dr.Dobb's DIGEST

The Art and Business of Software Development November 2010

Bugs Are Good, Yes? No!?

By Jonathan Erickson,
Editor In Chief

Like most people, I try to avoid buggy code. But John Bender isn't most people. He embraces buggy code. He writes buggy code. Buggy code is John's passion. Oh! Did I mention that John isn't a programmer by trade? Instead, John Bender is a postdoctoral research associate in the department of biology at Case Western Reserve University, which explains his interest in bugs — real ones, like cockroaches.

While most of us see little redeeming value in cockroaches, John seems to think they would make a great model for small rescue robots. Discovering how cockroaches use all six legs to walk, run, and turn at the same time isn't easy. However, John has figured it out using a pair of high-speed cameras and a custom computer program written in Python. The end result is that John and his research team can simultaneously extract the 3D movement of a cockroach's 6 legs and 26 leg joints. Here's how...

Digital video of a walking cockroach was collected in grayscale at 500 fps from two synchronized, calibrated cameras. The researchers improved the legs' visibility by painting white dots on the joints. Compared to manual digitization of 26 points on the legs over a single, 8-second bout of walking (or 106,496 individual 3D points), the software achieved approximately 90 percent of the accuracy with 10 percent of the labor. They reduced the complexity of the tracking problem by securing a cockroach on a leash and allowing it to walk in place on a lightly oiled glass surface; the algorithms implemented are extensible to free walking. The software is written in Python and freely available at http://sourceforge.net/apps/mediawiki/legger/. The user interface was built in Python using the WxWidgets library. The researchers also used the open-source computer vision software library OpenCV (http://sourceforge.net/projects/opencvlibrary/) to calculate the positions of the cameras with an iterative camera calibration algorithm presented in "A Flexible New Technique for Camera Calibration" (http://research.microsoft.com/en-us/um/people/zhang/Papers/TR98-71.pdf).

The software (again available here) let John analyze in a matter of hours 106,496 individual 3D points, with about 90 percent accuracy. He estimated that the old method of analyzing the 3D movement of all 26 joints frame-by-frame would take at least a couple of weeks. By automating much of the work, the team also sought to eliminate some of the subjectivity of analyzing each frame by human eye.

As disgusting as cockroaches are, John's research is really interesting, particularly when he lifts the hood on the image processing. You can find all the details in the PLoSOne paper, "Computer-Assisted 3D Kinematic Analysis of All Leg Joints in Walking Insects" (http://www.plosone.org/article/info%3Adoi%2F10.1371%2Fjournal.pone.0013617), by John Bender, Elaine Simpson, and Roy Ritzmann.

To show the lassoed cockroaches walking in place (with the cameras beneath the pane of glass), John put together a video available on Dr. Dobb's TV at http://www.drdobbs.com/tv/index.jhtml;jsessionid=HVZZXDRJSCW4RQE1GHPCKH4ATMY32JVN?bclid=1747276019&bctid=656263802001&bcpid=1753162249&cid, which also presents the visualization implemented in MATLAB.

I admit that John's bug-filled programming project is impressive. But I've also lived in too many apartments that could provide candidates for his research. Keep up the good work, John. We'll be watching — from afar.

Return to Table of Contents

Teaching Real-World Programming

Students meet for regular code reviews with senior programmers from area companies

By Larry Hardesty, *MIT News Office*

In the real world, code clarity is as important as software performance. Large software projects can involve hundreds of programmers, each working on a small corner of an application, and over the course of a project, personnel turnover can be high. Testing, revising, and updating software may require people to review code that they had no hand in writing. If the code isn't intelligible, engineers can waste a huge amount of time just figuring out how an existing program does what it does.

MIT professors of computer science Charles E. Leiserson and Saman Amarasinghe, who co-teach a class called "Performance Engineering of Software Systems" (http://stellar.mit.edu/S/course/6/fa10/6.172/index.html), believe that undergraduates should be taught to write clear code, not just running code. So this fall, for the second year in a row, students in the class won't just have their projects graded by their teaching assistants; they'll also have their code reviewed by senior programmers from the Boston area, who volunteer their time through a program that Amarasinghe and Leiserson call Masters in the Practice of Software Systems Engineering, or MITPOSSE.

The roughly 70 students in the class work on four major projects a semester, in two-person teams. After each project is submitted, students from separate teams are paired, and each pair meets with one of the professional programmers for a code review. The code review lasts between 60 and 90 minutes, and each of the 20-odd professionals meets with two pairs of students. The code reviews don't count toward the students' final grades, but they are mandatory.

Leiserson and Amarasinghe introduced the class, which they teach only in the fall, three years ago. The first year, enrollment was lower than it is now, and the professors decided to hold their own half-hour code reviews with individual students after each project. "That killed us," Leiserson says. "Even though we had a small class at the time, it was clearly not scalable." So in the summer of 2009, Amarasinghe and Leiserson put the word out among friends and colleagues, former students, and members of the Computer Science and Artificial Intelligence Laboratory's Industry Affiliates Program that they were recruiting seasoned programmers to serve as mentors to their students in the fall. "We got a tremendous response," Leiserson says. Indeed, in both 2009 and 2010, the professors had to turn away applicants to the program. To Amarasinghe and Leiserson's knowledge, theirs is the first computer science class in the country to incorporate code review with professional software engineers.

According to Amarasinghe, the chief purpose of the MITPOSSE program is to help students develop their own programming styles. By "style" he means things like selecting names for variables, so that it's clear to later readers what they refer to; avoiding the use of so-called "magic numbers," numerals that are simply inserted into equations without explanation (the better practice, Amarasinghe explains, is to assign the number a name that indicates its purpose, and then use the name in subsequent equations); indenting lines of code; and, most important, adding comments — lines of text that appear next to the program code and explain what it's doing, but which the computer ignores when converting code into an executable program.

In many computer-science classes, Amarasinghe says, professors trying to preserve intelligibility will insist on a particular style of coding, which may not be natural for some students and, he says,

Dr.Dobb's

EDITOR-IN-CHIEF
Jonathan Erickson

EDITORIAL
MANAGING EDITOR
Deirdre Blake
COPY EDITOR
Amy Stephens
CONTRIBUTING EDITORS
Mike Riley, Herb Sutter
WEBMASTER
Sean Coady

VICE PRESIDENT, GROUP PUBLISHER
Brandon Friesen
VICE PRESIDENT GROUP SALES
Martha Schwartz

AUDIENCE DEVELOPMENT
CIRCULATION DIRECTOR
Karen McAleer
MANAGER
John Slesinski

DR. DOBB'S
600 Harrison Street, 6th Floor, San Francisco, CA, 94107. 415-947-6000.
www.drdobbs.com

UBM LLC

Pat Nohilly Senior Vice President, Strategic Development and Business Administration
Marie Myers Senior Vice President, Manufacturing

TechWeb

Tony L. Uphoff Chief Executive Officer
John Dennehy, CFO
David Michael, CIO
Bob Evans Senior Vice President and Content Director, InformationWeek Global CIO
Joseph Braue Senior Vice President, Light Reading Communications Network
Scott Vaughan Vice President, Marketing Services
John Ecke Vice President, Financial Technology Network
Beth Rivera Vice President, Human Resources
Fritz Nelson Vice President and Editorial Director

techweb™

can actually lead to bad code. "The way we look at programming, it's like writing an English paper," he says. "If you are in English class, there's no set way of writing." What's important is that a programmer's style be consistent, not that it slavishly ape some model.

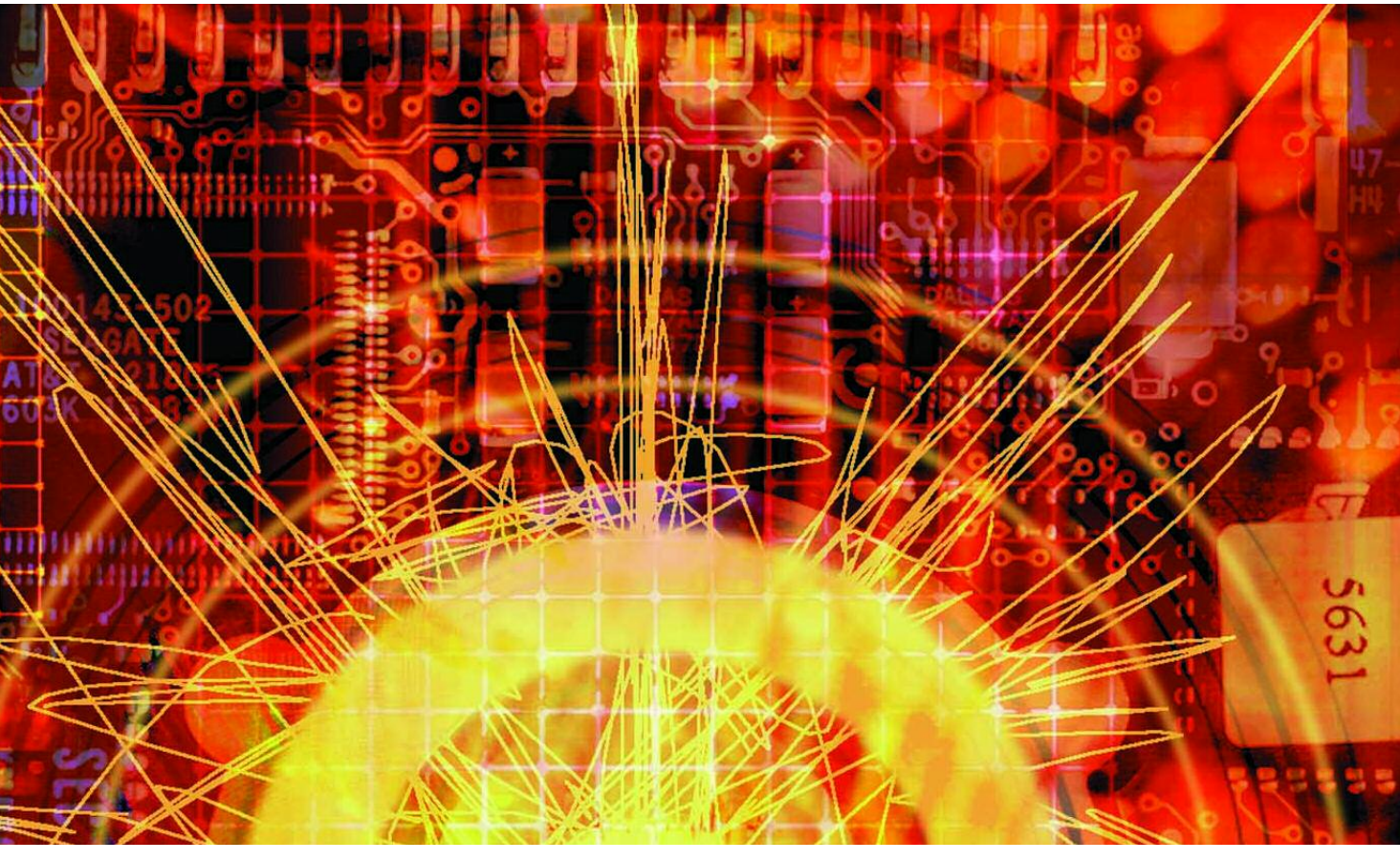
Of course, all computer-science professors emphasize the importance of good comments, but busy students usually try to get their programs working and then stick in comments in whatever time they have left. Reid Kleckner, who took Amarasinghe and Leiserson's class as an undergraduate last year and, as a master's candidate, is a teaching assistant this year, says that the MITPOSSE reviews can help some students get a better grasp of the purpose of comments. "Usually, students hear, 'Comment your code,' and they say okay, line by line, 'This is what this line does, this is what this line does.'" But, Kleckner explains, comments are in fact more useful when they are spare and descriptive. A program might, for instance, include some clever but obscure procedure that makes it run more efficiently. Simply commenting in the margin that, say, the procedure rounds a number up to the nearest power of two may be all the information the reader needs. "You don't need to say, here are the arithmetic operations that I am performing, and they achieve this result. Here's my mathematical proof," Kleckner says.

Moreover, Kleckner says, with the MITPOSSE approach, "It's not just graduate students telling you how to write code. It's coming from a figure of authority, in some sense." He laughs and adds, "Not that the T.A. is not a figure authority." But a T.A. is not a prospective employer.

Indeed, for the mentors, some of the appeal of volunteering for the program may very well be the opportunity to scout talent. So Leiserson and Amarasinghe have drawn up a set of rules that the mentors agree to abide by, including "no recruiting, hints at job opportunities, offers of summer internships, lab tours, free dinners, etc.," until "the semester is over and no power relationship exists."

But many of the mentors simply find the work intrinsically satisfying. Barry Perlman, who as an independent software consultant doesn't have any job vacancies to fill, volunteered with the program last year and came back again this year. "I felt like I was doing something useful," he says. "The world doesn't need more software engineers; the world needs better software engineers." Andrew Lamb, too, has joined the MITPOSSE for a second year in a row. He works for a small software company, and though he acknowledges that "it's nice to get that name out there," the company isn't currently looking for any new hires. But, Lamb says, "when they have some question and I'm able to answer it, and their eyes light up, and they say, 'I get it, I get it,' that's very cool."

