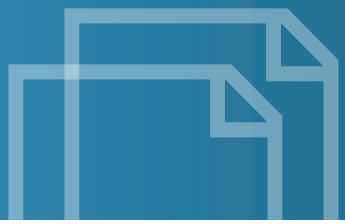



6 Steps to **Continuous Delivery**

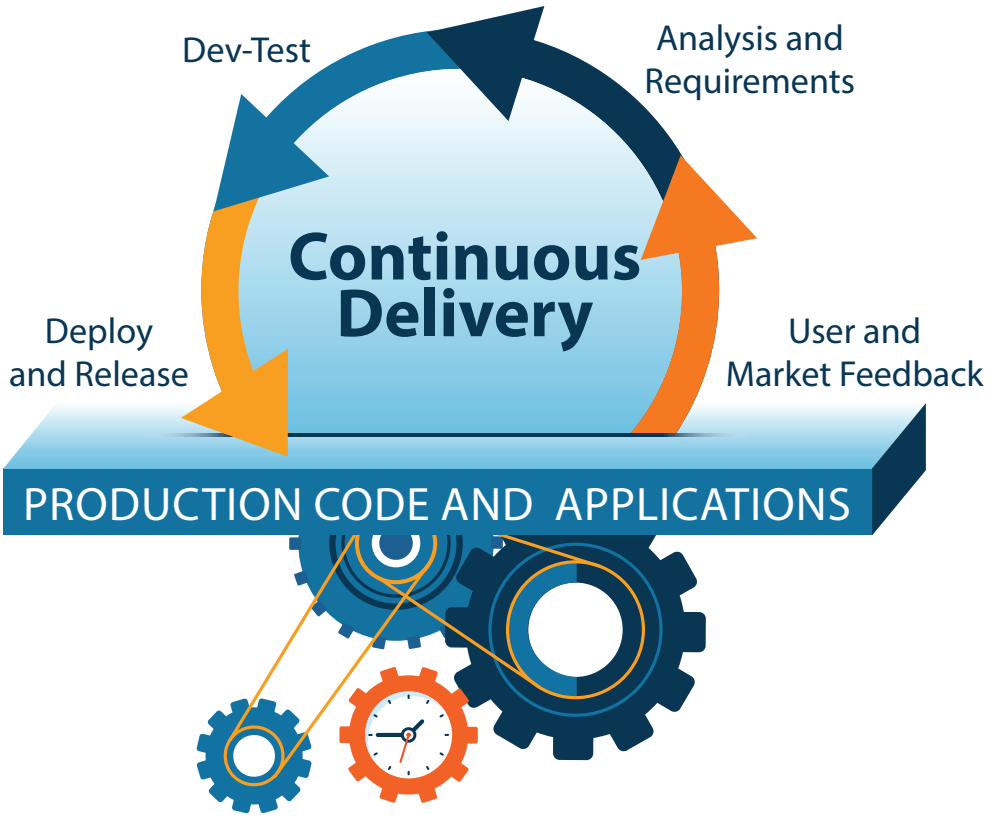
How to Increase DevOps Velocity and Produce More Stable Code

zend™

Understanding Continuous Delivery

Today's business reality is dominated by two powerfully disruptive forces:

- **Business Is Accelerating.** Markets are changing more rapidly. Customers are demanding immediate satisfaction. Products become obsolete more quickly. Business intelligence is becoming real-time data. Strategic business decisions are happening in real time.
- **Business Is Software Dependent.** Unique business value is embedded in software products and tools that businesses use every day. These tools drive automated manufacturing and supply chain workflows, baggage handling at the airport, traffic management, marketing automation, worker collaboration and communication, and apps consumers use to purchase your products and services. Software has become the lifeblood of business activity.



In a software-dependent world, one of the most important management considerations for any business involves the capability to deliver software that makes that business unique, competitive, and innovative. To achieve a level of business agility necessary to thrive in an accelerating business environment, businesses must be able to quickly modify not only products and services but business processes too. That means having the ability to rapidly modify software. How do businesses do this?

Unfortunately, many businesses do a poor job of it, and the reality for them is that they will not keep up. They will ultimately lose to their more agile competitors.

That said, many businesses are discovering that they can accelerate software development and delivery to meet real-time business demands. They are doing this through a development process known as Continuous Delivery.

