**IDC OPINION**

With Java's ubiquitous presence as an enterprise application development platform, it is easy to forget that the language was originally developed for use in embedded devices. A prototype of the platform, then called the Green Project, was first demonstrated on a personal digital assistant device in 1992. Twenty years later, even as Java has become a leading enterprise software platform, it continues to play a leading role in the embedded space. In fact, in the growing and transforming embedded systems industry, Java's set of rich system services are increasingly more attractive.

- Java was the first universal programming environment for mobile devices. In 2011, Java was the single most widely installed programming environment in mobile phone devices, supporting handsets from all the major manufacturers.

- Java is widely used in embedded systems, having made enormous inroads in what is historically a compiled language world. For example, Java is embedded in almost all Ericsson devices shipping today, almost all Ricoh multifunctional printers, and all Blu-ray Disc players. Java is expected to continue to gain traction in areas where devices present an application platform for developers.

- Thanks to the continued march of computing power, Java Platform, Standard Edition Embedded (Java SE Embedded) will increasingly be the platform of choice for a growing roster of embedded requirements, allowing developers to leverage the most familiar Java APIs and the rich ecosystem of Java tools with minimal compromise.

- The Java Platform, Micro Edition (Java ME) and its many device subconfigurations and API subsets are on a road map to be rationalized, aligned, and converged. IDC believes that a modular implementation, Java SE version 9, will begin to simplify the complexity of choices facing adopters.

- The Java ecosystem is healthy and remains on a growing trajectory with more programming languages than ever now hosted on the Java Virtual Machine (JVM), making it possible to adapt Java for more embedded uses.

- The embedded systems industry is projected to continue to expand rapidly, reaching $2.1 trillion in 2015 from $1.2 trillion in 2010. Importantly, the market for intelligent systems, where Java's rich set of services are most needed, is projected to grow from 56% of all embedded systems in 2010 to 78% in 2015.
# TABLE OF CONTENTS

**In This White Paper**  
1

**Situation Overview**  
1

- Embedded Application Characteristics
- Embedded Systems Trends
- The Evolution of Embedded Application Development
- A Compact History of Embedded Java
- Java ME Configurations and Profiles
  - Connected Limited Device Configuration
  - Connected Device Configuration
- Java ME Adoption
- Java SE Embedded
- Benefits of Java for Embedded Use

**Future Outlook**  
9

- Alignment of Java ME with Java SE
- Improved Small Device Development
- Integration of Web Technologies
- New APIs and Other Improvements

**Challenges/Opportunities**  
10

- Framework Proliferation
- Java Evolution Process
- Free and Open Source

**Conclusion**  
11

# LIST OF FIGURES

| P | 
|---|---|
| 1 | Revenue Growth in Traditional Embedded and Intelligent Embedded Systems, 2010–2015 ..... 2 |
| 2 | 2010–2015 CAGRs for Intelligent Embedded Device Revenues by Industry ......................... 3 |
IN THIS WHITE PAPER

This paper examines the state of Java in embedded applications, tracing the history and evolution of Java as a platform embedded in various types of devices. We start with a review of the unique characteristics of embedded systems and the pervading trends in the space. We then examine the various Java subplatforms available for embedded application development. Finally, we assess the road map outlined by Oracle for the evolution of Java to support modern embedded requirements and conclude with an overall assessment of the technology, including the challenges and opportunities it faces.

SITUATION OVERVIEW

Embedded applications are broadly defined to encompass a wide range of requirements, but their essential characteristic is that they constitute components of larger electromechanical systems with which they interface. In contrast to software applications developed and deployed on general-purpose computers, embedded applications usually operate in highly customized environments and impose complex and specific design constraints. Embedded systems have been with us for over five decades, coming into the limelight in aerospace applications such as the Apollo Guidance Computer, which was designed for the specific purpose of supporting NASA's Apollo program and its space missions. While traditional embedded systems will continue to grow, they are also evolving rapidly from fixed function and disconnected systems to more flexible and interconnected systems, which IDC calls intelligent systems.