Return to Table of Contents



Industry-Focused Development

Domain-specific languages can simplify the process

By Jon Barcellona

WorldDoc provides Web-based health management tools, such as personal health records, that health insurers and employers can buy for people covered under their plans. The idea is that people will use the information systems to stay healthier, thus living better and lowering costs for insurers and employers. The company's fully hosted central database engine integrates an individual's health risk assessment, medical and pharmacy claims, laboratory test results, and biometric data to provide care management such as personalized medical goals, find gaps in care, and suggest actions to stay well.

WorldDoc's vision is to build a "healthcare intervention engine" that integrates with its existing member healthcare portal. Such a system would let clinicians write rules to process health data information, medical claims, and other information for millions of members. The rules would match a person's health profile with specific interventions, which could be delivered as e-mails, text messages, Web-based coaching, mail, phone calls, and via a smartphone application. It's a highly complex vision, and it needs to execute against millions of members' health data while maintaining a high level of performance.

To do the software development, WorldDoc hired a four-person agile development team from my company, Digital Architects; it wanted a production application in one year. We had a basic development road map in mind from past experience:

1. Clearly define the problem domain.
2. Don't build what we could buy.
3. Start small and rapidly deliver a proof of concept.
4. Deliver the main project using agile principles — especially "deliver working software early and often."

Based on our experience, we decided at the outset to create a new domain-specific language (DSL), this time for the medical domain for medical rules. A DSL is a computer programming language that targets a particular domain — healthcare in this case — instead of a general-purpose language, such as Java or C++ that's designed to solve any kind of software problem. DSLs offer numerous benefits for projects like WorldDoc's. Users (clinicians in this case) understand them, thereby facilitating communication; they increase the productivity of software developers by raising the level of abstraction; and the system can be quickly updated if user requirements change — a must for a project like this.

The most critical and time-consuming aspect of such a "rules engine" project was the initial analysis of the language — understanding the healthcare domain and translating it into a simple programming language that would be intuitive to the clinicians. Another challenge was creating an intuitive user interface for clinician rule entry. Additionally, we learned that the data access layer needed to be mapped from the language elements to the WorldDoc healthcare data, so that future changes to the data structure could be changed via mapping of the language elements to the data, rather than changing the DSL itself.

Define The Problem

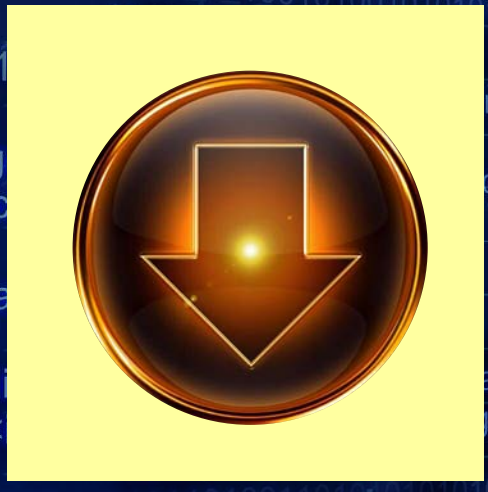
The project sprawled out in three big areas, according to Ben Say, World Doc's CIO. One, the clinical rules went very deep. Two, there's a huge variety of member preferences. And three, there's tremendous breadth of possible intervention the system could recommend to patients. Using a DSL let us automate more of the development.

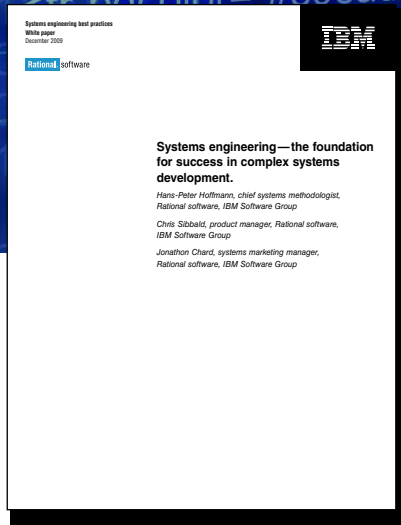
With the DSL approach settled upon, we began defining the problem domain by categorizing the tasks at hand into problem parameters, then reorganizing those tasks into technology areas. In doing so, the problem looked like this: Create a DSL for medical rules that:

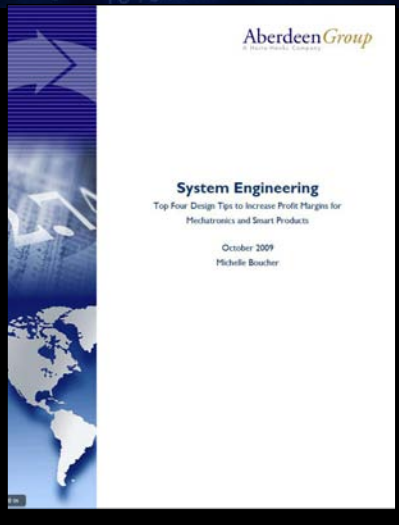
- Clinicians find intuitive;
- Integrates with the existing member health data and intervention process;
- Creates rules that execute swiftly enough to handle millions of members.

This led us to realize that we'd also need to build a Web-based portal for authoring rules that clinicians could use. It would also have to allow authoring and validation of rules syntax and data elements, as well as support full-featured rules testing and workflow to manage processes from the draft phase into rules production.

Click this button to download the Systems Content Resource Bundle!







Systems Engineering

The foundation for success in complex systems development

System Engineering

Top four design tips to increase profit margins for mechatronics and smart products

Building smart products

Best practices for multicore software development

Exclusive Downloads!

Compliments of IBM

Powered by Dr.Dobb's

Build Or Buy?

In identifying the tools we'd use to implement the system, an important element of the build-versus-buy decision is knowing the critical differentiation of the project — that's what should receive the heaviest investment. For WorldDoc, the design and implementation of the medical rule DSL and execution speed were the critical factors.

The rule authoring was important, but it was really a means to an end, not the core differentiation. Knowing that, and that we didn't want to overinvest in the portal, simplified how we created the Web-based Rules Authoring Portal. We needed a portal that could deliver a solid Web experience with workflow capability that integrates into existing databases as well as the new rules engine API. We selected the OutSystems Agile Platform (http://www.outsystems.com/), which simplified this process by building most of the user experience through its visual DSL that generates a .NET implementation. The OutSystems platform is a visual tool for creating and implementing not just the user interface but the entire code logic. More important for this project, it includes a customizable domain-specific language for developing and changing Web business applications. That allowed most of our resources to focus on delivering the core rules engine. But authoring a DSL language, building parsers and lexers, and integrating to external data is a big task. This demanded automation.

To help automate the DSL authoring, we turned to Antlr (short for "Another Tool for Language Recognition," see http://www.antlr.org/). Antlr is a language tool that provides a framework for constructing recognizers, interpreters, compilers, and translators from grammatical descriptions containing actions in a variety of target languages. While the toolkit includes visual modeling and some interpreting capability, its most important feature for us is that it generates C# code to do the language parsing, lexing, and code generation.

With the decision to include Antlr in the mix, our implementation strategy became clear: Let clinicians author medical rules in our simplified DSL, then generate C# code for these rules, and finally compile these rules into native .NET code for high-performance execution.

Rapid Proof Of Concepts

At this point, we had a good hypothesis on how we were going to solve this business problem. The proof, however, is in the implementation.

We executed a two-month proof of concept in which we delivered a slice of functionality through each technology component, verifying that each would operate and integrate as advertised. This phase also included a clinician usability review to ensure that our new DSL was actually well designed for healthcare users. During this process, we proved the concepts and gathered many lessons learned for targeting the final product — both in DSL changes and technology refactoring.

Several implementation challenges led to technology changes during the process. The WorldDoc system is a Web application and therefore the rule editor had to be Web based. The team considered JavaScript, applets, and Flash, and initially developed a fully functional editor utilizing JavaScript. It worked, but the response time and maintainability wasn't good enough. The response time affected the user experience, which is critical when creating numerous rules based on hundreds of healthcare-related definitions. Many rules editors have "rich functionality" that must be customized for different browsers, which then must be tested and updated for every new version of the browser. The team decided to convert the WorldDoc editor to Flash, to avoid costly maintenance and support. But it did several JavaScript iterations before making the call to convert the editor to Flash. Our team had never implemented several of the technologies and integration methods we used, and many of them had never been implemented within the healthcare industry — Antlr, OutSystems, Editor Code Complete, and Syntax Highlight to name a few. We needed proof-of-concept projects to know whether the technologies would work together well, and we had to do so prior to estimating project tasks.

Deliver Using Agile Principles

Our project plan consisted of 16 sprints, with two dedicated to refactoring and performance tuning. About 25 percent of each sprint was planned for adjustments and refactoring. Each had a demonstrable deliverable with a focus on the current release. To manage scope, we ruthlessly avoided any feature description that started with the word "Someday …" And like all complex projects, we encountered our share of issues, feature triage, and spikes in team effort. Still, overall the process was smooth.

The WorldDoc project was an interesting technology challenge. We authored a DSL for clinicians with an easy-to-use Web-based user experience, by leveraging two approaches:

- A visual modeling DSL (OutSystems Agile Platform) to build the user experience in .NET; and
- A text-based DSL to author the WorldDoc healthcare intervention language and generate the C# code for the execution engine.

We managed risk by proving all the technology and integration points early in the process and kept the project on schedule by limiting scope, delivering early and often, and planning for our learning curve and refactoring needs.

This project couldn't have been delivered in this time frame and resources without the use of these DSL toolsets. The cost and complexity involved in evaluating and learning a DSL toolset can be a deterrent, but once you've delivered a complex project using the proper DSL for your problem domain, you'll never want to do it the old way again. As the popularity of DSL toolkits expands, we look forward to further toolkit improvements, which will continue to reduce developer ramp-up time.

— Jon Barcellona is principal architect for Digital Architects (http://www.digitalarch.com/).

Return to Table of Contents

Managing Open Source Licensing

Semiconductors present a big challenge — and bigger opportunity — when it comes to open source

By Kamal Hassin

Software is a massive enabler for the semiconductor industry. It's difficult to find a device that isn't enabled by software in some way. Evolving from simple components to complex systems on a chip (SOC), semiconductor-based products now require many categories of software. The list starts with drivers to configure operating modes. On top of that, real-time operating systems (RTOS), SDKs, networking and security, administration, media formatting and compression, and of course an endless list of applications. Unsurprisingly, the semiconductor industry now spends more on software development than on all other R&D aspects.

How do developers build all of this software at a reasonable cost? Increasingly, they supplement custom coding with outsourced code, commercial libraries, and open source software. Open source has become a significant component of all software development, intentionally and sometimes unintentionally, thanks to the abundance of available code, its apparent free cost, and a high degree of stability and security.

But while open source code can appear to be free, it is not without obligation. It typically comes laden with licensing and copyright responsibilities that are enforceable by law. Even accidental infringements can result in fines and injunctions, which can play havoc with manufacturing schedules, inventory control, and supply chain management. Given the significant deployment volumes for semiconductors and the products into which they embed, it is important to ensure that software licensing obligations are managed. Ignoring these obligations or simply being ignorant of them can have significant consequences, as some recent legal cases have shown. For example, a lawsuit regarding an open source component called "BusyBox" affected many large companies, including Verizon, Samsung, and others. Another case filed by the Software Freedom Law Center (SFLC) against Cisco sought an injunction for the distribution of Linksys firmware as well as damages and costs.

There are a number of approaches to license management, ranging from doing nothing to fully automated real-time scanning of software to detect and report license obligations. All approaches can be viewed from a cost perspective, with the aim of maximizing developer productivity while minimizing legal risk. The cost of managing software license obligations is analogous to managing defects in the development process. It is well understood that the earlier a defect is identified and corrected, the less expensive it is — and the same is true for licensing obligations. Likewise, the earlier a development organization identifies "licensing bugs" that attract unacceptable license obligations, the less expensive it is to adjust the software to achieve licensing compliance.

Principle Aspects of License Management

A complete approach to assuring open source license compliance will generally include three major aspects:

1. Definition of a licensing policy which must be met by all software projects.
2. Auditing of all project software to detect any third-party source code including open source that is unacceptable based on the licensing policy.
3. Corrective processes to ensure that all released software conform to the licensing policy.

The creation of the licensing policy is an important step as it forms the basis upon which decisions regarding acceptance of open source software will be made. The licensing policy should be defined in accordance with both the business goals of the organization as well as its engineering processes, and generally requires the involvement of business and engineering managers, as well as legal counsel.

Software auditing is required to ensure production code conforms to the licensing policy, although the audit implementation can take a number of forms depending on the preference of the development organization. Audits can range from ad hoc developer training and post-development cycle auditing to proactive automated approaches such as periodic and real-time auditing.

Options for Managing Open Source Licensing

The following options are available to address license compliance at different points in the development process.

