

Still looking for a domain name? **SAVE 25%** Use Code: **SEARCH**  
[GET STARTED](#)

[Tweet](#)

Like 0

[Share](#)[G+](#)[Permalink](#)

## Passing the C++ Test

By Al Stevens, April 01, 1998

[Post a Comment](#)

**What kind of questions can you expect if you are interviewing for a programming job that requires C++? Al provides a list of questions and answers that all C++ applicants should know.**

*Al, a contributing editor for Dr. Dobb's Journal, can be contacted at [astevens@ddj.com](mailto:astevens@ddj.com).*

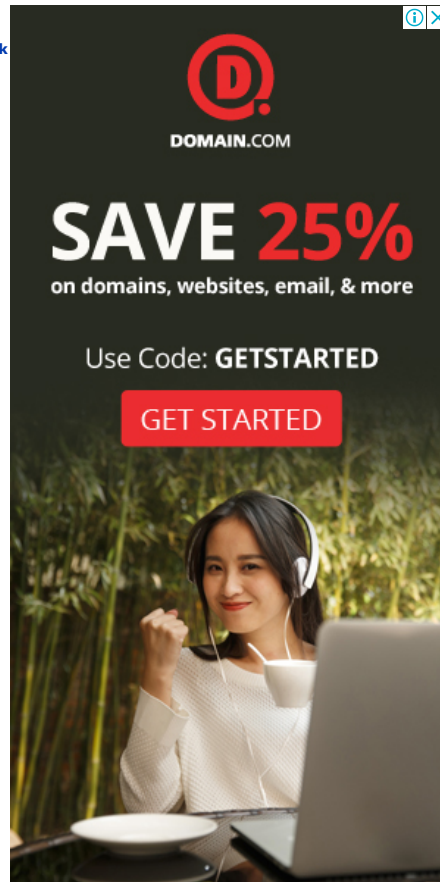
In my August 1996 *Dr. Dobb's Journal* "C Programming" column, I reported that someone with two to five years experience programming C++ can, according to a report in *ComputerWorld*, command \$70,000 on either coast and \$65,000 in the central part of the country. The opportunities seem even better for programmers with Win32 programming experience, particularly Windows NT C++ and Microsoft Foundation Classes (MFC) programming experience.

Of course, no one expects the typical college graduate to have two or more years of solid experience programming a particular platform. The range of your experience depends on how your time was distributed between the usual class workload, lab time, research projects, and any part-time jobs you might have had. Nonetheless, there is a demand for qualified people and a shortage of qualified people. Employers will be looking to entry-level recent graduates to take up the slack. That's where you come in. You'll be competing with all those other grads for those jobs, and, if the *ComputerWorld* report is a forecast of what's coming, in two years, this summer's entry-level programmers will be the veterans knocking down those respectable salaries. The trick is to get your foot in the door now. Let's see if we can help prepare you to impress the folks who are interviewing new programmers.

Windows NT programming is Win32 programming, which, unless you are writing deep systems-level code, is the same thing as Windows 95 programming. If you don't have an NT system handy, latch onto a Windows 95 PC and a copy of Visual C++ 4.0 or later. I don't want you to neglect your studies these last few months -- you have to get through finals before anyone talks to you about a job -- but there are a few things you can do to prepare to demonstrate an understanding of the programming environments that employers want to discuss. That demonstration could be just the advantage that edges out the competition.

You need to show in a brief interview that you understand and can work with C++ in the Win32 development environment. Those are two different curves, both of which are steep. Let's start with C++.

