Dr. Dobb's DIGEST

The Art and Business of Software Development September 2010

Itanium Innovations

By Jonathan Erickson,
Editor In Chief

The rumor is that the only reason I attend the Itanium Innovation Awards event every year is because of the free food and open bar. Not so, although the shrimp cocktail and clam chowder were especially tasty this year. No, the reason I attend is that it gives me the opportunity to meet some developers from around the world who have built some amazing systems, not to mention I've had the honor the last couple of years to be a judge. This year was no exception.

According to the Itanium Solutions Alliance (http://www.itaniumsolutions.org), which sponsored the event, a panel of "distinguished" judges selected winners in four categories:

- **Computationally Intensive Applications.** The winner in this category was the University of Malaga, which was commissioned by the city of Malaga, Spain, to create an accurate solar irradiation map and develop software that calculates the amount of solar energy that can be captured at any point of the terrain, as well as the parameters that allow a good exploitation of solar energy. For this task, they used an Itanium-based HP Superdome with a 64-bit dual-core Itanium processor.
- **Data Center Modernization.** And the winner in this category was Future Electronics of Canada, a global distributor of electronic components that operates in 169 locations in 42 countries in the Americas, Europe, and Asia. They underwent a successful migration from a MIPs processor to Itanium-based HP Integrity NonStop NS1200 platform. The team was able to complete the entire process on time, with no disruption to users, and a build a stronger platform for future growth.
- **Mission-Critical Data.** Yodobashi Camera of Japan built a new foundation for its mission-critical computing initiatives using their flexible Itanium-based Integrity servers from HP. The company launched a new service-oriented architecture and disaster-recovery environment that drastically reduced the time needed for launching new applications and greatly improved resource utilization.
- **Humanitarian Impact** (announced in June). The 2010 winner in the Humanitarian Impact category is the Spanish nonprofit organization COMPUTAEX. Their supercomputer is based on two shared-memory HP Integrity Superdome SX2000 supernodes and has been used to solve leading-edge social problems requiring more than 256 processors and up to two terabytes (TB) of RAM. COMPUTAEX is helping researchers obtain simulation results very quickly, leading to innovative solutions to Spain's social, environmental, and scientific challenges, such as climate prediction, agriculture and farming, cancer studies, and environmental impact forecasts. The reliable Itanium-based infrastructure enables researchers to run their simulations without the fear of losing research hours and gives them the assurance of accurate and quick results.

Your first question probably has to do with when and how did I become "distinguished." I'm not quite sure either, but I go with the flow. Secondly, just to ensure you know what I mean when I say "Itanium": The Itanium is a family of 64-bit Intel microprocessors formerly referred to as "IA-64" (see http://www.intel.com/products/processor/itanium/index.htm). The processors are marketed for use in enterprise servers and high-performance computing systems. The architecture originated at Hewlett-Packard (HP), and was later jointly developed by HP and Intel.

Congratulations not only to the winners, but to all of the finalists who attended the event from Russia, China, Finland, and elsewhere — all of whom developed some amazing mission-critical systems.

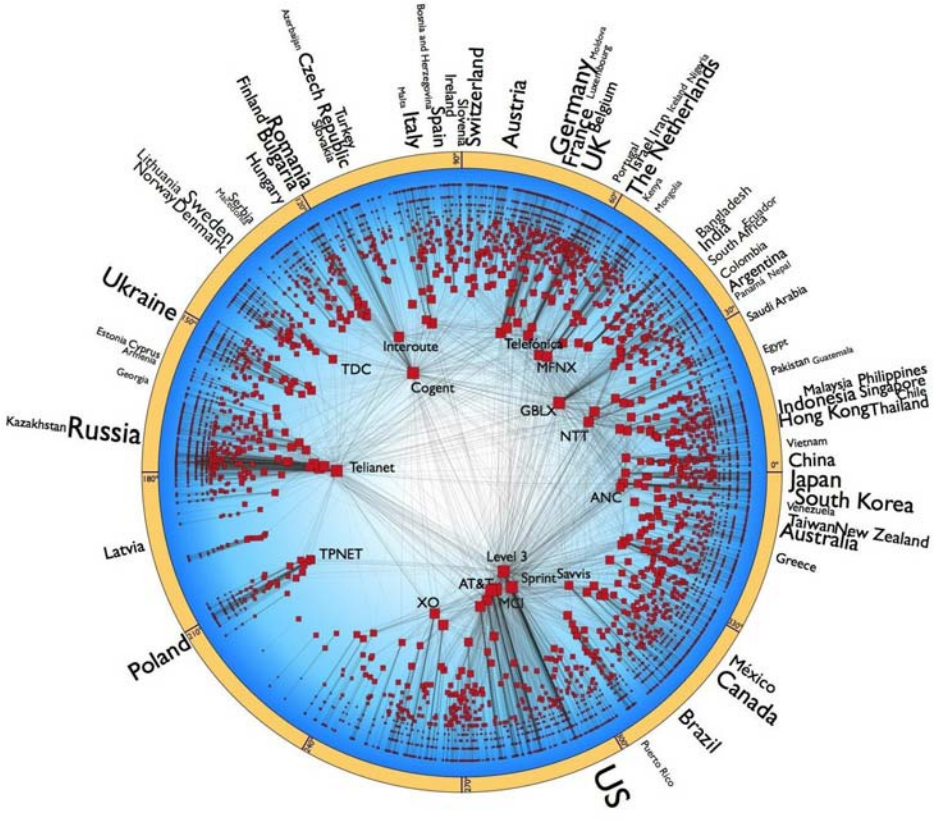
Return to Table of Contents

First Geometric 'Atlas' of the Internet Created

Mapping new paths for a stressed-out Internet

Computer scientists at the San Diego Supercomputer Center (http://www.sdsc.edu/) and the Cooperative Association for Internet Data Analysis (CAIDA; http://www.caida.org/home/) at the University of California, San Diego, in a collaboration with researchers at Universitat de Barcelona in Spain (http://www.ub.edu/web/ub/en/) and the University of Cyprus (http://www.ub.edu/web/ub/en/), have created the first geometric "atlas" of the Internet as part of a project to prevent our most ubiquitous form of communication from collapsing within the next decade or so.

CAIDA researcher Dmitri Krioukov, along with Marian Boguna (Universitat de Barcelona) and Fragkiskos Papadopoulos (University of Cyprus), discovered a latent hyperbolic, or negatively curved, space hidden beneath the Internet's topology, leading them to devise a method to create this Internet map using hyperbolic geometry:



Hyperbolic map of the Internet. Image courtesy of Dmitri Krioukov, SDSC/CAIDA

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
John Siefert, Senior Vice President and Publisher, InformationWeek Business Technology Network
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 Executive Producer, TechWeb TV

techweb™

In their paper "Sustaining the Internet with Hyperbolic Mapping", the researchers say such a map would lead to a more robust Internet routing architecture because it simplifies path-finding throughout the network.

"We compare routing in the Internet today to using a hypothetical road atlas, which is really just a long encoded list of road intersections and connections that would require drivers to pore through each line to plot a course to their destination without using any geographical, or geometrical, information which helps us navigate through the space in real life," said Krioukov, principal investigator of the project.

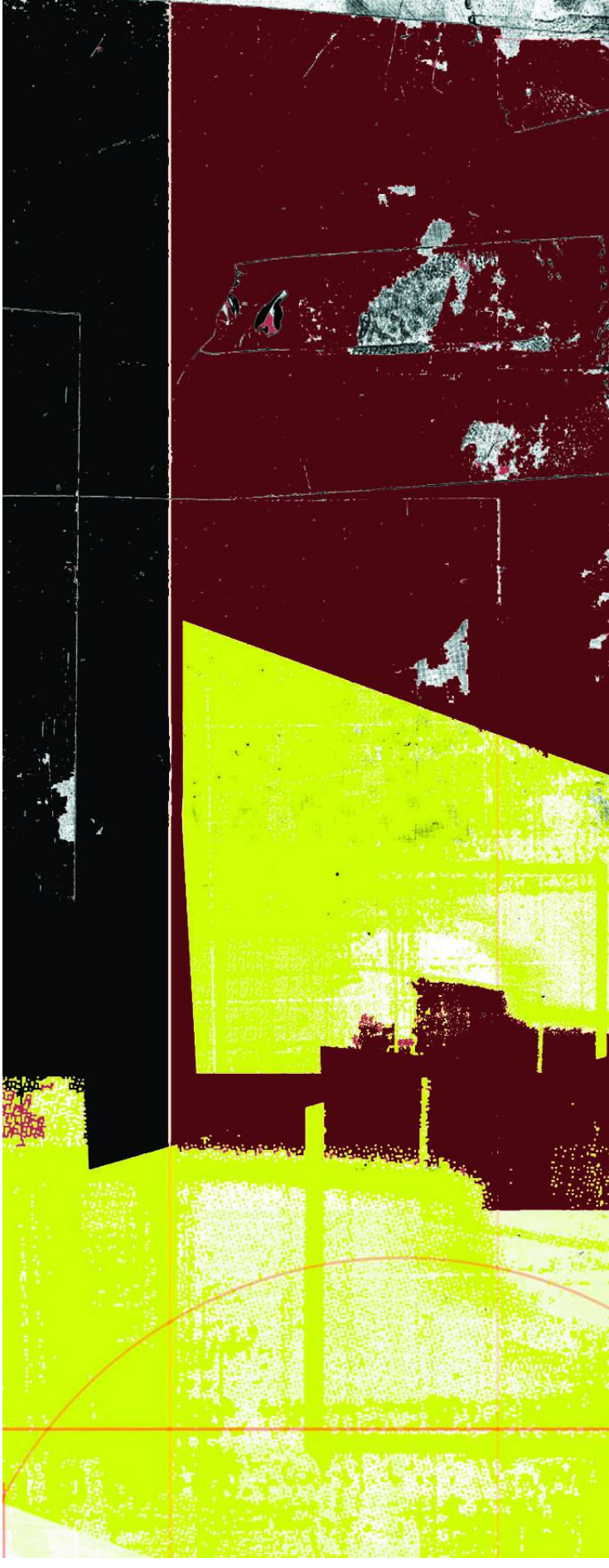
Now imagine that a road — or in the case of the Internet, a connection — is closed for some reason and there is no geographical atlas to plot a new course, just a long list of connections that need to be updated. "That is basically how routing in the Internet works today — it is based on a topographical map that does not take into account any geometric coordinates in any space," said Krioukov, who with his colleagues at CAIDA have been managing a project called Archipelago (http://www.caida.org/projects/ark/), or Ark, that constantly monitors the topology of the Internet, or the structure of its interconnections.

Like many experts, however, Krioukov is concerned that existing Internet routing, which relies on only this topological information, is not really sustainable. "It is very complicated, inefficient, and difficult to scale to the rapidly growing size of the Internet, which is now accessed by more than a billion people each day. In fact, we are already seeing parts of the Internet become intermittently unreachable, sinking into so-called black holes, which is a clear sign of instability."

In their paper, the researchers employ Ark's data and statistical inference methods to build a geometric map of the Internet. They show that routing using such a map would be superior to the existing routing, which is based on pure topology.

Instead of perpetually accessing and rebuilding a reference list of all available network paths, each router in the Internet would know only its hyperbolic coordinates and the coordinates of its neighbors so it could route in the right direction, only relaying the information to its closest neighbor in that direction, according to the researchers. Known as "greedy routing", this process would dramatically increase the overall efficiency and scalability of the Internet. "We believe that using such a routing architecture based on hyperbolic geometry will create the best possible levels of efficiency in terms of speed, accuracy, and resistance to damage," said Krioukov.

However, the researchers caution that actually implementing and deploying such a routing structure in the Internet might be as challenging, if not more challenging, than discovering its hidden space. "There are many technical and non-technical issues to be resolved before the Internet map that we found would be the map that the Internet uses," said Krioukov.



Return to Table of Contents

Real-World Software Security

BSIMM compares your secure development efforts to others'

By Gary McGraw

Building secure software isn't as simple as adding cryptography and authentication. Nor is it a matter of plunking down a firewall in front of your Web apps. It's about adjusting the software development lifecycle, teaching developers about security, choosing the right tools and techniques for writing code, and adapting the development culture to care about security. Each of these activities is made easier and more doable with not just careful planning but also objective measurement.

Many companies have large-scale software security initiatives. I know of 75 such efforts from my research alone. They cover a broad spectrum of industries, from financial services and independent software vendors to healthcare and energy providers. Any company that develops software or relies on it needs a software security initiative and metrics to measure its effectiveness.

Over the last decade, a number of software security methodologies have evolved, including Microsoft's Secure Development Lifecycle; the Comprehensive, Lightweight Application Security Process, part of the Open Web Application Security Project; and the Seven Touchpoints for Software Security. All are grounded in good software engineering and require attention to security throughout software's lifecycle. Companies must understand common risks, design for security, and subject all software artifacts to thorough, objective risk analyses and testing.

What's Really Happening

That's what companies ought to be doing anyway. For a look at what they're actually doing, BSIMM (short for Building Security In Maturity Model and pronounced "bee-sim") is an extensive research effort to analyze software security activities of companies such as Adobe, Bank of America, Capital One, EMC, Google, Intel, Microsoft, Symantec, VMware, and Wells Fargo. I'm one of the researchers.

BSIMM currently describes the work of 635 people in 30 companies who together have 130 years of experience working on software security. The success of these companies' programs hinges on having an internal group devoted to software security. The average size of these groups at BSIMM participants is 22 people, helped by about 40 others — developers, architects, and people directly engaged in and promoting software security. These companies have an average of 5,061 developers — so around 1% of the development team is directly promoting security.

