CHAPTER 2

Overview of Java 2 Platform, Micro Edition (J2ME™)

2.1 Java 2 Platform

Recognizing that one size does not fit all, Sun Microsystems has grouped Java technologies into three editions, each aimed at a specific area of today's vast computing industry:

- *Java 2 Platform, Enterprise Edition (J2EE™)* for enterprises needing to serve their customers, suppliers, and employees with scalable server solutions.
- *Java 2 Platform, Standard Edition (J2SE™)* for the familiar and well-established desktop computer market.
- *Java 2 Platform, Micro Edition (J2ME™)* for the combined needs of:
  - consumer and embedded device manufacturers who build a diversity of information devices,
  - service providers who wish to deliver content to their customers over those devices, and
  - content creators who want to make compelling content for small, resource-constrained devices.

Each Java platform edition defines a set of technologies that can be used with a particular product:

- Java Virtual Machines that fit inside a wide range of computing devices,
- libraries and APIs specialized for each kind of computing device, and
- tools for deployment and device configuration.

Figure 2.1 illustrates the Java 2 Platform editions and their target markets, starting from the high-end platforms on the left and moving towards low-end platforms on the right. Basically, five target markets or broad device categories are identified. Servers and enterprise computers are supported by Java 2 Enterprise Edition, and desktop and personal computers by Java 2 Standard Edition. Java 2 Micro Edition is divided broadly into two categories that focus on "high-end" and "low-end" consumer devices. Java 2 Micro Edition is discussed in more detail later in this chapter. Finally, the Java Card™ standard focuses on the smart card market.

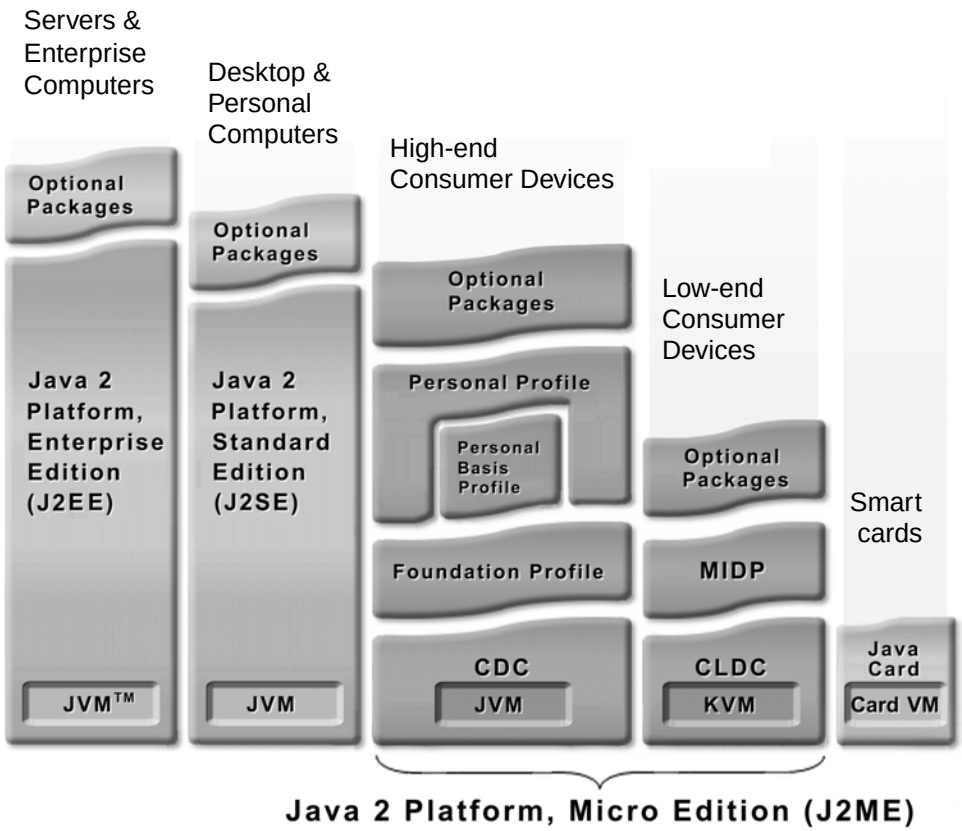

Figure 2.1 Java 2 Platform editions and their target markets

2.2 Java 2 Platform, Micro Edition (J2ME)

Java™ 2 Platform, Micro Edition (henceforth referred to as *Java 2 Micro Edition* or J2ME) specifically addresses the large, rapidly growing consumer space, which covers a range of devices from tiny commodities, such as pagers, all the way up to the TV set-top box, an appliance almost as powerful as a desktop computer. Like the larger Java editions, Java 2 Micro Edition aims to maintain the qualities that Java technology has become known for, including built-in consistency across products, portability of code, safe network delivery, and upward scalability.