- **Do nothing:** This option ignores the compliance issue because it carries the lowest up-front cost, but imposes the highest business risks and largest corrective costs as a product moves closer to launch.
- **Developer training:** Some companies consider developer training and project planning is sufficient enough. This, however, can be an expensive, labor-intensive option given the increasing diversity of software licenses, the high cost of developer training, and the constant churn within the development environment. With this option, compliance depends solely on busy developers and is prone to human error.
- **Post-development license audits:** Auditing late in the project lifecycle does not impact the development workflow and can be implemented manually or using automated software tools. This option does, however, lead to more expensive rework due to additional system retesting cycles.
- **Periodic assessment:** Licensing analysis during development allows for corrections along the way if license violations are detected. This type of analysis can be automated and tends to be less expensive than post-development assessment since changes and re-tests are always easier to undertake earlier rather than later in the cycle.

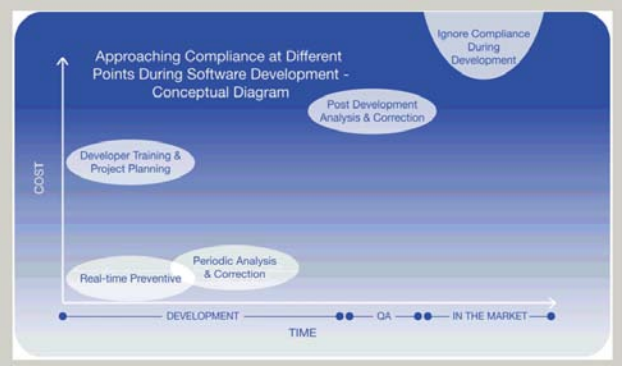


Figure 1: Software development lifecycle phases.

- **Real-time preventive assistance at the developer workstation:** The most proactive approach is to audit software in real time at the developer workstation. The development process is not disturbed and the cost of corrections is minimal as there is no impact to system integration and testing. This process can be automated and generally requires very little developer training.

Regardless of the type of auditing approach, the overall goal is to minimize the time and cost of correcting the final software release so that it meets all functional, quality, and license compliance requirements. Figure 1 illustrates how the cost of detecting and correcting "licensing bugs" escalates in later stages of the development process.

Each organization must consider their approach, balancing the short-term cost of developer training and tools versus the potential longer term cost of post-release legal problems.

Automating Open Source License Management

Tools are available to automate open source license detection and analysis. They can operate on demand, on a periodic schedule or in real-time within the development process. Generally such tools find compliance problems sooner, lowering the overall cost of license compliance and significantly reducing any risk of legal issues that could affect the deployment of semiconductor-based products.

— Kamal Hassin is Director of R&D and Product Management at Protecode (http://www.protecode.com/). He can be reached at khassin@protecode.com.

Return to Table of Contents

Post-Mortem Debugging Revisited

Debugging after the fact

By Stefan Wörthmüller

Several years ago I wrote an article entitled "Post-Mortem Debugging" (see http://www.drdobbs.com/tools/185300443), which included source code to create stack dumps with function names from C/C++ programs that crashed on client sites without any modification on the client's machine. This was done in two steps (available on Windows and Linux/UNIX):

1. Install a fault handler in your application that dumps the call stack to a text file if an error has occurred.
2. Run a program (MapAddr.exe) to add function names to the memory addresses of the stack dump by matching them to a map file from the programs build.

Since then, I've received feedback that led me to a number of different approaches which I summarize in this article.

Retrieving Line Numbers from the Stack Dump (Windows Only)

Visual Studio stores all debug information in a program database, which resides in a file with the extension ".pdb". This database file is created for debug builds by default, but it is easy to create them for optimized release builds as well:

```
In the Project Setup dialog select the following 2
Options:

In the C/C++ Section
        Debug Information Format: Program Database
(/Zi)
In the linker Section
        Generate Debug Info. YES
```

This will generate a .pdb file for the selected build, that should be archived for each released version. It also enables the possibility to debug the release version of your program just as you would do with the debug version.

.pdb files are stored in a binary format that is not documented, but it's easy to access it through a COM API. The Debug Information API: DIA (http://msdn.microsoft.com/en-us/library/t6tay6cz%28VS.80%29.aspx) is provided by Microsoft starting from Visual Studio 2008. Using this API, it is straightforward to get the following information for every code address:

- Unmangled function name (human readable including the list of parameters)
- Source file name
- Line number within that source file

Figure 1 shows a stack dump created from the same FaultApp as used for demonstration in Post-Mortem Debugging. While adding the line number to the stack dump may only seem like a "nice to have" feature, it ultimately proved to be very valuable once I started using it. Before having line numbers, I always had to examine the entire function. Now I only have to check a single line of code, and the problem often becomes obvious immediately.

Accessing the pdb file involves a number of COM-Calls bundled together in a class file. Adding the usage of this class to MapAddr was simple and straightforward. MapAddr can now be called with pdb files as well (as before) using map files. The full source code can be downloaded at http://i.cmpnet.com/ddj/images/article/2010/code/PostMortemDbg_2010.zip.

64-Bit Windows

The fault handler of the first version used x86 assembler calls to unwind the stack. But when porting to 64 bits, I found no way to write stack frames reliable with assembler instructions with optimization enabled. While there are a number of Win32 API functions to unwind the stack, only one (RtlCaptureStackBackTrace; see http://msdn.microsoft.com/en-us/library/bb204633%28VS.85%29.aspx) seems to work on x64 as expected. It is limited to retrieve 62 stack frames, which is sufficient in most cases, although this limit seems questionable.

```
*****************************************************
*** A Programm Fault occured:
*** Error code C0000005: ACCESS VIOLATION
*****************************************************
***    Address: 004398E8
***      Flags: 00000000
*****************************************************
*** CallStack:
*****************************************************

  Fault Occured At $ADDRESS:004398E8
                    ======== 000398E8 == belongs to FunctionC(int,int)
    File d:\projects\mapaddr\faultapp.cpp Line 29
                             with 64 00 00 00 10 00 00 00 80 FE 12 00 00 00 00 00 00 E0 FD 7E

***  0 called from $ADDRESS:00439969
                    ======== 00039969 == belongs to FunctionB(int)
    File d:\projects\mapaddr\faultapp.cpp Line 41
                             with 64 00 00 00 54 FF 12 00 00 00 00 00 00 E0 FD 7E CC CC CC CC

***  1 called from $ADDRESS:004399E5
                    ======== 000399E5 == belongs to FunctionA()
    File d:\projects\mapaddr\faultapp.cpp Line 51
                             with 00 00 00 00 00 00 00 00 00 E0 FD 7E CC CC CC CC CC CC CC CC

***  2 called from $ADDRESS:00439A68
                    ======== 00039A68 == belongs to main
    File d:\projects\mapaddr\faultapp.cpp Line 64
                             with 01 00 00 00 70 34 4D 00 B8 34 4D 00 84 5C FF C1 00 00 00 00

***  3 called from $ADDRESS:0044B573
                    ======== 0004B573 == belongs to __tmainCRTStartup
    File f:\dd\vctools\crt_bld\self_x86\crt\src\crt0.c Line 327
                             with

***  4 called from $ADDRESS:0044B32D
                    ======== 0004B32D == belongs to mainCRTStartup
    File f:\dd\vctools\crt_bld\self_x86\crt\src\crt0.c Line 195
                             with 00 00 00 00 00 00 00 00 00 E0 FD 7E 05 00 00 C0 C8 FF 12 00

***  5 called from $ADDRESS:7D4E7D42
         with

***  6 called from $ADDRESS:00000000
*************************************************************
```

Figure 1

Other Post-Mortem Techniques: Windows Mini Dumps

Minidumps store the current state of a program in a binary file that can be loaded into Visual Studio or WinDbg. When loaded into a debugger, a minidump shows the full symbolized stack and all local variables just as it would during a debug session. That is, of course, more comfortable than symbolizing a logfile.

Minidumps can be created a number of ways:

- Using Dr.Watson (http://support.microsoft.com/kb/275481/en-us) to be called on a program fault.
- Calling it explicitly from within the program itself (http://www.debuginfo.com/articles/effminidumps.html); i.e., by a installed Structured Exception handler, as in the approach described in the first issue of this article.
- Using a tool such as WinDbg (http://www.windbg.org/) or MiniDumpTool (http://bill.atwill.com/).

The disadvantages of minidumps are that they are Windows specific and they cannot be batch processed (as possible using a utility such as MapAdrr.exe).

Other Post-Mortem Techniques: Windows Error Reporting

You probably have seen a dialog similar to Figure 2, which was introduced with Windows XP.

To tell the truth, I was surprised to learn that Microsoft will send error reports even for programs that I have written but never published. However, I realized that I could use this mechanism for my personal applications. Windows Error Reporting (WER; see http://www.microsoft.com/whdc/winlogo/maintain/StartWER.mspx) is a service offered by Microsoft basically for free. To sign up you need a VeriSign certificate. With the certificate you can sign up for WER and obtain the error reports that users have accepted to upload using the dialog box (see Figure 2). These reports are just minidumps as described.

Some considerations about WER:

- Applications that have installed an exception handler using *SetUnhandledExceptionFilter* (as used in the first article) sometimes crash without calling the installed exception filter. In such cases, WER still can be used, thus giving an additional last chance to find bugs.

Exclusive Downloads!
Systems Content Resource Bundle

Click here to download the Systems Content Resource Bundle!

Compliments of IBM

Powered by Dr.Dobb's

- WER does not offer the ability to add any additional data or description by users. When using your own mechanism for error reporting (possibly a small separate utility) it may ask users to report information about how to reproduce the problem as well as sending additional data files to reproduce the bug. Such a utility also can be called when other problems occur, and will not lead to a crash.

It seems best to use WER as an additional service to report problems. Note also that there have been a number of reports about problems signing up for WER, especially when using a certificate from a provider other than VeriSign.

Other Post-Mortem Techniques: Google Breakpad

According to Google, breakpad "is a library and tool suite that allows you to distribute an application to users with compiler-provided debugging information removed, record crashes in compact "minidump" files, send them back to your server, and produce C and C++ stack traces from these minidumps (http://code.google.com/p/google-breakpad/). Breakpad can also write minidumps on request for programs that have not crashed. Breakpad is currently in use by Google Chrome, Firefox, Google Picasa, Camino, Google Earth, and other projects as well."

This is basically the same mechanism as propagated by MapAddr in my first article, but of course Google has enhanced the functionality:

- Breakpad works on multiple platforms (Windows, Linux, Mac). The minidump format has been adapted on other platforms.
- It includes functions to transmit a crash report directly to your HTTP server.
- It can also handle other errors such as C++ pure virtual function calls or invalid parameter calls and it can be triggered explicitly to create a crash report.

Breakpad seems to be well developed and can probably handle a number of conflicts that a simple utility like MapAddr cannot (i.e., loading DLLs to a different base address than the preferred

Figure 2

one). Unfortunately, its documentation is only fragmented. Even the documentation about installing and compiling is not correct. To compile breakpad for Windows, you first must install Python, then run the script:

```
src\tools\gyp\gyp.bat src\client\windows\breakpad_client.gyp
```

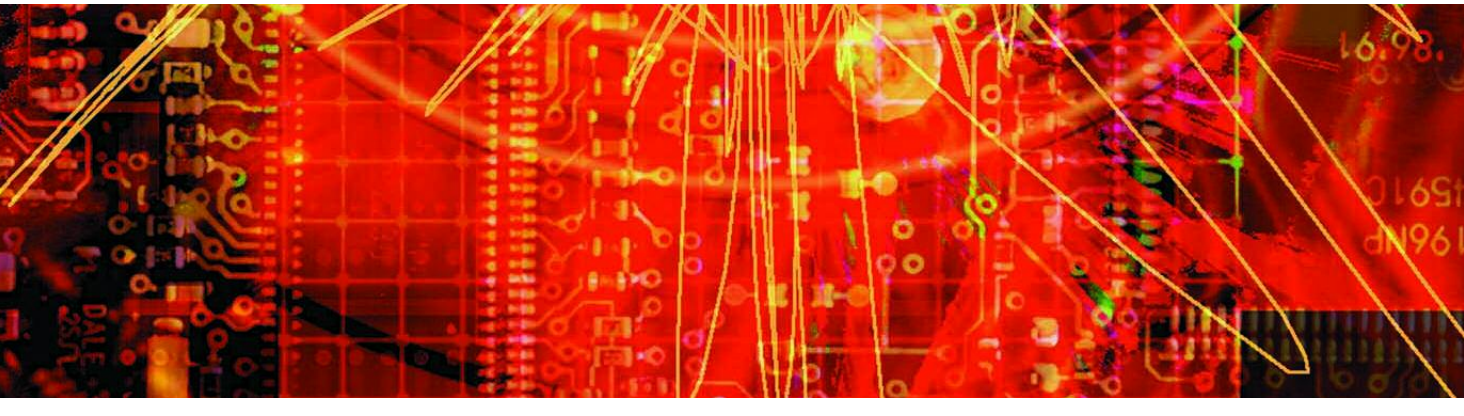
while the current directory is the root directory of breakpad. This will create Visual Studio Solution and project files; for other platforms, make files are provided.