BSIMM delineates 109 activities, such as getting upper management buy-in when it comes to compliance and privacy, and protecting the integrity of software by using code signing. These activities are organized in 12 categories such as compliance and policy, code review, and security testing. The 12 practices are divided into four groups — governance, intelligence, touchpoints, and deployment; see Table 1. For BSIMM's most recent study, researchers counted each time they observed one of the 109 activities.

The Software Security Framework (SSF)			
Governance	Intelligence	SSDL Touchpoints	Deployment
Strategy and Metrics	Attack Models	Architecture Analysis	Penetration Testing
Compliance and Policy	Security Features and Design	Code Review	Software Environment
Training	Standards and Requirements	Security Testing	Configuration Management and Vulnerability Management

Table 1: Activities are organized in 12 categories and divided into four groups.

The greatest value of BSIMM is in measuring and comparing software security initiatives among companies in different industries. To give you some idea of the analysis capabilities provided by the BSIMM, Figures 1 and 2 show the average maturity level over some number of organizations for the 12 practices.

Figure 1 shows data from all 30 BSIMM firms, while Figure 2 shows data from the top 10 firms (as determined by recursive mean score). In Figure 3, we calculated the level of activity in each of 12 practices for a company, then we averaged those scores for companies in the same industry. We've done this for financial services companies and ISVs in Figure 3.

By creating spider charts like these from the BSIMM data, it's possible to compare and contrast how an industry approaches software security. Not surprisingly, the data shows that financial services companies emphasize compliance and policy activities and the creation of security features and design to a greater extent than ISVs. However, the most striking result is the huge amount of overlap between the two disparate industries. Financial services and software companies generally do the same thing when it comes to software security.

Even more interesting than pooling data from multiple companies is comparing the scores of an individual company with the BSIMM averages. This can give a clear indication of how a company's software security initiative stacks up against that of others. Figure 4 shows a hypothetical company called "Firm" compared with the BSIMM average (computed over 30 firms). The hypothetical company Firm is clearly ahead of the game in some practices (strategy and metrics, security features and design, code review, and penetration testing) and just as clearly behind in some others (compliance and policy, architecture analysis, and software environment). A BSIMM spider chart provides data essential to informing an enlightened software security initiative strategy .

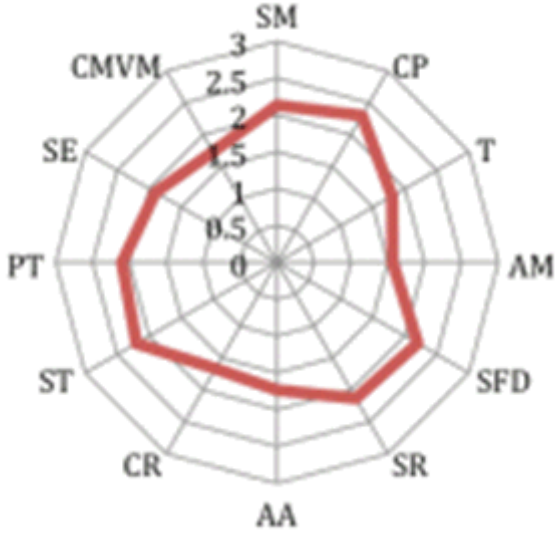


Figure 1

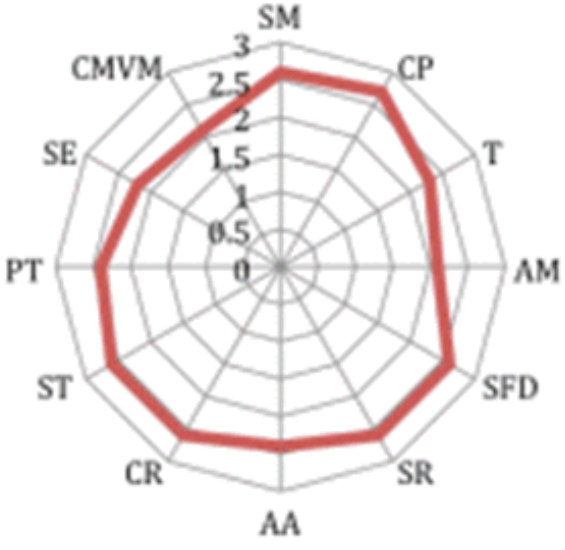


Figure 2

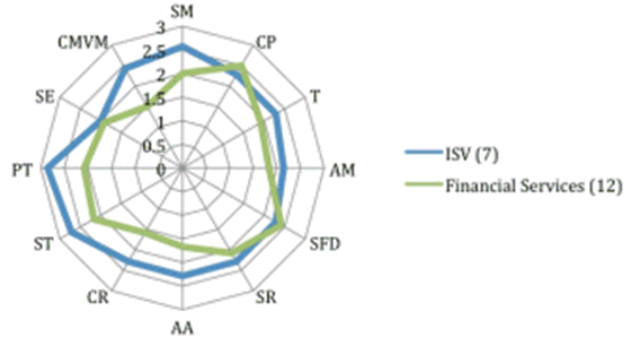


Figure 3

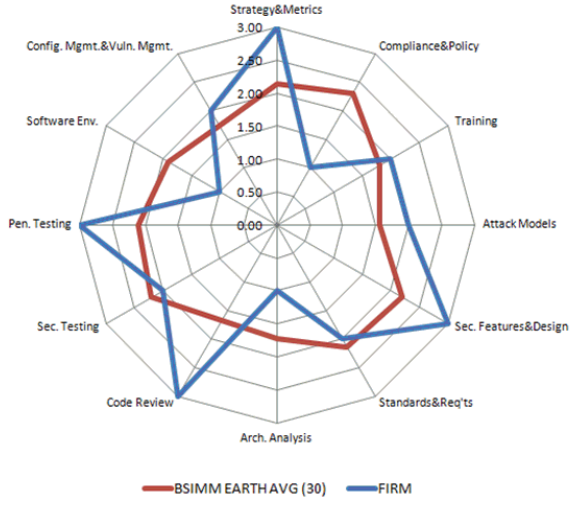


Figure 4

BSIMM Scorecard for: FIRM Raw Score: 41

Governance			Intelligence			SSDL Touchpoints			Deployment		
Activity	Observed	FIRM	Activity	Observed	FIRM	Activity	Observed	FIRM	Activity	Observed	FIRM
[SM1.1]	18	1	[AM1.1]	12	1	[AA1.1]	22		[PT1.1]	28	
[SM1.2]	18		[AM1.2]	20		[AA1.2]	18	1	[PT1.2]	17	1
[SM1.3]	16		[AM1.3]	14		[AA1.3]	19	1	[PT2.1]	17	
[SM1.4]	24	1	[AM1.4]	10		[AA1.4]	15		[PT2.2]	10	
[SM1.5]	13		[AM2.1]	7	1	[AA2.1]	9		[PT2.3]	11	
[SM2.1]	12		[AM2.2]	9	1	[AA2.2]	6		[PT3.1]	9	1
[SM2.2]	13		[AM2.3]	13	1	[AA2.3]	11		[PT3.2]	5	
[SM2.3]	16		[AM2.4]	9		[AA3.1]	5				
[SM2.4]	19	1	[AM3.1]	2		[AA3.2]	3				
[SM3.1]	7	1	[AM3.2]	2							
[SM3.2]	4										
[CP1.1]	24	1	[SFD1.1]	29	1	[CR1.1]	10	1	[SE1.1]	11	1
[CP1.2]	24		[SFD1.2]	16	1	[CR1.2]	19	1	[SE1.2]	30	1
[CP1.3]	26	1	[SFD2.1]	18		[CR1.4]	20	1	[SE2.2]	16	
[CP2.1]	13		[SFD2.2]	11		[CR2.2]	11		[SE2.3]	7	
[CP2.2]	18		[SFD2.3]	10	1	[CR2.3]	8	1	[SE2.4]	13	
[CP2.3]	13		[SFD3.1]	5	1	[CR2.4]	12	1	[SE3.2]	6	
[CP2.4]	9		[SFD3.2]	10		[CR2.5]	11				
[CP2.5]	17					[CR3.1]	7	1			
[CP3.1]	4					[CR3.2]	1				
[CP3.2]	7					[CR3.3]	2	1			
[CP3.3]	5										
[T1.1]	24		[SR1.1]	22	1	[ST1.1]	21	1	CMVM1.1	21	1
[T1.2]	6		[SR1.2]	13		[ST1.2]	9	1	CMVM1.2	22	
[T1.3]	5	1	[SR1.3]	12	1	[ST2.1]	18	1	CMVM2.1	18	1
[T1.4]	11		[SR1.4]	11		[ST2.2]	16		CMVM2.2	11	
[T2.1]	14		[SR2.1]	10	1	[ST2.3]	5		CMVM2.3	11	1
[T2.2]	13	1	[SR2.2]	8		[ST3.1]	7		CMVM3.1	2	
[T2.4]	14		[SR2.3]	13	1	[ST3.2]	10		CMVM3.2	4	
[T2.5]	7	1	[SR2.4]	13		[ST3.3]	3				
[T3.1]	4		[SR2.5]	11	1	[ST3.4]	4				
[T3.2]	3		[SR3.1]	10							
[T3.3]	4										
[T3.4]	2										

Table 2: 110 activites from BSIMM, shown in 4 domains and 12 practices. Observed count of firms (out of 30) observed performing this activity. Yellow = a most common activity, performed by 20 or more firms out of 30; Red = where we did not observe a most common activity; Green = where we did observe a most common activity ; Light Blue = a practice where the Firm's high-water mark score is below the average; Dark Blue = a data -driven candidate activity for increasing practice maturity.

Table 2 is filled to the brim with useful data, including a mapping of whether the most commonly observed activities in the BSIMM (the top 15 that are highlighted in yellow) are found in the target company called "Firm" (resulting in a red or green square). Each individual activity can be compared, as can the 12 practices. Note that those practices where the firm is behind the average in the spider chart are marked in the scorecard as "blue shift" practices, and the BSIMM scorecard can automatically suggest activities to consider adopting based on how commonly they are observed in the real world.

BSIMM released its original study describing software security in nine companies in March 2009. In May, it released BSIMM2, covering 30 companies and providing statistically significant results. BSIMM plans to add more companies, and it's also been remeasuring current participants to mark the evolution in the maturity of their software security initiatives.

For Adobe, BSIMM provides a useful framework and data repository that lets its Secure Software Engineering Team measure its activities against other companies in the industry, says Brad Arkin, senior director of product security and privacy. He adds that it also helps the security team to answer questions such as "Are there things we don't do now but could or should be doing?" and "Are there things we should be doing differently because of a shift in the threat landscape?"

Leveraging BSIMM

How can you leverage the BSIMM model? For starters, examine and organize your security initiatives along the lines of BSIMM's 109 activities. Think about which will work in your culture and which you're already doing. Download the BSIMM2 model (it's free and available at www.bsimm2.com). Compare your data with the BSIMM average and make your own spider charts.

If you want an objective measurement, you can join the project, be scored by the authors of the model, and have your data added to it. A BSIMM score will give you a baseline that you can use to compare your software security initiatives to other companies and show progress as your initiative matures. Bottom line, this data will let you create a software security strategy that's directly informed by what others are doing and what works.

— Gary McGraw is CTO at Cigital, a software security and quality consulting company, and the author of the Software Security Library. *He can be contacted at http://www.cigital.com/gem/.*

Return to Table of Contents

The Windows DLL Loading Security Hole

Many Windows apps — even Microsoft's and maybe your own — are vulnerable to "remote binary planting" attacks

By Larry Seltzer

In August, 2010, the Slovenian security research firm ACROS Security issued an advisory for a vulnerability in Apple's iTunes (see http://www.acrossecurity.com/aspr/ASPR-2010-08-18-1-PUB.txt). The details in the advisory accidentally exposed a much larger story that ACROS was working on quietly: Many Windows applications — including many famous ones and possibly your own — are vulnerable to the same attack. Luckily, you can fix the problems and I'll show you how.

"Remote Binary Planting" is a new variation on an old attack, first identified 10 years ago by Georgi Guninski (http://archives.neohapsis.com/archives/win2ksecadvice/2000-q3/0117.html). He showed how an attacker could load a malicious DLL by including it in the same directory as a data file opened by the user through the Windows shell. At the time, the problem was that when Windows searched for a DLL to load, the current directory was high in the search order. When you load a data file in the shell, Windows sets the file's directory as the current directory when loading the file, so all you had to do was to name your DLL and the calls exported from it the same as the one expected by the application.

Here's how it works: The attacker places an enticing data file in a folder on a network share or web folder. Probably through a social networking attack ("Open mlb.xls to see pennant race analysis") he induces a user to open the file. In the same directory as the file is a DLL with the same name as one needed by the application to open the file, and exporting a function expected by the application. With a vulnerable application, Windows will look for the DLL in the current directory — the one containing the data file — before finding the real one, and therefore will load it and run the malicious code.

In response to this issue, Microsoft changed the DLL search order (http://msdn.microsoft.com/en-us/library/ms682586%28VS.85%29.aspx), created the *SetDllDirectory* function (http://msdn.microsoft.com/en-us/library/ms686203%28VS.85%29.aspx), and issued guidance about proper loading of DLLs. It turns out that this advice is widely ignored, even within Microsoft.

ACROS's innovation in this attack was to show how to exploit it on a network share or remotely over the Internet on a WebDAV share. This raises the credibility of the attack scenario a great deal. Indeed, if an attacker gains access to a network, getting users to load data files from a network share within it should be relatively easy. This would be a great tool for an attacker perpetrating a targeted attack against your organization, where they have some knowledge of you already. All they need is the credentials of a relatively unprivileged user.

So what can you do? Follow Microsoft's guidance on DLL security (http://msdn.microsoft.com/en-us/library/ff919712%28VS.85%29.aspx). In particular, it's a bad idea to rely on the system PATH environment variable for your application to find its DLLs. The PATH lies behind the current directory in DLL loading priority:

1. The directory from which the application loaded.
2. The system directory.
3. The 16-bit system directory.
4. The Windows directory.
5. The current directory.
6. The directories that are listed in the PATH environment variable.