The high-level idea behind J2ME is to provide comprehensive application development platforms for creating dynamically extensible, networked devices and applications for the consumer and embedded market. J2ME enables device manufacturers, service providers, and content creators to capitalize on new market opportunities by developing and deploying compelling new applications and services to their customers worldwide. Furthermore, J2ME allows device manufacturers to open up their devices for widespread third-party application development and dynamically downloaded content without losing the security or the control of the underlying manufacturer-specific platform.

At a high level, J2ME is targeted at two broad categories of products:

- *"High-end" consumer devices*. In Figure 2.1, this category is represented by the grouping labeled *CDC* (Connected Device Configuration). Typical examples of devices in this category include TV set-top boxes, Internet TVs, Internet-enabled screenphones, high-end wireless communicators, and automobile entertainment/navigation systems. These devices have a large range of user interface capabilities, total memory budgets starting from about two to four megabytes, and persistent, high-bandwidth network connections, often using TCP/IP.
- *"Low-end" consumer devices*. In Figure 2.1, this category is represented by the grouping labeled *CLDC* (Connected, Limited Device Configuration). Cell phones, pagers, and personal organizers are examples of devices in this category. These devices have simple user interfaces (compared to desktop computer systems), minimum memory budgets starting from about 128–256 kilobytes, and low bandwidth, intermittent network connections. In this category of products, network communication is often not based on the TCP/IP protocol suite. Most of these devices are battery-operated.

The line between these two categories is fuzzy and becoming more so every day. As a result of the ongoing technological convergence in the computer, tele-

communication, consumer electronics, and entertainment industries, there will be less technical distinction between general-purpose computers, personal communication devices, consumer electronics devices, and entertainment devices. Also, future devices are more likely to use wireless connectivity instead of traditional fixed or wired networks. In practice, the line between the two categories is defined more by the memory budget, bandwidth considerations, battery power consumption, and physical screen size of the device rather than by its specific functionality or type of connectivity.

Because of strict manufacturing cost constraints, the majority of high-volume wireless devices today, such as cell phones, belong to the low-end consumer device category. Therefore, this book focuses only on the CLDC and MIDP standards that were specifically designed for that category of products.

2.3 Key Concepts of the J2ME Architecture

While connected consumer devices such as cell phones, pagers, personal organizers, and TV set-top boxes have many things in common, they are also extremely diverse in form, function, and features. Information appliances tend to be special-purpose, limited-function devices. To address this diversity, an essential requirement for the J2ME architecture is not only small size but also modularity and customizability.

In general, serving the information appliance market calls for a large measure of flexibility in how computing technology and applications are deployed. This flexibility is required because of

- the large range of existing device types and hardware configurations,
- the different usage models employed by the devices (key operated, stylus operated, voice operated),
- constantly improving device technology,
- the diverse range of existing applications and features, and
- the need for applications and capabilities to change and grow, often in unforeseen ways, in order to accommodate the future needs of the consumer.

The J2ME architecture is intended to be modular and scalable so that it can support the kinds of flexible deployment demanded by the consumer and embedded markets. To enable this, the J2ME environment provides a range of Java Virtual Machine technologies, each optimized for the different processor types and memory footprints commonly found in the consumer and embedded marketplace.

For low-end, resource-limited consumer products, the J2ME environment supports minimal configurations of the Java Virtual Machine and Java libraries that embody just the essential capabilities of each kind of device. As device manufacturers develop new features in their devices or service providers develop new and exciting applications, these minimal configurations can be expanded with additional libraries that address the needs of a particular market segment. To support this kind of customizability and extensibility, three essential concepts are defined by the J2ME architecture:

- *Configuration*. A J2ME configuration defines a minimum platform for a "*horizontal*" category or grouping of devices, each with similar requirements on total memory budget and processing power. A configuration defines the Java language and virtual machine features and minimum class libraries that a device manufacturer or a content provider can expect to be available on all devices of the same category.
- *Profile*. A J2ME profile is layered on top of (and thus extends) a configuration. A profile addresses the specific demands of a certain "*vertical*" market segment or device family. The main goal of a profile is to guarantee interoperability within a certain vertical device family or domain by defining a standard Java platform for that market. Profiles typically include class libraries that are far more domain-specific than the class libraries provided in a configuration. One device can support multiple profiles.
- *Optional package*. Some APIs are applicable to a large number of devices and device families. A J2ME *optional package* is a set of APIs that is layered on top of (and thus extends) a profile. An optional package typically contains functionality that is independent of any particular vertical market segment or device family. The main goal of an optional package is to allow the definition of APIs that can be added flexibly on top of a number of different profiles. One device can support multiple optional packages.

