CDC: JAVA™ PLATFORM TECHNOLOGY FOR CONNECTED DEVICES

Java™ Platform, Micro Edition
White Paper
June 2005

Table of Contents

Chapter 1

Introduction

From a developer's perspective, the APIs for desktop PCs and enterprise systems have been a daunting combination of complexity and confusion. Over the last 10 years, Java™ technology has helped simplify and tame this world for the benefit of everyone. Developers have benefited by seeing their skills become applicable to more systems. Users have benefited from consistent interfaces across different platforms. And systems vendors have benefited by reducing and focusing their R&D investments while attracting more developers. For desktop and enterprise systems, "Write Once, Run Anywhere"™ has been a success.

But if the complexities of the desktop and enterprise world seem, well, complex, then the connected device world is even scarier. Embedded CPUs vary far more than their cousins in the desktop and enterprise world. Software platforms based on Linux have tremendous architectural benefits, but they don't have stable binary interfaces that promote ease of deployment. User experience models for connected devices vary greatly because their purposes vary more than desktop PCs. Connectivity mechanisms vary with wireless technologies such as cellular and Bluetooth, and attached technologies like USB and Firewire. At the same time, the sheer number of products and the rate of innovation dictates lower volumes, which reduces budgets for system software.

Figure 1 describes the variety of connected devices available today. To provide a software platform for any one of these device categories would be difficult, but to provide a network-deployable software platform that spans the wide range of CPUs, operating systems, and GUI user experience models is very demanding.

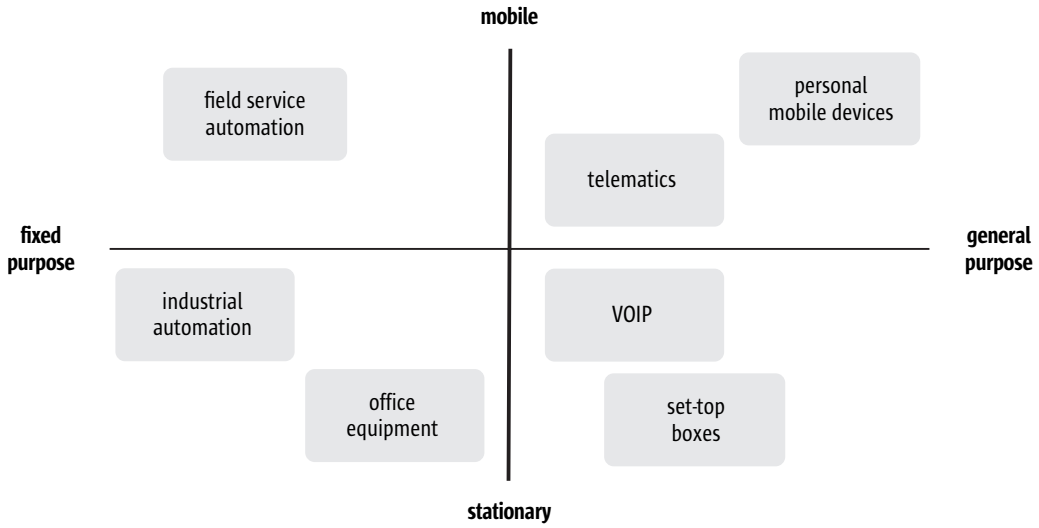

Figure 1: CDC Target Device Categories

Developers, device vendors, and service providers all need a stable and flexible software platform. Complex device matrices represent support nightmares and impede software deployment.

Java technology can provide the same benefits for the connected device world as it has for the desktop and enterprise worlds. The central question is how to handle the variety and change within this world. The Connected Device Configuration (CDC) offers a solution: a standards-based framework for supporting a wide variety of connected devices with Java technology. CDC is based on proven Java technology and is well-equipped to handle the heterogeneous world of embedded processors, operating systems, and GUIs. This white paper describes CDC's problem domain and gives a technical overview of its features and capabilities.

Enterprise Mobility

Connected devices such as PDAs and smart phones have evolved over the last decade from experimental devices to the useful tools they are today. At the core of this evolution is the relationship between the device and the mobile user. Successful product designers have learned to balance the form-factor requirements of size and battery life with intuitive, user-interface mechanisms, while providing software that meets the personal needs of mobile users: scheduling, addresses, and other personal reference information.

The emerging generation of connected devices adds network connectivity to this mix. At Sun, we believe that this addition will spark a new kind of relationship between mobile users and the agile enterprise. While first-generation connected devices provided mobile users with information tailored to their personal needs, connected devices allow them to plug into enterprise applications and Web services to extend the reach of business logic.

With this new relationship come new challenges for software developers to quickly create mobile applications for a variety of connected devices. As we will see, CDC provides a framework for building and deploying mobile applications that easily integrate with enterprise systems.

Chapter 2

Connected Devices in Transition

CDC spans the needs of many different products, applications, and users. To understand this strategy, it helps to see the problems CDC was designed to address.

Connected Devices Today

In recent years, we have seen the following trends in product designs for connected devices:

- **Form factors.** These have remained relatively constant. Hand-held devices such as PDAs and mobile phones are limited by the physical dimensions of their users. Advances in display technology show up as improvements in color, brightness, and pixel density, not in displays that expand the size of the commonly accepted form factors. User-input mechanisms remain simple, usually based on buttons, touchscreens, pen input, and jog dials.
- **Computing resources.** These have grown consistently. While processor speeds have not improved as fast as they have for desktop and server systems, performance of embedded processors is still increasing. Because battery technology does not improve as quickly as processor, storage, and display technologies, power management is a key design criteria for connected devices. Multimedia and time/location-based services add to the core functionality offered by connected devices.
- **Network connectivity.** This is the most important addition to the emerging generation of products. At the transport layer, network connectivity can be persistent across a large network (such as a cellular network) or focused on local connections and transactions (such as a Bluetooth network). Collectively, these network technologies will radically expand the use and integration of connected devices. Access to consumer-oriented Web content and services may be experimental, but secure mobile access to enterprise data will provide real value to corporate users.