Embedded Application Characteristics

Embedded application requirements have unique characteristics that set them apart from conventional computing workloads:

- Specificity of purpose of the device, appliance, or industrial setting
- Limitations or constraints on the resources of the hosting hardware platform
- Interfacing with electromechanical sensors and doing so, in many cases, under time constraints without loss of data
- Hardened settings to support exceptional robustness due to requirement for headless operation and minimal attention after deployment

These constraints have a variety of implications, such as extensive testing and security reviews to support the type of hardening often required and to meet the serviceability and supportability profile expected in embedded systems. One sacrifice often made by embedded systems designers is that of transparent portability. Embedded development is known to be more complex and demanding in time and effort than traditional software development. Nevertheless, precisely because of the difficulty of programming embedded systems, and because of the desire to maintain a rich choice of processor architectures to meet design and cost constraints, a portable and productive development platform such as Java has made significant inroads.
Embedded Systems Trends

Embedded systems are computer-based products with a limited range of functions and dedicated applications as opposed to mainstream computing segments, such as personal computers and servers. Examples of embedded systems include mobile phones and tablets, set-top boxes, digital TVs, and network routers, as well as industrial automation, transportation, and medical systems. Embedded systems are undergoing a transition to more intelligent and connected devices, which IDC labels intelligent systems. It is important to note that all manner of embedded systems are affected by the shift to intelligent systems, not just mobile or consumer devices. IDC estimates that embedded systems accounted for $1.2 trillion of system value worldwide in 2010. This figure is expected to grow at a compound annual growth rate (CAGR) of 19% to reach $2.1 trillion by 2015, far outstripping the growth rate of the IT industry overall. Figure 1 highlights the shift in the opportunity between traditional and intelligent embedded systems.

The transition to intelligent embedded systems is being driven by the following factors:

- The growth in semiconductor density propelled by Moore’s law. The trend is for more processors in systems, more cores in processors, and more transistors in cores and at the same time decreases in cost, size, and power consumption.
The trend to automate all manner of devices from household appliances, light switches, and door locks to complex building security systems. Consumer and industrial items are constantly being made more intelligent. More computing technology is embedded in devices than ever, with the growth accelerating and encompassing traditionally dumb devices. Even outside of mobile phones, recent examples of this phenomenon are the proliferation of smart picture frames and the pervasiveness of automobile electronic control systems (e.g., antilock braking systems, airbag sensing systems) in the past 10 years.

An explosion in mobile devices, including tablets, smartphones, and feature phones, which are acquiring increased intelligence on an ongoing basis.

The increasing requirement for continuous monitoring and tracking of events around us to comply with greater security or to monetize behavior (e.g., in-store advertising to mobile consumers).

The expansion in the level of automation between devices leading to systems designed largely for machine-to-machine (M2M) communication and thus no longer directly correlated with the number of human users.

While this transition is taking place across all embedded systems, two sectors in particular are expected to outstrip all others in their growth rates: healthcare and energy. Figure 2 shows the projected compound annual growth rates for the various categories of systems, including computers, consumer systems (including media tablets), and communication systems (including mobile phones and smartphones). New application areas in healthcare and energy, such as telehealth monitoring devices and smart grid home gateway devices, are on the doorstep of more widespread use.

**Figure 2**

2010–2015 CAGRs for Intelligent Embedded Device Revenues by Industry

Source: IDC, 2011
The Evolution of Embedded Application Development

Embedded systems have come a long way from the Apollo days, when custom circuitry was built specifically for an application. Often requiring interfaces with electromechanical systems akin to today’s mobile phone sensors, embedded systems required real-time capability, which implied that the code had to work under very precise timing constraints to capture and process signals from the electromechanical system or sensor without loss. Early embedded application software was written in the processor’s assembler language to maximize control over timing. As computing evolved and a number of popular processors became more widely used, high-level language compilers became more commonly available. The C programming language and its C++ object-oriented evolution eventually became the mainstay for embedded application development, allowing embedded systems vendors to tap into one of the largest ecosystems of skills, tools, and services in the programming language world. For vendors of processors and programmable components targeting embedded systems, delivering a cross compilation environment for C/C++ and a tool chain for application development for the use of the embedded application developer was the first order of business.

A number of end-to-end operating systems and application development tool chains have been on the market, catering to embedded development since the 1980s (e.g., Green Hills Software and Wind River Systems, now owned by Intel Corp.). Typically, hardware vendors have had to license these environments, or work with their vendors in partnership, to ensure that their processors are usable by embedded application developers. The task of preparing new hardware for embedded applications required supporting a relatively complete operating system and tapping the specialized tool chain made available to program for it. The arrival of Java on the scene in the mid-1990s, with its specific focus on portability and the elevation of the abstraction level of programming to a more productive level without a significant sacrifice in performance, was a landmark moment for computing in general and the embedded systems industry in particular.