Microsoft refers to this order as "Safe Search Order" or "SafeDllSearchMode" (see http://msdn.microsoft.com/en-us/library/ms682586%28VS.85%29.aspx), and it is enabled with a registry key. This setting has been default-on since Windows XP SP2 and supported since Windows XP SP1. If you are working with clients older than that, then DLL search order may be the least of your problems, as even XP SP2 is no longer supported.

Once you understand the problem, it seems rather obvious and it's remarkable that it hasn't been noticed before

Of course, if you specify a fully qualified path in your call to *LoadLibrary* (http://msdn.microsoft.com/en-us/library/ms684175%28VS.85%29.aspx), *LoadLibraryEx* (http://msdn.microsoft.com/en-us/library/ms684179%28VS.85%29.aspx), *CreateProcess* (http://msdn.microsoft.com/en-us/library/ms682425%28VS.85%29.aspx), or *ShellExecute* (http://msdn.microsoft.com/en-us/library/bb762153%28VS.85%29.aspx), no search is necessary. So that's always the best way, if you can. You can also make sure your app runs the correct DLL by using DLL Redirection (http://msdn.microsoft.com/en-us/library/ms682600%28VS.85%29.aspx) or a Manifest (http://msdn.microsoft.com/en-us/library/aa375365%28VS.85%29.aspx). You can specify a particular search order for the app with the *SetDllDirectory* function (http://msdn.microsoft.com/en-us/library/ms686203%28VS.85%29.aspx).

Microsoft's Dynamic-Link Library Security page (http://msdn.microsoft.com/en-us/library/ff919712%28VS.85%29.aspx) also has instructions for how to use the Process Monitor tool (http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx) to observe exactly what DLLs your apps are loading and from where.

You may have already noticed a steady stream of application security updates related to this issue, including some from Mozilla (http://www.mozilla.org/security/announce/2010/mfsa2010-52.html). A recent security update of Safari from Apple (http://support.apple.com/kb/HT4333) fixes a nasty variation on the same bug, that of the EXE load search order. When you load executables, you have to be aware of the same problem. Mitja Kolsek of ACROS says that "the search path for executables is actually much worse, even worse than it used to be for DLLs before Microsoft introduced the 'safe search order.'" Unless you specify the full location of the executable, Windows will first look in the current directory.

Once you understand the problem, it seems rather obvious and it's remarkable that it hasn't been noticed before. This also represents a gap in the guidelines of the Microsoft Security Development Lifecycle (SDL; www.microsoft.com/security/sdl/default.aspx), the secure development practices created by Microsoft for its own use and available to anyone through the Creative Commons License (http://blogs.msdn.com/b/sdl/archive/2010/08/26/microsoft-sdl-and-the-creative-commons.aspx). I asked Microsoft and it admitted as much:

> *"Given the newly-found Internet attack vector, we are currently updating the SDL to include guidance and tooling for developers and testers so they can prevent or find, triage, and fix potential DLL preloading vulnerabilities in their products. Once we have finished the SDL guidance, we will make a version of the guidance and tooling documentation publicly available as soon as possible."*

You might think that the answer would be to remove the current directory from the search order, and I asked Microsoft about just that. It seems this possibility is not in the cards, as it would break a substantial number of customer installations, and this is the clearest evidence you could ask for that this is a serious problem. It proves that the number of apps which rely on the current directory for DLL loading is large.

For the long term, Microsoft has released an update enabling a new registry key called "CWDIllegalInDllSearch" (http://blogs.msdn.com/b/sdl/archive/2010/08/26/microsoft-sdl-and-the-creative-commons.aspx), which lets users and administrators control the loading of DLLs in their own systems. You can't rely on this being loaded and Microsoft won't be turning it on by default. With no systemic solution coming from Microsoft. it's up to you, the developer and publisher of the application, to make sure it's done right and that any fixes get to your users.

— Larry Seltzer is an independent analyst focusing on security issues and a founder of OpenGov Solutions (http://www.ogsnj.com/).

Return to Table of Contents

XPERF and the Windows Performance Toolkit

XPERF adds tremendous value in the performance troubleshooting process

By Mario Hewardt

Far too often, performance is overlooked and tackled too late in the release schedule. And while developers spend long hours chasing elusive — but "standard" — bugs (crashes, resource leaks, and the like), performance bugs go unnoticed for long periods of time or not addressed at all. After all, "it's not crashing, just taking a little longer than expected".

I would bet that all developers have been in situations where all the standard bugs have been taken care of and the application works satisfactorily in staging — but once taken to production, comes to a grinding halt. After spending weeks troubleshooting the problem (all the while revenue is impacted) by going through log files, performance counters, and other interesting data points, the problem is finally isolated and a fix implemented and deployed.

One of the interesting challenges when troubleshooting performance problems is that of using a suitable toolset that can quickly help you find the root cause of the problem. It is common for performance problems to lie outside of your direct code base. Imagine a credit-card processing server where the RAID controller has been incorrectly configured, causing dropped transactions. In this case, you can look at your credit-card transaction code until you are blue in the eyes. What is needed is a tool that tells you about the system as a whole to help identify bottlenecks.

Fortunately, that tool exists — the Microsoft Windows Performance Toolkit. In this article, I focus on a subset of the toolkit called XPERF, a powerful tool that helps with overall system performance analysis.

Installing XPERF

XPERF is part of the Microsoft Windows Performance Toolkit (MSWPT), which in turn is part of the Windows SDK. Since the MSWPT is a fairly lean toolkit, it might seem like overkill to have to download the entire Windows SDK to get just that toolkit. Fortunately, the Windows SDK Web Installer lets you specify a subset of the Windows SDK to download and install and MSWPT is part of the Development Tools subset. To use the Windows SDK Web Installer, go to the Microsoft website and follow the download instructions for the particular Windows SDK version. For example, on my machine, I would use the following link to install the Windows 7 SDK:

```
http://www.microsoft.com/downloads/details.aspx?FamilyID=c17ba869-9671-4330-a63e-1fd44e0e2505&displaylang=en
```

Once the Windows SDK Web Installer starts running, I deselect all the options except for the Development Tools. An interesting caveat here is that once the installation completes you're almost all the way there but not quite. The installer doesn't actually install MSWPT, rather it copies the installer package (MSI) to your installation path. The final step in the installation process is to manually invoke the MSI to get the toolkit installed. For example, on my machine, I installed the Development Tools to the following folder:

```
C:\Program Files\Microsoft SDKs\Windows\v7.0
```

Upon successful installation, the following files are available under the bin folder:

- wpt_ia64.msi
- wpt_x64.msi
- wpt_x86.msi

Each of the installation packages represents the toolkit for a given architecture. If you run the package corresponding to the architecture of choice, the installation process is straightforward and installs the toolkit in the folder of choice. On my machine, I left the default options and it installed into:

```
C:\Program Files\Microsoft Windows Performance Toolkit
```

In this folder, you will see the two binaries that we will be using throughout this article:

- xperf.exe — Drives the configuration and collection of performance data
- xperfview.exe — Used to analyze the performance data collection from a previous run of xperf.exe

Architecture

As with any type of troubleshooting, success is directly proportional to the amount of diagnostics data that is available for analysis. I'm sure we've all been in a situation where we were debugging a heap corruption only to find out that the crash dump file we were analyzing didn't have full memory information contained within it, thereby limiting the success of finding the root cause. The same is true when debugging performance problems — the more data that is available, the higher the success rate. By increasing the amount of diagnostics data that is logged, however, we also increase the pressure on the system as a whole. With any type of tracing, one of the single biggest questions is always how much can we trace before it adversely affects the system?

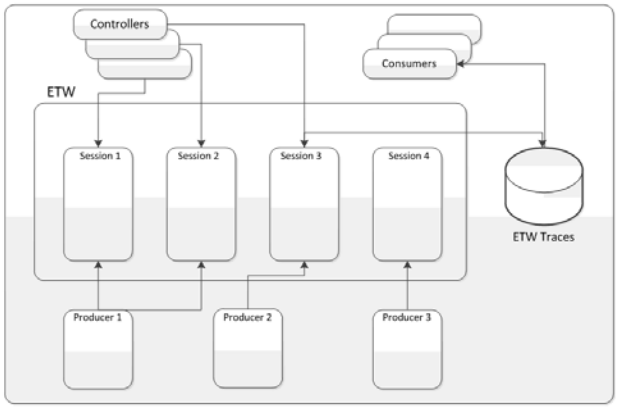


Figure 1: ETW Architecture.

The answer is that it depends on the tracing technology that is utilized. I can very easily write a simple tracer that writes and flushes to a file every time the log method is called. Of course, that approach is highly inefficient — I probably won't be able to log enough data to be useful before the system is affected. XPERF faces that same problem. For it to be a useful tool (across the system as a whole) it must be able to log large amounts of diagnostics data. How does it go about doing that without affecting the system overall? The answer lies in Event Tracing for Windows (ETW), a general-purpose and super-efficient tracing mechanism that is built into Windows (both in kernel and user mode). In addition to being extremely efficient, ETW lets you dynamically enable/disable logging without having to restart the system and/or application. Figure 1 illustrates the overall architecture of ETW.

Fundamentally, ETW consists of four primary components:

- **Session:** The base entity responsible for writing traces to the trace files.
- **Producer:** A producer writes event traces to one or more sessions.
- **Controller:** A controller can configure ETW sessions as well as enable and/or disable producers.
- **Consumers:** Applications that read event traces from the ETW trace logs.

Based on the architecture depicted in Figure 1, XPERF is really just a controller (in the sense that it can control what is being collected, i.e., the providers) as well as a consumer since it can display the results of the collection in a meaningful fashion to the user. For example, I can configure XPERF to collect certain kernel mode diagnostics data and then use the resulting file to analyze the results.

The key takeaway here is that XPERF is based on an incredibly powerful tracing mechanism native to the Windows operating system. It has the ability to tell (control) Windows exactly what to collect in an extremely efficient manner — far more efficient than any other tracing mechanism.

Running XPERF

Once you understand how XPERF utilizes the powerful ETW trace mechanism, how can we actually control what is being collected? As you've probably already guessed, XPERF is able to collect a ton of useful information and as such has an equally powerful configuration story in the form of a command-line tool called xperf.exe. I won't go through every single option available, rather I'll focus on some of the most common scenarios in how to enable/disable tracing.

Let's start by taking a simple look at how to use XPERF to collect diagnostics information. Start by running the following command from the MSWPT installation folder:

```
xperf.exe -on Base
```

The *Base* argument specifies a predefined set of commonly used providers to start collection diagnostics information. Upon

successful execution, anything you do on that system will be recorded based on the enabled providers. Please note that a lot of information is traced, so you may want to limit the amount of time that you spend trying to reproduce a particular problem. To give you an idea, on my machine I only ran for about 20 seconds and ended up with a trace file that was 8MB in size.

When you are ready to stop tracing, the following command can be used:

```
xperf.exe -d <filename>
```

where filename is the name of the result file (typically ending with the extension .etl). Now, the result file is in a raw format, meaning that it's nearly impossible to decipher using just a regular editor. Instead, you have to use either different parameters to the xperf.exe tool or use the xperfview.exe tool, which displays the results in an easy-to-digest form. Before we take a look at those tools, it is also important to mention that instead of using one of the pre-canned accumulative collection arguments (such as *Base*), you can specify data providers by using the *-providers* switch. To find out which providers are registered on the system, use the following command:

```
xperf.exe -providers
```

Analyzing XPERF Diagnostics Data

Now that have we all the diagnostics data collected, it's time to turn our attention to how to best look at the resulting file. Again, the xperfview.exe tool has a nice graphical user interface that makes it easy to digest the plethora of information collected by XPERF. Figure 2 shows an example of xperfview.exe when run against the generated results file:

```
xperf.exe -on Base
```

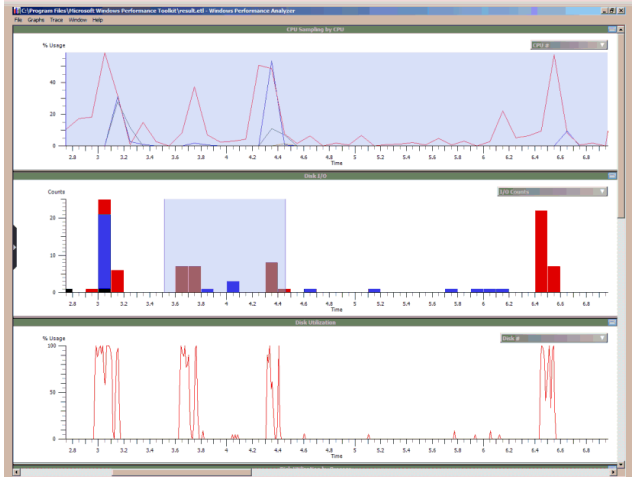


Figure 2: Running xperfview.exe.

You can see in Figure 2 how xperfview.exe provides an easy way to analyze the resulting information. Each section of the output corresponds to a particular provider (such as CPU usage, I/O Counts, etc.). To narrow down the investigation to a particular section of any given provider, highlight the section followed by right-clicking and selecting "Zoom To Selection" in the context menu. You can customize the different providers that are shown by selecting the left-most arrow tab in the middle of the screen, which will bring up a list of providers that you can either select or deselect.

Another useful XPERF feature is the ability to collect stack traces of the ongoing thread activity. For stack traces to work, you need to make sure that you have the correct symbols and that the symbol path is set to the symbol location. For example, if I wanted to take periodic stack traces I can use the following command:

```
xperf.exe -on Base -stackwalk Profile
```

Note that having the correct symbol path set is only required during the analysis stage and not the collection stage.