What Users Want

The relationship between users and their connected devices has created several key user expectations:

- **Personalization.** The fundamental applications for connected devices help users track personal information such as schedules, contact lists, and other personal reference information. These are often called personal information management (PIM) applications. Over time, as a mobile user interacts with this personal data to modify or add to it, PIM applications become a part of the user's daily routine. In addition to adding personal reference information, users want the ability to configure a device to suit their own tastes and work style. This can range from adding special application software for personal or work needs, to changing the appearance of the GUI to suit the user's taste.
- **Usability.** The software experience for a connected device must be simple and predictable, because users have a low tolerance for learning how to use these devices. Complex user-interface metaphors borrowed from desktop systems are often not successful, because connected devices are smaller and their input mechanisms are simpler. Users perform navigation and selection functions more often than data entry. Single-click navigation techniques are more successful because they work within the user's narrower attention span.
- **Security.** As a mobile user's experience with a connected device expands, the need for security becomes paramount. Mobile users need to know that their rapidly accumulating personal information is safe, and corporate IT departments need assurance that enterprise data is accessed only by trusted users.

What Developers Want

To a developer, the range of connected devices looks like a complex group of devices with different processors, operating systems, and GUI libraries. The effort to learn even a few of the native programming interfaces of these devices is a daunting task, and device-level programming seems unnecessary because enterprise applications mainly access, display, and manipulate data. Faced with so many different devices and interfaces, how can a developer build portable skills as well as portable code?

By focusing on APIs that are shared across a range of devices, developers can focus their energies on domain-specific issues that are important to their customers.

What Service Providers Want

The Java technology experience is not limited to device manufacturers and software developers. Service providers and carriers want to provide scalable offerings to enterprise customers without a complex device matrix. For example, deploying enterprise e-mail services requires the cooperation of device vendors, service providers, and enterprise IT departments. This requires integrating connected devices into their networks and back-end servers. The Java™ Platform, Micro Edition (Java ME, formerly J2ME) can help solve the device matrix problem while the Java Platform, Enterprise Edition (Java EE, formerly J2EE) can help solve the server-side scalability problem.

What Enterprises Want

With the arrival of Web technology, corporate IT departments have gone through several phases:

- **Intranets.** These allow users within a corporate network to use standard Web browsers to access enterprise services running on central servers.
- **Virtual Private Networks.** These allow corporate IT departments to extend the availability of enterprise services to sites outside of their corporate network firewalls. For example, remote workers can access enterprise services from their home PCs.
- **Web services.** These and other enterprise application integration (EAI) strategies allow enterprises to export their business systems to customers and trusted partners. For example, a search engine might allow a licensed partner to integrate search services into a Web application.

The next stage of this evolutionary process is to provide access to enterprise services for mobile users. This improves collaboration and brings up-to-date information into business decisions. But with these rich capabilities come integration challenges in the form of different kinds of mobile client devices and the enterprise technology for managing them. Connected devices that can be lost or stolen present important security issues for enterprises. Java technology can be used at each stage: Java EE technology on the server side, matched with Java ME technology in the client.

Chapter 3

Java Technology Leads the Way

Open systems have always been a major focus of Sun's product strategy. Today, Java technology is built around a set of standards developed in cooperation with industry leaders. These standards allow both device vendors and software developers to see what's coming so they can build products that interoperate, thus protecting customer investment and promoting vertical integration.

Java technology standards are developed through the Java Community Process℠ (JCP℠ - www.jcp.org) by expert groups drawn from several industries. The JCP program is an open standards organization that manages the evolution of the Java Platform. While Java technology was originally created by Sun, the JCP program is a mechanism for third-party developers and licensees to lend their voices to its continued evolution.

The JCP standardization process is built around three interrelated components: the Java Specification Request (JSR), the Reference Implementation (RI), and the Technology Compatibility Kit (TCK). As Figure 2 suggests, the JCP program requires an RI and a TCK for each JSR standard.

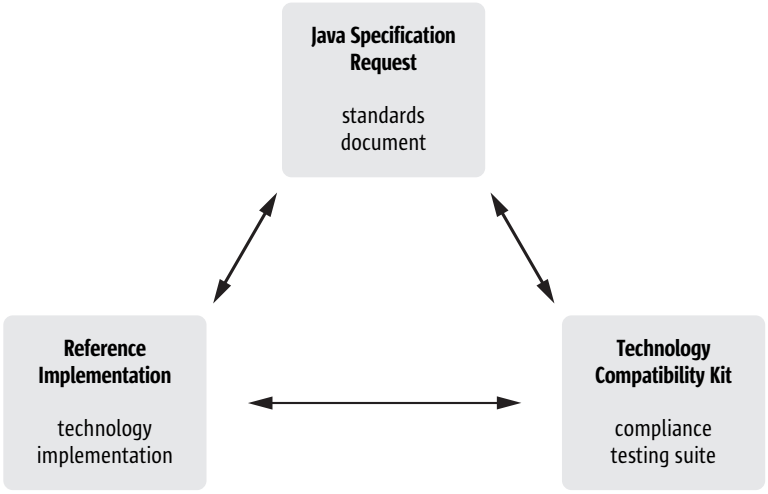

Figure 2: JCP Standardization Model

From Java Specification Requests ...

The JCP program delivers its Java technology standards in documents called **Java Specification Requests** (JSRs). These documents cover a wide range of technical issues across the spectrum of Java technologies. Three JSR categories are especially relevant to the Java ME platform. These are described in Table 1.

