The Intelligent Internet of Things with Axeda and Oracle Java Embedded

by Eric J. Bruno, Oracle
Jeremy Kornwitz, Phil Lombardi, and Mats Samuelsson, Axeda

ORACLE WHITE PAPER | JUNE 2014

Disclaimer

Table of Contents

Disclaimer 1

The Technical Requirements of the Internet of Things 1

Cloud-Based Internet of Things 2

Java and the Internet of Things 3

- Java and Native Development 3
- Java ME and the Internet of Things 3
  - What's New in Java ME 8 4
  - Java ME 8 Profile Details 4
  - End-to-End Connectivity and Enterprise Integration 5
- Java SE Embedded 8 and the Internet of Things 6

Axeda Java ME 8 Toolkit Library and Agent Solutions 8

Getting Started with Java and Axeda 9

The Technical Requirements of the Internet of Things

Wikipedia defines the Internet of Things (IoT) as: "The next wave of technology evolution where billions of devices have a virtual representation in the Internet. It is a world where devices communicate with each other, the environment and users, a world where appropriate information can be accessed in the right quantity and condition, at the right time and the right place." Although IoT often relies upon the use of embedded devices, distributed geographically and communicating via the Internet Protocol (IP) to help people bridge the online and physical worlds, it's not restricted to this. We're just now beginning to see the potential value of IoT, and its forms and uses will extend well beyond what we know today.

IoT promises to revolutionize the world by connecting billions of microcontroller-based devices with millions of web-based applications and billions of users in new ways. While driven by many industry participants as the next frontier or revolution in human existence, there are substantial underlying technologies and costs driving these new possibilities.

With new IoT modules and systems-on-a-chip (SOC), it is now possible to create connected devices without adding substantial product costs. Moore's law ensures that LAN, WiFi and Cellular, Bluetooth and LPAN connection technologies can be added at low incremental cost. Removing this barrier to entry creates tremendous opportunities for new products, applications and business models. As in any new industry, initial examples may seem strange (do we really need a connected refrigerator, thermostat, or car, for example?) but after a while, people around the world will wonder how they could have lived without them. In the history of electronics and consumer products, we have seen this time and time again.

As with all technology-based businesses, new products and applications need a technology foundation to build upon, tools to use, and a supplier infrastructure to provide the components. Eventually each new industry creates its own framework that incorporates existing technology and new piece parts. While it is difficult to accurately forecast what the structures and frameworks will be, a safe bet is that IoT will use some key aspects of the current technology industry applied in new ways, such as:

» Microprocessors and communication chips from the rapidly evolving semiconductor industry where Moore's law brings a doubling of capabilities every 18 months. Industry players compete to deliver new chip solutions for current and potential markets, and IoT is very hot in this industry.

» The Internet with its combination of device-based agents and web servers that today deliver billions of application uses every day.

» Cloud computing with its ability to seamlessly support applications and computing anywhere in the world, at any time.

» Connectivity infrastructures such as mobile, wireless and fixed networks that are constantly looking for new ways to drive revenue.

» Development, design and manufacturing tools needed for the creation of new products and applications.

» New innovations that will end up being catalysts for IoT's rapid creation and growth. Examples range from ubiquitous connection technologies, universal device registration/authentication and new unheard of services and applications.

» Business models for growth providing positive feedback for ease of use and adoption.

For this to come together there is need for devices, connectivity and application frameworks that are not only good enough and easy enough to be widely adopted, but low cost as well. The industry needs to evolve from existing solutions (A.K.A beachheads) that are widely used and deployed today, to provide a foundation for further growth while addressing key requirements. To that end, IoT needs to:

- Be as simple as turning on a device and logging on to a web site.
- Leverage standard application and web development tools such as Java and HTML/HTML5/JavaScript.
- Make use of existing communications infrastructures such as the Internet and telecom networks (mobile and wired) in order to support connectivity for billions of devices.
- Make use of existing hardware and software technologies while driving the customization and evolution of them. This includes leading hardware technologies, software platforms and tools.
- Make use of existing computing infrastructures in order to support the data processing and applications opportunities for which the Internet of Things will pave the way. This includes scalable distributed cloud computing and data management solutions, based on flexible deployment scenarios like SaaS, PaaS and IaaS.

The right combination of these existing solutions will help deliver robust implementations in a timely manner, while avoiding the need to reinvent every aspect of IoT solutions from scratch.