I put together a list of questions that employers can ask potential programming employees to qualify them as to the extent of their C++ knowledge. These, I think, are



**SAVE 25%**  
on domains, websites, email, & more

Use Code: **GETSTARTED**  
[GET STARTED](#)

### Recent Articles

[Dr. Dobb's Archive](#)  
[Farewell, Dr. Dobb's](#)  
[Jolt Awards 2015: Coding Tools](#)  
[Thriving Among the APIs](#)  
[The Long Death of Project Hosting Sites](#)

### Most Popular

[Stories](#) [Blogs](#)

[RESTful Web Services: A Tutorial](#)  
[Lambda Expressions in Java 8](#)  
[Developer Reading List: The Must-Have Books for JavaScript](#)  
[An Algorithm for Compressing Space and Time](#)  
[Why Build Your Java Projects with Gradle Rather than Ant or Maven?](#)

This month's Dr. Dobb's Journal

representative of the kinds of things that you will be asked. I published the questions and my opinions about appropriate answers in my column. From the response I got from readers, I learned that many practicing C++ programmers cannot answer some of these questions. The pop quiz, as one reader labeled it, sent many programmers back to their reference books. They had fallen into a rut, using a comfortable subset of the language, and, as a consequence, were not using C++ to its full potential. Other readers suggested more questions and a broader approach to qualifying an applicant. Those suggestions caused me to evaluate my approach and adjust it.

I divided applicants into three groups: those who understand C++ well enough to use it as an improved C; those who have experience designing C++ classes; and those who are so motivated that they keep up with the recently approved specification of standard C++.

I will state here that anyone who has read and understands my book, *Teach Yourself C++* (MIS Press, about to be released in its 5th edition), or who has completed the tutorials on *The Al Stevens Cram Course on C/C++* CD-ROM (available from *Dr. Dobb's Journal*), can ace all three parts of this quiz. If you consider that statement to be a shameless plug, then you fully understand my motives. It's true, nonetheless. Let's get on with it.

A lot of what follows was taken directly from my August and September 1996 "C Programming" columns in *Dr. Dobb's Journal*. I reworded some of it to reflect your concerns and to incorporate some changes and corrections that other readers sent to me.

### Questions for All C++ Applicants

Here's the first group of C++ questions and my opinions about what some acceptable answers would be. These questions do not cover C++ wall-to-wall, of course. I selected them as being typical of the kinds of things that all C++ programmers should be expected to know. There are five questions. Three correct answers is a good score.

**Q:** How do you link a C++ program to C functions?

**A:** By using the *extern "C"* linkage specification around the C function declarations.

You should know about mangled function names and type-safe linkages. Then you should explain how the *extern "C"* linkage specification statement turns that feature off during compilation so that the linker properly links function calls to C functions. Another acceptable answer is "I don't know. We never had to do that." Merely describing what a linker does would indicate to me that you do not understand the issue that underlies the question.

**Q:** Explain the scope resolution operator.

**A:** The scope resolution operator permits a program to reference an identifier in the global scope that has been hidden by another identifier with the same name in the local scope.

The answer can get complicated. It should start with "colon-colon," however. (Some readers had not heard the term, "scope resolution operator," but they knew what `::` means. You should know the formal names of such things so that you can understand all communication about them.) If you claim to be well into the design or use of classes that employ inheritance, you tend to address overriding virtual function overrides to explicitly call a function higher in the hierarchy. That's good knowledge to demonstrate, but address your comments specifically to global scope resolution. Describe C++'s ability to override the particular C behavior where identifiers in the global scope are always hidden by similar identifiers in a local scope.

**Q:** What are the differences between a C++ *struct* and C++ *class*?

**A:** The default member and base class access specifiers are different.



**This month**, Dr. Dobb's Journal is devoted to mobile programming. We introduce you to Apple's new Swift programming language, discuss the perils of being the third-most-popular mobile platform, revisit SQLite on Android, **and much more!**

[Download the latest issue today. >>](#)

### Upcoming Events

#### Live Events

#### WebCasts

Enterprise Connect 2019 - Build Your Comms & Collaboration Future - Enterprise Connect Orlando 2019  
Communications & Collaboration 2022 at EC19  
Orlando March 18-21 - Enterprise Connect Orlando 2019

### Featured Reports

[What's this?](#)

Cloud Collaboration Tools: Big Hopes, Big Needs  
Hard Truths about Cloud Differences  
State of Cloud 2011: Time for Process Maturation  
SaaS 2011: Adoption Soars, Yet Deployment Concerns Linger  
Database Defenses