Continuous Delivery is a highly collaborative approach to software development that breaks down silos between traditional job roles so that everyone is responsible for a successful release. In a successful Continuous Delivery environment, it is not possible for a person in one role to pass a problem on to the next person in the process.

Continuous Delivery improves the quality of code by providing instant feedback to developers. How fast is "instant?" In a Continuous Delivery environment, newly checked-in code can be built and tested in minutes. Developers get feedback while the solution is still fresh in their minds.

Continuous Delivery also minimizes risk by decoupling deployment from release. All code is developed and tested to a fully production-ready state. Problems that show up during testing become the entire team's top priority. When the code is truly done, release to real production use can happen at any time, with confidence that "turning on" the functionality will not break anything.

It's not difficult to see the value in working this way. The real question is, how do you get there?

What Can Continuous Delivery Do for You?

17-23%
improvement in business performance because of increased revenue, faster time to market, improved competitive positioning, and enhanced customer experience[4]

30 times
more frequent code deployments[5]

50%
fewer code failures[5]

12 times
faster at restoring service[5]

Less time
fixing code

Significantly lower
production downtime caused by software problems

A Definition of Continuous Delivery

Continuous Delivery is an end-to-end process for taking a functional requirement from concept to production as fast as is necessary to meet business needs. It depends on frequent deployments of small units of code rather than infrequent deployments of big releases. Successful continuous delivery also depends on DevOps—deep collaboration between development and operational IT functions. DevOps is not the same thing as Continuous Delivery; rather, it is a necessary component of Continuous Delivery.

Recent surveys show that Continuous Delivery provides up to a 23 percent improvement in business performance because of faster time to market, improved competitiveness, greater customer satisfaction, and increased revenue.[1] Surely this means that businesses are rushing to implement Continuous Delivery processes of their own, right? Well, not exactly.

Some companies have adopted Continuous Delivery and made it a central aspect of their culture—Amazon and Facebook are good examples—but many struggle with the concept. This paper outlines six steps that any organization can take to realize the advantages of Continuous Delivery. Before reviewing those steps, however, let's first see why some organizations struggle with its implementation.

Many Recognize the Advantages, but Obstacles Stand in the Way

Business managers know that if they could iterate process and product changes on a dime, they could do more than simply react to competitive challenges. They could also disrupt the market with their own competitive vision. This is what they need from their IT people.

Development leaders and IT organizations are torn between these business mandates and the knowledge that they can only push their developers so far before unstable code starts to appear. Many in development and on IT teams see the theoretical advantages of properly executed Continuous Delivery. They see data showing that Continuous Delivery produces faster release cycles, cleaner code, and applications that are more reliable in production. Even so, many organizations believe that implementing such a process is not possible in their situation. In fact, many organizations fail to implement Continuous Delivery for a lot of the same reasons:

- The process changes are too complicated, especially for a team and IT department already stretched thin.
- Few beyond the work group recognize the specific advantages of Continuous Delivery.
- No management structure is in place to support the close DevOps cooperation needed for successful Continuous Delivery.
- There is a general lack of buy-in from upper management.

3 Common Myths of Continuous Delivery

1	**Continuous Delivery Is Too Complicated**	Not true	The path to Continuous Delivery is an incremental one that begins wherever you are now. Starting small and implementing incrementally is something any organization can do with minimal workflow disruption.
2	**High Throughput Means More Unstable Code**	Not true	Frequent deployments of small units of code are less risky than infrequent deployments of big releases. Data show that code developed in this way is more stable in production.
3	**Automated Deployment and Testing Are Riskier Than Manually Controlled Processes**	Not true	It is well established that repetitive manual tasks are highly prone to human error. Data show that 60 percent of production failures are caused by human error.[2]

Overcoming Legacy Practices: Big Bang vs. a Phased Approach

Most IT organizations are bound by processes, procedures, and workflows built up over years. People in those organizations work hard to deliver stable code as fast as is practical. Everyone has a role to play. Developers create code and verify that it works, then they hand it off to someone else for integration testing. At that point, the developers are done, except when the code breaks in a test or production environment or something breaks after release.

In a traditional siloed IT environment, developers spend up to 50 percent of their time finding and correcting software issues3 while working to meet business demands for new functionality. It is no wonder the most common reason for not moving to Continuous Delivery (which would reduce software error rates and time spent "fixing") is that no one has time to make the necessary process changes. They are all too busy "keeping things running."

How can they possibly adopt new methods and tools, begin working in an entirely new and more collaborative way, and implement new roles and functions while still doing all the things they need to do? They can't, of course.