To view stack traces in xperfview.exe, select the area of interest in the CPU usage chart, right click on the selection, and choose "Summary Table." In the new window, click on the Columns menu item and select Stack. Another column name "Stack" is now added and you can use the + button to expand and get call stack information.

Summary

Performance problems can be tricky to troubleshoot in an efficient matter. One of the main difficulties is knowing which part of the system is causing the performance degradation. I've often seen developers spend weeks looking at their application logs trying to figure out why the system is running poorly when, in reality, a system-wide analysis needed to be performed to figure out a configuration problem on the server. It is a daunting task to step outside the application boundary and troubleshoot a system as a whole. Fortunately, we do have a great tool in XPERF that adds tremendous value in the troubleshooting process.

— *Mario Hewardt is the founder of The High-Tech Avenue (http://www.thehightechavenue.com/), a consulting and educational company that specializes in Windows and .NET. Mario is also the author of* Advanced .NET Debugging *and* Advanced Windows Debugging.

Return to Table of Contents

The Arrival of HTML 5

Lots of new features, all eagerly awaited

By Alexander V. Korostov and Dmitry J. Paramzin

HTML (short for the "Hyper Text Markup Language") is one of the underpinning technologies of the modern Web with the lion's share of web users' Internet activities founded on it. For nearly two decades HTML has worked well while undergoning a number of changes in response to the growing requirements of web users and vendors. It now stands on the brink of the next change — the coming of HTML 5. At present, the Internet already contains a handful of HTML 5 specification outlines which partially cover HTML 5 features and conceptions. In this article, we review the current state of HTML and describe the most significant HTML 5 innovations.

History

The initial version of HTML was pioneered by Tim Berners-Lee in the 1980s. The idea was based on and influenced by the SGML format (Standard General Markup Language) used in CERN.

The first description of HTML was introduced to the general public in 1991 in a document called HTML Tags. This document described 20 elements included in the initial design of HTML. The first HTML specification, HTML 2.0, was published as IETF RFC 1866 in 1995 (http://tools.ietf.org/html/rfc1866). HTML 1.0 was a draft specification.

In 1994, the W3C organization was established with a directive to administer expansion of common web protocols and to advance web interoperability. All subsequent versions of the HTML specification were published as W3C Recommendations. HTML 3.2 (http://www.w3.org/TR/REC-html32) was published as the W3C Recommendation in January 1997 and HTML 4.0 (http://www.w3.org/TR/REC-html40-971218/) was published in December 1997, with minor edits issued in 1998, 1999, and 2000.

The Web Hypertext Application Technology Working Group (WHATWG; http://www.whatwg.org/) took leadership of HTML development and started work on HTML 5.0 specification in 2004 under the name "Web Applications 1.0" (see http://www.whatwg.org/specs/web-apps/current-work/multipage/). As of now, more than 13 years have passed since the first publication of the HTML 4.0 W3C Recommendation, and HTML 5.0 is still in a working draft state. Nevertheless, many parts of the draft are stable and have already been implemented in some browsers.

And Then There Was DOM

HTML 5 uses a completely different approach to language definition in comparison with those of the past. In contrast with the previous versions of HTML, which were based on document syntax, HTML 5 uses the Document Object Model (DOM) as the foundation of the language. DOM is an in-memory language-independent representation of the document and all its elements (objects or nodes). This shifting of the language "source" to DOM enables abstract specification of HTML elements without relying on concrete formatting syntax. This also allows transparent usage of multiple DOM serialization syntaxes and application of different programming languages, typically JavaScript, for manipulating DOM objects from within scripts. Such manipulation is possible because DOM defines methods along with data attributes.

The authors of HTML 5.0 have chosen the Interface Definition Language (IDL; http://en.wikipedia.org/wiki/Interface_description_language) for describing the interface of the DOM elements in a language-neutral manner. IDL has C++-like syntax and provides various methods for defining OOP concepts, including but not limited to attributes (including read-only), methods, constants, and inheritances. The following is an

excerpt from the HTML 5.0 specification that illustrates the power of IDL. This example is designed for developers familiar with object-oriented programming languages such as C++, Java, or C#.

```
interface HTMLMediaElement : HTMLElement {
  // error state
  readonly attribute MediaError error;

  // network state
     attribute DOMString src;

     readonly attribute DOMString currentSrc;
     const unsigned short NETWORK_EMPTY = 0;
     const unsigned short NETWORK_IDLE = 1;
     const unsigned short NETWORK_LOADING = 2;
     const unsigned short NETWORK_NO_SOURCE = 3;
  ...
     void load();
     DOMString canPlayType(in DOMString type);
     void play();
     void pause();
  ...
     // controls
    attribute boolean controls;
    attribute float volume;
    attribute boolean muted;
  ...
    // timed tracks
    readonly attribute TimedTrack[] tracks;
    MutableTimedTrack addTrack(in DOMString label, in DOMString
kind,
          in DOMString language);
};
```

Note that IDL only defines the attributes and methods of elements, not their textual representations. For textual representation, the HTML 5 specification uses HTML and XHTML serialization types. In regards to web application authors, this change merely means that they are allowed to write their documents in any of these syntax styles.

While IDL provides numerous advantages, it also has shortcomings in comparison with DTD, the method previously used for defining HTML 4 elements. Its primary disadvantages are:

- Lack of an obvious way to specify allowable elements hierarchies (i.e. which elements can be children of other elements and vice versa).
- Absence of a means for defining the constraints of attributes. It is impossible to specify which attributes are required and which are optional. It is also not possible to specify a set of allowable values.

The authors of HTML 5 have chosen IDL, in place of DTD or the more powerful XSD, because it can abstractly define elements and their behaviors without relying on concrete markup syntax, an impossible task for DTD and XSD.

Not By Text Alone

Although HTML was initially a text markup language, its abilities evolved from its "textual" bounds long ago. New native elements in HTML 5 allow authors to operate non-textual information such as video and sound. HTML 5 has not made major developments in image support.

The new <*video*> element (http://www.whatwg.org/specs/web-apps/current-work/multipage/video.html#video) provides the ability to embed videos inside web documents without using third-party plug-ins like Adobe Flash or Microsoft Silverlight. Below is a simple example of a video element with explicit width and height specifications for a poster image (e.g. representative frame from the video):

```
<video src="good-dog.flv" width=200 height=120 controls
       poster=preview.jpg />
```

The next example illustrates a more complex scenario where a <*video*> element may include several sources of a video encoded using different codecs and alternative content to be shown if a browser does not support the video:

```
<video width=200 height=120 controls poster=preview.jpg>
  <source src=video.ogv type=video/ogg>
  <source src=video.mp4 type=video/mp4>
  Your browser does not support video.
  Please <a href=video.ogv>download it</a>.
</video>
```

Note: An even more complex example may include an <*object*> element inside the <*video*> tag to play the video using Flash if the browser does not support the <*video*> tag out-of-the-box. The <*audio*> element (http://www.whatwg.org/specs/web-apps/current-work/multipage/video.html#audio) is very similar to the <*video*> element. Both DOM elements extend the same *HTMLMediaElement* for those with like attributes (the vast majority of them) and methods. Here is an example of that tag:

```
<audio src="audio.oga" controls>
  Your browser does not support audio.
</audio>
```

Note that there are many additional features of both elements not shown in the examples above such as buffering, auto-playing, and the ability to associate external tracks containing subtitles, captions or descriptions with media content. The bottom line is that, by using video and audio elements with a corresponding DOM interface, web developers can embed and manage multimedia resources without the headache related to embedding external plug-ins for different browsers.

One of the chief innovations related to data representation is the <*canvas*> element (http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html#the-canvas-element), which lets JavaScript developers paint dynamic bitmap content and 2D shapes on the screen. The <*canvas*> element, in combination with modern speedy JavaScript engines, opens incredible possibilities in the creation of online games and advertisements. The HTML 5.0 specification defines the *HTMLCanvasElement.getContext(contextId)* method that returns canvas context for drawing. Context APIs are not defined in HTML 5.0 specification itself. Context API for 2D rendering is defined in a separate HTML Canvas 2D Context W3C Recommendation (http://www.w3.org/TR/2dcontext/) that is also currently in a draft state. This 2D drawing API is rich enough to render complex 2D scenes. Dynamism can be achieved by clearing and re-rendering the whole scene or only parts of it. Below is a simple example of

defining a *canvas* element and rendering a red rectangle using 2D context API:

```
<canvas id="canvas" width="200" height="200">
  Your browser does not support HTML5 Canvas.
</canvas>

<script type="text/javascript">
  var canvas = document.getElementById('canvas');
  var context = canvas.getContext('2d');
  context.fillStyle    = '#f00'; // red
  context.fillRect(0, 0, 100, 100);
</script>
```

HTML 5.0 also allows embedding SVG (Scalable Vector Graphics; http://www.w3.org/Graphics/SVG/) scenes inside a document by using the *<SVG>* element (http://www.whatwg.org/specs/web-apps/current-work/multipage/the-map-element.html#svg-0). SVG is a separate specification of another technology allowing us to define 2D scenes. Compared with *<canvas>*, SVG provides a means for high-level operations of drawing static and dynamic scenes. Basically, SVG is an XML format for defining shapes, text, and other visual objects in a scene. As a result, a developer using SVG defines objects to be rendered (i.e. declarative approach and vector graphics) instead of programmatically drawing these objects (i.e. imperative approach and bitmap graphics) when using *<canvas>*. SVG scenes can be rendered by a browser, a third-party application, or a plug-in.

SVG is also much more convenient and powerful for developing dynamic scenes. All defined objects are remembered in a tree similar to DOM and can be manipulated by using languages like ECMAScript (JavaScript) embedded in SVG to change their attributes. This greatly simplifies implementation of dynamic scenes: A developer needs only to change the attributes of a defined object (e.g. position, size, color) and need not implement any code for clearing or re-rendering the scene. This is in sharp contrast with *<canvas>*, where a developer must programmatically clear and re-render the scene. Note that SVG 1.1 supports animation (gradual timed change of position or other attributes) out-of-the-box. Here is an example of the same red rectangle defined in SVG:

```
<svg width="200" height="200" version="1.1"
    xmlns="http://www.w3.org/2000/svg">
 <rect width="100" height="100" style="fill:rgb(255,0,0)"/>
</svg>
```

A quality critique that helps one decide between using SVG and Canvas by exploring the advantages and disadvantages of each technology for different tasks can be found at http://dev.opera.com/articles/view/svg-or-canvas-choosing-between-the-two/. Note that SVG is an older, but less-supported technology compared to HTML 5 canvas.

Offline Potential

Some time ago, a new specification for client-side database support with interesting applications was introduced. While this feature had vast potential, it has been excluded from current specification drafts (http://www.whatwg.org/specs/web-apps/current-work/multipage/) due to insufficient interest from vendors who use various SQL back-ends. As such, the only offline feature currently available in HTML 5 is flexible online/offline resource management using cache manifests (http://www.whatwg.org/specs/web-apps/current-work/multipage/offline.html#offline).

Cache manifests allow an author of a document to specify which referenced resources must be cached in browser data store (e.g., static images, external CSS and JavaScript files) and which must be retrieved from a server (e.g., time-sensitive data like stock price graphs, responses from web services invoked from within JavaScript). The manifest also provides a means for specifying fall-back offline replacements for resources that must not be cached.

This mechanism allows the author to compose HTML documents that can be viewed offline. It also enables an explicit notification to a user that some functionality is not available in offline mode. For example, given the web page with stock graph:

```
<!DOCTYPE HTML>
<html manifest="stock-graph.manifest">
 <head>
  <title>Stock Graph</title>
  <script src="js/stock-graph.js"></script>
  <link rel="stylesheet" href="style/stock-graph.css">
 </head>
 <body>
   <img src="img/logo.jpg" />

   Data: <img src="api/stock-graph.jpg" />
 </body>
</html>
```

The following cache manifest (stock-graph.manifest) specifies the caching policy for the example presented above (cache everything except the stock price graph that is replaced by another image when offline):

```
CACHE MANIFEST

# cacheable resources:
CACHE:
js/stock-graph.js
style/stock-graph.css
img/logo.jpg

# non-cacheable resources:
NETWORK:
api/stock-graph.jpg

# offline replacements for non-cacheable resources:
FALLBACK:
api/stock-graph.jpg img/stock-graph-unavailable.jpg
```

Let's Not Reinvent the Wheel

Some solutions, previously implemented in web pages by independent developers using JavaScript code, are now embedded in browsers. These integrated solutions include drag-and-drop mechanisms, history navigation, data validation in forms, and new types of "input" elements, among others. These innovations enrich user interactions and make dynamic, behavioral programming simpler.

Let's start with an overview of drag-and-drop support (http://www.whatwg.org/specs/web-apps/current-work/multipage/dnd.html#dnd). Before HTML 5, drag-and-drop support was a tough task for those who wanted to implement it from scratch (of course there are JavaScript libraries that significantly simplify this). HTML 5 makes drag-and-drop simple by providing draggable

attributes to mark draggable elements and several useful callback functions for elements being dragged:

- *Dragstart*: Stores data from objects being dragged in a specific *DragEvent* event.
- *Dragend*: Handles the end of the drag process (e.g. removes the moved element from DOM). Drop target can be any element that handles the following 3 events:
- *Dragenter*: Determines whether to accept or ignore an element being dragged.
- *Dragover*: Specifies the visual effect that notifies a user when the element is dragged over the drop target.
- *Drop*: Handles the drop.

A working drag-and-drop example that can be found in HTML 5 (http://www.whatwg.org/specs/web-apps/current-work/multipage/dnd.html#introduction-5) or in dedicated articles is lengthy and may seem complex at first sight, but this is expected since drag-and-drop is one of the most complex features in graphical UI development.