Configurations, profiles, and optional packages are discussed in more detail below. Configurations, profiles, and optional packages use the capabilities of the Java Virtual Machine (JVM), which is considered to be part of the configuration. The virtual machine usually runs on top of a *host operating system* that is part of the system software of the target device. The high-level relationship between the

different software layers—the JVM, configuration, profiles, optional packages, and the host operating system—is illustrated in Figure 2.2.

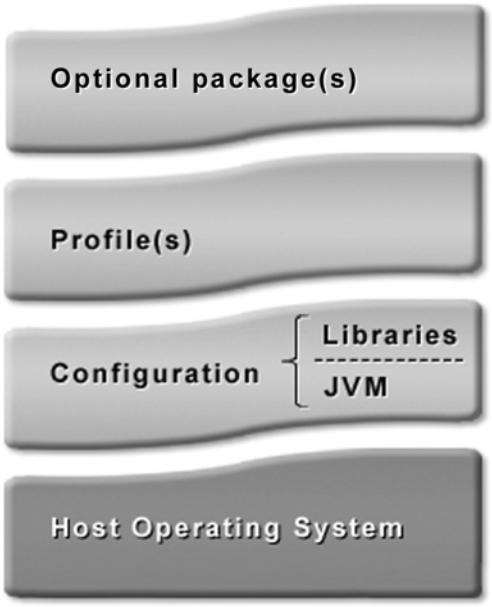

Figure 2.2 Software layers in a J2ME device

J2ME configurations, profiles, and optional packages are defined through industry collaboration using the Java Community Process (JCP). For further information on the Java Community Process, refer to the Java Community Process web site (`http://jcp.org/`).

2.3.1 Profiles

Application portability is a key benefit of Java technology in the desktop and enterprise server markets. Portability is an equally critical element in the consumer device space. However, application portability requirements in the consumer space are very different from portability requirements demanded by the desktop and server markets. In most cases, consumer devices differ substantially in memory size, networking, and user interface capabilities, making it very difficult to support all devices with just one solution.

In general, the consumer device market is not so homogeneous that end users can expect or require universal application portability. Rather, in the consumer

space, applications should ideally be fully portable within the same device family. For example, consider the following types of consumer devices:

- cellular telephones,
- washing machines, and
- electronic toys.

It seems clear that each of these represents a different market segment, device family, or application domain. As such, consumers would expect useful applications to be portable within a device family. For example:

- A discount broker's stock trading application is generally expected to work on different cell phones, even though the phones are from different manufacturers.
- It would be annoying if a highly useful grape-juice-stain-removing wash cycle application available on the Internet runs on an old brand-X washer but not a new brand-Z washer.
- A child's birthday party could be less enjoyable if the new toy robot does not "talk to" or "play games with" the new electronic teddy bear.

On the other hand, consumers do not expect the stock trading application or an automobile service program to run on the washing machine or the toy robot. In other words, application portability *across* different device categories is not necessarily very important or even meaningful in the consumer device space.

In addition, there are important economic reasons to keep these device families separate. Consumer devices compete heavily on cost and convenience, and these factors often translate directly into limitations on physical size and weight, processor power, memory size, and power consumption (in battery-powered devices). Consumers' wallets will usually favor devices that have *the right functionality at the right price*. In other words, the devices must perform the desired functions well, but they do not have added cost for unnecessary features.

Thus, the J2ME framework provides the concept of a *profile* to make it possible to define Java platforms for specific vertical markets. A profile defines a Java platform for a specific vertical market segment or device category. Profiles can serve two distinct portability requirements:

- A profile provides a complete toolkit for implementing applications for a particular kind of device, such as a pager, set-top box, cell phone, washing machine, or interactive electronic toy.

- A profile may also be created to support a significant, coherent group of applications that might be hosted on several categories of devices. For example, while the differences between set-top boxes, pagers, cell phones, and washing machines are significant enough to justify creating a separate profile for each, it might be useful for certain kinds of personal information management or home banking applications to be portable to each of these devices. This could be accomplished by creating a separate profile for these kinds of applications and ensuring that this new profile can be easily and effectively supported on each of the target devices along with its "normal" more device-specific profile.