Or can they? When thinking about it in this way, IT managers envision a wholesale, ground-up process change that is huge and likely hugely disruptive. Such an undertaking is referred to as the big bang approach to Continuous Delivery, but it does not need to be that way.

The entire philosophy of Continuous Delivery is all about making many, frequent, incremental changes to an application and then quickly testing those changes so that problems can be fixed right away, while they are still small problems. This philosophy is just as valid for implementing Continuous Delivery itself. Let's see how an incremental approach to Continuous Delivery works.

Where to Begin: A 6-Step Approach to Continuous Delivery

IT organizations come in many shapes and sizes and operate at different levels of process maturity. They do not all need to start at ground zero to build Continuous Delivery into their workflow, but everyone needs to start somewhere. We recommend the following six steps regardless of whether the organization is starting at the beginning, stuck with entrenched traditional practices, or already on its way:

Each of these six steps has specific implications for different aspects of the development workflow. The following table details how these six steps play out in the context of the build, package and deploy, test, and release and monitor phases of a development cycle.

6 Steps

1. **Start Small**

   This is especially important for those organizations starting new or updating a legacy practice. Begin with a project or work group in which change is manageable, goals can be set, and results can be measured.

2. **Get Organizational Buy-in**

   It is essential that both business management and IT team members be committed to developing a Continuous Delivery workflow.

3. **Implement Continuous Integration to Make Integration a Non-event**

   Implement a continuous integration solution along with automating application deployment to testing and staging environments and implementing a basic set of automated unit tests that occur with every build. The goal here is to ensure that all code is production ready at all times.

4. **Reinforce Best Practices, and Automate the Deploy-to-Production Pipeline**

   This means that every piece of code must be accompanied by a unit test ensuring the functionality of the code. Ideally, these unit tests should be automatically executed every build. Builds also need to happen regularly and frequently. Also, the entire team, including business management, need to meet regularly to review which processes are working well and what needs improvement.

5. **Adopt Automation Across the Delivery Process**

   Employ zero-touch continuous builds, automated system deployments, and comprehensive automated testing. Metrics, performance monitoring, and performance analytics are available to the entire team.

6. **Build on Success**

   Expand a successful Continuous Delivery practice to other projects and work groups.

Step	Description	Build	Package and Deploy	Test	Release and Monitor
1	**Start Small**	Begin with a project or a team rather than an entire enterprise. Continuous Delivery changes the traditional workflow of familiar development and operational tasks. Starting small minimizes disruption. A common starting point is to standardize on two essentials: a continuous integration system and a common application platform that spans development, staging, and production. Together, these provide a foundation of consistency and visibility across the stages of application delivery. The next step is to automate one process—application packaging and deployment is a popular starting point.			
2	**Get Organizational Buy-in**	Gain management buy-in for the cooperative DevOps mind-set, and set new expectations for a Continuous Delivery workflow. This includes: • Demonstrating the advantages of Continuous Delivery and adoption of DevOps practices. • Breaking down silos in the traditional development and operations process: ºEveryone is responsible for every release. ºEveryone (developers, testers, business operations, deployment engineers, IT operations) attends regular planning and review meetings together. ºPlan on frequent, incremental releases rather than infrequent, large releases.			
3	**Implement Continuous Integration to Make Integration a Nonevent**	Implement a continuous integration solution to automate the build, packaging, testing, and deployment steps. Builds are scheduled or automatically triggered whenever code is checked in. Build one or more times per day as a best practice. Ensure automatic assembly of application artifacts, including dependencies and configuration to a deployable application package.	Automate deployment to the testing and staging environments, including automatic provisioning of the standardized environment, which should resemble production as much as possible. Automate deployment to specialized test environments to ensure specific quality requirements such as load and capacity testing.	Implement a basic set of automated unit tests. The unit tests are executed by the continuous integration system with every build for faster feedback and quality validation. Immediately respond to failure notifications by giving them highest priority.	Decouple deployment and release. When testing is successfully completed, the application code package should be ready for production at any time. Each group (developers, testers, release engineers, operations) should begin creating standardized dashboard reports.
4	**Reinforce Best Practices, and Automate the Build-to-Production Pipeline**	Every piece of code must be accompanied by a unit test ensuring the functionality of the code. Developers must frequently check code in to the version control system. Establish a build server as part of a standardized build process managed by the continuous integration system. Define how source and artifacts are retrieved for builds on the build server. Automate build steps, and perform frequent builds on a regular schedule.	Use fully automated packaging and deployment. The deployment process should be consistent across all environments and performed automatically using the same application package. Perform a "smoke test" immediately after completing the packaging in which the application is deployed to a test server and the successful functionality of the application is validated.	Automate as many tests as possible to extend instant feedback during the build process by running unit tests on each code file. Automate functional testing to test the deployed application build and verify successful operation. Advance to automating performance and scalability testing. Always report failures and their related diagnostics information so that problems can be fixed quickly and code remains production ready.	The code is always production ready. Toggling on new functionality becomes a business decision. Production releases should be automated and repeatable to generate zero downtime. Once in production, the application health and performance is monitored so that availability, performance, and other application issues are immediately identified and addressed.
5	**Adopt Automation Across the Delivery Process**	Employ zero-touch continuous builds using the continuous integration solution. Each code commit triggers an automated build-deploy-test cycle ending in a deployable application package when the test criteria have been met.	Employ zero-touch packaging and deployment across environments. Adopt automatic release to testing and staging. With approval, the same build is deployed to production in the same manner using the tested and verified application package.	Ensure comprehensive automated testing at multiple levels starting from unit testing, static code analysis, and deployment package smoke testing, which are done during the continuous integration phase. Continue with automated functional, load, and capacity testing to complete the full test suite and validate the quality of each build.	Real-time health metrics, application faults, and performance metrics are automatically gathered, and the team is notified when critical problems occur based on predefined monitoring and notification rules. Code-level diagnostics in production allow developers to troubleshoot production problems quickly and efficiently without the need to reproduce the problems in the lab.
6	**Build on Success**	When a Continuous Delivery infrastructure and workflow have been established, extending it within the organization becomes a process of replicating the practices in additional application groups. Most companies find that adopting Continuous Delivery across the organization and with different business applications is accelerated when there is a higher level of standardization on tools, techniques, and application platforms, which allows processes and practices to be replicated and shared between application delivery processes.			