Summary

There are a number of ways to capture stack dumps from program crashes and add symbol information to them. Such information helps to resolve bugs and to improve the quality of a program. Post-mortem debugging saves time in finding bugs, as a stack dump often is sufficient to reproduce and solve a bug. The effort needed to acquire this information is minimal.

— Stefan is founder of www.RED-SOFT-ADAIR.com, a company that rescues software development projects in a technical crisis. He can be contacted at StefanWoe@gmail.com.

Return to Table of Contents



BlackBerry DevCon 2010: A Developer's Perspective

A peek at the BlackBerry playbook

By John M. Wargo

At this year's BlackBerry Developer Conference 2010 (http://www.blackberry-devcon.com/), Research In Motion (RIM) announced a new class of mobile hardware and some tools to help both consumer and enterprise developers enhance their applications. In this article I highlight the announcements made at the conference and show how they'll affect everyday mobile developers.

The BlackBerry Playbook

There was a lot of excitement this year as developer's prepared for DevCon 2010. Beginning a week before the conference, pundits around the world announced that RIM would be unveiling their new tablet device, what everyone was calling the "BlackBerry BlackPad".

RIM took a different approach this year with the conference, scheduling morning sessions on the first day of the conference, then holding the Opening Keynote in the afternoon. RIM Co-CEO Mike Lazaridis gave at hint at the beginning of the session by talking about an upcoming special announcement while holding his hands like he was holding a tablet PC. He talked a bit about how BlackBerry was doing in the market, then finally got around to the big announcement. They launched the device by showing a video that highlighted the capabilities of the device then finally gave us the name — the BlackBerry Playbook.

It's smaller than I expected; a 7-inch screen in letterbox format (1024x600) and only 0.4 inches thick with dual 1GHz processors and 1 GB of memory. It runs the QNX operating system that RIM acquired through its acquisition of the QNX Software earlier this year; it's an operating system that's widely used in network hardware, automobiles, and too many other places to list here. It's a POSIX-based system, but will run BlackBerry applications (through a Java virtualization layer RIM will create for it). It's designed to deliver the full web experience, utilizing Apple's WebKit rendering engine and including support for HTML 5 and Flash 10.1. RIM worked very closely with Adobe to create the software for the device, and their constant reminders that Apple doesn't allow the full web experience (by blocking Adobe Flash) were humorous.

It's a companion device, designed to work alone or in conjunction with a BlackBerry handheld device. It has Wi-Fi capabilities (supporting 801.11 a/b/g/n), so it can stand alone when near a Wi-Fi hotspot. When on the go, away from any Wi-Fi networks, it will leverage its Bluetooth connection to a BlackBerry device and be able to pull email, calendar, contacts, and more directly from the BlackBerry (rather than requiring that the Playbook retrieve the same data from a server). There's an assumption that the Playbook can use the companion BlackBerry device's cellular connection to connect to remote data sources, but very little was said about this capability. US Carriers haven't said anything either about whether a Playbook will require a tethering plan in order to use the BlackBerry device's connection, but I assume they will if they can.

From a developer's standpoint, RIM did not provide a lot of detail. Expect that RIM's existing development tools will be updated to include support for the Playbook. Since it's a completely different hardware architecture and operating system, it's likely you'll need to learn a new set of APIs to build applications for this device (unless you leverage the BlackBerry VM provided). This ultimately is a good thing since RIM has been getting hammered in the market recently for the perceived limitations on its devices. This switch to a new operating system (RIM is unclear on this one, but it's also likely future BlackBerry devices will run this same OS) allows RIM to do some interesting things and streamline the APIs available to developers.

For now, when you build applications for the device, you'll be building them in HTML or Flash. Expect RIM to release a suite of development tools for the platform soon after it's released. Members of RIM's ISV Alliance program will be able to get their hands on the new tools as soon as they're available.

BlackBerry Messenger Social Platform

For developers creating applications that only target BlackBerry users, RIM announced the BlackBerry Messenger Social Platform (BBMSP). BlackBerry Messenger is essentially a free instant messaging platform that leverages RIM's BlackBerry infrastructure to allow BlackBerry devices to send messages between themselves. BlackBerry users have always had the ability to send PIN to PIN messages to other BlackBerry devices (even across carriers). BlackBerry Messenger is essentially a chat and contact management application built on top of PIN to PIN messaging.

With the BlackBerry Messenger Social Platform, RIM is providing an additional suite of APIs developers can use to leverage BlackBerry Messenger capabilities/services into their applications. This includes chat, content sharing and invitations, contacts, and more. One example that RIM demonstrated showed how the developers of the Kobo eReader used these new APIs to enhance the capabilities provided by their program. When reading or learning about a book, these features let users suggest a book to their friends or even chat among themselves about a particular book directly from within the eReader application. The ability to recommend and chat is already available through other applications such as Facebook, Twitter, and Google Talk for example, but this provides an easy way to provide the same capabilities within a custom application, increasing customer loyalty, saving time for users, and allowing these types of interactions to remain within the context of the application.

The BlackBerry Messenger Social Platform is expected to be available in Spring, 2011. RIM believes BlackBerry Messenger is an important piece of technology, but while providing interesting capabilities, this service is limited by the fact that it only works on the BlackBerry platform. While BlackBerry has become very popular with consumers, it's not likely that every member of a group will all be carrying BlackBerry devices. So, to build in the types of features provided by the BlackBerry Messenger Social Platform into your application, the APIs will make it easy to do on a BlackBerry, but will require developers to handcraft a similar solution for their application running on other mobile platforms.

BlackBerry Advertising Service

Announced at the 2009 conference and finally available today is the BlackBerry Advertising Service (BBAS). This service gives BlackBerry developers with a single suite of APIs to use to include advertisements from multiple ad sources into their applications while keeping 60 percent of the revenue generated by those ads. RIM has already built relationships with many advertising networks including Amobee, Jumptap, Lat49, Millennial Media, and Mojiva and expects to continue to add networks over time. They even went so far as to announce networks they haven't yet solidified relationships with including Buzzcity, NavTeq, Placecast, Sympatico, Transpera, Where, xAD, and more.

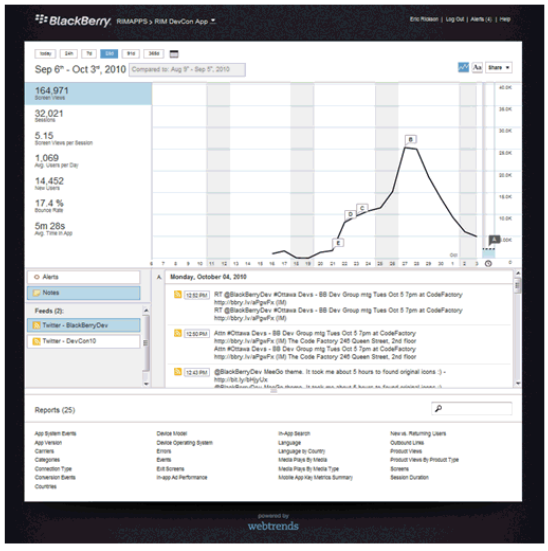


Figure 1: BlackBerry Analytics Service Console.

As RIM describes it, the service consists of three components: a suite of APIs a developer would use to leverage the service, a mediation platform (what RIM calls iys web console for managing ad allocations, analytics, tracking, reporting, and more) and the relationships RIM has created with the multitude of ad vendors the service works with. Using this service, developers will be able to serve up various types of ads and integrate them with their applications different ways and with other applications on the device as well (adding an event to the user's calendar or adding a restaurant's contact information to the address book). As with the BlackBerry Messenger Social Platform APIs, the BlackBerry Advertising Service only helps a developer working on a BlackBerry application. If the application under development targets multiple mobile platforms, the developer will be able to use the BlackBerry Advertising Service for the BlackBerry version of the application but will have to look at other services (iAd for iOS for example) for other targeted platforms.

BlackBerry Analytics Service

Another surprise announcement made by RIM at the conference is that they worked with Webtrends to deliver free, enterprise class analytics for BlackBerry applications. With the increased popularity of BlackBerry applications and the tremendous growth of the BlackBerry App World, RIM recognized that application vendors needed an easy way to tell how much their applications were being used and which features/screens were the most popular.

A developer implements the service by implementing calls to the analytics service from within their applications. Once an application has been instrumented and is out in the wild, developers can view metrics for their application from a web console similar to the one shown in Figure 1. The service is expected to be available in early 2011.

The WebWorks Application Platform

One of the most interesting application technologies RIM created recently was BlackBerry Widgets. They announced the technology right about the time they announced the end of life for BlackBerry MDS Runtime (essentially a drag-and-drop, point-and-click application builder for BlackBerry) and Widgets was basically seen as its replacement. At this year's DevCon, RIM renamed the tools for creating Widgets to "BlackBerry WebWorks Platform".

With the WebWorks tools, developers build web applications that run within a Java application container; the architecture for the application is as in Figure 2. The application's code (HTML, CSS, JavaScript) plus additional resources (images, videos, and more) are all packaged into a Java application (essentially an app with a single screen encompassing a big browser field). RIM provides special JavaScript libraries a developer can use to interact with device hardware such as the camera and other BlackBerry applications such as the calendar, address book, and email client. If a particular API is not available or if there's custom code that needs to be executed in an application, developers can create JavaScript extensions (written in Java) that can be called from any WebWorks application. WebWorks applications can even be configured as Push listeners, allowing back-end applications to push data into a WebWorks application rather than requiring that the client-side application request updates from the server. WebWorks applications support GPS, text messaging, background processing, interacting with other applications, and much more — pretty much anything you can do in a BlackBerry Java application you can do in a WebWorks application.

The benefits of WebWorks applications are:

- Developers can build BlackBerry applications using technologies they are already comfortable with rather than learning Java and the BlackBerry Java APIs.
- Applications built on this platform deploy like Java applications, mostly because they are Java applications, through the BlackBerry App World, using Over the Air (OTA) Push or via OTA Pull — you can't do that today with standard web applications without using third-party tools.
- WebWorks applications leverage the same security model available to Java applications, so enterprise administrators, through the BlackBerry Enterprise Server (BES), can finely tune what the applications can and cannot do on a device.
- Because the web application is running on-device, within the container, the application can operate when network connectivity is not available and can leverage a local SQLite database for local storage.

It's pretty amazing what you can do with these applications; at the conference, RIM demonstrated many WebWorks applications and you'd be hard pressed to tell the difference between them and native Java applications. They're web applications, so through graphics and CSS plus the available interface APIs, you can essentially make the applications look and act any way you want. RIM

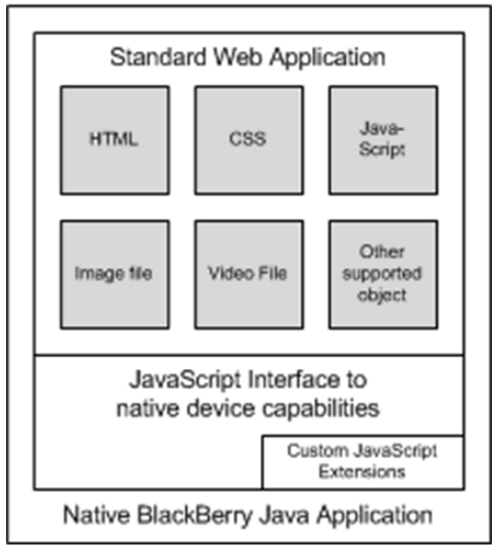


Figure 2: BlackBerry WebWorks Application Architecture.

provides thousands of JavaScript APIs and continues to add more. At this time, most essential BlackBerry Java APIs are available to WebWorks applications through JavaScript.

The platform consists of a plug-in for either Eclipse or Microsoft Visual Studio; it allows developers to write, test, and debug applications all from within the IDE. The BlackBerry simulators plug into the IDE as well, so when you're testing an application, you're doing it right alongside the IDE (with real-time debugging). The core of the platform is a packaging utility that takes a web application's files and packages them up into a Java application. This can be accomplished via a command-line tool or directly within the IDE. If you're looking for something to compare WebWorks to, think of PhoneGap or Appcelerator Titanium.

By leveraging the WebKit rendering engine, it looks like RIM is putting a lot of effort behind this newly renamed platform. I expect that because it's so easy to build these applications, you'll find most BlackBerry development shifting over to this platform. There will still be a place for building BlackBerry Java applications, but only for high performance, very complicated applications or hardware device drivers that can't be built any other way.

There has been a lot of pressure on RIM to release its development tools using an open source license; considering how often they get sued, it's been a really hard issue for them to address. With WebWorks, RIM is making its first big contribution to the Open Source community. RIM has released the full source code for WebWorks to GitHub (http://github.com/blackberry/WebWorks). RIM expects developers to take Java APIs that haven't been exposed through the WebWorks JavaScript libraries and publish their own libraries that leverage them. Expect RIM to make additional open source contributions over time.