Cloud-Based Internet of Things

Cloud computing can potentially address many of the challenges of IoT. For example, the cloud can:

- Connect millions or billions of devices worldwide
- Handle and host large numbers of cloud based applications that use data from IoT devices
- Facilitate a common data model so that different types of devices can speak a common language and be used by a wide range of applications for different purposes
- Manage large amounts of data generated by IoT that needs to be processed or stored
- Provide management solutions for IoT devices to handle device problems, connectivity, service and updates
- Create a secure communication and data layer between IoT devices and their applications in an attempt to be hacker-proof
- Provide tools that make it easy to develop and deploy complete end-to-end IoT solutions that are competitive in the market place

The Axeda IoT Cloud Platform is a secure and scalable cloud solution to connect, manage and deploy IoT solutions. Based on years of experience with demanding industrial customers, the cloud-based platform handles all the key functions of an IoT cloud, such as:

- The basics:
  - Connectivity - how to connect via LAN, WiFi, Mobile, protocols, and so on
  - Device Management (managing billions of devices without interfering with device use)
  - Scalability and deployment
  - Data Processing, Management and Storage Business/Consumer Application Support (APIs and Web Services for easy application access to device data and control of devices)
- The applications and integrations: easy tools and simple IoT application development solutions
  - Consumer facing
  - Business facing
  - Business integration
  - Operational integration

- Device technologies that provide easy ways to create IoT products
    - Plug-and-play device support
    - Software integration support
    - Chip set, SOC and module support
    - Support of industry standard development tools

While the basic and application/integration functions are part of the IoT cloud solution, device technologies will always be supplied by companies specializing in this space. This includes device manufacturers, software vendors and semiconductor manufacturers. However, the most critical components of the entire IoT infrastructure are the development tools and environments that simplify and accelerate development of new IoT devices and services. The world of billions of connected devices requires the ability to develop these in a fast, efficient and competitive way. Java, the most widely used development language in the world along with Oracle's Java Virtual Machine, will play a critical role in enabling IoT devices.

Java and the Internet of Things

Java, born on an embedded device as part of Project Green, is the most widely used development language in the world with over 9 million developers. Given its ability to run on a wide range of devices—from sensors with limited CPU and memory to rack mounted servers—no other technology today is better positioned to power an IoT strategy than Java. Java-powered devices in this ecosystem will communicate with each other, with users, and with the real world via sensors and relays to gather knowledge and to reason about the world around us in real time.

Oracle Java ME Embedded and Oracle Java SE Embedded are ideally positioned to be the foundation for device and gateway application development. By using a combination of these VM offerings, Java can be used for IoT development, end-to-end, from the device to the data center.

Java and Native Development

With Java's built in just-in-time (JIT) compiler—present in Java SE, Java SE Embedded, and Java ME —Java also makes for an excellent alternative to native C/C++ embedded development. JIT compilation compiles Java class files down to machine code, capable of executing as fast as or even faster than compiled C applications. This is possible due to the Java virtual machine's (JVM) ability to dynamically optimize the machine code as the application executes; something statically compiled languages cannot.

With binaries available for many embedded platforms, including ARM, PowerPC, and Intel, Java eliminates the need to build custom tool chains for native development, and insulates the embedded developer from changes in hardware platforms as well. Whether you're moving from development boards to production devices, upgrading your device hardware, or changing hardware platforms altogether (i.e. from PowerPC to ARM), your Java applications will continue to work without requiring any changes.

Java ME and the Internet of Things

Java ME is a proven embedded platform with over 15 years of maturity, and is based on a multitasking virtual machine highly optimized for each of the hardware platforms (i.e., ARM9, ARM11, ARM Cortex-M) it was designed to run on. Java ME includes support for a wide range of connectivity, including wired and wireless networking and a variety of peripheral devices, all with low memory and processing requirements.

Overall, Java ME is purpose built to connect embedded devices of all sizes. It enables headless operation, a robust and secure application environment, and remote software provisioning and management, all built on the industry standard Java language and VM. This allows you to decouple your application from the multitude of variations of hardware and operating systems in the embedded world. With Java ME, you don't need to worry about tool chains or related complexities—you just use Java to focus on business value and time to market.

What's New in Java ME 8

The release of Java ME 8 includes a convergence of both Java SE and Java ME, enabling a more consistent developer experience and more code and library re-use. This includes Java SE language features such as generics, assertions, enumerations, autoboxing, and more, as well as a number of popular Java SE APIs such as java.util.* and Collection classes.