[More >>](#)

This is one of the commonly misunderstood aspects of C++. Believe it or not, many programmers think that a C++ *struct* is just like a C *struct*, while a C++ *class* has inheritance, access specifiers, member functions, overloaded operators, and so on. Some of them have even written books about C++. Actually, the C++ *struct* has all the features of the *class*. The only differences are that a *struct* defaults to public member access and public base class inheritance, and a *class* defaults to the private access specifier and private base class inheritance. Getting this question wrong does not necessarily disqualify you because you will be in plenty of good company. Getting it right is a definite plus.

**Q:** How many ways are there to initialize an *int* with a constant?

**A:** Two.

There are two formats for initializers in C++ as shown in [Example 1](#). [Example 1\(a\)](#) uses the traditional C notation, while [Example 1\(b\)](#) uses constructor notation. Many programmers do not know about the notation in [Example 1\(b\)](#), although they should certainly know about the first one. Many old-timer C programmers who made the switch to C++ never use the second idiom, although some wise heads of C++ profess to prefer it.

A reader wrote to tell me of two other ways, as shown in [Examples 2\(a\)](#) and [2\(b\)](#), which made me think that maybe the answer could be extended even further to include the initialization of an *int* function parameter with a constant argument from the caller.

**Q:** How does throwing and catching exceptions differ from using *setjmp* and *longjmp*?

**A:** The throw operation calls the destructors for automatic objects instantiated since entry to the try block.

Exceptions are in the mainstream of C++ now, so most programmers, if they are familiar with *setjmp* and *longjmp*, should know the difference. Both idioms return a program from the nested depths of multiple function calls to a defined position higher in the program. The program stack is "unwound" so that the state of the program with respect to function calls and pushed arguments is restored as if the calls had not been made. C++ exception handling adds to that behavior the orderly calls to the destructors of automatic objects that were instantiated as the program proceeded from within the try block toward where the throw expression is evaluated.

It's okay to discuss the notational differences between the two idioms. Explain the syntax of try blocks, catch exception handlers, and throw expressions. Then specifically address what happens in a throw that does not happen in a *longjmp*. Your answer should reflect an understanding of the behavior described in the answer just given.

One valid reason for not knowing about exception handling is that your experience is exclusively with older C++ compilers that do not implement exception handling. I would prefer that you have at least heard of exception handling, though.

It is not unusual for C and C++ programmers to be unfamiliar with *setjmp/longjmp*. Those constructs are not particularly intuitive. A C programmer who has written recursive descent parsing algorithms will certainly be familiar with *setjmp/longjmp*. Others might not, and that's acceptable. In that case, you won't be able to discuss how *setjmp/longjmp* differs from C++ exception handling, but let the interview turn into a discussion of C++ exception handling in general. That conversation will reveal to the interviewer a lot about your overall understanding of C++.

## Questions for Class Designers

The next group of questions explores your knowledge of class design. There are eight questions. Five out of eight is a good score.

**Q:** What is your reaction to this line of code?

## Featured Whitepapers

[What's this?](#)

[Mid-Market Mayem: Cybercriminals Wreak Havoc Beyond Big Enterprises](#)  
[Managing Access to SaaS Applications](#)  
[Top Six Things to Consider with an Identity as a Service Solution](#)  
[Rogue Wave Tools and Libraries for Big Data](#)  
[Challenging Some of the Myths About Static Code Analysis](#)

[More >>](#)



## Most Recent Premium Content

### Digital Issues

#### 2014

**Dr. Dobb's Journal**  
 November - **Mobile Development**  
 August - **Web Development**  
 May - **Testing**  
 February - **Languages**

#### Dr. Dobb's Tech Digest

DevOps  
 Open Source  
 Windows and .NET programming  
 The Design of Messaging Middleware and 10 Tips from Tech Writers  
 Parallel Array Operations in Java 8 and Android on x86: Java Native Interface and the Android Native Development Kit