BlackBerry Enterprise Application Middleware (BEAM)

So far, everything RIM announce at the conference was consumer focused. Granted you can use WebWorks to create enterprise

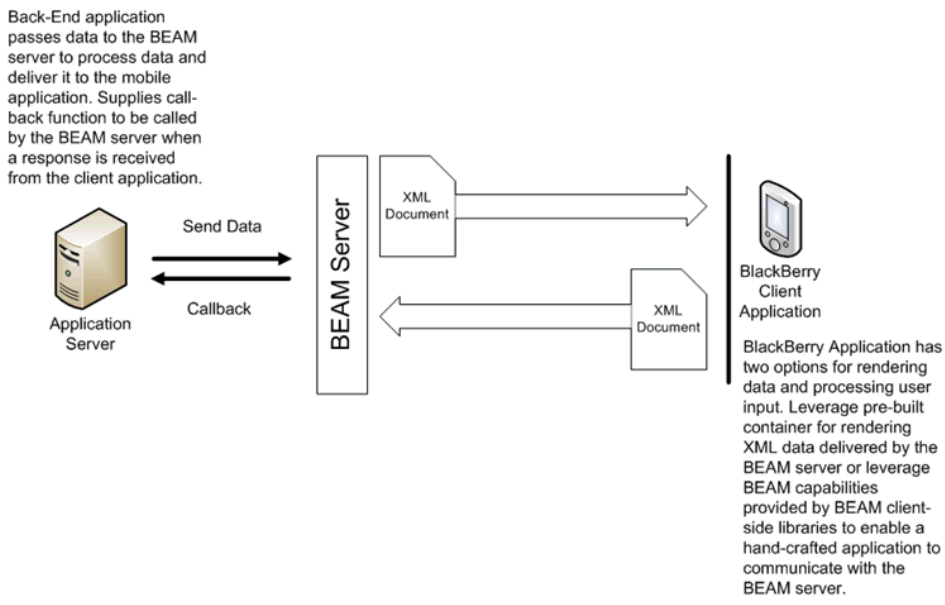


Figure 3: The BlackBerry Enterprise Application Middleware Platform Architecture.

applications, but most development done today on the BlackBerry platform is for consumer applications. Even the sample WebWorks applications were consumer applications. So, what about the enterprise?

To make it easier for enterprises to create applications for BlackBerry devices, RIM announced the BlackBerry Enterprise Application Middleware (BEAM) platform. It really wasn't clear from the announcement what RIM was announcing here, but I figured out how it works and thought I'd share it with you.

If you think about how you would build a mobile application that communicates with back-end data sources, your application would be responsible for connecting to the server to request the data then processing the results. Because the application is running on a mobile device that could have intermittent network connectivity, your application would need to build in the necessary logic to detect network availability (usually through a listener interface) and queue requests if needed. As a special bonus, it would be good if the back-end server could queue up responses as well, detecting that the requesting device is not available and delivering results to the device once the device reappears. Rather than forcing your application to periodically query the server for updates, you could also build your back-end system to leverage the BlackBerry platform's push capabilities to push new data when it becomes available rather than making the device application come get it. How does BEAM fit in here? Well, BEAM essentially does a lot of that stuff for you automatically. The purpose of the platform is to isolate the developer from dealing with many of the issues I just described.

BEAM is a middleware gateway that sits inside of an organization's firewall and processes requests made by a back-end application then delivers data to a device application (as shown in Figure 3). An application uses a published interface (in the form of RESTful web services) to deliver data to the BEAM server, which in turn delivers it over the air to the mobile application in the form of an XML document. When the device application returns data to the server, an XML document containing the data is sent to the BEAM server, which then calls a callback function to return the data to the calling program. In this model, application data is pushed to devices (whether they're connected to a BES or using the BlackBerry Internet Service (BIS)) through the same mechanism. All of the work a developer would need to do to setup and manage the push process is all provided by the platform.

Client-side capabilities are provided by a suite of libraries a developer can use in one of two ways. At the simplest level, the client-side libraries can be used as a simple runtime container that renders the enterprise data in what's essentially a CRUD (create, read, update, delete) application. By leveraging this approach, the developer has little control over the functionality provided by the application, but at the same time gets a very simple and quick-to-market solution to use for the client application. If the application needs more than the simple runtime container can provide, the developer can hand craft an application that uses the libraries to provide connectivity to the BEAM server and beyond then use the standard BlackBerry Java APIs to provide whatever UI and additional functionality is needed in the application.

Macintosh Support

Since most mobile developers develop for multiple mobile device platforms and an Apple Macintosh computer is required to develop for an iPhone or iPad, mobile developers pretty much need to

have a Macintosh. A limiting factor for BlackBerry developers has been that the BlackBerry development tools will only run in Windows (32-bit windows even for some of the tools or simulators) and therefore you would either need two computers (one running Windows and another running Mac OS) or have some sort of Windows virtualization running on a Macintosh.

At last year's conference, RIM announced that it would be working to enable some of its tools to run on a Macintosh and this year RIM delivered. During his presentation this year, Chris Smith (a senior director in RIM's development organization) casually pointed out that the demo was being performed on a Mac Book. RIM is delivering on the promise to allow the BlackBerry development tools to run on a Macintosh. The tools are currently available as a tech preview, so you can download them at http://na.blackberry.com/eng/developers/javaappdev/macosx.jsp and play around with them as RIM puts on the finishing touches. It's implemented as an Eclipse plug-in, just like the same tools on Windows, although the tech preview does not integrate with any BlackBerry device simulators at this time. To test/debug Java applications on a Macintosh, you'll need to cable a physical device to the IDE and use it that way. Even though the BlackBerry is a Java device, the BlackBerry simulators are currently delivered as Windows executables. It's not clear whether RIM will make the device simulators available on the Macintosh in a future release of the tools or whether you'll be stuck using physical hardware for your testing. I imagine they will port the simulators over, but likely only future devices will be converted to run as Java applications since porting all of them may not be worth the effort.

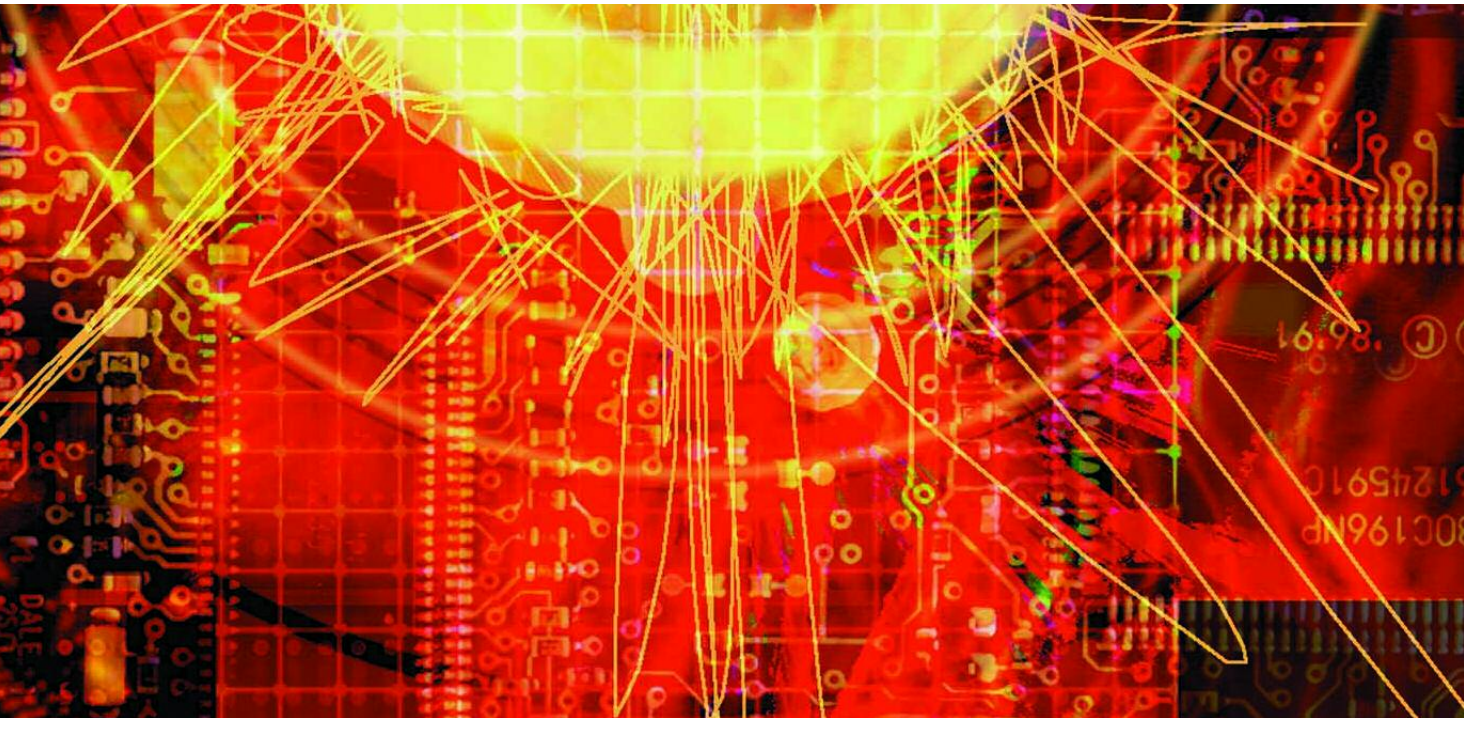
RIM really didn't say when it expects the final product to be released. One of the things I noticed when I worked at RIM was that just getting a product passed through legal sometimes took a year or more to complete. While other vendors release new tools with every new version of a product, RIM developers have to wait at least an additional 6 months or more before production-ready versions of RIM's tools become available. Once the production version is finished, it will likely be a while before you'll actually see them in the wild.

BlackBerry UI Designer

At last year's conference, RIM demonstrated a preview version of the BlackBerry UI Designer. Implemented as an Eclipse plug-in, the UI Designer allows BlackBerry developers to paint their application user interfaces just like they can on Android and the iPhone. The UI Designer was released as a tech preview and developers around the world have been waiting for the production release. I fully expected the product to be released at this year's conference, but apparently there's a problem with the product. It's done, the folks at RIM tell me, but Google acquired the company they worked with to create the tools and now everyone is waiting for legal issues to be resolved before they release the product. Maybe RIM will have them all worked out by next year's conference.

— John is the author of BlackBerry Development Fundamentals *(http://www.bbdevfundamentals.com).*

Return to Table of Contents



Developing a Silverlight UI for Windows Phone 7

Silverlight for Windows Phone, the application development platform for Windows Phone 7 Series

By Gastón Hillar

Silverlight for Windows Phone, the application development platform for Windows Phone 7 Series, supports core Silverlight features while providing access to the phone's unique capabilities through managed .NET code. Developers with some experience in XAML and managed .NET code will be able to use the new developer tools to create Windows Phone 7 Series applications. In this article, I show how to build a UI from scratch using the new development tools, SDKs, and the mobile device emulator, assuming that you are familiar with Silverlight and Visual Studio 2010 basics. First, you must download and run the Silverlight for Windows Phone Developer Tools installer. These Developer Tools are available in their Beta version, and therefore, it is convenient to download the installer by clicking the Download button in the Silverlight for Windows Phone page at http://www.silverlight.net/getstarted/devices/windows-phone/. You will always find the latest version available in the link included in this page.

Once you finish the installation process, you will notice a new folder in Start -> Programs, Microsoft Visual Studio 2010 Express. If you had any version of Visual Studio 2010 installed, you don't need to start Microsoft Visual Studio 2010 Express for Windows Phone, included in the Visual Studio 2010 Express folder. You can develop Silverlight for Windows Phone applications in your Visual Studio 2010 version that was installed before running the installer for the new tools.

Start Visual Studio 2010, select File -> New -> Project and the new "Silverlight for Windows Phone" item will appear within the Installed Templates list for Visual C#. Figure 1 shows the three types of projects that you can create targeting Silverlight for Windows Phone:

- Windows Phone Application. An application without navigation support.
- Windows Phone List Application. An application with navigation support.
- Windows Phone Class Library. A class library that you can use in other applications.

If you select "XNA Game Studio 4.0" within the Installed Templates list for Visual C#, you will notice that there are new project types related to Windows Phone. In fact, Silverlight for Windows Phone can use the XNA Framework features for audio capture and playback, access the media library, and Xbox LIVE.

Introducing Silverlight for Windows Phone

When you create a new Windows Phone application, the new solution includes the Silverlight

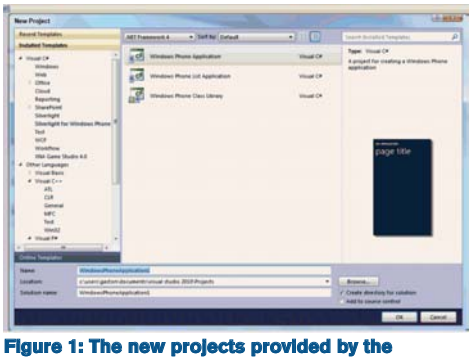


Figure 1: The new projects provided by the Silverlight for Windows Phone template.