While higher-level engineering considerations, such as system costs and target environments, dominate decisions around embedded platform selection, technical considerations are also crucial to such decisions. In particular, developer availability and skills, as well as developer productivity and the quality of tools, can determine whether embedded systems projects are successful in the marketplace or not. Other developer considerations include the desired level of code reuse across projects or devices and whether the devices are intended to attract an ecosystem of developers and additional ISVs. From an application development perspective, we observe the following shifts taking place in the embedded space:

- The increased level of embedding entire computing subsystems in the form of complete processor boards or even system-on-a-chip (SoC) designs into electromechanical systems. This trend has driven the use of more complete application platforms to develop, maintain, and manage this computing power.
The increased need to leverage ecosystem dynamics in the selection of platform
technologies. Developers align with platform ecosystems where they are more
likely to adopt affiliated technologies from within the ecosystem rather than from
outside of it. A few large ecosystems have developed around Java, compiled
languages, and Microsoft languages that allow developers and ISVs to tap into
rich tool chains and extensive support communities. Developer ecosystem
dynamics favor a convergence between the embedded and the general-purpose
set of technologies to a certain degree to leverage the best tools and
technologies and the broadest skills available.

**A Compact History of Embedded Java**

The paradox of the term "embedded Java" is that Java was initially designed and
conceived to run in embedded devices. Java was precisely focused on the portability
and productivity problems faced by embedded developers. As such, by design, Java
was aimed at providing developers with a higher level of abstraction in the
programming model and productivity-enhancing services in the runtime, such as
automatic memory management. Soon after Java was born, and perhaps in a
testament to its compelling value proposition, Java was rapidly being licensed and
ported for a variety of general-purpose computing platforms. Sun, being a vendor of
general-purpose computing devices in the form of workstations and servers, was
clearly interested in this trajectory. Nevertheless, the team responsible for bringing
Java to life in the early 1990s continued to promote it for use in embedded devices,
up to and after its formal release in 1995 for general-purpose computing.

In 1991, some two years before inventing Java and over 15 years before the iPhone
changed the trajectory of mobile computing, the Green Project team built a touchscreen
prototype device with wireless connectivity using C++. The device, called Star7, even
ran applications, and a few were developed by the team, such as a remote control
application, an electronic program guide application, and a checkbook-balancing
application. Experience with Star7 was crucial in arriving at the final architecture for
Java with its dynamic JVM technology able to run programs on any device, thus
supporting the concept of a downloadable applet running inside of a browser.

Java was designed with a VM architecture, meaning that the language was coupled
with a runtime that provided an execution machine implemented in software and that
Java language applications were translated to the instructions for this machine just
before they were executed. The VM architecture allowed programmers to code at a
high level of abstraction, and thus be more productive, while retaining the portability
of their application as their projects required transitioning to other processor
architectures. Supporting new processors simply needed a JVM implementation or a
new port of one. Isolating the complexity of delivering portable code to the process of
creating a Java VM for any new underlying processor platform was one of Java’s key
innovations. Typically, Java was made available for the major computing hardware
platforms, and any interested party, such as a vendor of a new processor chip, would
license the Java Platform and port the JVM to its hardware, allowing applications to
run. In this way, the new hardware platform was made programmable by any Java-
skilled developer and was able to take advantage of an ever-growing ecosystem of
tools to support the design, development and testing of Java applications.
Many flavors of Java were birthed and sunsetteted before a more durable formula for embedding Java emerged. As early as 1998 and before Java 2 was released, several forms of Java became available with various subsets of class libraries and JVMs. In those early days, two flavors of embedded Java were aimed at embedded devices: PersonalJava for larger but computing-capacity constrained devices such as set-top boxes and EmbeddedJava for even smaller devices that required minimal or no display capabilities (at the time), such as pagers and printers. Additionally, an ultracompact version of Java known as Java Card was designed for 8-bit smartcard applications with memory requirements as small as 512 bytes.

**Java ME Configurations and Profiles**

With the release of Java 2 in 1998, the embedded configurations were evolved and rebranded so that the family of embedded subsets of Java became branded as Java 2 Micro Edition (J2ME) and the capabilities were organized to handle the variety of different device types and sizes according to a specific scheme. Any particular implementation of J2ME is made up of the following:

- A configuration, which provides the libraries and virtual machine capabilities for a broad class of devices
- A profile, which is a set of APIs that support a specific range of devices
- Optional packages, which support further capabilities through additional technology-specific APIs

In the transition to Java ME, PersonalJava gave way to the Connected Device Configuration (CDC) and EmbeddedJava became Connected Limited Device Configuration (CLDC). Two main levels of Java ME evolved to support devices with varying levels of physical constraints, and both achieved tremendous traction in the embedded space.