An additional alignment feature is the binary compatibility between Java ME 8 and Java SE 8, meaning that Java .class and .jar files can now be shared and re-used across Java 8 VMs. Java ME 8 accomplishes this while still maintaining backward compatibility with earlier versions of Java ME, along with its focus on resource constrained devices.

Java ME 8 Profile Details

Java ME 8 contains a set of profiles meant to target different classes of devices based on a common range of capabilities. Optional packages may be added to a profile set to further customize Java ME 8 to your target device's use case. Table 1 below describes each profile, along with the packages within, and the footprint in terms of memory and storage requirements.

TABLE 1: JAVA ME 8 PROFILES AND REQUIREMENTS

Profile Description	Storage and Memory Requirements
Java ME Embedded 8 Minimum Profile Set » *Required core APIs* » *Application model and packaging*	Minimum: » *128KB RAM (*see note)* » *1MB Storage (i.e. Flash)* Recommended: » *256KB RAM* » *2MB Storage*
Java ME Embedded 8 Standard Profile Set » *MEEP 8 Minimum Profile Set* » *Multi-tasking* » *Application management* » *Shared libraries* » *Events* » *Enhanced security model* » *Optional packages based on use case*	Minimum: » *512KB RAM* » *2MB Storage* Recommended: » *1MB RAM* » *3MB Storage*
Java ME Embedded 8 IMP-NG Profile Set » *MEEP 8 Minimum & Standard Profile Sets* » *MIDlet, Wireless messaging, Media, Media Control, RMS, PDA, Web services, Security & Trust services, Location, Contactless communication, and XML packages*	Minimum: » *1MB RAM* » *2MB Storage*
Java ME 8 Full Profile Set » *MEEP 8 Minimum & Standard Profile Sets* » *All optional packages*	Minimum: » *2MB RAM* » *4MB Storage*

Note: MEEP 8 Minimal Profile Set, optimized for single- function devices. Actual footprint will vary based on target device and use case.

Target devices or use cases for each profile include:

TABLE 2: JAVA ME 8 PROFILES AND USE CASES

Profile	Target Device / Use Case
Java ME Embedded Profile (MEEP) 8 Minimum Profile Set	Smallest set of smart devices (i.e. sensors,)
MEEP 8 Standard Profile Set	Low-end embedded systems, Mid-range microcontroller based devices, industrial control systems, remote monitoring devices, network infrastructure, medical devices, automotive, and so on. Capabilities and footprint increase as optional features are added.
MEEP 8 IMP-NG Profile Set	Allows execution legacy Java ME IMP-NG applications.
MEEP 8 Full Profile Set	High-end microcontroller based devices, full Java functionality, i.e. home multimedia controllers/gateways, multi-function printers

End-to-End Connectivity and Enterprise Integration

Each of the Java ME 8 profiles is ideal for a wide range of devices in the Internet of Things. From smart sensors or meters, to remote communications gateways with complex event processing capability, Java ME offers a good balance of value and efficiency. Unifying your IoT deployment on Java, end-to-end, simplifies the development and deployment processes, frees you from hardware fragmentation, updates and changes, and allows you to leverage a single set of developers and tools.

The Generic Connection Framework (GCF), along with the Wireless Communication and Web Service APIs, allows Java ME 8 based devices to work seamlessly with cloud-based services. This capability is crucial when deploying remote devices and infrastructure as part of an IoT solution. For instance, with GCF you get secure, consistent, IP and IPv6 support with IP multicast, TLS security, reliable modem communication, UDP datagrams, and support for many protocols including HTTP/HTTPS. The included Access Point API and Cellular API offers support for multiple network interfaces including cellular communication, allowing you to build secure remote WANs for sensor arrays or other IoT use cases.