Another important feature of HTML 5 is the updated HTML browsing session history navigation API (http://www.whatwg.org/specs/web-apps/current-work/multipage/history.html#history). The *window.history* object now allows users to traverse through the history, modify historical entries from JavaScript, and remember the state of the current document on a client. Modification of the state entries allows changing the current URL, as well as the title, without reloading the page. This becomes especially handy in AJAX applications where a page is not reloaded but a user performs some operations on it, such as retrieving data and page content changes. Changing the URL allows encoding the current state of the page in the URL (which is remembered by a browser) and restoring the state automatically (by processing the URL) when the user navigates back to this page. Moreover, the *history.pushState()* method allows recollection of the state (arbitrary JavaScript object) on the client and restoration of it in JavaScript by handling the *window.popstate* event (this feature can be used even without changing the URL).

The next feature we discuss is the major upgrade of HTML forms (http://www.whatwg.org/specs/web-apps/current-work/multipage/forms.html#forms). The HTML 5 specification introduces many new types of inputs (http://www.whatwg.org/specs/web-apps/current-work/multipage/the-input-element.html#the-input-element) and powerful client-side validation methods (http://www.whatwg.org/specs/web-apps/current-work/multipage/forms.html#client-side-form-validation). HTML 4 had basic input elements such as text input, combo box, checkbox, etc. However, in real life application developers frequently need other types of inputs for entering colors, dates, time, numeric ranges, phones, emails, and URLs. Thus a user is forced to enter all these types of values as text, a process application that is anything but user-friendly. There are third-party JavaScript libraries that provide custom controls (constructed from standard ones) for such data, however they complicate development and reduce overall performance. Moreover, some very useful features cannot be implemented by third-party libraries because they do not have access to the platform. For example, the following make entering data much easier:

- Open the user's address book and let him/her choose email addresses and phone numbers.
- Allow a user to browse through the History and select a URL or copy the URL of the page loaded in the next tab.
- Show a color picker that is developed for a specific platform, taking into account its characteristics (e.g. small screen, input method, touch screen).
- Popup special virtual keyboard specific for input type on touch screen smart phones.

Now, all of the aforementioned features have become implementable thanks to previously introduced input types.

Forms validation is another major enhancement attributed to HTML 5. Depending on input type, the developer can specify different common constraints: required (i.e. a field becomes mandatory), pattern, list of allowable values, length restriction, and maximum and minimum numeric values. It is also worth mentioning the new placeholder attribute (http://www.whatwg.org/specs/web-apps/current-work/multipage/common-input-element-attributes.html#the-placeholder-attribute) that enables showing of hints or examples of a value in the input before any value is entered (a browser may indicate such text with a certain color or font).

Below is an example of a form with new types of inputs, validations, and placeholders:

```
<form method="post"
      action="/addContact">
 <p><label>Name: <input name="name" required></label></p>
 <p><label>Telephone: <input type="tel" name="tel"></label></p>
 <p><label>E-mail: <input type="email" name="email"
           placeholder="email@example.com" required></label></p>
 <p><label>Birthday: <input type="date" name="dob"
           max="2010-01-01"></label></p>
 <button type="submit" name="submit">Send</button>
</form>
```

Unfortunately, no browser supported all of these input types and features when this text went to press. Below you can see examples of how these advanced controls are rendered in Opera 10.53 (the page http://www.miketaylr.com/code/html5-forms-ui-support.html has been specifically developed for testing the support of new input types).

Function Precedes Form

The HTML 5 specification continues the trend of becoming a semantic-oriented language. Authors of HTML 5 encourage developers to separate information from presentation. All presentation instructions must be placed in CSS and pure presentation HTML elements have become deprecated (e.g. * *, *<big>*, *<font>*, *<s>*, *<u>*, etc). Note that the meaning of *<i>* and *<b>* elements have been corrected: These elements now represent "alternate voice or mood, or otherwise offset from the normal prose…" and "stylistically offset from the normal prose, such as keywords, product names …" respectively.

Along with deprecated presentation tags and changed definitions, HTML 5 introduces many new purely semantic elements (http://www.w3.org/TR/html5-diff/#new-elements), including:

- Section semantic:
  <article>: An independent piece of content of a document, such as a blog entry or newspaper article.
  <aside>: A piece of content that is only slightly related to the rest of the page.
  <header>: A group of introductory or navigational aids.
  <footer>: A footer for a section, which can contain information about the author or copyright information.
  <nav>: Represents a section of the document intended for navigation.
- Group semantic:
  <figure>: Can be used to associate a caption together with some embedded content, such as an image or video.
- Text semantic:
  <progress>: Completion of a task, such as downloading or when performing a series of expensive operations.
  <meter>: Represents a measurement, such as disk usage.
  <time>: Date and/or time.
  <details>: Additional information or controls that the user can obtain on demand. The summary element provides its summary, legend, or caption.
  <output>: Some type of output, such as a calculation done through scripting.

The idea of these tags is to provide additional semantic information about the content to browsers, information aggregators, and web crawlers. This has become especially important for search engines (to properly categorize, split, and rate the significance and relevance of each part of content), mashups, and other information aggregation tools (e.g. price comparison catalogs, blog aggregators, etc).

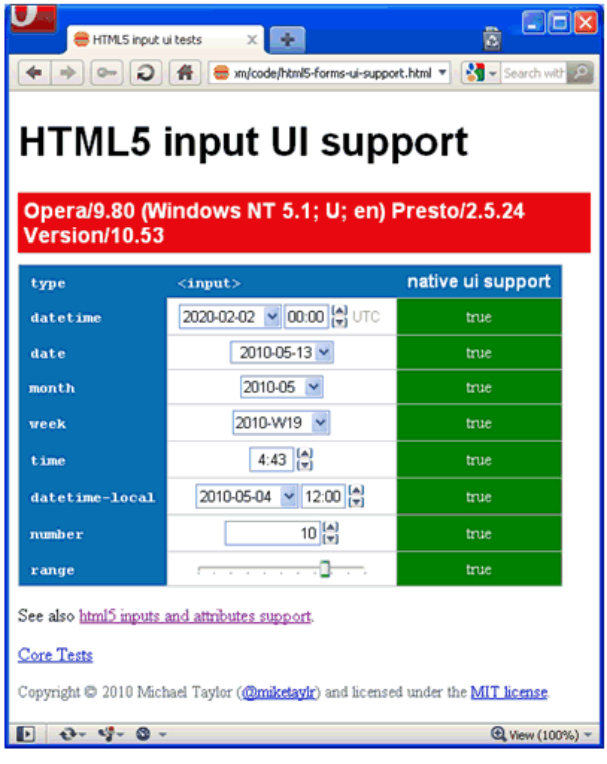


Figure 1: HTML5 controls in Opera.

REST in Forms

REST (short for "Representational State Transfer") is a popular web application architecture style (http://en.wikipedia.org/wiki/Representational_State_Transfer). The REST application can be characterized by a clear separation between clients and servers, stateless communications with the server (no client context is stored on the server between requests), and a uniform client-server protocol that can be easily invoked from other clients. Applied to HTTP, it encourages usage of URI for identifying all entities and standard HTTP methods like GET (retrieve), POST (change), PUT (add), and DELETE (remove) for entity operations. However, before HTML 5 it was impossible to send PUT and DELETE HTTP requests straight from the HTML form. This forced developers to use JavaScript or encode method names using additional parameters (e.g. *_method*).

HTML 5 now fully supports issuing PUT and DELETE requests from HTML forms without any workarounds. This is an unobtrusive, but ideologically important innovation that brings more elegance into web architecture and simplifies the development of HTML UI for REST services. Below is an example of using the PUT method for adding a new contact via a REST service:

```
<form method="put"
      action="https://mycontacts.com/api/contacts/">
 <p><label>Name: <input name="name"></label></p>
 <p><label>Telephone: <input type="tel" name="tel"></label></p>
 <p><label>E-mail: <input type="email" name="email"></label></p>
 <button type="submit" name="submit">Add</button>
</form>
```

Communicating Documents

Now documents opened in browsers can exchange data using messages (http://www.whatwg.org/specs/web-apps/current-work/multipage/comms.html#web-messaging). Such data exchange may be

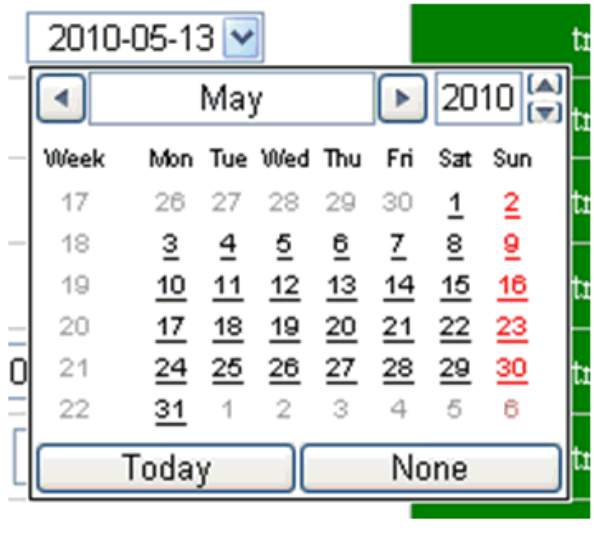


Figure 2: Opera's native date picker.

useful on a web page that includes several frames with the data loaded from different origins. Usually, a browser does not allow JavaScript code to access/manipulate the objects of other documents opened from a different origin. This is done to prevent cross-site scripting and other malicious and destructive endeavors. However, sometimes it is required to exchange data between frames, such as a parent document and an inner frame or to provide an API that allows controlling the behavior when the page is embedded in another page. To address this need, the new *onmessage* event handler has been introduced for a *Window* object. JavaScript from another frame can send events by using the *Window.postMessage()* method (this method is permitted to be invoked by JavaScript loaded from another origin).

```
window.addEventListener('message', onHello, false);

function onHello(msg) {
  if (msg.origin == 'http://example.com' && msg.data == 'Hello')
{
      // send similar message back
      e.source.postMessage('Hello', msg.origin);
      helloMsgCounter++;
  }
}
```

Note that it is essential to validate an input event (origin and/or content) before executing an associated logic to avoid injection of malicious data (executing malicious commands). It is also important to carefully consider the target origin and the data to be sent to avoid sending confidential content to other windows.

Realization

HTML 5 brings many useful and eagerly awaited features. However, it will take time to become an official W3C Recommendation since the specification is quite sizable and refers to other draft specifications (e.g. CSS 3).

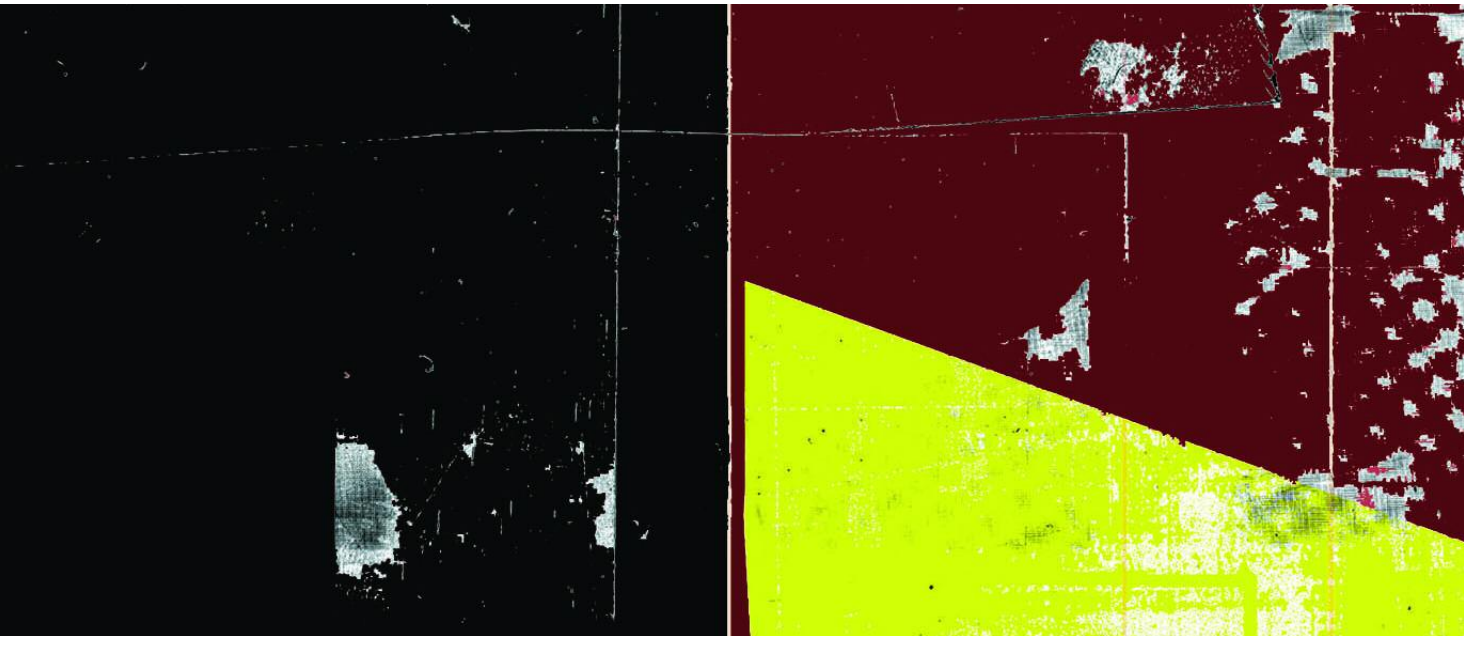
Whereas the standard is still under construction, many of its features are already supported by popular web browsers. The actual state of the implemented features is indicated in the HTML draft (http://www.whatwg.org/specs/web-apps/current-work/multipage/). A random browser can be tested for support of HTML 5 features via an online test. Some useful compatibility tables are available online, such as http://caniuse.com/.