It is possible for a single device to support several profiles. Some of these profiles are very device-specific, while others are more application-specific. Applications are written "for" a specific profile and are required to use only the features defined by that profile. Manufacturers choose which profile(s) to support on each of their devices, but are required to implement all features of the chosen profile(s). The value proposition to the consumer is that any application written for a particular profile will run on any device that supports that profile.

In its simplest terms, a profile is a contract between an application and a vertical market segment. All the devices in the same market segment agree to implement all the features defined in the profile, and the application agrees to use only those features that are defined in the profile. Thus, portability is achieved between the applications and the devices served by that profile. New devices can take advantage of a large and familiar application base. Most importantly, new compelling applications (perhaps completely unforeseen by the original profile designers and device manufacturers) can be dynamically downloaded to existing devices.

At the implementation level, a profile is defined simply as a collection of class libraries that reside on top of a specified configuration and that provide the additional domain-specific capabilities for devices in a specific market segment.

In our example above, each of the three families of devices (cell phones, washing machines, and intercommunicating toys) could be addressed by a separate J2ME profile. The only one of these profiles in existence at the current time is the MIDP, designed for cell phones and other two-way communication devices.

2.3.2 Configurations

In the J2ME environment, an application is written "for" a particular profile, and a profile is "based upon" or "extends" a particular configuration. Thus, all of the features of a configuration are automatically included in the profile and may be used by applications written for that profile.

A configuration defines a Java platform for a "horizontal" category or grouping of devices with similar requirements on total memory budget and other hardware capabilities. More specifically, a configuration:

- specifies the Java programming language features supported,
- specifies the Java Virtual Machine features supported, and
- specifies the basic Java libraries and APIs supported.

The J2ME environment is designed so that it can be deployed in more than one configuration. Each configuration specifies the Java language and virtual machine features and a set of libraries that the profile implementer (and the applications using that profile) can safely assume to be present on all devices when shipped from the factory. Profile implementers must design their code to stay within the bounds of the features and libraries specified by that configuration.

In its simplest terms, a configuration defines a "lowest common denominator" platform or building block for device manufacturers and profile implementers. All the devices with approximately the same amount of memory and processing power agree to implement all the features defined in the configuration, and the profile implementers agree to use only those features defined in the configuration. Thus, portability is achieved between the profile and the devices served by that configuration.

In our example above, each of the three profiles (for cell phones, washing machines, and electronic toys) would most likely be built upon the same configuration, the CLDC. This configuration provides all the basic functionality to serve the needs of each of these, and perhaps many more, profiles.

To avoid fragmentation, there are a very limited number of J2ME configurations. As depicted in Figure 2.1, only two standard J2ME configurations are available:

- *Connected, Limited Device Configuration (CLDC)*. This configuration focuses on *low-end consumer devices*. Typical examples of CLDC target devices include personal, mobile, battery-operated, connected information devices such as cell phones, two-way pagers, and personal organizers. This configuration includes some new libraries, not drawn from the J2SE APIs, designed specifically to fit the needs of small-footprint devices.
- *Connected Device Configuration (CDC)*. This configuration focuses on *high-end consumer devices*. Typical examples of CDC target devices include shared, connected information devices such as TV set-top boxes, Internet TVs,

and high-end communicators. This configuration includes a much more comprehensive set of Java libraries and virtual machine features than CLDC.

Figure 2.3 illustrates the relationship between CLDC, CDC, and Java 2 Standard Edition (J2SE). As shown in the figure, the majority of functionality in CLDC and CDC has been inherited from Java 2 Platform, Standard Edition (J2SE). Each class inherited from the J2SE environment must be precisely the same or a subset of the corresponding class in the J2SE environment. In addition, CLDC and CDC may introduce a number of features, not drawn from the J2SE, designed specifically to fit the needs of small-footprint devices.