MainPage.xaml page. However, the design view for this page shows a preview of the controls that compose the user interface within a Windows Phone 7 screen, as in Figure 2. You can drag and drop controls to the design surface and check the layout preview for a Windows Phone 7 screen. As happens when working with Silverlight in Visual Studio 2010, if you make changes to the XAML code, these changes will be reflected on the design surface that emulates a Windows Phone 7 screen.

According to the information provided in the design resources, all Windows Phone 7 devices will have WVGA screens at 800 x 480 pixel resolution, no matter the screen size. This single resolution makes it simpler to design the UI. The great drawback is that you cannot emulate multitouch gestures with a mouse or a touchpad in your developer workstation.

However, if you don't have a multitouch monitor, you still have a chance to test some multitouch gestures without having to deploy the project to the phone. There is an interesting project on CodePlex, Multi-Touch Vista (http://multitouchvista.codeplex.com), that allows you to work with multiple mice to emulate two fingers on the screen and their multitouch gestures. In fact, the latest version of Multi-Touch Vista provides a Windows 7 compatible driver that enables multiple mice and is compatible with the Windows Phone 7 emulator. For example, you can use a laptop's touchpad as one of the pointers and a USB mouse connected to the same laptop as the second pointer. (If you're interested in working with Multi-Touch Vista, you can read an excellent step-by-step tutorial written by Michael Sync at http://michaelsync.net/2010/04/06/step-by-step-tutorial-installing-multi-touch-simulator-for-silverlight-phone-7. This tutorial explains how to install and configure the driver to work with the Windows Phone 7 emulator.)

The structure of a Windows Phone application project is similar to a classic Silverlight application project. However, Windows Phone adds some proprietary references and files. Figure 3 shows the project structure for a sample WPBusinessApp project. This is the default structure that Visual Studio 2010 creates for a Windows Phone application.

You will notice the WMAppManifest.xml file within the Properties folder. This XML file defines the resources and capabilities of the application from the point of view of the operating system. The following code snippet shows the default contents for this file, considering that the project's name is *WPBusinessApp*:

```
<?xml version="1.0" encoding="utf-8"?>
<Deployment
xmlns="http://schemas.microsoft.com/windowsphone/2009/deployment"
AppPlatformVersion="7.0">
<App xmlns="" ProductID="{41bdea14-687b-4815-93b3-a51759f18a09}"
Title="WPBusinessApp" RuntimeType="Silverlight" Version="1.0.0.0"
Genre="apps.normal" Author="WPBusinessApp author"
Description="Sample description" Publisher="WPBusinessApp">
<IconPath IsRelative="true"
IsResource="false">ApplicationIcon.png</IconPath>
<Capabilities>
<Capability Name="ID_CAP_NETWORKING" />
<Capability Name="ID_CAP_LOCATION" />
<Capability Name="ID_CAP_SENSORS" />
<Capability Name="ID_CAP_MICROPHONE" />
<Capability Name="ID_CAP_MEDIALIB" />
<Capability Name="ID_CAP_GAMERSERVICES" />
<Capability Name="ID_CAP_PHONEDIALER" />
<Capability Name="ID_CAP_PUSH_NOTIFICATION" />
<Capability Name="ID_CAP_WEBBROWSERCOMPONENT" />
</Capabilities>
<Tasks>
<DefaultTask Name ="_default"
NavigationPage="MainPage.xaml"/>
</Tasks>
<Tokens>
<PrimaryToken TokenID="WPBusinessAppToken"
TaskName="_default">
<TemplateType5>
<BackgroundImageURI IsRelative="true"
IsResource="false">Background.png</BackgroundImageURI>
<Count>0</Count>
<Title>WPBusinessApp</Title>
</TemplateType5>
</PrimaryToken>
</Tokens>
</App>
</Deployment>
```

The WMAppManifest.xml file includes the list of capabilities that the application needs from the phone in *<Capabilities>*. By

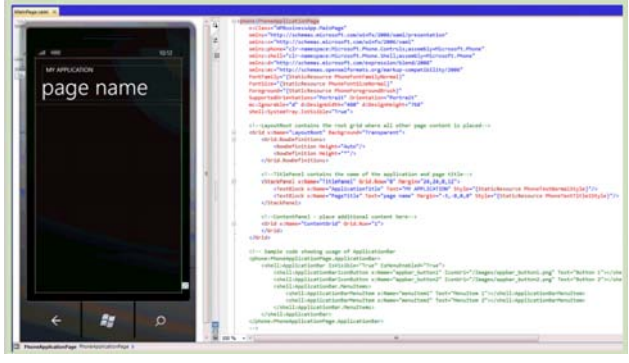


Figure 2: Design view for MainPage.xaml in Visual Studio 2010 and its XAML code on the right side.

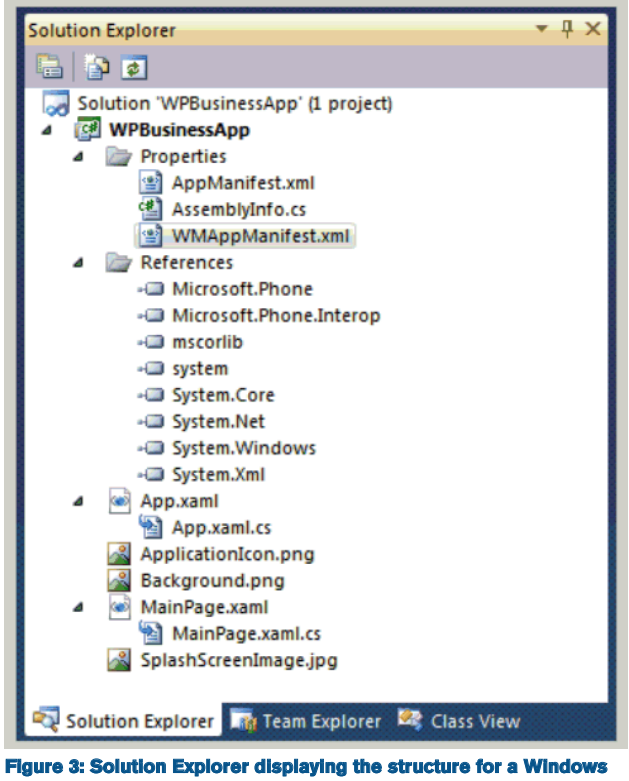


Figure 3: Solution Explorer displaying the structure for a Windows Phone Application.

default, the manifest requests all the capabilities. However, you should remove the unnecessary ones in order to create a more secure application. If you try to use a feature that corresponds to a capability that the application doesn't request in the manifest, you will get an *UnauthorizedAccessException* exception. If the application is installed, it will have the capabilities that it requests.

The names for the capabilities are very easy to understand. For example, *ID_CAP_LOCATION* means that you need to access location services included in the *System.Device.Location* namespace. However, some capabilities are related to many namespaces and classes. Jaime Rodriguez wrote a very interesting post about the Windows Phone capabilities security model that includes detailed information about the namespaces related to each capability at http://blogs.msdn.com/b/jaimer/archive/2010/04/30/windows-phone-capabilities-security-model.aspx.

The WMAppManifest.xml file also defines the icon, the background image, and the title for the application. Luckily, you can specify icons as PNG bitmap files. The default icon is *ApplicationIcon.png* and the default background is *Backgroud.png*. You can edit XML code or you can change the values for these properties in the Application page in the project's properties, as in Figure 4.

By default, the project includes two Windows Phone related references, *Microsoft.Phone* and *Microsoft.Phone.Interop*. *Microsoft.Phone* provides access to *Microsoft.Phone.Controls* and *Microsoft.Phone.Shell*. If you have to work with sensors, you must add *Microsoft.Devices.Sensors* as a new reference.

Understanding the Code-Behind File for App.xaml and the Splash Screen

If you have some experience with Silverlight and C#, you will be familiar with App.xaml and its code-behind file, App.xaml.cs. The C# code adds some phone-specific initialization code. The *App* class provides easy access to the root frame with the public *RootFrame* property:

```
public PhoneApplicationFrame RootFrame { get; private set; }
```

PhoneApplicationFrame is *MicrosoftPhone.Controls.PhoneApplicationFrame*. *App.xaml.cs* uses the *Microsoft.Phone.Controls* and *Microsoft.Phone.Shell* namespaces.

The *App* class constructor calls the *InitializePhoneApplication* method, which adds phone-specific initialization code that displays a splash screen. This code snippet shows the code for this method with the classic Silverlight initialization and the new phone-specific line:

```
public App()
{
// Global handler for uncaught exceptions.
// Note that exceptions thrown by ApplicationBarItem.Click
// will not get caught here.
UnhandledException += Application_UnhandledException;
// Standard Silverlight initialization
InitializeComponent();
// Phone-specific initialization
InitializePhoneApplication();
}
```

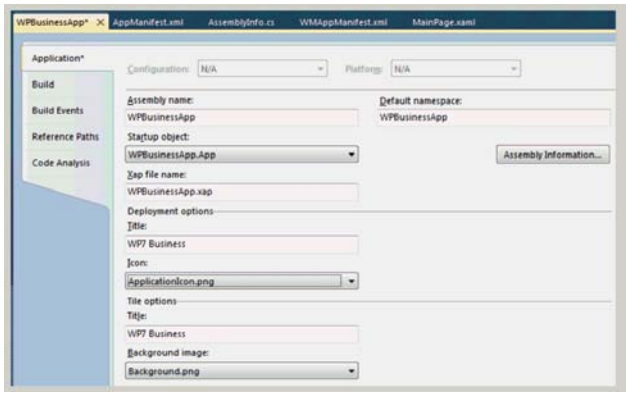


Figure 4: Project's properties displaying the Application page.

The following lines show the code for the *InitializePhoneApplication* and *CompleteInitializePhoneApplication* methods. The *InitializePhoneApplication* method creates the new frame and keeps the splash screen visible until the application is ready to render. The *CompleteInitializePhoneApplication* method sets the new frame as *RootVisual* because it is attached as the event handler for *RootFrame.Navigated*. Don't make changes to these methods.

```
private bool phoneApplicationInitialized = false;
// Do not add any additional code to this method
private void InitializePhoneApplication()
{
if (phoneApplicationInitialized)
return;
// Create the frame but don't set it as RootVisual yet;
// this allows the splash screen to remain active until the
// application is ready to render.
RootFrame = new PhoneApplicationFrame();
RootFrame.Navigated += CompleteInitializePhoneApplication;
// Handle navigation failures
RootFrame.NavigationFailed += RootFrame_NavigationFailed;
// Ensure we don't initialize again
phoneApplicationInitialized = true;
}
// Do not add any additional code to this method
private void CompleteInitializePhoneApplication(object sender,
NavigationEventArgs e)
{
// Set the root visual to allow the application to render
if (RootVisual != RootFrame)
RootVisual = RootFrame;
// Remove this handler since it is no longer needed
RootFrame.Navigated -= CompleteInitializePhoneApplication;
}
```

While the application is loading, the emulator and the device will show a splash screen. This splash screen is a 480 x 800 pixel resolution bitmap, *SplashScreenImage.jpg*, with a 24-bit color depth and included in the project. You can replace this JPEG bitmap with your desired splash screen but you have to make sure that you use the same name, set its Build Action to Content and Copy to Output Directory to Do Not Copy. Figures 5 and 6 show the default splash screen and a customized version. The splash screen bitmap must use the 24-bit color JPEG format to work properly. If you use a PNG file instead of a JPEG file, the splash screen won't be displayed. Remember that the user can rotate the device before designing the customized splash screen.