Best Practices for Successful Continuous Delivery

Implementing a fully functional Continuous Delivery process is not easy, and it will not happen quickly. Most organizations take two to three years to implement a mature Continuous Delivery practice.

Some aspects of this process are much more challenging than others. For instance, one of the most difficult parts of all is instilling the discipline to develop unit tests along with code. Yet, that is essential for the automated builds and testing that are the foundation of continuous integration and rapid feedback to developers.

Successful Continuous Delivery practices share these characteristics:

- **The Entire Development Team Must Accept That Continuous Delivery Means That Code Is Always Production Ready.** This places the highest team priority on correcting any issue that threatens production readiness, such as a test failure. Successful implementations use system tools to enforce priorities. For example, it may not be possible for any developer to check in new code if there is an active failure notification that no one is working to fix.
- **Strong Version Control Is Critical.** This needs to happen through a source control system that stores every possible artifact (tests, dependencies, configuration elements, third-party components), meaning everything needed to build and provision the entire application environment. Continuous Delivery without version control is like a building without a foundation.
- **Code and Tests Must Be Developed Together, and Tests Need to Run Automatically at Build Time.** This is essential for providing instant feedback to developers and accelerating bug resolution.
- **Automate As Much of the Build-Deploy-Test Pipeline As Possible and As Early As Possible.** This accelerates the development and testing process, and it reduces the possibility of human error.
- **Continuous Delivery Cannot Happen Without Continuous Integration.** Continuous integration is the constant merging of new code into the main trunk and testing it in production environments. It is essential to Continuous Delivery because it is the core process for testing code at build time and fixing bugs quickly. Continuous integration depends on automating these key functions:
  - Automate builds to pull and assemble all application artifacts and dependencies.
  - Automate code quality checks by performing automated unit tests and static code analysis immediately at build time.
  - Automate packaging so that tested code, artifacts, dependencies, and configurations are quickly assembled into deployable packages.
- **Continuous Delivery Cannot Happen Without Automated Provisioning and Deployment.** These are essential parts of a successful Continuous Delivery workflow.
- **Development, Operations, and Business Managers Need Shared Visibility into Application Behavior.** This enables quick feedback on problems, including insights into both the application and the infrastructure, which can lead to rapid problem resolution.