#### 2013

January - **Mobile Development**  
 February - **Parallel Programming**  
 March - **Windows Programming**  
 April - **Programming Languages**  
 May - **Web Development**  
 June - **Database Development**  
 July - **Testing**  
 August - **Debugging and Defect Management**  
 September - **Version Control**  
 October - **DevOps**  
 November - **Really Big Data**  
 December - **Design**

#### 2012

January - **C & C++**  
 February - **Parallel Programming**  
 March - **Microsoft Technologies**  
 April - **Mobile Development**  
 May - **Database Programming**  
 June - **Web Development**  
 July - **Security**  
 August - **ALM & Development Tools**  
 September - **Cloud & Web Development**  
 October - **JVM Languages**  
 November - **Testing**  
 December - **DevOps**

```
delete this;
```

**A:** It's not a good practice.

A good programmer will insist that the statement is never to be used if the class is to be used by other programmers and instantiated as static, extern, or automatic objects. That much should be obvious.

The code has two built-in pitfalls. First, if it executes in a member function for an extern, static, or automatic object, the program will probably crash as soon as the delete statement executes. There is no portable way for an object to tell that it was instantiated on the heap, so the class cannot assert that its object is properly instantiated. Second, when an object commits suicide this way, the using program might not know about its demise. As far as the instantiating program is concerned, the object remains in scope and continues to exist even though the object did itself in. Subsequent dereferencing of the pointer can and usually does lead to disaster.

A reader pointed out that a class can ensure that its objects are instantiated on the heap by making its destructor private. This idiom necessitates a kludgy *DeleteMe* kind of function because the instantiator cannot call the delete operator for objects of the class. The *DeleteMe* function would then use "delete this."

I got a lot of mail about this issue. Many programmers believe that *delete this* is a valid construct. In my experience, classes that use *delete this* when objects are instantiated by users usually spawn bugs related to the idiom, most often when a program dereferences a pointer to an object that has already deleted itself.

**Q:** What is a default constructor?

**A:** A constructor that has no arguments or one where all the arguments have default argument values.

If you don't code a default constructor, the compiler provides one if there are no other constructors. If you are going to instantiate an array of objects of the class, the class must have a default constructor.

**Q:** What is a conversion constructor?

**A:** A constructor that accepts one argument of a different type.

The compiler uses this idiom as one way to infer conversion rules for a class. A constructor with more than one argument and with default argument values can be interpreted by the compiler as a conversion constructor when the compiler is looking for an object of the type and sees an object of the type of the constructor's first argument.

**Q:** What is the difference between a copy constructor and an overloaded assignment operator?

**A:** A copy constructor constructs a new object by using the content of the argument object. An overloaded assignment operator assigns the contents of an existing object to another existing object of the same class.

First, you must know that a copy constructor is one that has only one argument, which is a reference to the same type as the constructor. The compiler invokes a copy constructor wherever it needs to make a copy of the object, for example to pass an argument by value. If you do not provide a copy constructor, the compiler creates a member-by-member copy constructor for you.

You can write overloaded assignment operators that take arguments of other classes, but that behavior is usually implemented with implicit conversion constructors. If you do not provide an overloaded assignment operator for the class, the compiler creates a default member-by-member assignment operator.

This discussion is a good place to get into why classes need copy constructors and overloaded assignment operators. By discussing the requirements with respect to data member

pointers that point to dynamically allocated resources, you demonstrate a good grasp of the problem.

**Q:** When should you use multiple inheritance?

**A:** There are three acceptable answers: "Never," "Rarely," and "When the problem domain cannot be accurately modeled any other way."

There are some famous C++ pundits and luminaries who disagree with that third answer, so be careful.

Let's digress to consider this issue lest your interview turn into a religious debate. Consider an *Asset* class, *Building* class, *Vehicle* class, and *CompanyCar* class. All company cars are vehicles. Some company cars are assets because the organizations own them. Others might be leased. Not all assets are vehicles. Money accounts are assets. Real-estate holdings are assets. Some real-estate holdings are buildings. Not all buildings are assets. Ad infinitum. When you diagram these relationships, it becomes apparent that multiple inheritance is an intuitive way to model this common problem domain. You should understand, however, that multiple inheritance, like a chainsaw, is a useful tool that has its perils, needs respect, and is best avoided except when nothing else will do. Stress this understanding because your interviewer might share the common bias against multiple inheritance that many object-oriented designers hold.