A few of the features that are currently supported, or partially supported, in the latest releases and beta versions of popular browsers are canvas, audio and video elements, cross-document messaging, offline cache management, and SVG. Unfortunately, at this writing one of most widely distributed browsers — Internet Explorer 8 — does not support HTML 5. Current Internet Explorer 9 Platform Preview (http://ie.microsoft.com/testdrive/) slightly improves this situation but its HTML 5 capabilities are still far behind other popular browsers.

It seems that currently HTML 5 may be used for internal web applications developed for specific browsers or devices with default browsers that support HTML 5 features, such as the Apple iPhone and the iPod. Authors of JavaScript libraries can also utilize HTML 5 features if execution platforms support it by having a fallback implementation of the same functionality that will work for older versions of other browsers.

— Alexander Korostov is a System Architect at Axmor Software, IBM Solutions Group. Dmitry Paramzin is a Senior Software Engineer at IBM Solutions Group, Axmor Software.

Return to Table of Contents



No End to DLL Hell!

DLL Hell is back on the Windows programming scene — and with a vengence

By Stefan Wörthmüller

For most of us, the term "DLL Hell" is a Windows throwback from the '90s that we never thought we'd hear again. But not only has DLL Hell come back on the Windows programming scene, it has done so with a vengence: In the 1990s, your application sometimes crashed due to use of the incorect DLL version. In 2010, your application may not run at all.

DLLs were introduced with the first releases of Windows, mainly for the following of reasons:

- It's a fundamental part of the operating system. Most system parts are provided by DLLs, so that many programs can use them at the same time (i.e. to draw text).
- To save disk space and RAM (disks as well as RAM were very expensive and of limited size at the time).
- To make system parts serviceable by replacing DLLs with a newer version including fixes and improvements and to make these updates available to all programs instantly.

DLLs were not only introduced for the core OS, but made available for programs as well. Microsoft frameworks such as MFC, .NET, ATL, GdiPlus, as well as for the C (CRT) and C++ runtime libraries made use of DLLs. But in the process, a complex mechanism was exposed: Parts of programs (Microsoft's DLLs) became maintainable without notice to (or even a chance to intervene by) the program.

Since an OS is a "living thing" with new hardware, drivers, and viruses showing every day, it is a good idea to have the capability to handle new problems without installing an entirely new OS. This is a different story for programs, however. While programs should be maintained and possibly update themselves frequently, like the OS they should do so consistently. If we update some parts of a program, the program itself might be confronted with modified behavior that it is not prepared for. This leads us to the original problem that DLLs introduced: What is part of the OS, and what belongs to a program?

Everything would have been okay if the implementation of DLLs had been clean and non-ambiguous. Unfortunately that wasn't the case, as Microsoft and others began releasing different versions of the same (system and non-system) DLLs that had no proper versioning. Installers of programs overwrote existing versions of DLLs, leading to the first iteration of DLL-Hell: Installing an application could possibly break other applications (because they were built for a different "version" of the DLL). And even worse, uninstalling an application could remove some DLLs that other applications depended on. Most applications installed their DLLs in the system directory (and many still do) in hopes of sharing them with other programs, but there was no way to keep two different versions of a DLL in the system folder as the file name usually remained unchanged.

Many steps were taken to stop DLL-Hell. Some helped, but all of them introduced new problems — file protection, version resources, the system keeping a usage count for each DLL (installers asking you on unistallation to confirm the removal of a specific DLL). Ron Burks wrote a delicious article entitled "A Brief History of Windows Programming Revolutions" (http://www.drdobbs.com/windows/225701475) in which he detailed the process.

The most recent approach to get rid of these problems is the side-by-side manager introduced with Windows XP. According to Microsoft online documentation (http://msdn.microsoft.com/en-us/library/aa376307%28VS.85%29.aspx):

> *Side-by-side assemblies are used by the operating system as fundamental units of naming, binding, versioning, deployment, and configuration. Every side-by-side assembly has a unique identity. One of the attributes of the assembly identity is its version....*
>
> *Starting with Windows XP, multiple versions of side-by-side assemblies can be used by applications running at the same time. Manifests, and the assembly version number, are used by the loader to determine the correct binding of assembly versions to applications.*

Let's start by defining a couple of terms:

- Assembly: A number of DLLs boundled together with a manifest file.
- Manifest: An XML-File containing a unique ID for the assembly as well as the exact version number (the same which is included in the VERSIONINFO resource of each DLL) and information about the assemblies update policy.

Correct versioning of software is critical. A version number is a unique ID of a software package. You can think of it like a person's fingerprint, which makes how little care vendors sometimes take with it surprising. A common denominator of version numbers may be:

- The first and second part of the version number specifies a products release. Each release usually has different functionality and other major differences.
- The next (or if existent the next two) part of the number specifies the bug fix (also called "maintenance") and/or build number. Different versions of those parts should not have modified functionality; higher numbers should indicate more stable releases. Of course, in practice sometimes bug fixes introduce unwanted modification of existing behavior. Sometimes without notice to the manufacturer notwithstanding thorough testing. Sometimes accepted as a welcome improvement. Sometimes introducing problems.

Keeping versions properly distinct is easy. Some manufacturers just use the whole version String in the filename of the DLL. Another approach is based on the version info resource. As a general rule, no two released binaries shall have identical version numbers.

Experiences

When you start your program running Windows XP, you may encounter one of the following messages:

```
"Unable to execute program"
```

or:

```
"Error Code: 0xC0150002 : The application failed to initialize
properly. Click on OK to terminate the application."
```

Or even other messages, none of which are particularly informative regarding the cause of the problem.

If you are lucky, you'll come up with the idea to look into the Systems Event log (not the Applications Event log as some might guess). And yes, there may be a bit more information:

```
"Dependent Assembly Microsoft.VC80.CRT could not be found and
Last Error was The referenced assembly is not installed on your
system."
```

or:

```
"Component identity found in manifest does not match the
identity of the component requested."
```

That's better, but it still doesn't tell you what version it was searching, and why it didn't use a different version existing on the system. On the machine where I reproduced this message, four similar versions of this very DLL exist in the side-by-side repository:

```
8.0.50727.1433
8.0.50727.3053
8.0.50727.4053
8.0.50727.42
```

The version requested was:

```
8.0.50727.762
```

I know that because I wrote the program and I inspected the manifest file used to link the program. Otherwise, I'd have no idea what version is needed.

Things are better with Windows 7: System messages are more informative and starting with Windows Vista a new tool has been introduced: sxstrace, which is designed to retrieve information about the root causes (http://blogs.msdn.com/b/junfeng/archive/2006/04/14/576314.aspx). To use sxstrace:

1. Start sxstrace in logging mode
2. Reproduce the problem
3. Stop sxstrace logging
4. Parse the created log file and convert it to human-readable information

Sometimes this works, sometimes it doesn't. In any event it is a tool that the typical user will be unable to handle. A support tool may collect entries from the systems event log and transmit them to the support center — but it isn't able to use sxstrace. That's a task for a administrator. If you managed to run sxstrace successfully, the output is some diagnostic text that explains in more detail what the problem is. But it still won't tell you where to find the exactly required DLL. Microsoft used to have a page that helped locate specific DLL versions (http://support.microsoft.com/dllhelp), but alas, this service has been terminated.

So far the program concerned still cannot be started by Windows. The only solution is to "somehow" find the needed DLLs and install them. If this "somehow" does not work, the program cannot be used at all.

Consequences

The more assemblies a program uses, the more conflicts may occur and, in many cases, the more size is required for the installer as

	Conflicts	Size of Installer	Size Used on Disk
All Microsoft Libraries bound statically	None	Minimum for an application to ship.	Minimum per Application. No redundancy. No unused functions will be linked to the executable.
All Microsoft DLLs	Matching version must be installed on Target systems	All needed DLL Assemblies must be shipped and installed in the correct version.	Adding potentially more DLLs to the system including all functions - not only those needed.
Using third-party libraries bound statically	None	Minimum for an application to ship.	Minimum per application. Function calls in third-party library will be linked to the same single implementation as the ones called from your code
Using third-party DLLs	May be caused by third-party DLLs requiring certain assemblies	Different third-party DLLs may very well require different versions of "system" DLLs thus requiring an installer to ship and install multiple versions of these.	Possibly strongly redundant, if multiple versions of (i.e.) CRT had to be installed.

Table 1

well as on disk. A DLL would save disk space, if it is used by a large number of applications. But in practice Microsoft now ships so many different assembly versions, that they increase disk space used instead of saving it.

Table 1 shows the most typical ways an application is assembled and their consequences.

Solutions

To avoid DLL-Hell in 2010, the best way is:

- Don't use public DLLs or assemblies at all.
- Don't use the side-by-side manager.
- Include all DLLs privately in the applications bin-directory and use private assemblies when possible if assemblies cannot be avoided.

The best way to defeat all DLL conflicts is: **Don't use DLLs at all, use static libraries instead**. It's also possible to bypass side-by-side and assemblies, if desired. By default, Visual Studio creates a manifest file for your project, but its easy to remove this in the Project settings and to work the old way, just using the DLLs in your application's folder.

And, of course, these options may not be practical for all applications — for large applications DLLs are a way to keep the executable file small and loading fast, and also to keep things maintainable without reinstalling the whole application.

If in question, statically bound libraries are much better, even if they cause some redundancy. Redundancy is not avoidable when different third-party DLLs require different versions of the same Microsoft DLLs.

DLL-Hell in 2010

Assume you have a new great tool —say, the side-by-side manger that resolves all conflicts and keeps track of everything, if installed properly. That's an improvement. But shouldn't there be some fall-back mechanism that makes it possible to get things running? Shouldn't it be possible to use an existing DLL of a very similar version, instead of not allowing a program to run? The latter is a catastrophe for users and for support (if existent). This seems especially hard to understand in the case of a long concluded part such as the C Standard Library.

And shouldn't there be more straight and distinct information to vendors and to system administrators? Experienced administrators who were confronted with 0xC0150002 in my presence had no idea how to proceed. Amusingly, Microsoft apparently published an article "The End of DLL Hell" a while ago, but all links to it (even those on www.microsoft.com) point to a page containing the text "This content has been removed."

It is worth noting that DLLs no longer save disk space. In fact, the opposite is true. If you have a look into the side-by-side repository (found in Windows' WinSXS directory), you will find that it's 5 GB on Windows 7 and even 7 GB on Windows Server 2008, containing 40,000 files with numerous versions of roughly all Microsoft Libraries. That's about 50% of the disk space used by Windows in total. We needed s solution, but we created a monster.

— 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

Q&A: Locking Down Wireless Networks

Wireless has its own security issues

by Jonathan Erickson

Md Sohail Ahmad, a senior wireless security researcher at AirTight Networks, which develops wireless security products, recently talked with *Dr. Dobb's* editor-in-chief Jon Erickson about the state of wireless security.

Dr. Dobb's: What's unique about wireless security?
Ahmad: Wireless brings a fundamental paradigm shift to security in that the network access and data transmission is now over the air, which is a shared and unbounded medium (radio waves spill way beyond your premises). This means that without appropriate security, it's very easy for an unauthorized user sitting in the parking lot outside your building or even few blocks away to steal your data in the air and similarly to gain access to your organization's network through the air. The Wi-Fi (802.11) is an open standard, operates in an unlicensed frequency band, and hardware and software is available off-the-shelf, which maximizes this risk. This risk is absent in wired networks.

Dr. Dobb's: What's the biggest wireless threat?
Ahmad: Wi-Fi access points (APs) are inexpensive and available off the shelf. Most portable computers like laptops and netbooks as well as smartphones (e.g., iPhone, Blackkberry) and consumer devices (e.g., printers) come with Wi-Fi built in. These devices are flooding the enterprise, in turn putting enterprise security into the hands of end-users more than ever before. End-users could inadvertently (or maliciously) put enterprise security at risk, for instance, by plugging an unsecured Wi-Fi AP into the enterprise network, or sharing their enterprise network access with outsiders over Wi-Fi. This threat from unmanaged, unsecured Wi-Fi devices and the flood of Wi-Fi endpoints into the enterprise and the lack of awareness among administrators and users about it, is the biggest wireless threat.

Dr. Dobb's: What's the biggest security mistake companies make?
Ahmad: As far as wireless security is concerned, the biggest mistake organizations can make is to apply wired network security philosophy and solutions to implement wireless security. Often network administrators make the mistake of thinking that having solid wired security is enough to take care of Wi-Fi security risks. On the contrary, unsecured Wi-Fi attached to the enterprise network can open a backdoor for hackers to enter the wired enterprise network while completely bypassing all wired security measures such as firewalls, wired IDS/IPS, and content filters.

Dr. Dobb's: Are there emerging wireless security standards?
Ahmad: Given that Wi-Fi transmissions are over the air, the security of inflight wireless data starts with strong encryption and authentication. The earliest Wi-Fi security protocol was the Wired Equivalent Privacy (WEP) protocol, which is now known to be broken; the latest is Wi-Fi Protected Access version 2 (WPA2), which uses the advanced encryption standard and 802.1x based authentication. While WPA2 has so far done its job in protecting Wi-Fi networks from outsiders, a flaw in the protocol that was recently demonstrated at BlackHat Arsenal 2010 and Defcon18 exposes WPA2-secured Wi-Fi networks to insider attacks.