Table 1. JSR Categories for Java ME Platform

JSR Category	Description	Examples
Configuration (required)	Provides the most basic set of class libraries and virtual machine capabilities for a broad range of devices.	CDC supports network-connected devices and provides a subset of the APIs in the Java SE, formerly J2SE, platform.
Profile (required)	Defines additional APIs that support a narrower range of devices. A profile is built on a specific configuration.	Foundation Profile can be used for network printers without GUIs, while Personal Profile can be used for high-end PDAs. Both of these profiles are based on CDC.
Optional package(s)	Defines a set of technology-specific APIs.	The Java Database Connectivity (JDBC™) Optional Package provides database access, while the RMI Optional Package enables distributed application development. Both of these optional packages are compatible with each of the CDC profiles.

By selecting a configuration, a profile, and any number of optional packages, a product designer defines the Java runtime environment for a given device based on the Java ME technology. This approach has the dual benefits of fitting within a framework of established industry standards while meeting the performance, footprint, and functionality requirements of commercial products. The standards developed through the JCP program provide a consistent set of APIs that can be shared across a broad range of products from different vendors.

... to Reference Implementations

Reference Implementations (RIs) are implementations of JSRs that demonstrate core technology. Sun makes some RIs available for download for demonstration and testing purposes. Sun has also developed many optimized implementations of Java ME specifications that are available to licensees for use with their own platform-level products.

... to Technology Compatibility Kits

Technology Compatibility Kits (TCKs) are test suites that ensure an implementation of a JSR is compliant with the requirements of that specification. The purpose of a TCK is to broadly test the conformance of an implementation to a JSR specification, ensuring that the Java platform behaves consistently across different implementations.

Like any implementation of a JSR, an RI must also pass the TCK tests prior to being made available. See the JCP program Web site (www.jcp.org) for information about where to obtain RIs and TCKs for a given JSR.

The CDC technology family is an interrelated group of RIs and TCKs. While RIs are usually based on common development platforms, Sun also makes available optimized implementations for targeted embedded platforms. In addition, Optimized Implementations (OIs) of JSRs support strategic platforms and are made available under commercial license.

Chapter 4

Java Platform, Micro Edition Technologies

Java technology is divided into the platforms shown in Figure 3. Like its counterparts Java EE and Java SE (formerly J2SE), Java ME provides a set of APIs developed through the JCP program.

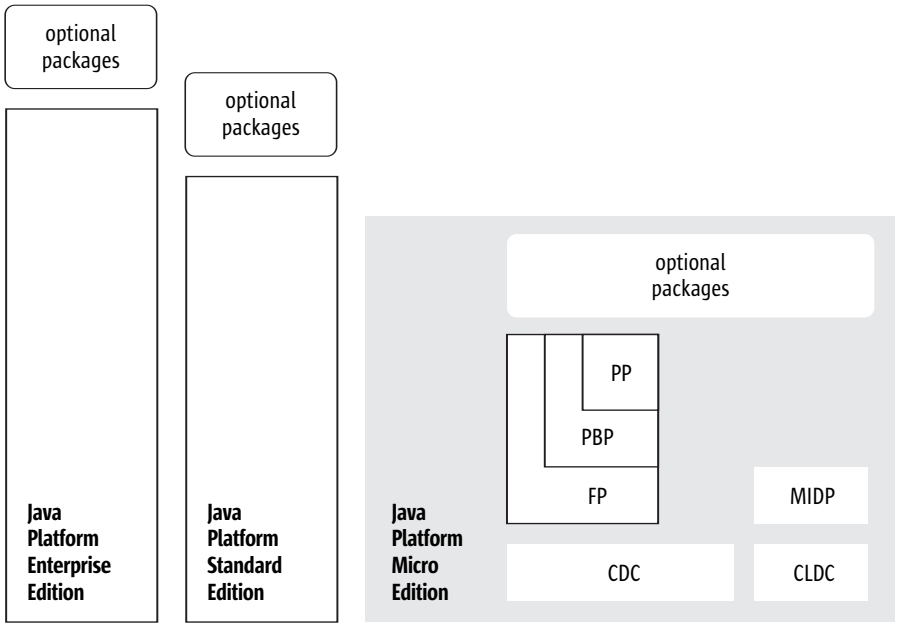

Figure 3: Java Platform Architecture

Java ME technology is targeted at connected devices ranging from wireless phones to PDAs to set-top boxes. To be relevant in this product arena, Java technology must be adaptable, because these product categories are constantly evolving as vendors add new features and identify new product niches.

Java ME supports a much broader range of target devices than Java SE and Java EE technology, and these devices cannot all be supported with a single set of APIs. The JCP standards process described in the previous section provides a mechanism for Java technology to grow and adapt to change. Flexibility is needed to address both limited resource issues and product capabilities. To accomplish this, Java ME allows product designers to choose from a menu of configurations, profiles, and optional packages to create a product-specific, Java runtime environment.

Configurations

A *configuration* is a basic set of APIs and virtual machine features needed to support a broad range of devices. The first step to making Java ME technology relevant to so many different device categories was to organize it into the configurations described in Table 2.

Table 2. Java ME Configurations

Java ME Configuration	Device Examples
Connected Device Configuration (CDC) (JSR-218)	PDAs, mobile phones, VOIP phones, telematics, set-top boxes, network printers, routers, residential gateways, and Embedded Java runtime environments for enterprise-class server applications.
Connected Limited Device Configuration (CLDC) (JSR-139)	Mobile phones and low-end PDAs.

CDC

CDC is designed around the two goals of Java SE compatibility and support for resource-constrained devices. Java SE compatibility allows developers to leverage their investments in Java SE technology, including libraries, tools, and skills. Support for resource-constrained devices allows device vendors to offer a feature-rich Java runtime environment that can support secure, mobile, enterprise applications.