**Q:** What is a virtual destructor?

**A:** The simple answer is that a virtual destructor is one that is declared with the *virtual* attribute.

The behavior of a virtual destructor is what is important. If you destroy an object through a pointer or reference to a base class, and the base-class destructor is not virtual, the derived-class destructors are not executed, and the destruction might not be complete.

**Q:** Explain the ISA and HASA class relationships. How would you implement each in a class design?

**A:** A specialized class "is a" specialization of another class and, therefore, has the ISA relationship with the other class. An *Employee* ISA *Person*. This relationship is best implemented with inheritance. *Employee* is derived from *Person*. A class may have an instance of another class. For example, an *Employee* "has a" *Salary*, therefore the *Employee* class has the HASA relationship with the *Salary* class. This relationship is best implemented by embedding an object of the *Salary* class in the *Employee* class.

The answer to this question reveals whether you have an understanding of the fundamentals of object-oriented design, which is important to reliable class design.

There are other relationships. The USESA relationship is when one class uses the services of another. The *Employee* class uses an object (*cout*) of the *ostream* class to display the employee's name onscreen, for example. But if you get ISA and HASA right, you usually don't need to go any further.

**Q:** When is a template a better solution than a base class?

**A:** When you are designing a generic class to contain or otherwise manage objects of other types, when the format and behavior of those other types are unimportant to their containment or management, and particularly when those other types are unknown (thus the genericity) to the designer of the container or manager class.

Prior to templates, you had to use inheritance; your design might include a generic *List* container class and an application-specific *Employee* class. To put employees in a list, a *ListedEmployee* class is multiply derived (contrived) from the *Employee* and *List* classes. These solutions were unwieldy and error-prone. Templates solved that problem.

### Questions for ANSI-Knowledgeable Applicants

There are six questions for those who profess knowledge of the progress of the ANSI committee. If you claim to have

that much interest in the language, you should know the answers to all these questions.

**Q:** What is a mutable member?

**A:** One that can be modified by the class even when the object of the class or the member function doing the modification is *const*.

Understanding this requirement implies an understanding of C++ *const*, which many programmers do not have. I have seen large class designs that do not employ the *const* qualifier anywhere. Some of those designs are my own early C++ efforts. One author suggests that some programmers find *const* to be such a bother that it is easier to ignore *const* than to try to use it meaningfully. No wonder many programmers don't understand the power and implications of *const*. Someone who claims to have enough interest in the language and its evolution to keep pace with the ANSI deliberations should not be ignorant of *const*, however.

**Q:** What is an explicit constructor?

**A:** A conversion constructor declared with the *explicit* keyword. The compiler does not use an explicit constructor to implement an implied conversion of types. Its purpose is reserved explicitly for construction.

**Q:** What is the Standard Template Library?

**A:** A library of container templates approved by the ANSI committee for inclusion in the standard C++ specification.

An applicant who then launches into a discussion of the generic programming model, iterators, allocators, algorithms, and such, has a higher than average understanding of the new technology that STL brings to C++ programming.

**Q:** Describe run-time type identification.

**A:** The ability to determine at run time the type of an object by using the *typeid* operator or the *dynamic\_cast* operator.

**Q:** What problem does the namespace feature solve?

**A:** Multiple providers of libraries might use common global identifiers causing a name collision when an application tries to link with two or more such libraries. The name-space feature surrounds a library's external declarations with a unique namespace that eliminates the potential for those collisions.

This solution assumes that two library vendors don't use the same namespace, of course.

**Q:** Are there any new intrinsic (built-in) data types?

**A:** Yes. The ANSI committee added the *bool* intrinsic type and its true and false value keywords and the *wchar\_t* data type to support character sets wider than eight bits.