Dr. Dobb's: How safe are airports for wireless users?
Ahmad: Not only are airports rife with Wi-Fi hotspots that are used by travelers, but many businesses and critical operations (e.g., baggage handling, ticketing) are increasingly relying on Wi-Fi. This makes airports a prime target for Wi-Fi based attacks. AirTight Networks has conducted a Wi-Fi security study by scanning over 30 airports worldwide. The report from this study is available here. Most Wi-Fi hotspots on airports use Open Wi-Fi

configuration, which is extremely insecure for Wi-Fi users. Given the well-known risks of using Open Wi-Fi, some hotspot providers (e.g., Tmobile) are migrating to the WPA2 protocol for Wi-Fi security. But travelers using these hotspots should be aware of the recently uncovered Hole196 vulnerability in the WPA2 protocol that exposes the inherent lack of inter-user data privacy among authorized users of a WPA2-secured Wi-Fi network. Details of WPA2 Hole196 vulnerability are available here.

Dr. Dobb's: What' s the easiest thing companies can do to enhance wireless security?

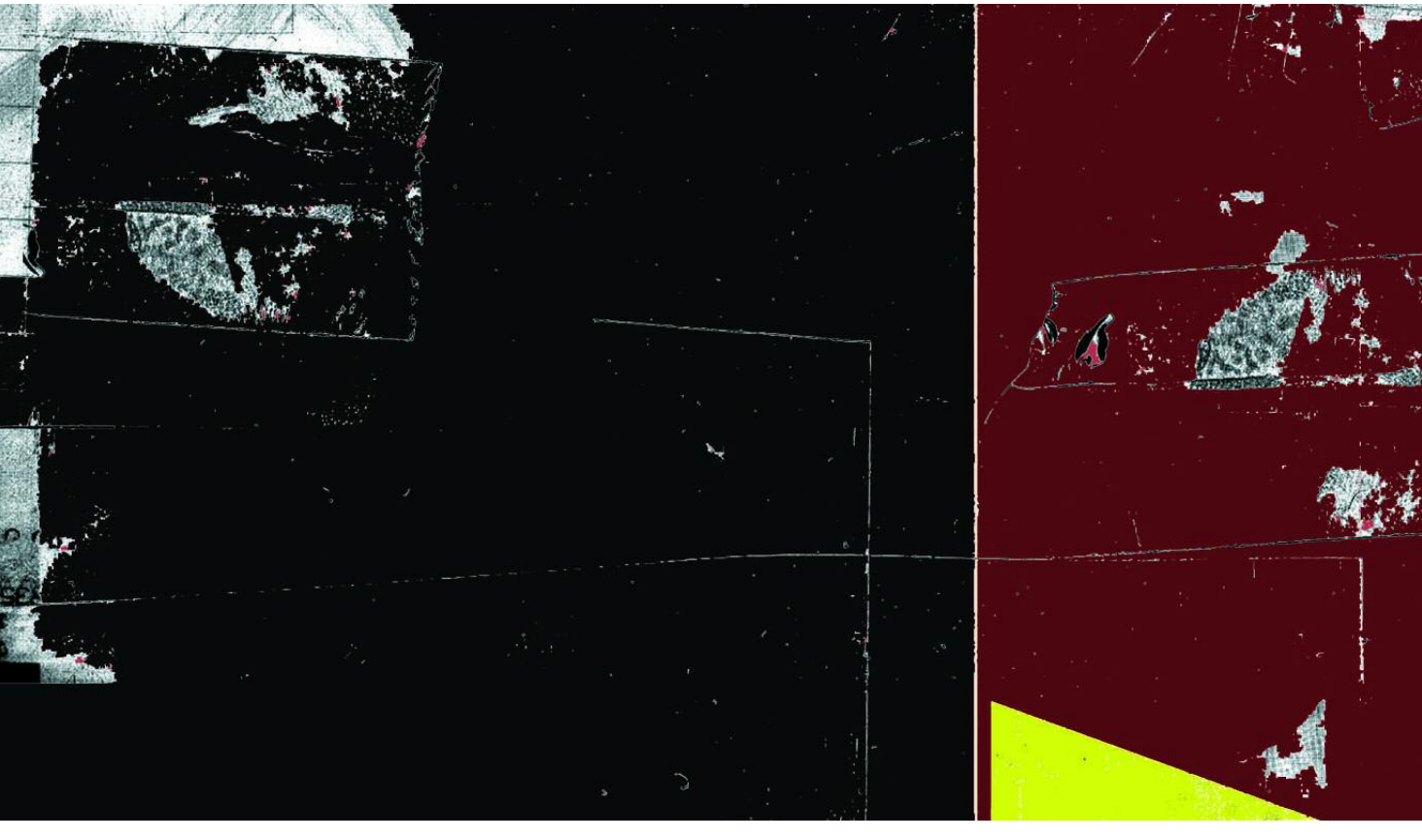
Ahmad: Like any security, there are no shortcuts to wireless security. Companies should follow some best practices to protect their networks from wireless vulnerabilities and threats:

- Use WPA2 (AES and 802.1x) to secure your Wi-Fi network from outsiders.
- Use a wireless intrusion prevention system to continuously monitor your airspace and proactively detect, block, and locate threat-posing Wi-Fi devices or those that may be violating your company's security policy.
- Educate end-users about risks from careless use of Wi-Fi on their endpoints on premises or on the road, such as at airports and coffee shops.

Here is a checklist of best practices for wireless computing on public networks:

1. Remove undesirable wireless networks from your wireless network connection profile
2. Remove any peer-to-peer network from your wireless network connection profile
3. Connect only to trusted networks that are known to you
4. Disconnect immediately if you accidentally connect to an unknown network
5. Turn off your wireless card when you do not need to be connected over wireless
6. Use a VPN client when connecting over unsecure public Wi-Fi hotspots
7. When a VPN client is not available, use a secure Web browser (SSL)
8. Avoid accessing confidential, sensitive, or valuable information over unencrypted connections
9. Upgrade your wireless software regularly. Always keep your laptop's wireless drivers updated to the latest version
10. Review points 1 – 9 every time you are connecting wirelessly.

Return to Table of Contents



Jolt Product Excellence Awards: App Libraries and Frameworks

Telerik's Premium Collection for .NET

By Robert A. DelRossi

Telerik's Premium Collection for .NET, this year's recipient of the Jolt Product Excellence Award in the Application Libraries and Frameworks category, is both deep and wide. It's deep in that the components included deftly handle a significant number of common developer needs. It's wide in that it covers the range of Microsoft platforms including ASP.NET, Silverlight, Windows Forms, and WPF. Telerik rounds the package out with additional libraries for report generation, object relational mapping, and code analysis.

There's the usual array of customizable, skinnable editors, trackbars, tree views, menus, grids, and so forth. There is also an impressive set of charting, scheduling, and data visualization components. We Jolt Award judges were particularly impressed with the Telerik Presentation Framework that lets you achieve WPF-like visual effects in standard Windows Forms applications.

Most people these days are socially conditioned to view beauty as being only skin deep and not the basis for making quality assessments. So while it's tempting to be impressed by just the sheer appearance of the components (the company has clearly invested in aesthetics), we found that the various project wizards and smart property inspectors simplified the work of implementation. We also liked the company's uncomplicated licensing scheme, its vast online support offerings, and the very complete collection of sample code that can jumpstart your own development work.

Congratulations Telerik!

Return to Table of Contents

Autotools: A Practitioner's Guide to GNU Autoconf, Automake

By Mike Riley

Autotools: A Practitioner's Guide to GNU Autoconf, Automake and Libtool
by John Calcote
No Starch Press
$44.95 print + Ebook, $22.95 Ebook

Prior to reading this book by author John Calcote, I was in the Autotools hater camp. I never took the time to learn the toolset's intricacies and in John's words, found myself "fighting the system." Has the book made me an Autotools believer?

The book begins with an introduction to the GNU Autotools packages and the systems it runs on best (POSIX systems like Linux/UNIX, Mac OSX with caveats, and for the techno-masocist, Windows with Cygwin — maybe) and the native languages supported (C, C++, Objective C, Fortran, Fortran 77, and Erlang, with partial support for other languages like Java), followed by a chapter on open source software project structure and organization via the GNU Coding Standards (GCS) and the Filesystem Hierarchy Standard (FHS). The next series of chapters dive into the actual utilization of Autotools, from creating project configuration scripts, Autoconf Makefil.in templates, and Automake Makefile.am files. Chapters 6 and 7 cover Libtool for abstracting shared library generation functionality, as well as library versioning and runtime dynamic module management. The next two chapters use Novell's open source FLAIM project (of which the author is the project administrator) to demonstrate how to move a manually managed system to an "autoconfiscated" one. Chapter 10 discusses the M4 macroprocessor as it applies to Autoconf so as to learn how to write your own Autoconf macros. Finally, the book concludes with a chapter on various tips and tricks such as cross-compiling and determining which dependencies should accompany your project's source packages.

The book has plenty of variable reference tables, command-line input/output examples and switches, Bourne shell scripts, macro printouts, best practice recommendations, and honest recognition of the passion that developers have about their tools. This refreshing approach shows how the author is in tune with the real-world practices and curmudgeons who are resistant to change. Source code listings can be downloaded from the book's website as well as updates and sample chapter PDFs.

So will I start using Autotools more frequently as a result of reading the book? Probably not, but at least now I understand the advantages of what Autotools bring to the developer toolbox. And who knows, perhaps when the next big C-based *nix project hits, Autotools will be on the list of open source build management tools to consider employing.

Return to Table of Contents

C++ Compilation Speed

By Walter Bright

I often hear the complaint that C++ code tends to be slow to compile, sometimes even taking overnight. Slow compiles were one of the motivations for exported templates, and are even listed as one of the reasons for the development of the Go language. It's a real problem, and since I'm in the C++ compiler business I get asked about this.

Why is C++ compilation slow? As it's reasonable to assume that C++ compiler implementers are pretty good at writing fast code, there must be something inherent in the language. C++ compilers do vary widely in their compilation speeds. But that isn't the whole story, since other languages routinely compile orders of magnitude faster, and it can't be true that the good compiler guys only implement other languages!

I've been working on C++ compilers since 1987. Back then, machines were extremely slow relative to today, and I paid enormous attention to trying to make the compiler fast. I've spent a lot of time doing performance profiling and tweaking the guts of the compiler to make it fast, and found what aspects of the language slow things down.

The reasons are:

1. The 7 phases of translation [1]. Although some of these can be combined, there are still at least 3 passes over the source text. I never was able to figure out how to reduce it below 3. A fast language design would have just one. C++0x exacerbates this by requiring that trigraph and \ line splicing be unwindable to support raw string literals [2].
2. Each phase is completely dependent on the previous one, meaning there's no reliable way to look ahead and, for example, look for *#include*'s and fire off an asynchronous read in advance for them. The compiler cannot look ahead to see if there's a raw string literal and so not do trigraph translation; it must do the trigraphs, and keep some sort of undo list. I've never figured out a way to parallelize C++ compilation other than at the gross level that *make* provides with the *-j* switch.
3. Because *#include*'s are a textual insertion, rather than a symbolic one, the compiler is doomed to uselessly reprocess them when one file is *#include*'d multiple times, even if it is protected by *#ifndef* pairs. (Kenneth Boyd tells me that upon careful reading the Standard may allow a compiler to skip reprocessing *#include*'s protected by *#ifndef* pairs. I don't know which compilers, if any, take advantage of this.)
4. There's a tendency for source files to just *#include* everything, and when it's all accounted for by the compiler, there's often a truly epic amount of source text that has to be processed for every .cpp file. Just *#include*'ing the Standard results, on Ubuntu, in 74 files being read of 37,687 lines (not including any lines from multiple *#include*'s of the same file). Templates and the rise of generic programming has exacerbated this, and there's increasing pressure to put more and more of the code of a program into header files, making this problem even worse.
5. The meaning of every semantic and syntactic (not just lexical) construct depends on the totality of the source text that precedes it. Nothing is context independent. There's no way to correctly preparse, or even lex, a file without looking at the *#include* file contents. Headers can mean different things the second time they are *#include*'d (and in fact, there are headers that take advantage of this).
6. Because of (5), the compiler cannot share results from compiling a *#include* from one TU [3] to the next. It must start again from scratch for each TU.
7. Because different TUs don't know about each other, commonly used templates get instantiated all over again for each TU. The linker removes the duplicates, but there's a lot of wasted effort generating those instances.

Precompiled headers address some of these issues by making certain simplifying assumptions about C++ that are non-Standard, such as a header will mean the same thing if *#incude*'d twice, and you have to be careful not to violate them.

Trying to fix these issues while maintaining legacy compatibility would be challenging. I expect there to be some signficant effort to solve this problem in the C++ Standard following C++0x, but that's at least 10 years out.

In the meantime, there isn't much of a solution. Exported templates were deprecated, precompiled headers are non-Standard, imports were dropped from C++0x, often you don't have a choice about which compiler to use, etc. Effective use of the *-j* switch to *make* is the best solution out there at the moment.

I'll do a follow up about language design characteristics that make for high-speed compilation.

Notes

[1] Paraphrased from C++98 2.1, the seven phases are:

1. Trigraph and Universal character name conversion.
2. Backslash line splicing.
3. Conversion to preprocessing tokens. The Standard notes this is context dependent.
4. Preprocessing directives executed, macros expanded, *#include*'s read and run through phases 1..4.
5. Conversion of source characters inside *char* and string literals to the execution character set.
6. String literal concatenation.
7. Conversion of preprocessing tokens to C++ tokens.