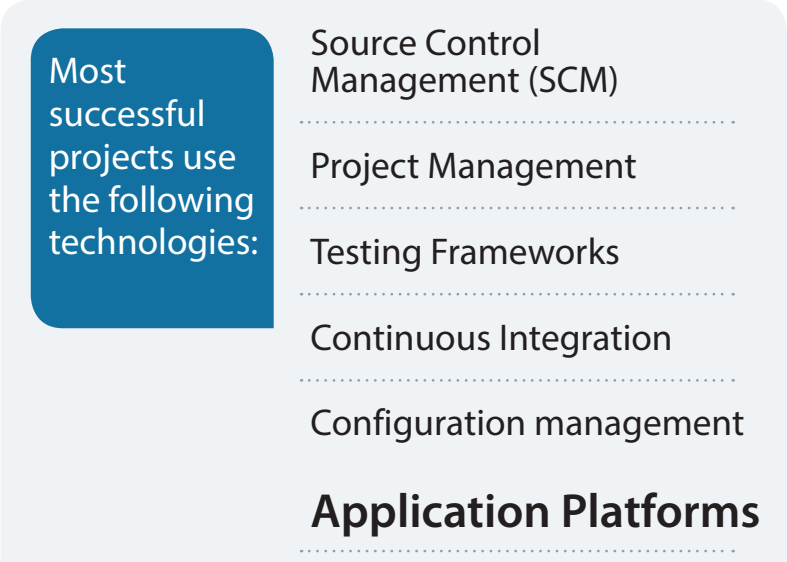
Continuous Delivery Means That **Code Is Always Production Ready**

The Essential Role of an Application Platform for Continuous Delivery

A successful Continuous Delivery practice depends on three key elements; the people that work in it, the processes they adopt, and how well they collaborate towards their common goals. But without the proper technology to support the initiatives, Continuous Delivery is doomed to fail.

It is critical that the participants can gain a common understanding of the status of their project, that they can detect problems early and fix them quickly, and that they can implement new functionality with confidence.

Most successful projects use at least the following technologies to support their efforts:



- **Source Control Management (SCM).** It is critical to be able to determine which parts combine to create releasable software, who has contributed what, how to revert back to the last working version of software and more. The application must be constructed in an automated, repeatable and consistent manner to eliminate configuration and versioning mismatches which mandates that source code and related artifacts be stored in managed repository. Popular systems are Git and Subversion.
- **Project Management.** Teams divide projects in tasks and assign them to persons. Good systems help teams understand their velocity, prioritize tasks, understand their dependencies and alert when things go off track. There are very many systems for this, some of the more popular ones are Asana, Jira, and Pivotal.
- **Testing Frameworks**. Testing can be automated all the way from code unit testing (JUnit, PHPUnit, JSUnit, etc.), to API's (SoapUI and more), functional testing (Selenium), to load (Soasta) and more. Automation is critical to enable reliable test results and fast feedback on the quality of the release, so developers can fix problems before their code moves to staging and production.
- **Continuous Integration.** CI-systems 'see' when new code is checked in to a SCM and fire off the automated processes that assemble the code, provide the dependencies, run tests, and release the software to their target servers. CI-systems like Jenkins and Bamboo are core technology for Continuous Delivery.
- **Configuration management.** It is important that the physical or virtual infrastructure that is used in development, test, staging and production is consistent and predictable – preferably defined in code. Popular tools are Ansible, Chef and Puppet, although many organizations still use homegrown scripting approaches for that purpose.