CDC supports the full Java virtual machine specification, including full class loading and core library features. At the class library level, CDC uses modified Java SE class libraries whose interfaces have been scaled to meet the needs of resource-constrained devices and whose implementations have been optimized for small memory environments. In the interest of resource conservation, some class libraries based on Java SE have modified interfaces, while others have been removed entirely. The result is a flexible Java runtime environment that fits comfortably within a memory budget of two MB of RAM and two MB of ROM.

CLDC

CLDC was designed to meet the memory footprint requirements of cellular phones. To meet these requirements, CLDC made accommodations in both the virtual machine and the class library. The first CLDC virtual machines (CLDC 1.0) omitted certain virtual machine features, such as floating point. The CLDC class library contains a small subset of core Java SE classes that are shared between Java SE, CDC, and CLDC. Certain Java SE features such as class reflection, thread groups, and application-defined class loading are not present in the CLDC class libraries.

The benefits of these differences are significant. The memory footprint for early CLDC implementations achieved the target memory goal of 128 to 256 KB, well within the memory budgets of low-end to mid-range mobile phones. Recent versions of CLDC have taken advantage of greater memory footprints by bringing back some virtual machine features and adding features to the CLDC class library.

In a nutshell, CLDC was designed around limited memory requirements, and CDC was designed to achieve as much Java SE compatibility as possible within constrained device resources.

Profiles

Profiles provide core APIs for connected devices.

A *profile* is an additional set of APIs that support a narrower range of devices. Profiles provide a product designer with flexibility for supporting different kinds of connected devices with a compatible Java runtime environment. Instead of defining a single, monolithic API, Java ME gives a designer choices for different device categories. Table 3 describes the various CDC profiles:

Table 3. Java ME Profiles

Profile	JSR	Description	Product Information
Foundation Profile	219	Foundation Profile is the most basic CDC profile. In combination with the class library provided by CDC, Foundation Profile provides basic application-support classes such as network support and I/O support. In particular, it does not include any support for graphics or GUI services.	`java.sun.com/products/foundation`
Personal Basis Profile	217	Personal Basis Profile provides a structure for building lightweight component toolkits based on a limited GUI toolkit based on AWT, JavaBeans runtime support, and support for the xlet application programming model. In addition, Personal Basis Profile includes all of the Foundation Profile APIs.	`java.sun.com/products/personalbasis`
Personal Profile	216	Personal Profile provides full AWT support, applet support, and limited bean support. In addition, Personal Profile includes all of the Personal Basis Profile APIs. Personal Profile also represents the migration path for PersonalJava™ technology.	`java.sun.com/products/personalprofile`

Optional Packages

Optional packages provide choices for technology- specific features.

Optional packages give a product designer additional choices for supporting specific technologies. Table 4 describes the optional packages that are compatible with CDC.

Table 4. Java ME Optional Packages for CDC

Optional Package	JSR	Description
RMI	66	Provides a subset of the Java SE RMI. It exposes distributed application protocols through Java interfaces, classes, and method invocations and shields the application developer from the details of network communications.
JDBC	169	Provides a subset of the JDBC 3.0 API that can be used by Java application software to access tabular data sources including: spreadsheets, flat files, and cross-DBMS connectivity to a wide range of SQL databases.
AGUI	209	Provides a modified implementation of Swing for providing rich GUIs and Java 2D™ technology for providing advanced graphics and imaging features. The AGUI optional package is based on both Personal Basis Profile and Personal Profile.
Security	219	Provides a security framework based on Java SE, including SSL, cryptography, authentication, and authorization features.
Web Services	172	Provides standard data access from Java ME clients to Web services.

A CDC Java Runtime Environment

Putting it all together: A Java ME runtime environment includes a configuration, a profile, and any number of optional packages.

CDC standards provide a great deal of flexibility for designing a product based on the CDC Java runtime environment. During the product design phase, a designer can select a configuration, a profile, and any number of optional packages. For example, a PDA designer might include Personal Profile, Remote Method Invocation (RMI) Optional Package, and Java Database Connectivity (JDBC™) Optional Package. Application developers can then reference these APIs during software development.

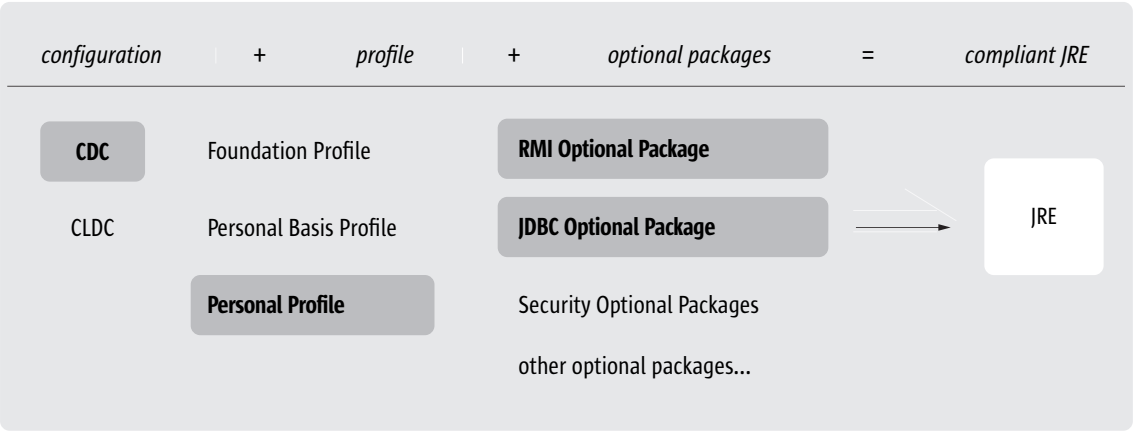

Figure 4: An example Java runtime environment

It is important to understand that different CDC-based products may have different APIs within the CDC standards framework. Choices made by a CDC product designer determine the API target for a CDC application developer.