Understanding MainPage.xaml in a Windows Phone Application

The following code snippet shows the original code for MainPage.xaml. These lines shows the code that provides an example of the *ApplicationBar* buttons uncommented. By default, these lines appear commented, and therefore, you don't see the application bar buttons in the design view. If you uncommment the code that begins with *<phone:PhoneApplicationPage.ApplicationBar>*, you will see the application bar buttons. Figure 7 shows the document outline for MainPage.xaml. The document outline allows you to understand the different controls that compose the basic UI.

```
<phone:PhoneApplicationPage
x:Class="WPBusinessApp.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:phone="clrnamespace:
Microsoft.Phone.Controls;assembly=Microsoft.Phone"
xmlns:shell="clrnamespace:
Microsoft.Phone.Shell;assembly=Microsoft.Phone"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markupcompatibility/
2006"
FontFamily="{StaticResource PhoneFontFamilyNormal}"
FontSize="{StaticResource PhoneFontSizeNormal}"
Foreground="{StaticResource PhoneForegroundBrush}"
SupportedOrientations="Portrait" Orientation="Portrait"
mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="696"
shell:SystemTray.IsVisible="True">
<!—LayoutRoot contains the root grid where all other page
content is placed—>
<Grid x:Name="LayoutRoot" Background="Transparent">
```

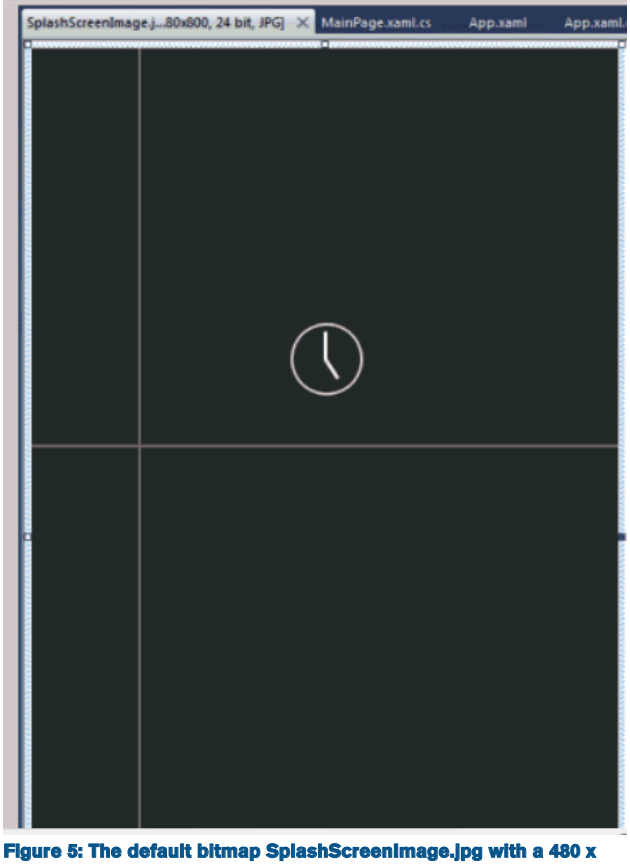


Figure 5: The default bitmap SplashScreenImage.jpg with a 480 x 800 pixel resolution.

```
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>
<!—TitlePanel contains the name of the application and
page title—>
<StackPanel x:Name="TitlePanel" Grid.Row="0"
Margin="24,24,0,12">
<TextBlock x:Name="ApplicationTitle" Text="MY
APPLICATION" Style="{StaticResource PhoneTextNormalStyle}"/>
<TextBlock x:Name="PageTitle" Text="page name"
Margin="-3,-8,0,0" Style="{StaticResource
PhoneTextTitle1Style}"/>
</StackPanel>
<!—ContentPanel - place additional content here—>
<Grid x:Name="ContentGrid" Grid.Row="1">
</Grid>
</Grid>
```

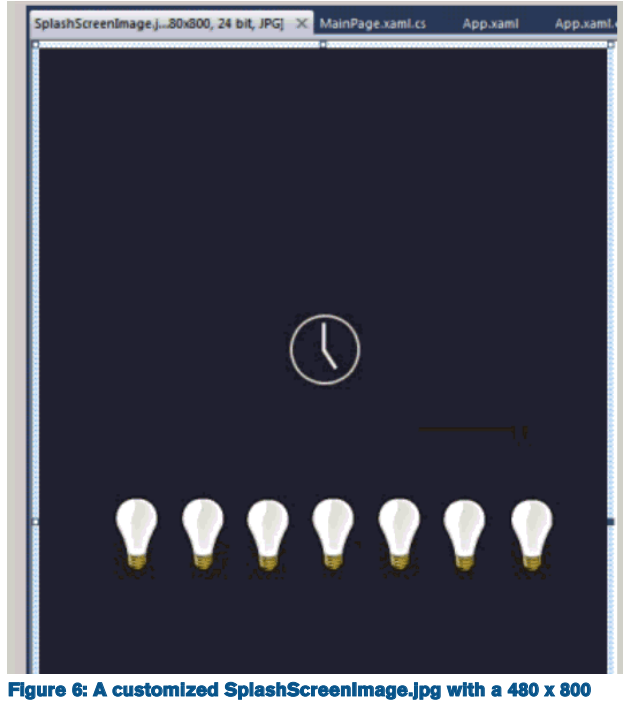


Figure 6: A customized SplashScreenImage.jpg with a 480 x 800 pixel resolution.

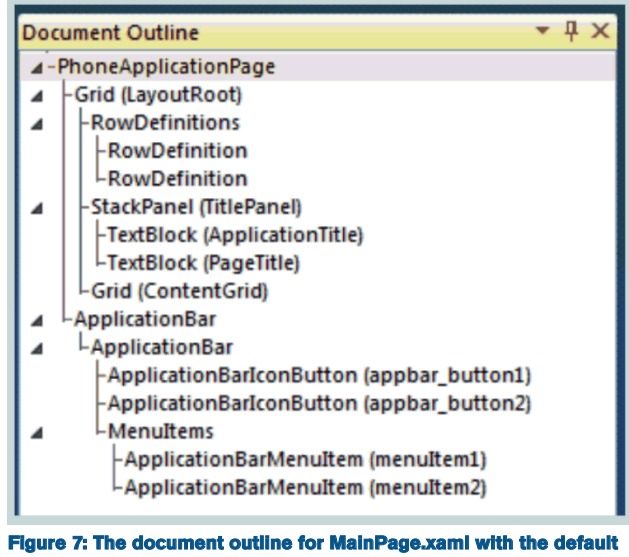


Figure 7: The document outline for MainPage.xaml with the default sample code that shows the usage of ApplicationBar.

```
<!— Sample code showing usage of ApplicationBar—>
<phone:PhoneApplicationPage.ApplicationBar>
<shell:ApplicationBar IsVisible="True"
IsMenuEnabled="True">
<shell:ApplicationBarIconButton
x:Name="appbar_button1" IconUri="/Images/appbar_button1.png"
Text="Button 1"></shell:ApplicationBarIconButton>
<shell:ApplicationBarIconButton
x:Name="appbar_button2" IconUri="/Images/appbar_button2.png"
Text="Button 2"></shell:ApplicationBarIconButton>
<shell:ApplicationBar.MenuItems>
<shell:ApplicationBarMenuItem
x:Name="menuItem1" Text="MenuItem
1"></shell:ApplicationBarMenuItem>
<shell:ApplicationBarMenuItem
x:Name="menuItem2" Text="MenuItem
2"></shell:ApplicationBarMenuItem>
</shell:ApplicationBar.MenuItems>
</shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>
<!— End of sample code —>
</phone:PhoneApplicationPage>
```

LayoutRoot is the root *Grid* within *PhoneApplicationPage*. All the page content is included in *LayoutRoot*. However, it is very important to notice that an *ApplicationBar*, with no specific name, is also part of *PhoneApplicationPage*, because the application bar is a special shell control. *TitlePanel* is a *StackPanel* with two *TextBlock* controls:

- *ApplicationTitle*. By default, its *Text* property is set to "MY APPLICATION". You can use it to display the desired title for your application.
- *PageTitle*. By default, its *Text* property is set to "page name." If your application has many pages, you can use this *TextBlock* to specify the actual page. However, if the application just needs one page with controls, this *TextBlock* can consume unnecessary space. If you delete this *TextBlock*, the *StackPanel*'s height will auto-adjust its value. Thus, you can remove *PageTitle* when you need more space to place controls.

ContentGrid is the *Grid* where you have to place the controls. Figure 8 shows a *TextBlock* and a *TextBox* within *ContentGrid*. Silverlight for Windows Phone supports theming, and therefore, each control is going to have a different look according to the theme selected by the user in his/her device.

By default, the Visual Studio 2010 Toolbox provides the most common Windows Phone controls. However, there are important controls that don't appear and you have to add them by selecting Choose items — in the context menu for the Toolbox. One example

Figure 8: The design view applies the default theme for the controls that you add.

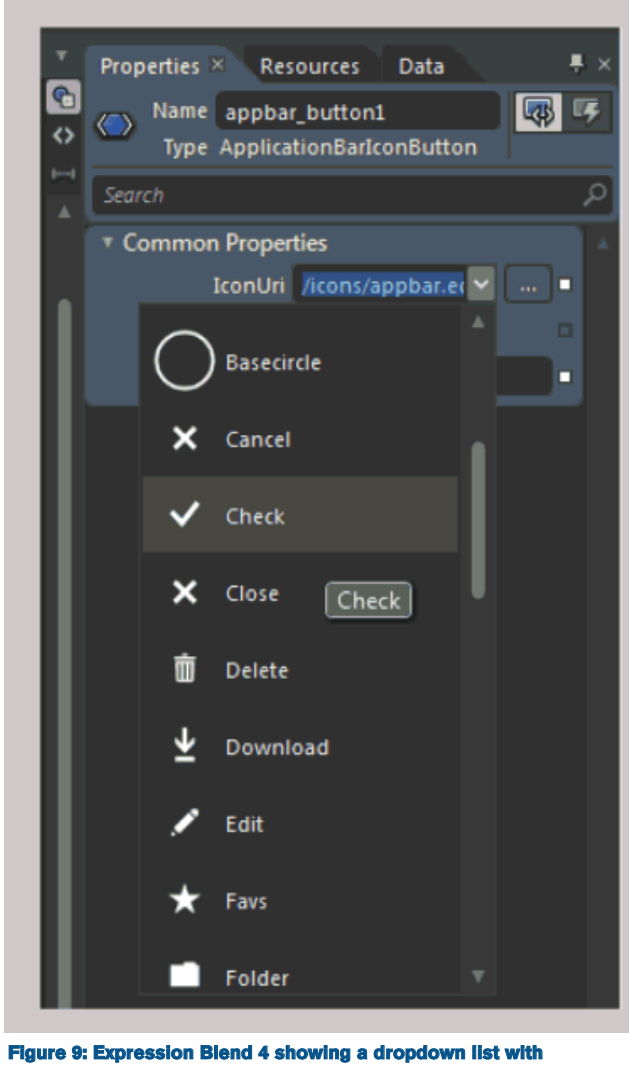


Figure 9: Expression Blend 4 showing a dropdown list with predefined icons for an ApplicationBarIconButton.

is the old *InkPresenter* control that is very useful to allow users to draw directly with their fingers.

You can use the 3D projections, introduced by Silverlight 3. However, if you don't want to write XAML code to add these projections, you will have to use Microsoft Expression Blend 4 for Windows Phone. In fact, if you have to create a complex UI, Expression Blend will simplify your work. Expression Blend also allows you to take advantage of the behaviors to simplify the creation of UI controls that respond to common multi-touch gestures.

Figure 10: Two ApplicationBarIconButton controls with their icons.

Figure 11: Windows Phone 7 emulator in action.

Expression Blend 4 provides a more accurate design view when you have to work with the *ApplicationBar*. You can select the desired icon for each button from a list of predefined icons, as in Figure 9.

The *ApplicationBar* is composed of many *ApplicationBarIconButton* controls. These icon buttons display a small icon within a circle, as in Figure 10. The *ApplicationBar* can also include *ApplicationBarMenuItem* controls.

You can attach a *Click* event handler for each of the *ApplicationBarIconButton* and the *ApplicationBarMenuItem* controls. Remember that the *ApplicationBar* control is optional.

When you run the project in Visual Studio 2010 or Expression Blend 4 for Windows Phone, the results of the build process will be deployed in the Windows Phone 7 emulator. The first time you run the application, the Windows Phone 7 emulator will require time to load. However, you don't need to close the emulator to enable another debugging session. It is convenient to leave the emulator running, make the necessary changes to the project, and run it again. If you close the emulator, you will need more time to run the project again. Figure 11 shows the emulator running a very simple UI.

If you click on the Start menu on the emulator, you will see the icon for Internet Explorer. You can access the menu with the icon for the new application by clicking on the next icon button (the arrow). Figure 12 shows the customized icon for a sample application. Figure 13 shows the customized splash screen in the emulator.

By default the project defines support for the portrait orientation in the *PhoneApplicationPage*. The following line indicates the values for *SupportedOrientations* and *Orientation*:

```
SupportedOrientations="PortraitOrLandscape"
Orientation="Landscape"
```

The location and size for the controls will vary according to the device orientation and the value for the *SupportedOrientations*

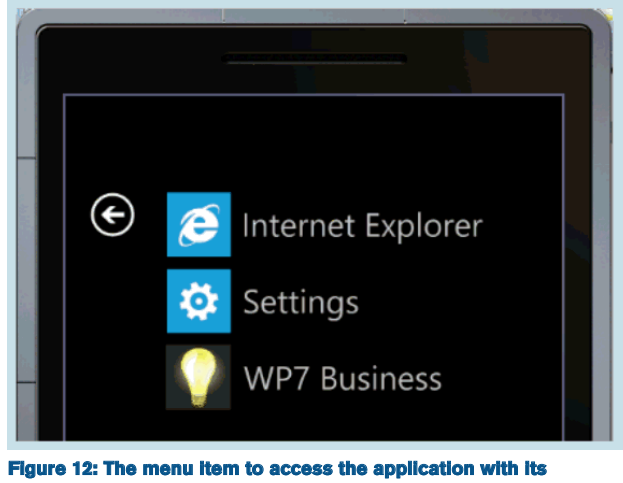


Figure 12: The menu item to access the application with its customized icon.