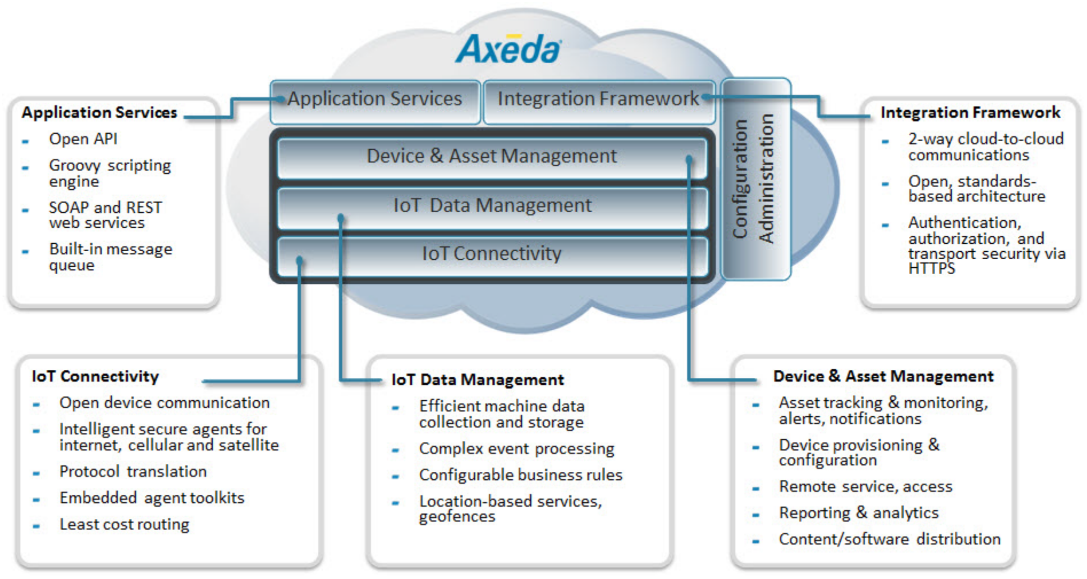
Axeda
Application Services
Integration Framework
Device & Asset Management
IoT Data Management
IoT Connectivity
Configuration Administration
Application Services
Open API
Groovy scripting engine
SOAP and REST web services
Built-in message queue
Integration Framework
2-way cloud-to-cloud communications
Open, standards-based architecture
Authentication, authorization, and transport security via HTTPS
IoT Connectivity
Open device communication
Intelligent secure agents for internet, cellular and satellite
Protocol translation
Embedded agent toolkits
Least cost routing
IoT Data Management
Efficient machine data collection and storage
Complex event processing
Configurable business rules
Location-based services, geofences
Device & Asset Management
Asset tracking & monitoring, alerts, notifications
Device provisioning & configuration
Remote service, access
Reporting & analytics
Content/software distribution

Figure 1. The Enterprise Capabilities of the Axeda IoT Platform

The Axeda IoT platform, which includes Axeda Agent technology and the Axeda Toolkit for Java ME 8, supports end-to-end enterprise capabilities using Java EE technology and the Oracle Database. Features include message queuing for reliable end-to-end data transfer, security based on SSL communication with authentication and authorization, device and asset management, tracking and monitoring, complex event processing, location services, and more (see Figure 1 above).

Java SE Embedded 8 and the Internet of Things

Java SE Embedded 8 brings the same Java VM used for desktop and server Java applications to the embedded world with implementations for x86, ARM, and PowerPC. This includes support for the entire Java SE 8 language and API set, along with compact profiles to help you tailor the footprint to your embedded environment. You can choose from one of the pre-defined profiles, and pick and choose from the optional components using tools that are new in Java SE Embedded 8.

In addition to headless versions, a new embedded version of JavaFX is available for graphical applications such as kiosks. Other new features include a reduced base Java runtime, a smaller VM, faster VM startup times, Java ME CDC convergence, Java Flight Recorder, and most of the new features of Java SE 8 such as Nashorn, Lambdas, and so on. Java Flight Recorder originated from JRockit Flight Recorder as part of the convergence between the HotSpot and Oracle JRockit JVMs, used to profile Java applications.

Out of the box, there are three Java SE compact profiles to choose from, including:

TABLE 3: AVAILABLE JAVA SE COMPACT PROFILES

Java SE Embedded 8 Compact Profile Description	**Storage and Memory Requirements**
Java SE Embedded 8 Compact Profile 1 » *Required core APIs* » *Logging and SSL included* » *Migration path for CDC applications*	Minimum: » *11MB (ARM,PPC), 16MB (x86) Storage*
Java SE Embedded 8 Compact Profile 2 » *Compact Profile 1* » *XML* » *JDBC* » *RMI*	Minimum: » *15MB (ARM,PPC), 23MB (x86) Storage*
Java SE Embedded 8 Compact Profile 3 » *Compact Profiles 1 and 2* » *Management APIs* » *Naming* » *Additional security* » *Compiler API*	Minimum: » *21MB to (ARM,PPC), 27MB (x86) Storage*
Java SE Embedded 8 Full » *Adds all desktop APIs, Web Services, and CORBA APIs*	Minimum: » *49MB to (ARM,PPC), 97MB (x86) Storage*
JavaFX 8 Embedded » *FX Base Graphics →* » *FX Controls →* » *Total FX package →*	Minimum: » *7.8MB storage* » *2.2MB additional storage* » *20.4MB storage total*