Chapter 5

CDC Technical Overview

The preceding chapters describe the technology landscape that connected devices occupy, the JCP standards model, and the CDC technology family. This chapter introduces the organization and structure of CDC technology, including its public APIs and application models.

CDC Class Library

CDC contains a Java class library derived from Java SE technology, but crafted to the needs of connected devices. The CDC class library includes a number of application support classes that software developers have been using for years to develop countless applications for desktop and server systems. CDC leverages this broad experience with standard Java SE APIs in the connected device space. While most CDC APIs are identical to their Java SE counterparts, some interfaces have changed, and the implementations of others have been tuned to the needs of resource-constrained devices. The result is a Java class library that allows developers to quickly migrate their code and skills from Java SE to CDC.

CDC HotSpot™ Implementation

CDC HotSpot Implementation is an optimized Java virtual machine designed specifically for connected devices. Sun has developed the CDC HotSpot™ Implementation, a Java virtual machine that is highly optimized for use with connected devices. CDC HotSpot Implementation adheres to the same Java virtual machine specifications required by Java SE technology-based Java virtual machines, but its performance, device support, resource footprint, and reliability characteristics are designed around the needs of consumer products and embedded devices. For more information about the CDC HotSpot Implementation, see: java.sun.com/products/cdc-hi/.

CDC API Overview

It's helpful to examine how the CDC APIs are organized by comparing Foundation Profile, Personal Basis Profile, and Personal Profile. Understanding these differences is an important part of CDC application development. As the name suggests, Foundation Profile provides a basic set of application support classes. The other profiles build on this by adding specific functionality.

Table 5: Comparison of Foundation Profile, Personal Basis Profile, and Personal Profile

Relationship	Package/Class	Description
Foundation Profile	`java.io`	Full J2SE 1.4.2 support for most core packages and classes. Some enterprise-level APIs have been removed to save space.
	`java.lang`	
	`java.lang.ref`	
	`java.lang.reflect`	
	`java.math`	
	`java.net`	
	`java.security`	
	`java.security.acl`	
	`java.security.cert`	
	`java.security.interfaces`	
	`java.security.spec`	
	`java.text`	
	`java.util`	
	`java.util.jar`	
	`java.util.zip`	
	`javax.microedition.io`	The Generic Connection Framework (GCF) provides an abstraction of various communication technologies so that applications can make network connections and perform I/O without referring to a specific connection type.
Added by Personal Basis Profile	`java.awt`	Support for lightweight components and some Java 2D graphics classes.
	`java.awt.color`	
	`java.awt.event`	
	`java.awt.image`	
	`java.beans`	Limited to runtime support. Design time manipulation of Java Beans must be managed by an external bean editor like an integrated development environment (IDE) based on Java SE technology.
	`java.rmi`	Limited RMI support for xlets, not intended for general-purpose use.
	`java.rmi.registry`	
	`javax.microedition.xlet`	Xlet support.
	`javax.microedition.xlet.ixc`	
	`java.applet`	Applet support.
Added by Personal Profile	`java.awt`	Support for heavyweight components and 2D graphics.
	`java.awt.datatransfer`	

From a developer's perspective, CDC 1.1 and its profiles look like a modified J2SE™ 1.4.2 runtime environment. Table 6 compares the packages in CDC 1.1 and its profiles with the packages in J2SE 1.4.2.

Table 6: Package Comparison of J2SE 1.4.2, FP 1.1, PBP 1.1, and PP 1.1

Package[1]	J2SE 1.4.2	FP 1.1	PBP 1.1	PP 1.1
java.applet	•	—	—	P[2]
java.awt.*	•	—	P[3]	P[3]
java.beans.*	•	—	P[4]	P[4]
java.io	•	•	•	•
java.lang.*	•	•	•	•
java.math	•	•	•	•
java.net	•	•	•	•
java.rmi.*	•	OP[5]	OP[5]	OP[5]
java.security.*	•	•	•	•
java.sql	•	OP[5]	OP[5]	OP[5]
java.text	•	•	•	•
java.util*	•	•	•	•
javax.accessibility	•	—	—	—
javax.naming.*	•	—	—	—
javax.rmi.*	•	—	—	—
javax.sound.*	•	—	—	—
javax.swing.*	•	—	OP[6]	OP[6]
javax.transaction	•	—	—	—
org.xml	•	OP[7]	OP[7]	OP[7]
org.omg.*	•	—	—	—
javax.microedition.io.*	—	•[8]	•[8]	•[8]
javax.microedition.xlet.*	—	—	•[9]	•[9]

•: full support, P: partial support, —: not supported, OP: replaced by an optional package

Application Models

CDC supports several application models that allow developers to handle different user needs and deployment scenarios. These range from the basic standalone application model that is similar to a conventional native application, to managed application models that offload the tasks of deployment and resource management to application management systems.

[1] Methods deprecated in J2SE have been removed from CDC and its related profiles and optional packages.
[2] In PP, `java.applet` is very similar to J2SE 1.4.2 except that `getAccessibleContext()` is not supported.
[3] In PBP, `java.awt` supports lightweight components and some Java 2D classes. PP includes full AWT support.
[4] PP and PBP include only runtime support for the `java.beans` package. PBP and PP support external IDEs based on J2SE, but not bean editors running directly on a CDC Java environment.
[5] `java.rmi.*` and `java.sql` have been replaced by optional packages. PBP and PP include a subset of `java.rmi` to support xlet communication, but these classes should not be used directly.
[6] The AGUI Optional Package (JSR 209) contains a subset of the Swing and Java 2D packages.
[7] The Web Services Optional Package (JSR 172) includes a programming model that is consistent with the J2SE platform.
[8] FP, PBP and PP include the Generic Connection Framework with extensions that include serial port connections.
[9] The xlet support packages `javax.microedition.xlet.*` have been added to PBP and PP.