**Connected Limited Device Configuration**

This configuration is designed for devices with limited memory, processing power, and graphical capabilities. CLDC is often used with the Mobile Information Device Profile (MIDP) to provide an application environment for mobile phones. Portable applications known as MIDlets are created by Java ME developers to the MIDP API subset and, in theory, can run on any conforming device. In reality, specific MIDlets have to be approved by a carrier to run on specific phones. CLDC with MIDP is used on a large range of popular mobile phone lines, including phones from Samsung, LG, Sony Ericsson, BlackBerry, and Nokia. The Oracle Java Wireless Client is a CLDC-based stack optimized for mobile phones. Because of the diversity of mobile phone devices, players in this field generally provide additional APIs and optional packages to support their devices.
**Connected Device Configuration**

The CDC is a specification of Java ME targeting higher-capacity devices such as set-top boxes. The CDC is more similar to Java SE, offering a richer set of services than the CLDC, such as fuller class verification. A set of profiles of increasing capabilities are available for CDC, including the Foundation Profile, the Personal Basis Profile, and the Personal Profile, each with attendant optional packages. The typical development model for CDC devices involves writing components known as Xlets, which are managed by the device's Application Management System (AMS). One of the most widely used implementations of CDC is the Blu-ray Disc Java (BD-J) standard, which is a part of every Blu-ray Disc playing machine. Other widely used devices that embed CDC include Cisco VoIP phones, Ricoh printers, and the Amazon Kindle eReader.

**Java ME Adoption**

The Java ME platform is a collection of technologies and specifications that can be combined to construct a complete Java runtime environment specifically to fit the requirements of a particular device or market. The process of navigating the choices of configurations, profiles, and options is complex, and while the technology has been widely adopted, the level of compatibility across devices has not been the paramount reason for adoption. What initially drove the selection of Java ME are time-to-market considerations and industry familiarity with the technology. Java ME achieved a network effect and leveraged ecosystem dynamics around the late 1990s and early 2000s to power almost the entire mobile phone industry. IDC estimates the count of mobile phones that have shipped with Java ME to exceed 2 billion. The arrival of the iPhone and the wave of new touch-based smartphones it engendered created competition for Java ME primarily in the way smartphones are displacing lower-end mobile phones. Java ME also has a wide adoption profile outside the mobile industry. Set-top boxes, bending machines, and a host of other types of embedded systems leverage the CDC configuration of Java ME. Embedded systems outside of mobile devices that use Java ME configurations and profiles continue to see significant growth in adoption and may one day rival the mobile phone device counts.

**Java SE Embedded**

Java technology is heavily used in embedded systems, and it can be argued that the proliferation of configurations and profiles for Java ME was the result of the need to leverage the ever richer Java ecosystem in the embedded space. A platform that is being considered more often for embedded applications is the Java SE platform.

Java SE Embedded, based on the latest Java SE 7 and Java SE 6 releases, is ideally used for systems supporting 70MB of memory or more. Java SE is a rich platform with a mature development environment that has benefited from the extensive refinement of large-scale usage in general-purpose settings. This technology is available in both headless and headful forms for the major operating systems, such as Linux and Windows on SPARC and x86 processors. For embedded devices that do not require a screen with graphical user interface, a headless form can be used. The headless form is a smaller footprint JRE, with up to 50% savings in space compared with the full JRE. Update 2 of Java SE Embedded 7 brings Server Compiler support on the ARMv7 architecture on Linux.
One major accommodation made by the Java SE Embedded release for low-memory environment is the ability to start with less default virtual memory allocated. Additionally, Java SE Embedded uses the latest garbage collector algorithm, Garbage First (G1), and features the option of client and server JITs. The distinguishing characteristic for Java SE Embedded is that despite the accommodations provided for embedded environments, the platform is Java SE compliant. This gives embedded system developers the option to choose a platform that leverages many of the advantages of full-blown Java for their development process.

Java SE Embedded 7 has been gaining traction, and the technology has been adopted in various medical application areas, such as imaging devices, where there are no hardware constraints but the need for high performance on modern multicore processors and reliable operation is critical.