[2] The example in the C++0x Standard is at 2.14.5-4:

```
const char *p = R"(a\
b
c)";
assert(std::strcmp(p, "a\\\nb\nc") == 0);
```

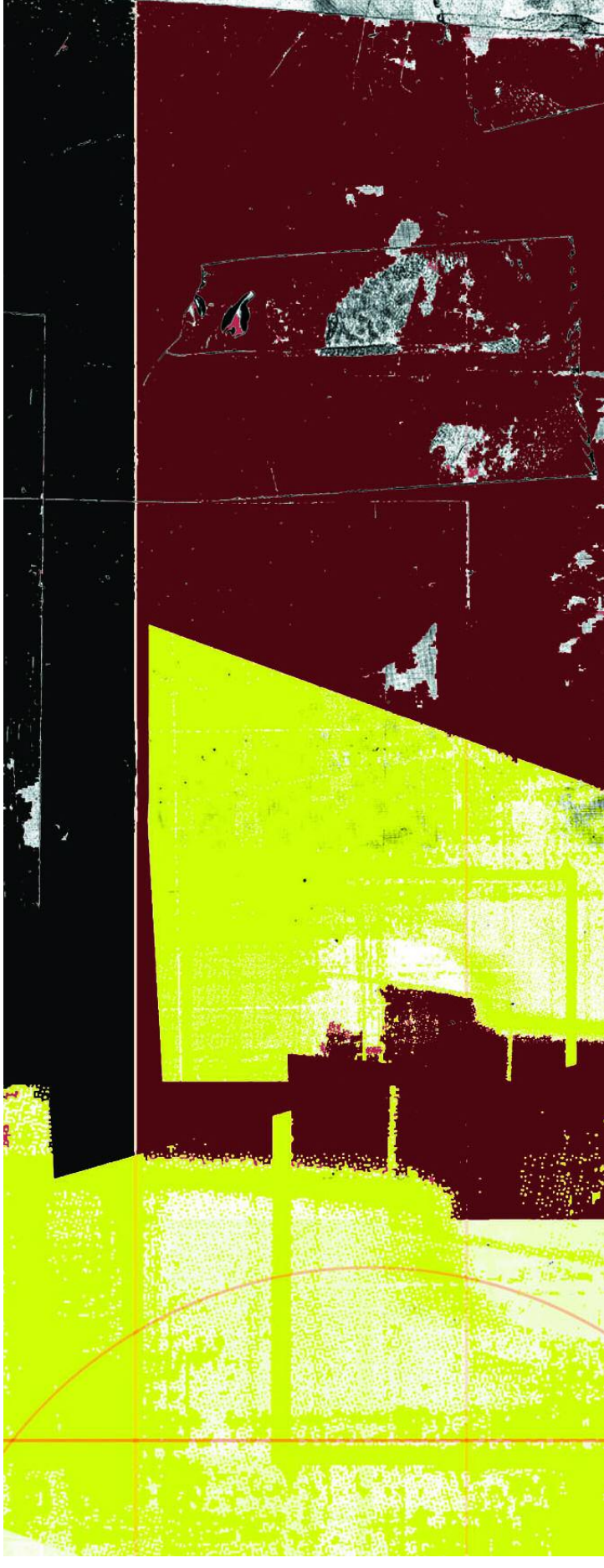
[3] A TU, or Translation Unit, is typically one C++ source file that usually has a .cpp filename extension. Compiling one TU results in one object file. The compilation process compiles each TU independently of any other TUs, and then the linker combines the object file output of those compilations into a single executable file.

Acknowledgments

Thanks to Andrei Alexandrescu, Jason House, Brad Roberts, and Eric Niebler for their helpful comments on a draft of this article.

CLICK HERE TO COMMENT ON THIS POST (http://www.drdobbs.com/blog/archives/2010/08/c_compilation_s.html)

Return to Table of Contents



Know When to Use an Active Object Instead of a Mutex

Got state? Hide it! And if it's shared and either popular or slow, make it asynchronous… it's just good encapsulation

By Herb Sutter

Let's say that your program has a shared log file object. The log file is likely to be a popular object; lots of different threads must be able to write to the file; and to avoid corruption, we need to ensure that only one thread may be writing to the file at any given time.

Quick: How would you serialize access to the log file?

Before reading on, please think about the question and pencil in some pseudocode to vet your design. More importantly, especially if you think this is an easy question with an easy answer, try to think of at least two completely different ways to satisfy the problem requirements, and jot down a bullet list of the advantages and disadvantages they trade off.

Ready? Then let's begin.

Option 1 (Easy): Use a Mutex (or Equivalent)

The most obvious answer is to use a mutex. The simplest code might look like that in Example 1(a):

```
// Example 1(a): Using a mutex (naïve implementation)
//

// The log file, and a mutex that protects it
File logFile = …;
mutex_type mLogFile( … );

// Each caller locks the mutex to use the file
lock( mLogFile ) {
  logFile.write( … );
  logFile.write( … );
  logFile.write( … );
} // unlock
```

If you've been paying attention to earlier installments of this column, you may have written it as shown in Example 1(b) instead, which lets us ensure that the caller doesn't accidentally write a race because he forgot to take a lock on the mutex (see [1] for details):

```
// Example 1(b): Using a mutex (improved implementation)
//

// Encapsulate the log file with the mutex that protects it
struct LogFile {
  // Hide the file behind a checked accessor
  // (see [1] for details)
  PROTECTED_WITH( mutex_type );
  PROTECTED_MEMBER( File, f );

  // A convenience method to avoid writing "f()" a lot
  void write( string x ) { f().write( x ); }
};

LogFile logFile;

// Each caller locks the entire thing to use the file
lock( logFile ) {
  logFile.f().write( … ); // we can use the f() accessor
                          // explicitly
  logFile.write( … ); // but mostly let's use the
  logFile.write( … ); // convenience method
}
```

Examples 1(a) and 1(b) are functionally equivalent, the latter is just more robust. Ignoring that for now, what are the advantages common to both expressions of our Option 1?

The main advantage of Option 1 is that it's correct and thread-safe. Protecting the log file with a mutex serializes callers to ensure that no two threads will be trying to write to the log file at the same time, so clearly we've solved the immediate basic requirement.

But is this the best solution? Unfortunately, Option 1 has two performance issues, one of them moderate and the other potentially severe.

The moderate performance problem is loss of concurrency among callers. If two calling threads want to write at the same time, one must block to wait for the other's work to complete before it can acquire the mutex to perform its own work, which loses concurrency and therefore performance.

The more serious issue is that using a mutex doesn't scale, and that becomes noticeable quickly for high-contention resources. Sharing is the root of all contention (see [2]), and there's plenty of potential contention here

Microsoft®
Silverlight®

One Silverlight, Many Opportunities.

Microsoft Silverlight is a powerful development platform for creating engaging, interactive applications for many screens across the Web, desktop, and mobile devices. Developers can write once, optimize everywhere to deliver engaging, high-quality consumer and enterprise experiences through all major browsers on Mac, Windows, and Linux client operating systems.

Get the free tools and resources you need.
Visit the Dr. Dobbs Microsoft Resource Center at
http://www.drdobbs.com/tv/ddjtvmsondemand.jhtml

Microsoft

on this global resource. In particular, consider what happens when the log file is pretty popular, with lots of threads intermittently logging things, but the log file's *write* function is a slow, high-latency operation — it may be disk- or network-bound, unbuffered, or slow for other reasons. Say that a typical caller is calling *logFile.write* regularly, and that the calls to *logFile.write* take about 1/10 of the wall-clock time of the caller's computation. That means that 10% of a typical caller's time spent inside the lock — which means that at most 10 such threads can be active at once before they start piling up behind the lock and throttling each other. It's not really great to see the scalability of the entire program be limited to at most 10 such threads' worth of work.

We can do better. Given that there can be plenty of contention on this resource, the only winning strategy is not to share it…at least, not directly. Let's see how.

Option 2 (Better): Use Buffering, Preferably Asynchronous

One typical strategy for dealing with high-latency operations is to introduce buffering. The most basic kind of buffering is synchronous buffering; for example, we could do all the work synchronously inside the calls to *write*, but have most calls to *write* only add the data to an internal queue, so that *write* only actually writes anything to the file itself every *N*-th time, or if more than a second has elapsed (perhaps using a timer event to trigger occasional extra empty calls to *write* just to ensure flushing occurs), or using some other heuristic.

But this column is about effective **concurrency**, so let's talk about asynchronous buffering. Besides, it's better in this case because it gets much more of the work off the caller's thread.

A better approach in this case is to use a buffer in the form of a work queue that feeds a dedicated worker thread. The caller writes into the queue, and the worker thread takes items off the queue and actually performs the writing. Example 2 illustrates the technique:

```
// Example 2: Asynchronous buffering
//

// The log file, and a queue and private worker thread that
// protects it
message_queue<string> bufferQueue;

// Private worker thread mainline
File logFile = …;
while( str = bufferQueue.pop() ) {      // receive (async)
  // If the queue is empty, pop blocks until something is
  // available.
  // Now, just do the actual write (now on the private thread).
  logFile.write( str );
}

// Each caller assembles the data they don't want interleaved
// with other output and just puts it into the buffer/queue
string temp = …;
temp.append( … );
temp.append( … );
bufferQueue.push( temp );               // send (async)
```

Note that in this approach the individual calls to *send* on multiple threads are thread-safe, but they can interleave with each other. Therefore, a caller who wants to *send* several items that should stay together can't just get away with making several individual calls to *send*, but has to assemble them into an indivisible unit and send that all in one go, as shown above. This wasn't a problem in Option 1, because the indivisible unit of work was already explicit in the Example 1(a) and 1(b) calling code — while the lock was held, no other thread could get access to the file and so no other calls could interleave.

Another minor drawback is that we have to manage an extra thread, including that we have to account for its termination; somehow, the private thread has to know when to go away, and Example 2 leaves that part as an exercise for the reader. I call this issue "minor" because the extra complexity isn't much, and termination is easy to deal with in a number of ways (note that Option 1 had a similar termination issue, too, to make sure it destroyed the file object), but I mention it for completeness — if you use a strategy like Example 2, don't forget to join with those background helper threads at the end of the program!

But enough about minor drawbacks, because Option 2 delivers major advantages in the area of performance. Instead of waiting for an entire *write* operation to complete, possibly incurring high-latency accesses and all the trimmings, now the caller only has to wait for a simple and fast *message_queue.push* operation. By never executing any part of the actual *write* on the caller's thread, callers will never have to wait for each other for any significant amount of time even if two try to write at the same instant. By thus eliminating throttling, we eliminate both performance issues we had with Option 1: We get much better concurrency among callers, and we eliminate the scalability problem inherent in the mutex-based design.

Guideline: Prefer to make high-contention and/or high-latency shared state, notably I/O, be asynchronous and therefore inherently buffered. It's just good encapsulation to hide the private state behind a public interface.

Oh, but wait — don't modern languages have something called "classes" to let us express this kind of encapsulation? Indeed they do, which brings us to Option 3…

Option 3 (Best): Use an Active Object to Encapsulate the Resource

In [3] and [4], we covered how to encapsulate a threads within an active object, which gives us a disciplined way to talk to the thread using plain old method calls that happen to be asynchronous, as well as to easily manage the thread and its lifetime just like any ordinary object. It turns out that this is a generally useful pattern, and if you didn't already believe me before, consider how naturally it helps us out with the *logFile* situation.

For Option 3, let's continue to buffer and do the work asynchronously just as we did in Option 2, except now use an active object to express it in code instead of having a distinct visible thread and message queue. Note that the queue buffer is still there, but now it's implicit and automated; we simply use the active object's message queue, and it happens naturally because we just turn *write*

into an asynchronous method on the active object so that the actual *logFile.write* call is sent via the internal queue to be executed on the active object's hidden private thread:

```
// Example 3: Private thread - expressed as active object (see [3]
for details)
//
class AsyncLogFile {
public:
  void write( string str )
    { a.Send( [=] { log.write( str ); } ); }
private:
  File log;          // private file
  Active a;          // private helper (queue+thread)
};
AsyncLogFile logFile;

// Each caller just uses the active object directly
string temp = …;
temp.append( … );
temp.append( … );
logFile.write( temp );
```

This has all the benefits of Option2, including full concurrency and scalability, but expressing it this way is significantly better. Why?

First, it's simpler and better encapsulated. Instead of exposing a raw queue and helper thread, we instead raise the level of abstraction and provide a simpler and more natural object-based interface to calling code. The caller gets to use the original and natural *logFile.write* "*object.method*" syntax instead of dealing with an exposed message queue.

Second, shutdown is greatly simplified. Now, whenever we're done using the *logFile* and destroy it, it naturally performs its own orderly shutdown of the private thread, including that it correctly drains its remaining messages (if any) before returning from the destructor (see [3] for details). We no longer have to worry about special code to arrange shutdown of the helper thread, because we've expressed the thread as an object and so we can just deal with its lifetime the same way we do with that of any regular object.

Summary

If you have high-contention and/or high-latency shared state, especially I/O, prefer to make it asynchronous. Doing so makes buffering implicit, because the buffering just happens as a side effect of the asynchrony. It also makes correct serialization implicit, because buffered operations are naturally performed in serial order by the single private thread that services them. Prefer to use the active object pattern as the most convenient and simple way to express an asynchronous object or resource.

In convenience, correctness, and (most especially) performance and scalability, this strategy can beat the pants off using a mutex to protect popular and/or slow shared resources. If you haven't tried this technique already in your code, pick a high-contention or high-latency shared resource that's currently protected with a mutex, test the performance of replacing the mutex with an active object, and see. Try it today.

References

[1] H. Sutter. "Associate Mutexes with Data to Prevent Races," *Dr. Dobb's Digest*, May 2010; www.drdobbs.com/224701827.

[2] H. Sutter. "Sharing Is the Root of All Contention," *Dr. Dobb's Digest*, March 2009; www.drdobbs.com/architecture-and-design/214100002

[3] H. Sutter. "Prefer Using Active Objects Instead of Naked Threads," *Dr. Dobb's Digest*, June 2010; www.drdobbs.com/high-performance-computing/225700095

[4] H. Sutter. "Prefer Using Futures or Callbacks to Communicate Asynchronous Results," *Dr. Dobb's Digest*, August 2010; www.drdobbs.com/cpp/226700179

— Herb Sutter is a bestselling author and consultant on software development topics, and a software architect at Microsoft. He can be contacted at www.gotw.ca.

Return to Table of Contents