Standalone Applications

The most basic application model is the standalone application model, which dates back to the beginning of Java technology. Developers use standalone applications for fixed-purpose designs such as utilities and productivity applications.

When the Java virtual machine is launched, it is given the name of the main application class for loading. This class must include a method named *main()* that handles the rest of the application's class loading, object creation, and method execution. Figure 5 illustrates this process. The standalone application interacts directly with the Java runtime environment to manages its own life cycle and system resource needs. When the main() method exits, the standalone application terminates.

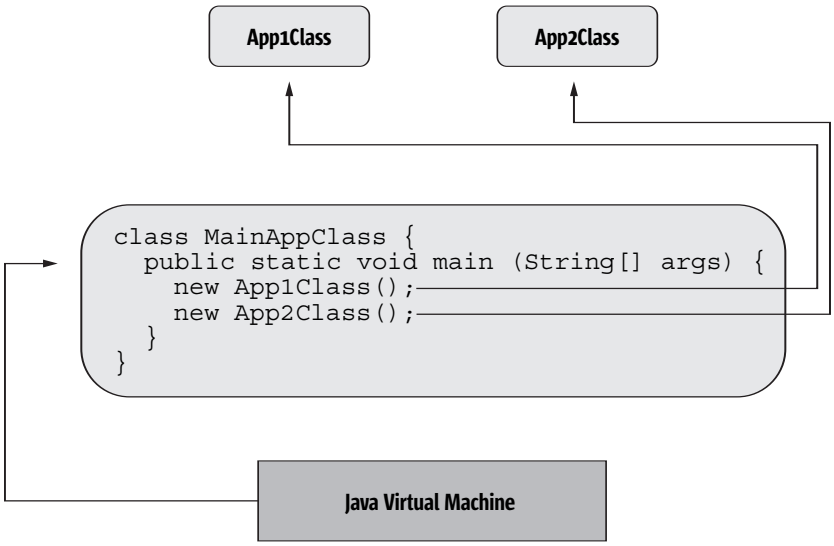

Figure 5: Standalone Application

Standalone applications serve a variety of purposes. For example, a bundled application such as a calculator could be a permanent part of a device-resident application suite. A vertical market application, such as a prescription drug compatibility database, might be installed separately by a customer or systems integrator.

Managed Applications: Applets

One of the first success stories of Java technology is the browser-based applet model, which allows a user to view and interact with dynamic Web content through a Web browser that contains an embedded Java runtime environment. When a user loads a Web page containing an applet, she can interact with the applet in ways not possible with static Web content. To enable this capability, the Web browser uses its embedded Java runtime environment to load and run the applet. The browser interacts with the Java runtime environment to manage the applet's life cycle and provide system-level services, such as a security sandbox and GUI system. Because java.applet.Applet is a subclass of java.awt.Panel, applets are GUI-based, which allows them to use some of the Abstract Windowing Toolkit (AWT) graphics and layout capabilities.

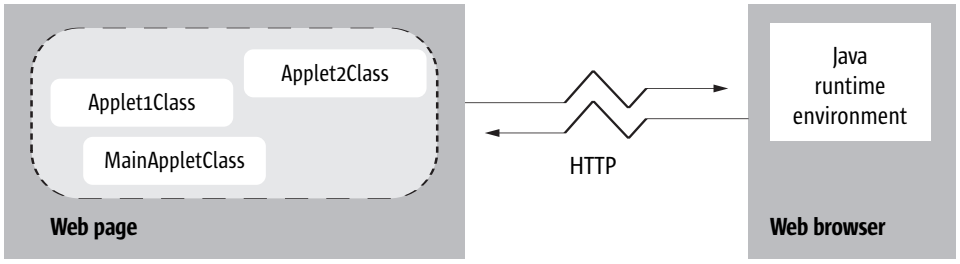

Figure 6: Applet Interface

In the example in Figure 6, the Java runtime environment is embedded into a Web browser. When the browser loads a Web page, it receives an HTTP stream that contains both the HTML content for the Web page and its applet classes. It renders the HTML content and hands off the applet classes to the Java runtime environment. The Java runtime environment then loads the main applet class, which includes methods for the major events in an applet's life cycle: initialization, starting, stopping, and destruction. This abstraction allows developers to avoid much of the system-related code normally associated with a standalone application, while protecting the client system from insecure Web content.

The main benefits of applets are a streamlined development model coupled with a deployment model that fits well with the dynamic nature of the Web. Personal Profile includes full applet support, so that it can be used with embedded Web browsers.

Managed Applications: Xlets

Both Personal Basis Profile and Personal Profile include support for the new xlet application model, which is similar in purpose to the applet application model, but different in design. The main differences are that xlets have no dependency on AWT, and they have a cleaner life cycle model. An xlet is loaded into an xlet manager, which manages its life cycle and provides it with system services through an XletContext. An xlet manager can handle multiple, dynamically loaded xlets that can communicate with each other through an RMI mechanism.

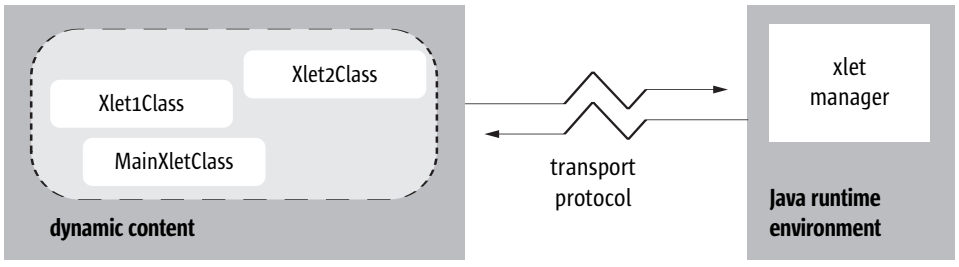

Figure 7: Xlet Interface

Because the xlet application model does not depend on AWT, xlets can be used in both GUI and non-GUI scenarios ranging from smart phones to set-top boxes.