Benefits of Java for Embedded Use

Java was designed to combine the productivity and portability of abstraction with the performance and compactness of native compilation. No matter which version of Java is used for embedded applications, Java SE or Java ME, and there are certainly trade-offs that system designers must weigh as they make their selection, a variety of benefits ensue when the use of Java is compared with the use of C or C++. Benefits include:

- Developer tools such as the comprehensive free NetBeans IDE with its built-in support for Java SE and ME capabilities, including the specific configurations, profiles, and APIs by the Java ME environment. The Java tool chain, whether in open source and community-developed software or commercial products, is one of the richest in the industry, including strong ecosystem support for all phases of the application life cycle. Developers can also bring to bear the Eclipse IDE and its expansive ecosystem of tools and plug-ins for more demanding projects.

- Code reliability and security as well as developer productivity stemming from Java’s strong type enforcement, array bounds checking, and elimination of pointer address arithmetic from the hands of developers. In Java, the runtime environment provides a variety of checks to avoid common security holes found in software, and developers are saved from the temptation to leverage address arithmetic in ways that are error prone and difficult to maintain.

- Developer productivity and maintainability improvements stemming from the addition of an exception handling mechanism and a portable and hardware-independent concurrency model through Java multithreading. The lack of the goto statement, which is replaced in Java with break/continue reserved words to handle loop exits and resumption, provides additional readability benefits.

- Readability advantages that ensue from the elimination of preprocessing steps available low-level compiled languages and thus providing improved program readability and maintainability.
- Overall efficiency and performance due to the compactness of bytecode and JAR file formats. In addition, bytecode verification conducted automatically by Java performs a variety of soundness and safety checks on program code.

- The flexibility of being able to access non-Java native code through the Java Native Interface (JNI) specification to accommodate either specific device-dependent functionality or the integration of existing code modules.

- The productivity advantages of automatic garbage collection, although this capability may be unsuited for some real-time embedded applications because of the nondeterminism in the garbage collection process.

**FUTURE OUTLOOK**

Since its acquisition of Sun Microsystems in January 2010, Oracle has been at work organizing and streamlining the Java offerings for embedded applications. Along with the many improvements that Oracle pushed through for the Java platform in general are specific road map items intended to simplify the choices and enhance the value proposition for Java in the embedded space. During the JavaOne 2011 Keynote, Oracle painted a picture of reinvigorated investment in Java for Embedded in the following key areas.

**Alignment of Java ME with Java SE**

It is expected that Java 9 will bring modularity features to the Java platform when it ships in 2015. The modularity of the complex and expansive Java SE platform will enable it to adapt to a variety of embedded form factors without compromising compatibility because modules not needed for a specific device can be eliminated to reduce Java's footprint. Thus, the CDC configuration of Java ME is expected to evolve into a Java SE 9 profile with the new virtual machine, based on the converged Oracle JRockit and Hotspot JVMs, also absorbing key CDC VM features. JavaFX is planned to be the graphics framework for the converged platform. Overall, this is expected to bestow the much-needed compact footprint for embedded applications with great performance, tools, and powerful user interface technology.

**Improved Small Device Development**

The next release of Java ME is planned to be in better alignment with Java SE, though significant alignment may not come until Java SE 9. The project calls for improving consistency between Java ME configurations through an update to the CLDC VM and libraries. New Java 7 language features will be ported to the CLDC VM, and the tooling interfaces will be brought into consistency. It is expected that new APIs will be introduced to handle mobile M2M and small device features. Finally, it is hoped that release timing will be synchronized across ME and SE starting with this release.
Integration of Web Technologies

Oracle aims to bring together Java embedded applications and JavaScript so that multiple application models (Java applications, Web applications, Web widgets) are supported. In particular, Oracle is investing in further integration and bridging between the Lightweight User Interface Toolkit (LWUIT) technology for Java ME and JavaScript applications.

New APIs and Other Improvements

To support embedded settings, Oracle plans new APIs for device access, graphics, nearfield communication, IMS, sensors, payment, telephony, location, and other mobile services. Oracle is also expected to continue to improve the Java ME JVM and the class library footprints. Work on CPU performance and efficiency across the various configurations and runtimes continues.

Challenges/Opportunities

Java has penetrated and occupied a broad chunk of the embedded application development space. Despite unprecedented success in mobile and embedded devices of all sorts, Java faces some challenges as it continues to compete for embedded workloads, and most of these challenges present Oracle with an opportunity to invest to improve the situation.