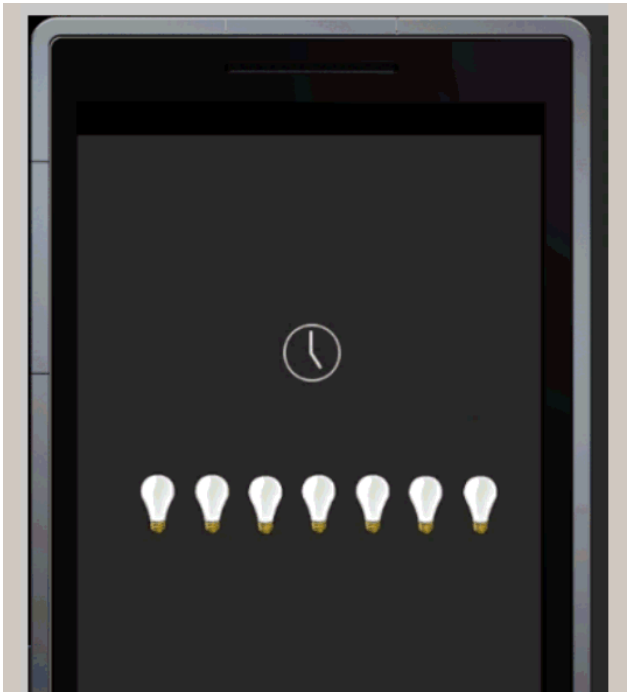


Figure 13: The customized splash screen that appears before the application launches in the middle of the animated transition.

property for the *PhoneApplicationPage*. If you want your application to support both portrait and landscape orientations, you have to specify "*PortraitOrLandscape*" for *SupportedOrientations*. Remember to test the different orientations with the emulator to avoid unexpected locations or sizes for the controls when the user rotates the device. Figure 14 shows an example of an application with a new orientation in the emulator.

Conclusion

Silverlight for Windows Phone 7 Series allows you to take full advantage of your Silverlight experience to create mobile applications. Visual Studio 2010 makes it simple to design the UI and debug new applications that target Windows Phone 7 Series and you can test and debug them with the help of the powerful emulator.

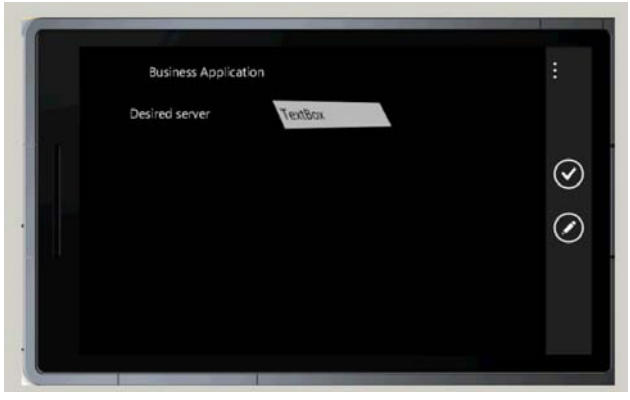


Figure 14: Windows Phone 7 emulator displaying an application with the device in landscape orientation.

Expression Blend 4 for Windows Phone allows you to reduce your development times when you need to create complex UIs. If you want to learn as fast as possible, you can work with Expression Blend in combination with Visual Studio 2010.

There are many new namespaces and you have to learn many new classes to interact with the phone capabilities. However, you still have to work with a combination of XAML and C#. Silverlight for Windows Phone 7 Series provides an excellent opportunity to create mobile, rich, and interactive user interfaces that interact with services on the cloud.

— Gastón Hillar is a frequent contributor to Dr. Dobb's and author of Professional Parallel Programming With C#: Master Parallel Extensions With .NET 4 (www.wiley.com/WileyCDA/WileyTitle/productCd-0470495995.html).

Return to Table of Contents

Q&A: DSL Has Its Risks, and Big Benefits

Domain-specific languages have a lot to offer

by Jonathan Erickson

Walid Taha is a professor of computer science at Sweden's Halmstad University, and an expert on domain-specific languages. He recently talked with *Dr. Dobb's* editor in chief Jonathan Erickson.

Dr. Dobb's: What kind of applications are well-served by a DSL?
Taha: Applications that you can build using tools such as Excel, AutoCAD, and Simulink. All of these are great examples of DSLs, although we don't always think of them as such.

Dr. Dobb's: What's the most exotic DSL you've seen?
Taha: Blender.org is way cool. A lot of kids seem to be using it to build all kinds of beautiful games and other pieces of art.

Dr. Dobb's: Do developers need special skills to build and use DSLs?
Taha: Developers need knowledge of or the ability to understand the domain, and a good understanding of what it takes to define and implement a language for the particular group of domain experts at hand.

Dr. Dobb's: Five or 10 years from now, won't unique, nonstandard DSLs be difficult to maintain, once the original developer is no longer around?
Taha: This is a valid concern, but it shouldn't be a deterrent from using DSL. This is, fundamentally, an economic concern and should be addressed as such. That also means that there are many aspects of that question that should be considered. For example, if the domain is small enough, and the software isn't free (as in open source), then sure, the implementation may ultimately not be maintainable. On the other hand, if the domain is large enough, and the software is free, there should be enough commercial interest to maintain and evolve the DSL.

Dr. Dobb's: Have DSL productivity gains for developers been proved, or is this just another sales pitch?
Taha: I would say neither. I don't know where we would be without Excel, and I know that a lot of really simple tasks would take me and millions of others much longer. Even though studies demonstrating productivity gains are rare, that doesn't mean that [the gains] aren't very real. In many cases, one sees DSLs that clearly let users get done in a few hours what would have normally taken several months (e.g., tools for simulation of systems). When the difference is huge, it often seems like taking the time to do a controlled study is of purely academic interest.

Return to Table of Contents

Jolt Product Excellence Awards: Develpment Environments

Visual Studio 2010

By Jonathan Erickson

Congratulations to Microsoft's Visual Studio 2010 (http://msdn.microsoft.com/en-us/vstudio/default.aspx), winner of this year's Jolt Product Excellence Award in the Development Environment category. Finally released earlier this year, Visual Studio 2010 has lived up to all of the hoopla surrounding its introduction. Not only does Visual Studio 2010 have lots of features, but they're features that are actually useful — and that work. Stuff like code contracts, static code analysis, improved Intellisense, multi-monitor support, tools for parallelism, built-in UML, the ability to multi-target applications, Silverlight 4 support, post-mortem debugging, support for F#, and more. Whew! Compiling a list of Visual Studio 2010 features sure accelerates the heart rate.

One reason Microsoft packed Visual Studio 2010 with all of these capabilities is that, instead of thinking of it as just another integrated development environment, Microsoft is pitching it as a full-fledged platform for developing all-things Windows — desktop, Web, mobile, enterprise, and everything in between. But being "full-fledged" requires tools and features, and lots of them.

The Jolt judges aren't alone in liking what they see with Visual Studio 2010. In the just completed Forrester-Dr. Dobb's Developer Technographics Survey, a study aimed at better understanding what's really going on in software development, we found that 52% of the 1000-plus developers polled rely upon Visual Studio 2010 as their primary IDE.

Visual Studio has come a long way since it was first rolled out with little fanfare and minimal features more than a decade ago. But the 1997 version of the IDE has little in common (other than the name) with Visual Studio 2010. It finally is the "full-fledged" IDE that it was set out to be.

Return to Table of Contents

The Linux Programming Interface Book Review

By Mike Riley

The Linux Programming Interface
by Michael Kerrisk
No Starch Press
$99.95 (Hardcover), $49.95 (Ebook)

The expanding role of Linux in our lives is undeniable. From the servers that run our favorite websites to the smartphones that keep us on the grid, Linux continues to make a major impact in the way we connect, transfer, and manipulate electronic data. However, few books have plumbed the depths of the heart of the Linux OS. Does this latest attempt by author and Linux hacker Michael Kerrisk satiate this market need? Read on to find out.

Coming in at just over 1,500 pages, *The Linux Programming Interface (TLPI)* reminds me of the Linux version of Charles Petzold's *Programming Windows* in terms of size, scope, and depth. Unlike *Programming Windows*, with the last published edition nearly 12 years old, *TLPI* is hot off the presses and intended for today's generation of computer software students and engineers. It also serves as a reminder of how this rapidly evolved OS is based on nearly 40-year-old OS design principles and languages. Knowing the C language is an obvious prerequisite to extract meaningful information from the book, as well as a desire to learn the inner workings of the Linux kernel.

The book begins with a brief history on the development of UNIX and Linux and the standards they employ, followed by a chapter on fundamental concepts (i.e., a breakdown of the kernel, which serves as an outline of the chapters to follow). System programming concepts (getting familiar with glib) are followed by two chapters on File I/O. Processes, memory allocation, users and groups, process credentials, and time round out the first 10 chapters. System limits, process information, file I/O buffering, file systems and attributes (and extended attributes), access control lists (ACLs), directories and links and monitoring file events are also explored. Several chapters on signals (fundamentals, handlers, advanced features), timers and sleeping, processes (creation, termination, monitoring), program execution and threads (synchronization, safety, cancellation, and more). The book continues with explanations of process groups, sessions, job control, priorities, scheduling, and managing process resources. Chapters on daemons, privileged programs, file and system capabilities, login accounting (the utmpx API), shared libraries, interprocess communication (IPC), pipes and FIFOs are then followed by several chapters on System V and topics (IPC APIs, message queues, semaphores, shared memory). More chapters follow on file locking, sockets (struct *sockaddr_un* and *socketpair()*, TCP/IP fundamentals, internet domains, TCP and UDP echo server designs, and advanced topics such as the netstat, tcpdump, *snedmsg()* and *recvmsg()* system calls), terminals and alternative I/O models (multiplexing, signal-driven, the epos API, and the *pselect()* system call for example). The book concludes its 64th chapter on UNIX 98 and BSD pseudoterminals. Six brief appendixes cover the strace system trace command, command-line parching, null pointer casting, kernel configuration, additional references, and solutions to selected exercises posed at the end of each chapter.

Before No Starch's *TLPI* came along, I had been recommending Wolfgang Mauerer's *Professional Kernel Architecture*, published two years ago by Wrox. *TLPI* benefits from having the latest Linux kernel to base its discussions on, as well as the 20+ years of UNIX development and teaching experience as well as the Linux kernel man-pages contributions that the author brings to the text. Still, the nearly one hundred dollar cover price will be a persistent barrier of entry for all but the most devoted students and developers of Linux kernel knowledge. There is no doubt after consuming *TLPI* that readers will be vastly more knowledgeable about the complexity underlying the Linux kernel, though for those merely interested so they can understand why and how certain popular system calls work the way they do, the Wrox title still holds a place on the bookshelf and is much easier on the wallet.

However, just as I was going to finish this review, the No Starch folks informed me that the electronic edition of *TLPI* is forthcoming and will be fifty dollars less than the hardcover print version. Of course, for those not yet equipped with a purpose-built electronic reading device or still prefer the physical ink on paper approach, No Starch is also offering a free version of the Ebook with the purchase of the hardcover edition from its online store.

Return to Table of Contents

The Neverending Awesomeness of Bash

By Mark Nelson

This semester, I'm teaching a class on Linux/UNIX and am enjoying it immensely. With every lecture, I'm reminded that you simply never stop finding new tools and tricks to use in an OS that is now well into middle age. One of my midterm questions from last week was a basic query regarding filename expansion under bash:

"Write a command to copy files chapt00.txt, chapt01.txt, through chapt15.txt from the current directory to your home directory. Make the command as short as possible."

The answer I was expecting was

```
cp chapt0[0-9].txt chapt1[0-5].txt ~
```

and that is indeed what I got from the majority of the students. Of course, a few were confused about the use of character classes and tried to get something like *chapt[00-15].txt* to work. But one student turned in something a bit more novel:

```
cp chapt{0[0-9],1[0-5]}.txt ~
```

My initial reaction was that this was simply a misguided attempt at filename expansion using broken syntax. But I was assured that this worked properly, and a quick look at the bash reference manual showed me the error of my ways. It turns out I was completely overlooking a nice feature of the shell: **brace expansion**. When the shell encounters a sequence of the format *preamble{comma-separated-list}postamble*, it iterates through the comma list of tokens inside the brackets and generates a new string for each one. The GNU manual example is for the sequence *a{d,c,b}e*, which generates a space separated list: *ade ace abe*.

What this means is what I took to be an incorrect file specification actually generated the sequence *chapt0[0-9].txt chapt1[0-5].txt*. Since brace expansion occurs before filename expansion, this results in the desired output. So this answer, being more terse than my expected response, is actually more right.

Our machines at school are using bash 3.2. If we upgraded to bash 4, we could use a numerical sequence with brace expansion to get an even better answer. The expression *chapt{00..15}.txt* should expand to the list of all 16 filenames. But bash 4.1 is still only slowly working its way into new distributions, so it may be a while before we can count on that syntax to work everywhere.

The moral of the story, of course, is that no matter how much time you spend working with the UNIX/Linux command line, there is always plenty more to learn. I should be paying UTD (http://www.utdallas.edu/) to teach this course, not the other way around.

CLICK HERE TO COMMENT ON THIS POST
(http://www.drdobbs.com/blog/archives/2010/10/the_neverending.html)

Return to Table of Contents