- **Application Platforms.** Arguably, this is the most critical part of the Continuous Delivery infrastructure. While servers, clouds , networks, and storage are the supporting infrastructure, the application code is driving the business value and the interaction with the customers. All team members, from management, to developers, to QA-teams and operations teams collaborate around the functionality of this platform. In most cases application platforms are specific to the core application language, such as Java (Red Hat's JBoss) and PHP (Zend Server). The application platform needs to support:
    - **Developer Productivity Tools** – For Continuous Delivery projects it is critical that developers be able to find problems before they check in their work and during the entire application release cycle. The more they can inspect the effects of their choices before an app goes into production, the more they can submit code that can be delivered automatically without interruption, directly to production.
    - **Consistent Environments** – The days of 'it worked on my computer, so I assumed it would work everywhere' are over. Having developers work on certified application environments, that have the same runtime environment, libraries and dependencies, as their test, stage and production environment avoids untold amounts of problem.
    - **App & Library Packaging** – When an application moves from one phase to the next in application development, it needs to be packaged in a way that it will find all its dependencies in the expected places at runtime. Applications must be deployed in a consistent, repeatable way to eliminate potential configuration issues and enable automation. This requires detailed knowledge of the application technology as can only be provided by a dedicated application platform.
    - **Cloud Elasticity and Dynamic Scaling** – As use fluctuates, applications need to scale by moved to clusters of front-end and back-end servers, which can be added or removed as needed. An application platform knows how to do this without manual intervention.
    - **Acceleration, Caching, Job Queuing** – In the Continuous Delivery pipeline, developers can see in detail how to optimize their application for optimum performance in production. Enabling them take responsibility for production characteristics of their applications, makes them more motivated and helps them deliver better apps faster.
    - **Robust application monitoring** - With configurable monitoring rules that provide error detection and isolation as well as insight into code execution and performance behaviors. It also needs to include advanced diagnostics. These capabilities ensure better communication and collaboration across the IT organization. And when problems arise, effective monitoring ensures rapid detection and quick diagnosis.

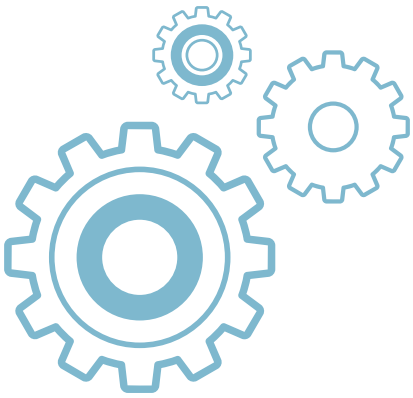
In summary, in addition to placing the appropriate focus on the process architecture and the collaboration approach, successful Continuous Delivery depends on having a platform capable of supporting all aspects of a highly automated build-to-deployment process.

Conclusion

There is plenty of evidence to show that Continuous Delivery allows organizations to deliver code more quickly, and the code they release is more stable. Today's market realities give clear competitive advantage to businesses that manage product and process changes more quickly. For many organizations, the question is not whether they should implement a Continuous Delivery practice but rather how they can do it.

By its nature, Continuous Delivery is an iterative process—many small changes tested and released continuously. This approach to developing code also works for implementing Continuous Delivery. Approaching Continuous Delivery incrementally and in phases is safer than attempting to make Continuous Delivery happen all at once. In the end, a phased approach produces a much stronger and more responsive software development practice.

This paper offered six steps that any organization can follow to implement Continuous Delivery. For many organizations, this process change can be difficult and painful, but it will also be hugely rewarding. In our software-dependent world, an organization's ability to quickly develop new software-dependent capabilities is one of the fastest and most effective business differentiators. An investment in Continuous Delivery could be the best investment an organization ever makes.



References and resources:

[1] ZDNet. (September 17, 2013). "DevOps Really Does Speed Things Up, [CA Technologies] Study Shows."

[2] ZeroTurnaround. (April 9, 2013). "DevOps/ItOps Productivity Report 2013."

[3] Sweeney, Kate. (December 11, 2012). "Software bugs cost more than double Eurozone bailout." Business Weekly.

[4] ZDNet. "DevOps Really Does Speed Things Up."

[5] IT Ops & DevOps Productivity Report 2013: Tools, Methodologies and People, RebelLabs

Corporate Headquarters: Zend Technologies, Inc. 19200 Stevens Creek Blvd. Cupertino, CA 95014, USA · **Tel** 1-408-253-8800 · **Fax** 1-408-253-8801
Central Europe: (Germany, Austria, Switzerland) Zend Technologies GmbH, St.-Martin-Str. 53, 81669 Munich, Germany · **Tel** +49-89-516199-0 · **Fax** +49-89-516199-20
International: Zend Technologies Ltd. 12 Abba Hillel Street, Ramat Gan, Israel 52506 · **Tel** 972-3-753-9500 · **Fax** 972-3-613-9671
France: Zend Technologies SARL, 105 rue Anatole France, 92300 Levallois-Perret, France · **Tel** +33-1-4855-0200 · **Fax** +33-1-4812-3132
Italy: Zend Technologies, Largo Richini 6, 20122 Milano, Italy · **Tel** +39-02-5821-5832 · **Fax** +39-02-5821-5400
Ireland: Zend Technologies, The Digital Court, Rainsford Street, Dublin 8, Ireland · **Tel** +353-1-6908019