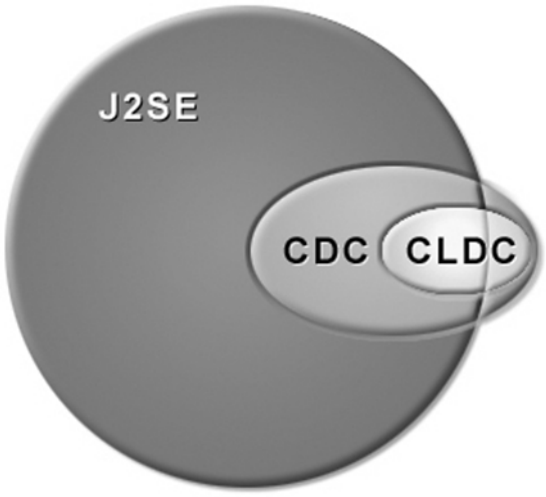

Figure 2.3 Relationship between J2ME configurations and Java 2 Standard Edition

The most important reason for the configuration layer in the J2ME environment is that core Java libraries needed across a wide variety of Java platform implementations are usually intimately tied with the implementation of a Java Virtual Machine. Small differences in the specification of a configuration can require a number of significant modifications to the internal design of a Java Virtual Machine and can require a substantial amount of additional memory footprint. Such modifications would be very expensive and time-consuming to maintain. Having a small number of configurations means that a small number of virtual machine implementations can serve the needs of both a large number of profiles

and a large number of different device hardware types. This economy of scale provided by the J2ME environment is very important to the success and cost-effectiveness of devices in the consumer and embedded industry.

2.3.3 Optional Packages

Soon after the introduction of the original CLDC and MIDP standards, it was realized that, in addition to configurations and profiles, there is a need for additional, general-purpose libraries that are not bound to a single device category or family. For instance, a location API that provides facilities for geographic positioning is potentially applicable to a broad number of devices and device families, and is not functionally limited only to a single profile. Similarly, a wireless messaging API could potentially be applied to a number of different types of devices and profiles. To address the need to create such broadly applicable, device family independent APIs, the concept of *optional packages* was added to the architecture of the J2ME platform.

Generally speaking, a J2ME optional package is an API that can be used to extend a profile. An optional package exposes specific functionality independent of any profile and is naturally distinct from any profile. Optional packages provide APIs that are intended for reuse by profile design teams and vendors of J2ME environments.

It is expected that optional packages may sometimes play an important role in the evolution of a profile. As APIs are developed for new technologies and features, the new APIs are initially developed as optional packages. However, as those APIs mature and new versions are created through the Java Community Process, optional packages may later be incorporated into a profile.

A summary of J2ME optional packages created so far for the wireless market is provided in the next section.

2.4 Evolution of the J2ME Platform

After the successful completion of the CLDC and MIDP standardization efforts, a number of additional J2ME standardization efforts have been launched to complement the functionality provided by the core standards. Most of these additional efforts define optional packages that can be deployed on top of MIDP. These efforts are summarized in Section 2.4.2, “Optional Packages for the Wireless Market” below. In addition, there are a number of more fundamental, core standardization efforts that are summarized in Section 2.4.1, “Core J2ME Standardization Efforts.”

2.4.1 Core J2ME Standardization Efforts

There are a number of standardization activities that have an important role in defining the overall J2ME architecture. These efforts can be viewed as "umbrella" activities that specify the ground rules for a number of standardization activities or provide additional instructions, recommendations, and clarifications for binding together the existing J2ME standards:

- JSR 68: J2ME™ Platform Specification
- JSR 185: Java™ Technology for Wireless Industry

JSR 68: J2ME™ Platform Specification

This specification defines the "ground rules" for the J2ME platform architecture and J2ME standardization activities. It formalizes the fundamental concepts behind J2ME, such as the notions of a configuration and profile, and defines how new J2ME APIs can be formed by subsetting existing APIs from the Java 2 Platform, Standard Edition (J2SE).

JSR 185: Java™ Technology for the Wireless Industry (JTWI)

This specification defines how various technologies associated with MIDP work together to form a complete handset solution for the wireless services industry. The specification provides an exposition of the overall architecture of the wireless client software stack, including a description of the following aspects:

- Which optional packages fit with which profiles?
- How does an end-to-end solution for interoperable Java applications work?
- How does the migration of applications occur, and to which profiles, as the devices become more capable?

A key goal of the *JTWI Specification* is to minimize the fragmentation of the Java APIs in the mobile handset market by creating a community that coordinates the API evolution and deployment as the industry continues to expand the capabilities of mobile devices. To accomplish this, the JSR 185 expert group coordinates the creation of new J2ME-related JSRs and provides recommendations for new optional packages that could take place within the context of the wireless development community as a whole.