TABLE 4: PACKAGES INCLUDED WITHIN EACH COMPACT PROFILE

Compact1	Compact2	Compact3	Full Java SE
java.io	*<Compact1>*	*<Compact1 and 2>*	*<Compact1, 2 and 3>*
java.lang	java.rmi	java.lang.instrument	java.applet
java.lang.annotation	java.rmi.activation	java.lang.management	java.awt .**(13 packages)
java.lang.invoke	java.rmi.dgc	java.security.acl	java.beans
java.lang.ref	java.rmi.registry	java.util.prefs	java.beans.beancontext
java.lang.reflect	java.rmi.server	javax.annotation.processing	javax.accessibility
java.math	java.sql	javax.lang.model	javax.activation
java.net	javax.rmi.ssl	javax.lang.model.element	javax.activity
java.nio	javax.sql	javax.lang.model.type	javax.annotation
java.nio.channels	javax.transaction	javax.lang.model.util	javax.imageio
java.nio.channels.spi	javax.transaction.xa	javax.management	javax.imageio.event
java.nio.charset	javax.xml	javax.management.loading	javax.imageio.metadata
java.nio.charset.spi	javax.xml.datatype	javax.management.modelmbean	javax.imageio.plugins.bmp
java.nio.file	javax.xml.namespace	javax.management.monitor	javax.imageio.plugins.jpeg
java.nio.file.attribute	javax.xml.parsers	javax.management.openmbean	javax.imageio.spi
java.nio.file.spi	javax.xml.stream	javax.management.relation	javax.imageio.stream
java.security	javax.xml.stream.events	javax.management.remote	javax.jws
java.security.cert	javax.xml.stream.util	javax.management.remote.rmi	javax.jws.soap
java.security.interfaces	javax.xml.transform	javax.management.timer	javax.print
java.security.spec	javax.xml.transform.dom	javax.naming	javax.print.attribute
java.text	javax.xml.transform.sax	javax.naming.directory	javax.print.attribute.standard
java.text.spi	javax.xml.transform.stax	javax.naming.event	javax.print.event
java.time	javax.xml.transform.stream	javax.naming.ldap	javax.rmi
java.time.chrono	javax.xml.validation	javax.naming.spi	javax.rmi.CORBA
java.time.format	javax.xml.xpath	javax.security.auth.kerberos	javax.sound.midi
java.time.temporal	org.w3c.dom	javax.security.sasl	javax.sound.midi.spi
java.time.zone	org.w3c.dom.bootstrap	javax.sql.rowset	javax.sound.sampled
java.util	org.w3c.dom.events	javax.sql.rowset.serial	javax.sound.sampled.spi
java.util.concurrent	org.w3c.dom.ls	javax.sql.rowset.spi	javax.swing.** (18 packages)
java.util.concurrent.atomic	org.xml.sax	javax.tools	javax.xml.bind
java.util.concurrent.locks	org.xml.sax.ext	javax.xml.crypto	javax.xml.bind.annotation
java.util.function	org.xml.sax.helpers	javax.xml.crypto.dom	javax.xml.bind.annotation.adapters
java.util.jar		javax.xml.crypto.dsig	javax.xml.bind.attachment
java.util.logging		javax.xml.crypto.dsig.dom	javax.xml.bind.helpers
java.util.regex		javax.xml.crypto.dsig.keyinfo	javax.xml.bind.util
java.util.spi		javax.xml.crypto.dsig.spec	javax.xml.soap
java.util.stream		org.ietf.jgss	javax.xml.ws
java.util.zip			javax.xml.ws.handler
javax.crypto			javax.xml.ws.handler.soap
javax.crypto.interfaces			javax.xml.ws.http
javax.crypto.spec			javax.xml.ws.soap
javax.net			javax.xml.ws.spi
javax.net.ssl			javax.xml.ws.spi.http
javax.script			javax.xml.ws.wsaddressing
javax.security.auth			org.omg.** (28 packages)
javax.security.auth.callback			
javax.security.auth.login			
javax.security.auth.spi			
javax.security.auth.x500			
javax.security.cert			

Java SE Embedded is meant to bring the power and performance of Java SE for desktop and server applications to embedded devices that have 32MB or more of RAM, while Java ME targets devices with as little of 128K of RAM. Java SE Embedded 8 also supports headful embedded applications, including the new JavaFX 8 Embedded GUI runtime for advanced, accelerated, graphics and animation support. Finally, Java SE Embedded 8 compact profiles help to tailor the runtime to your environment, and create a migration path for older Java ME CDC applications.