CLDC Compatibility

The migration path from CLDC to CDC technology is based on compatibility packages and common APIs:

Core CLDC (JSR 139) support. The core CLDC packages (`java.io, java.lang, java.lang.ref` and `java.util`) are subsets of their corresponding CDC and Java SE packages. Migrating non-GUI CLDC applications to CDC is straightforward, because CDC shares these same core APIs. CDC includes xlet support in the in the `javax.microedition.xlet.*` packages.

Generic Connection Framework (GCF, JSR 30) support. CLDC introduced GCF as a generic approach to connectivity, so that applications could avoid the API details of different networking technologies, while CLDC implementations could avoid the resource requirements of more feature-rich networking technologies. GCF provides broad support for different kinds of connection types and I/O mechanisms ranging from packet-based HTTP connections to video streams. CDC includes a GCF class library in the `javax.microedition.io` package.

MIDP (JSR 118) support. The MIDP GUI framework provided by CLDC is designed around the needs of mobile phones and low-end PDAs, and is very different from the GUI frameworks based on Java SE that are provided by Personal Basis Profile and Personal Profile.

Chapter 6

GUI Options and Tradeoffs

CDC offers several GUI options that allow designers to handle different product scenarios. Table 7 describes the main scenarios for CDC-based devices. It shows how the different CDC profiles and optional packages match up with different design scenarios.

Table 7: GUI Scenarios

Scenario	Description	Specification
No GUI	The target device does not have a GUI. For example, a network printer might have an embedded server that exports a Web-based interface.	Foundation Profile
Proprietary GUI	The target device has a simple native GUI that may display text or buttons. For example, a pager may only have a simple alphanumeric display.	Foundation Profile
Raw graphics device	The target device includes a raw graphics device but lacks a native GUI toolkit. In this case, a portable GUI toolkit like Qt or GTK can be used.	Personal Basis Profile or Personal Profile
Minimal GUI toolkit	The target device has a minimal GUI toolkit and does not require full AWT compatibility. Instead, the target device needs support for a technology-specific lightweight toolkit.	Personal Basis Profile
Native GUI toolkit	The target device already has a native GUI toolkit that can be used by both native applications and for AWT widget peer functionality.	Personal Profile
Applet support for Web browsing	The target device has Web browser functionality that also needs applet runtime support.	Personal Profile
Rich GUI toolkit	The target device may have a basic graphics interface, but needs a rich GUI toolkit as well as sophisticated graphics and imaging features.	AGUI Optional Package

From a developer's perspective, Personal Basis Profile and Personal Profile offer AWT APIs that are similar to their Java SE counterparts, with the addition of some low-level Java 2D™ APIs. From a device designer's perspective, two issues need more explanation: AWT compatibility and lightweight components. Understanding these concepts will make the purposes of the Personal Profile and Personal Basis Profile more clear.

AWT

The Abstract Window Toolkit (AWT) dates back to the original Java technology release. It is a basic GUI toolkit that provides an API for common GUI widgets like buttons and dialogs. AWT widgets can handle most of the GUI needs for the kinds of applications developed for connected devices. Because native platforms supply similar GUI widgets, the AWT performs an intermediary role by using these native widgets to supply a similar user experience on different platforms.

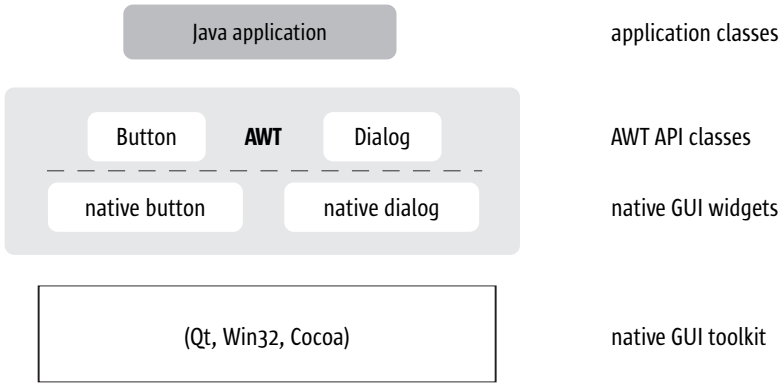

Figure 8: AWT and Native Toolkits

The AWT divides the task of supplying GUI services between high-level API classes and low-level implementation classes. Java application software interacts only with the high-level API classes, while the low-level "peer" classes create and manage the native widgets with which a user actually interacts. This architecture allows Java application software to use a single GUI API that runs without modification on different target platforms.

Lightweight Components

The core AWT system presents two problems: limited widget selection and portability. Thus, early in the evolution of Java technology, its architects identified the need for greater flexibility in GUI design and developed a mechanism for building widgets entirely with Java technology. These Java technology-based widgets are sometimes called lightweight components because they are rendered and managed entirely by the Java runtime environment.

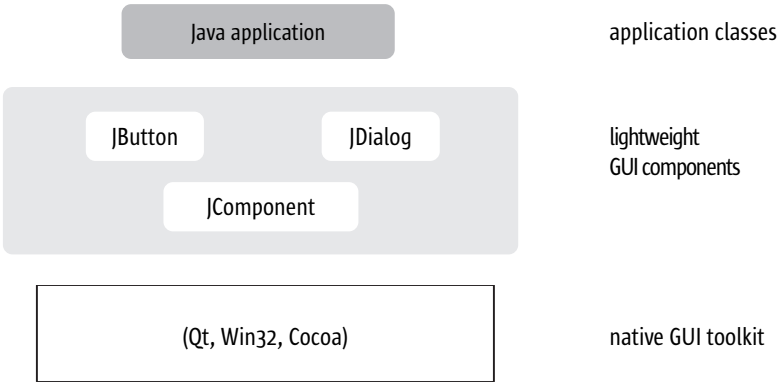

Figure 9: Lightweight Components

Developers can build their own special-purpose components and include them in applications or component libraries that can be shared by multiple applications. The best example of a large collection of lightweight components is Swing, which includes sophisticated table layout, pluggable look and feel, and accessibility features.