Framework Proliferation

Perhaps because Java has been pulled into such a broad range of devices, the number of configurations, API subsets, and JVM variations has multiplied. This proliferation has meant that systems designers and developers are presented with complex choices that can waste time and energy to evaluate. The choices developers make can have significant consequences because they permanently define the nature of applications written. Earlier decisions to implement a modular JVM and class libraries would have avoided many of the issues later with the different ME configurations or would have obviated the need to implement a Micro Edition subset in the first place. Oracle is now in the process of mending this problem by introducing modularity in Java SE. The path to convergence of the various frameworks will extend through multiple releases, and a more detailed road map for how CLDC can be cleaved out of a modular Java has to be outlined. Oracle has an opportunity to define a new set of aligned configurations as it evolves Java to further enhance its appeal and adoption.

Java Evolution Process

Java is evolved in a highly structured fashion through the Java Community Process (JCP). The process deals with the sensitive issue that while Java is the intellectual property of one vendor, it needs to be evolved through a trustworthy process to achieve the wide-scale adoption that will make it a useful standard. The JCP as a governance process has been instrumental in ultimately growing the Java ecosystem
to its current state. The trade-off with the JCP is the overhead it imposes on the speed of decision making. Recently, Oracle made the difficult decision to shift most of the modularity work promised for SE 8 into SE 9 — a delay of an additional two years. For a set of technologies as broadly used as Java, a predictable schedule is critical and potentially more important than shipping functionality. However, Java does not operate in a vacuum. There are many ongoing efforts in the industry to forge and build application platforms, not least of which is the Microsoft platform, which has successfully created a parallel ecosystem focused on Microsoft’s Windows operating system. The high level of democracy in the JCP has arguably cost Java its ability to keep up with the latest and greatest mobile devices. However, the JCP has also produced a scale of ecosystem not likely to be achieved without it. Oracle is unlikely to do away with this process because promulgating Java as a standard is one of Java’s key value propositions and indeed one of Oracle’s key marketing messages around its entire product line. Since the Sun acquisition, Oracle has steered the JCP, which was stalled in several places, more assertively. The measure of governance for Java is its decisive evolution and the predictable delivery on the road maps outlined for the various component technologies.

**Free and Open Source**

Java SE is open source through its reference implementation, OpenJDK, and is free for organizations that respect the GPL license restrictions. Java ME is partially in open source but is not free. Licensing Java ME pays for its maintenance and evolution, much like most software has been monetized in the past two decades. Despite this model, Java ME has gained huge adoption. Nevertheless, the question remains as to whether the open source community and more specifically the commercial ecosystem is satisfied with a GPL-style license or whether a more permissive Apache-style license is needed to maintain high levels of community investment. Java licensing may be costly, but many organizations prefer a supported model from the technology vendor. It can be argued that many platform adopters prefer a more controlled governance model such as the JCP, which provides for the support of industry standards and guides Java’s evolution in a way that ensures congruence and compatibility of Java’s ecosystem of enterprise products and services. Java’s traction and growth in the future will depend on Oracle getting the tension between governance and freedom right.

**CONCLUSION**

The measure of success for any new software platform is its ability to engender an ecosystem of additional vendors and technologies working both collaboratively and competitively to build added value. Java has been successful in building one of the largest platform ecosystems known in modern computing. The value of an ecosystem is reflected in the richness of the available products and services, which, once they reach a critical mass, can lead to accelerated adoption and long-term strategic safety and viability coveted by users of the technology. Developers feel safe investing in learning and building skills around Java and its ecosystem, precisely because of the scale of Java’s ecosystem.
Embedded systems are perched on the doorstep of a massive transformation that will take them to higher levels of intelligence. New intelligent embedded systems will experience tremendous growth in the next few years, especially in areas such as energy and healthcare. The demands of intelligent embedded systems are in strong alignment with the value proposition of the Java Platform. In its various subsets, Java offers a set of choices suitable to a broad set of needs and a road map to further improve the productivity of the Java ecosystem.

Copyright Notice

External Publication of IDC Information and Data — Any IDC information that is to be used in advertising, press releases, or promotional materials requires prior written approval from the appropriate IDC Vice President or Country Manager. A draft of the proposed document should accompany any such request. IDC reserves the right to deny approval of external usage for any reason.

Copyright 2012 IDC. Reproduction without written permission is completely forbidden.