The deciding factors when choosing between Java ME Embedded 8 and Java SE Embedded 8 are mainly RAM and graphics support. While both runtimes include support for non-x86 based devices (i.e. ARM), Java ME Embedded 8 should be your choice for memory constrained, headless application environments. Java SE Embedded 8 should be chosen when enough RAM is available, to better unify the development and runtime experience with server-based Java application development and deployment.

Axeda Java ME 8 Toolkit Library and Agent Solutions

The Axeda Java ME 8 Toolkit Library is designed to provide a simple, yet powerful way to quickly develop IoT devices using Java ME 8 and connect them to the Axeda IoT cloud. It uses the common domain API model used across Java 8, and a fluent builder API pattern that aligns well with the Java ME HTTP client. The use of asynchronous event handling means that the developer does not have to spend time managing threads.

To make I/O connectivity easy to implement, the toolkit library uses the existing Generic Connection Framework to create Connection APIs that are easy to understand, and also translates specific I/O functions into the IoT data models that are used by the Axeda IoT platform. Examples include:

» The ability to monitor a Passive Infrared (PIR) sensor connected to GPIO pins and send alarms when value rises to high.

```java
// Code excerpt to asynchronously monitor the PIR sensor
// connected to the board and send up an alarm when the
// value rises to high

GPIOPin pirPin;
int pirPinNumber = 7;
try {
    pirPin = (GPIOPin) DeviceManager.open(pirPinNumber);
    pirPin.setTrigger(GPIOPinConfig.TRIGGER_RISING_EDGE);
    pirPin.setInputListener(new PinListener() {
        @Override
        public void valueChanged(PinEvent event) {
            Alarm alarm = new Alarm.Builder("MotionDetected", 500).build();
            Transmission transmission = new Transmission().add(alarm);
            ammpClient.send(transmission);
        }
    });
} catch (IOException ex) {
    Logger.getLogger(RaspberryPiGoKit.class.getName()).log(Level.SEVERE, null, ex);
}
```

» The ability to provide location information.

```
// Method to poll GPS location and send to platform

private static void pollLocation() throws Exception {
    LocationProvider lp = LocationProvider.getInstance(null);
    QualifiedCoordinates qc = lp.getLocation(60).getQualifiedCoordinates();

    if (qc != null) {
        double lat = qc.getLatitude();
        double lon = qc.getLongitude();
        CoordinateLocation cl = new CoordinateLocation(lat, lon);
        Transmission transmission = new Transmission().add(cl);
        ammpClient.send(transmission);
    }
}
```

The Axeda Java ME 8 Toolkit Library allows integration of device logic with IoT platform capabilities, enabling device specific reactions to commands and data sent from the IoT platform. As an example, the Axeda Toolkit can restrict files or content downloads to occur only at specified times, controlled by the IoT device status and other conditions.

The Toolkit Library uses Axeda's Adaptive Machine Messaging Protocol (AMMP), a simple, byte-efficient, lightweight messaging protocol designed to facilitate machine-to-machine (M2M) communications. With it, you can build IoT connectivity directly into your product, with the ability to integrate with the Axeda IoT cloud service.

By incorporating critical IoT concepts discussed above, the Axeda Java ME 8 Toolkit library allows for:

» Rapid Development and Deployment of complete end-to-end IoT solutions.
» Simplified use, with examples and sample code, to enable any Java developer to become a successful embedded developer.
» Support of all relevant IoT use cases with an integrated API and Protocol feature set.
» The creation of secure solutions using encrypted communication.
» Device and application enhancements based on the interaction between devices and the IoT platform, which you define.

The Axeda Java ME 8 Toolkit can be used to develop robust production ready IoT agents on small footprint devices.

Getting Started with Java and Axeda

Getting started with Java and the Axeda IoT cloud is as easy as going to one web site, www.axeda.com/go-java8, which has directions for how to download the toolkits and use them to connect to the Axeda IoT cloud platform. This site also provides quick start examples using platforms like RaspberryPi, Qualcomm GOBI and other preferred developer prototyping environments. The instructions also refer to the Oracle Java 8 development platform and the Axeda Developer Platform:

» http://java.oracle.com
» http://developer.axeda.com

Together, these provide complete documentation associated with developing end-to-end applications using Java 8 and the Axeda IoT platform.

Time for the Internet of Things innovation revolution to begin.