Other apparent new types (*string*, *complex*, and so forth) are implemented as classes in the Standard C++ Library rather than as intrinsic types.

In my original column, I left out *wchar\_t* even though I should have known about it. Several readers wrote to correct me. I tell you this now to emphasize that even I would not have scored 100 percent on my own test.

## Understanding Win32

I haven't put together a list of questions that you should know about the Win32 API because the subject is way too broad to cover that way. There's no way that I could guess what an interviewer is likely to ask. Instead, I'm going to tell you how to become expert enough to get through an interview. You'll have to cram this work into your otherwise busy student's schedule. I don't know how to tell you where to find the time, but if you can do it, it's worth it. Forsake the social life for the next several months. Try to get by without sleeping. (It's easy. You do all the things you would otherwise do, but you do them tired.) You are investing in your future, and this might be the most profitable time

you'll ever spend. Don't, of course, neglect your other studies.

You'll need a Windows 95 (or NT) system with Visual C++. Run the tutorials that the VC++ online books include. Get comfortable with the programming environment -- the editor, the debugger, the online documentation, the class wizard, the resource editor, and so on.

Next, you need two good books about Windows and MFC programming. These books are the classic *Programming Windows 95*, by Charles Petzold, and *Programming Windows 95 with MFC*, by Jeff Prosise (both published by Microsoft Press). Read the Petzold book cover to cover. Do it at your computer and run the programs that Charles includes. Step through them with the VC++ debugger. Get a solid foundation in the Win32 API, the Windows event-driven, message-based programming model, and the Windows way of using resources -- menus, dialog boxes, controls, and so on.

Now, read Prosise. Learn the MFC application/document/view architecture. Learn how the MFC class library encapsulates those things and the Windows resources. Run all of Jeff's programs with the debugger just as you did Charles's. Be aware that the debugger steps into all the MFC code. If you find yourself in the depths of an MFC constructor, for example, use the "Step out of" command to get back to the example. Eventually you'll learn to anticipate that and step over statements that would lead you into the vapor.

Here's the best advice I can give you. With both these books, when you are running the example programs through the debugger, never step out of a line of code until you fully understand what the line of code is doing and why. If the book is unclear about it, and if, after wracking your brain and exhausting all your resources, you can't figure it out, send me a message at [astevens@ddj.com](mailto:astevens@ddj.com). If I can't figure it out, I'll find someone who can.

This offer is good until December 24, 1999. If you don't have a job by then, you can have mine, because that's when I plan to get out of this business to avoid the Year 2000 fiasco.

### How to Comport Yourself in the Interview

Bear with me now while I patronize you for a moment. I've sat through enough interviews to know what rings the chimes of an interviewer and what turns them cold. This is going to be like one of those motivational speeches that people buy on cassettes from late night TV pitchmen. Inasmuch as you already paid for the magazine, consider this one to be free.

Remember, no one expects you to demonstrate the breadth of experience and judgement that comes with years of experience. You should not be expected to know the relative merits of every compiler, operating system, application framework, and so on. You should know what those things are, and, being a smart young person, you are expected to have and voice strong opinions based on your limited experience, but those opinions need not demonstrate anything more than the youthful exuberance and enthusiasm to which you are naturally entitled. You are bullet-proof, beer is food, and all's right with the world.

Relax in the interview and be yourself. It is not an adversarial situation. The company really wants you to be the one. Most of us hate interviewing and hope above hope that the right person shows up in the first session so we can get out of having to do any more interviews.

Above all, don't pretend to know more than you do. Be truthful when you do not know an answer. Don't try to guess what the interviewer wants to hear. Don't be afraid to ask, "What are you getting at?" Tell the interviewer that, given the chance, you can learn whatever you lack in plenty of time to be of useful service to the company. Most programming is intuitive to someone like yourself who has the aptitude. Make them know that you understand that concept and can apply it. The successful applicant is assured and confident. Make the interviewer feel that by hiring you, the company will solve its programming problems.