Alternate GUI Interfaces

By itself, Personal Basis Profile does not provide a complete GUI library. Instead, it is meant as a basis for building alternate GUI libraries for market and product-specific purposes. Table 8 describes several industry specifications that are compatible with Personal Basis Profile.

Table 8: Alternate GUI Interfaces

Specification	URL	Description
Home Audio-Video interoperability (HAVi)	`www.havi.org`	A specification for networking digital home entertainment products used in MHP and OCAP.
Multimedia Home Platform (MHP)	`www.mhp.org`	A specification for enhanced broadcast interactive services. It defines an interface between interactive applications and the devices on which they run.
OpenCable Application Profile (OCAP)	`www.opencable.com`	A middleware specification for interactive services and MHP-based applications.
Advanced Graphics and User Interface (AGUI) Optional Package	JSR-209	See below.

AGUI Optional Package

Java SE includes Swing, a sophisticated user interface toolkit for desktop applications. Not all of Swing's feature set is appropriate for CDC class products. Therefore, the Advanced Graphics and User Interface (AGUI) Optional Package represents a subset of Swing for connected devices.

Describing all the differences between the AGUI Optional Package and its Java SE counterpart is beyond the scope of this white paper. Table 9 describes the important differences to show that Swing features have been retained, modified, or removed to provide features relevant to connected devices.

Table 9: AGUI Feature Comparison

Feature	AGUI Optional Package	Swing
Swing, a rich set of GUI components	subset	supported
Java 2D, a graphics and imaging toolkit	subset	supported
Image I/O	subset	supported
Pluggable look and feel	modified	supported
Accessibility toolkit	not supported	supported
Drag-and-drop toolkit	not supported	supported
Input method framework	modified	supported

Chapter 7
Security

Security is a principal feature of Java technology and has guided the evolution of the Java platform from its beginning. CDC includes different levels of security that give users, developers, service providers, and enterprises an application framework with a powerful security architecture.

- **Virtual machine security** includes runtime features such as class verification, and language features such as the omission of pointers. These security features have removed an entire class of security threats called buffer stack overflow. Early Java technology releases used these features to build a simple but powerful "sandbox" security model that allows Web browsers to download Java applets without exposing the user's system to extra risk.
- **Signed classes** extend the sandbox security model and verify the integrity and origin of a Java class file to the Java virtual machine that attempts to load it.
- **Policy-based security** was introduced in J2SE 1.2. Security policies give application developers fine-grained control over who can access the data and interfaces of an application's objects. A security policy file is a set of permissions and policies specified that can be modified by system administrators during deployment.
- **Cryptography** provides a standard way to encode software and data for secure transfer or archival. The Java security framework includes the Java Cryptography Architecture (JCA), which is a standards-based framework for providing, selecting, and using cryptographic functionality.

CDC includes the Java security framework included in the Java SE. This security framework is extensible, in that it is based on algorithm-independence, and interoperable, in that it can use different implementations of security services. Foundation Profile (JSR 219) includes three optional packages that use this framework to add security features:

- **Java Authentication and Authorization Service (JAAS).** Optional Package provides a pluggable framework for authentication and authorization. The authorization component allows the specification of access controls based on code location, code signers, and code executors in separate policy files that can be maintained by a system administrator. At runtime, a Java runtime environment can provide different login modules, such as keystore, without requiring application modification. Alternate login modules can be integrated at runtime to support systems like biometric or Kerberos authentication.
- **Java Cryptography Extension (JCE).** Optional Package extends the Java Cryptography Architecture to include encryption, key generation, key agreement, and message authentication code (MAC) generation services.
- **Java Secure Socket Extension (JSSE).** Optional Package provides services for using Secure Socket Layer (SSL) to encrypt data and authenticate communicating peers.

Chapter 8

Developer Tool Support

Developers invest their resources in more than just their source code. They develop reusable skills by using tools that increase their productivity and allow them to focus their efforts on adding domain-specific value. To help achieve these goals, CDC maintains compatibility with the Java SE developer tool interfaces described in Table 10.

Table 10: Developer Tool Interfaces

Category	Interface	Description
compiler	Java Virtual Machine Specification	Application developers can use the same compilers that are available for J2SE development. These can be command-line based tools such as `javac`, or part of an IDE.
debugger	Java Virtual Machine Debugger Interface	The CDC HotSpot Implementation Java virtual machine supports remote debugging through the Java Virtual Machine Debugger Interface (JVMDI).
profiler	Java Virtual Machine Profiler Interface	The CDC HotSpot Implementation Java virtual machine supports application profiling through the Java Virtual Machine Profiler Interface (JVMPI).
GUI development	JavaBeans™ Components	Though Personal Profile and Personal Basis Profile do not support the entire `java.beans` package, they do contain runtime support for `java.beans` technology-based components.
IDE integration	Unified Emulator Interface (UEI)	Defines an interface that allows an external developer tool to control a Java ME emulator. The Java ME Wireless Toolkit is an example of a UEI-based emulator that developer tools like the NetBeans™IDE and the Sun Java Studio line of IDEs.

CDC support of these developer tool interfaces allows developers to leverage their investments in conventional Java SE software, tools, and skills.

Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 USA **Phone** 1-650-960-1300 or 1-800-555-9SUN **Web** sun.com