2.4.2 Optional Packages for the Wireless Market

The following standardization efforts define optional packages that can reside on top of MIDP:

- JSR 120: Wireless Messaging API
- JSR 135: Mobile Media API
- JSR 172: J2ME™ Web Services Specification
- JSR 177: Security and Trust Services for J2ME™
- JSR 179: Location API for J2ME™
- JSR 180: Session Initiation Protocol (SIP) for J2ME™
- JSR 184: Mobile 3D Graphics for J2ME™
- JSR 190: Event Tracking API for J2ME™

Each of these efforts is summarized briefly below.

JSR 120: Wireless Messaging API

This JSR defines a set of optional APIs that provide standardized access to wireless communication resources, allowing third-party developers to build intelligent connected Java applications. The specification addresses the following wireless technologies:

- Short Message Service (SMS), and
- Cell Broadcast Service (CBS).

JSR 135: Mobile Media API (MMAPI)

This JSR specifies a multimedia API for J2ME, providing straightforward access and control of basic audio and multimedia resources and files. The MIDP 2.0 Sound API (introduced in Chapter 13 of this book) is a subset of the Mobile Media API defined by JSR 135.

JSR 172: J2ME™ Web Services Specification

This specification defines an optional package that provides standard access from J2ME devices to Web services. The JSR is designed to provide an infrastructure to

- provide basic XML processing capabilities,
- enable reuse of Web service concepts when designing J2ME clients to enterprise services,
- provide APIs and conventions for programming J2ME clients of enterprise services,
- adhere to Web service standards and conventions around which the Web services and Java developer community is consolidating,
- enable interoperability of J2ME clients with Web services, and
- provide a programming model for J2ME client communication with Web services, consistent with that for other Java clients such as J2SE.

JSR 177: Security and Trust Services API for J2ME™

The purpose of this JSR is to define a collection of APIs that provide security services to J2ME devices. These APIs are a necessary step for a device to become *trusted*, that is, to provide security mechanisms to support a wide variety of application-based services, such as access to corporate network, mobile commerce, and digital rights management.

Many of these services rely on the interaction with a "security element" (such as a smart card) in the device for secure storage and execution, as described below:

- Secure storage to protect sensitive data, such as the user's private keys, public key (root) certificates, service credentials, personal information, and so on.
- Secure execution, such as cryptographic operations to support payment protocols, data integrity, and data confidentiality.
- Custom and enabling security features that J2ME applications can rely on to handle many valued-added services, such as user identification and authentication, banking, payment, ticketing, loyalty applications, and digital media playing.

The JSR 177 specification defines an access model and a set of APIs that enable applications running on a J2ME device to communicate with a smart card inserted in the device, providing a flexible mechanism to allow service and equipment providers to define secure operations.

JSR 179: Location API for J2ME™

This specification defines an optional package that enables developers to write mobile, location-based applications for J2ME devices. The purpose of the specification is to provide a compact and generic API that produces information about the device's present physical location to Java applications. This specification defines a generic interface that works with most positioning methods, such as GPS and E-OTD.

JSR 180: Session Initiation Protocol (SIP) for J2ME™

The Session Initiation Protocol (SIP) is used to establish and manage multimedia IP sessions. This same mechanism can also be used to provide instant messaging, presence, and gaming services. This specification defines a general SIP API for J2ME devices based on the SIP protocol defined by IETF and 3GPP, and targeting resource constrained platforms.

JSR 184: Mobile 3D Graphics API for J2ME™

This specification defines a lightweight, interactive 3D graphics API, which sits alongside J2ME and MIDP as an optional package. The API is targeted at devices that typically have very little processing power and memory, and no hardware support for 3D graphics or floating point math. However, the API also scales up to higher-end devices that have a color display, a DSP, a floating point unit, or even specialized 3D graphics hardware. The API is designed to make rapid development of compelling 3D applications feasible and is intended to be flexible enough for a wide range of applications, including games, animated messages, screen savers, custom user interfaces, product visualization, and so on.

JSR 190: Event Tracking API for J2ME™

This specification defines an optional package that standardizes the tracking of application events in a mobile device and the submission of these event records to an event-tracking server via a standard protocol. The events can be used for purposes such as billing, usage tracking, application revocation, update notification, reviews and ratings, and so on.

The Event Tracking API is intended to work with devices supporting CLDC 1.0 and later. The API is designed as an optional package that can be used with many J2ME profiles, in particular MIDP 1.0 and MIDP 2.0.

For More Information

For additional information on each effort mentioned above, refer to corresponding pages on the Java Community Process web site (`http://jcp.org`).