Here's what else not to do. I know I just told you to be yourself, but, please, avoid making social and political statements with your appearance. You have no way of knowing what the interviewer's biases and prejudices are, and that's the person whose muster you must pass today, even if on the job you'd never see that person again. Sometimes the initial screening is done by personnel types who wouldn't know a byte if it bit them. Dress and groom yourself neatly and neutrally. Try to look capable and professional. You can retreat to your own style, slob or fop, after you get the job and prove your worth.

Express a willingness to relocate anywhere and work long hours on any kind of application. Insist, however, that you want to design programs and write code. You are not a typist, shipping clerk, computer operator, data-entry person, manual writer, coffee maker, driver, gopher, or tech-support phone person. Those are fine professions, but not for you. You are a programmer. That's what you do best and that's what they need most. Emphasize that you want to work with the best technical people that the company has so that you can learn. Interviewers like to hear that.

Finally, don't get discouraged. You won't get an offer for every interview, and you'll get some offers that are unacceptable. Keep your spirits up, apply for every job that looks appealing, and spend your evenings sharpening your programming skills at home. If you see a job that you want, don't be dissuaded by an imposing list of qualifications that you do not possess. Chances are, the company won't find anyone with all the right skills anyway. Go for it and convince the interviewers that you can learn what you need to know.

Do these things and the right job will find you.

**DDJ**

---

*Copyright © 1998, Dr. Dobb's Journal*

[1](#) [2](#) [3](#) [Next](#)

## Related Reading

- [News](#)
- [Commentary](#)
- [Java Plumb Unlock Threads](#)
- [Parasoft DevTest Shifts To Continuous](#)
- [Docker Clocks In On Azure](#)
- [New Relic Continues Developer Data Apps Push](#)
- [More News»](#)
- [Slideshow](#)
- [Video](#)
- [Developer Reading List](#)
- [Developer Reading List: The Must-Have Books for JavaScript](#)
- [Jolt Awards: Coding Tools](#)
- [2012 Jolt Awards: Mobile Tools](#)
- [More Slideshows»](#)
- [Most Popular](#)
- [Lambdas and Streams in Java 8 Libraries](#)
- [Read/Write Properties Files in Java](#)
- [Lambdas in C++11](#)
- [Jolt Awards 2015: Coding Tools](#)
- [More Popular»](#)

---

## More Insights White Papers



- [Top Six Things to Consider with an Identity as a Service Solution](#)
- [Encrypted Traffic Management for Dummies eBook](#)

[More >>](#)

**Reports**

- [Strategy: The Hybrid Enterprise Data Center](#)
- [Will IPv6 Make Us Unsafe?](#)

[More >>](#)


**Webcasts**

- [Transforming Operations - Part 1: Managing Outsourced Development in Telecommunications](#)
- [Client Windows Migration: Expert Tips for Application Readiness](#)

[More >>](#)



**INFO-LINK**


  
DOMAIN.COM

**SAVE**  
**25%**

on domains, websites,  
email, & more

Use Code:  
**GETSTARTED**

**GET STARTED**



Login or Register to Comment



**TECHNOLOGY GROUP**

Black Hat  
Content Marketing Institute  
Content Marketing World  
Dark Reading

Enterprise Connect  
GDC  
Gamasutra  
HDI

ICMI  
InformationWeek  
Interop ITX  
Network Computing

No Jitter  
Service Management World  
XRDC

**COMMUNITIES SERVED**

Content Marketing  
Enterprise IT  
Enterprise Communications  
Game Developers  
Information Security  
IT Services & Support

**WORKING WITH US**

Advertising Contacts  
Event Calendar  
Tech Marketing  
Solutions  
Contact Us  
Licensing

- 
- [Dr. Dobb's Home](#)
  - [Articles](#)
  - [News](#)
  - [Blogs](#)
  - [Source Code](#)
  - [Dobb's TV](#)
  - [Webinars & Events](#)
  - [About Us](#)
  - [Contact Us](#)
  - [Site Map](#)
  - [Editorial Calendar](#)