

## Differences between PersonalJava and MIDP Java Environments

### 1. Introduction

#### 1.1. Java Development Platform

This document investigates the differences between PersonalJava and the Mobile Information Device Profile, which make up the Java Environments available on mobile phones. They are introduced as follows

##### 1.1.1. PersonalJava

The PersonalJava profile is based upon the JDK1.1 but makes a number of package, classes and methods optional. It gives the capability to the developer to create Webapplets and other mobile phone applications. It is the first attempt by Sun to produce a Java application environment for mobile devices, it predates the CDC configuration and is the forerunner to J2ME. Symbian has ported Sun's PersonalJava 3.0.1 implementation of the PersonalJava Application Environment 1.1.1a specification. Symbian has implemented all of the PJAЕ's optional features, although the individual licensees may decide not to implement all of those features.

The Personal Profile will however provide a replacement for PersonalJava and extends the Foundation Profile and sits on top of the Compact Virtual Machine (CVM) and the CDC configured devices. As well as providing backward support for PersonalJava 1.1 and 1.2, it will provide J2ME developers with a clear roadmap on which to base application development.

##### 1.1.2. J2ME

J2ME is a subset of the J2SE edition and is the Java platform for consumer and embedded devices, defined through the Java Community Process and it also maintains the Java ethos of portability. J2ME was designed to take this in to account by grouping devices in to *configurations*, a vertical set of virtual machines and minimal set of class libraries providing the base set of functionality for the range of devices within each configuration.

Two configurations have been defined, the Connected Limited Device Configuration (CLDC) and the Connected Device Configuration (CDC).

The CLDC is the smaller of the two platforms and caters for devices, such as mobile phones with an intermittent network connection, slow processor (16 or 32 bit) and limited memory (128kb-512kb). The CDC is designed for devices, such as high end PDAs and communicators, with more memory (minimum of 2Mb) and a 32 bit processor.

The development environment for J2ME on the CLDC devices is the Mobile Information Device Profile (MIDP). It defines the classes for user interface, persistent storage and networking and application management. Combined with CLDC and its Kilo Virtual Machine (KVM), this profile provides a complete Java runtime environment for handheld devices with minimum memory and processor power. It allows programs to be downloaded, over the air, to the device from a service provider in the form of MIDlets.

## 1.2. Purpose and Scope

This document aims to provide developers with a background to the Java environment and demonstrate the differences between the two environments currently implemented on Symbian OS mobile phones, PersonalJava and the J2ME MIDP.

In some cases both environments exist on mobile devices at the same time, so the dilemma faces developers with which one to create applications for. This should help developers make an informed decision when deciding which profile to use and for what reasons by showing the capabilities and limitations of both.

## 1.3. Target Audience

This document is aimed at any developers wishing to develop Java applications for Symbian OS phones and are unsure which J2ME profile to use in creating those applications.

A previous knowledge of Java and to a lesser degree knowledge of the Symbian OS is also required.

## 1.4. Conventions used in this article

The font type 'Courier New' will be used to denote code segments and class or method names.

## 2. Symbian OS phones in the market

Before we investigate the Java environments available for mobile devices, first we should look at the Symbian OS phone supporting the J2ME framework

The Nokia Communicator, the 9210/9210i/9290, supports the PersonalJava Application Environment 1.1.1 along with the JavaPhone API extensions.

Nokia also has two further devices supporting Java. The two Series 60 devices, Nokia 7650 which is currently available and the Nokia 3650, yet to be available both support Java MIDP 1.0. The 7650 supports the additional Nokia UI API, as does the 3650, but this device also provides the Wireless Messaging API (JSR 120) and the Mobile Media API (JSR 135). Both phones run Java under the CLDC 1.0 configuration.

The Sony-Ericsson P800 provides developers with both Java environments. MIDP and PersonalJava are both available giving the developer plenty of scope when deciding which environment to use when designing applications.

## 3. Introduction of PersonalJava/JavaPhone API's

The PersonalJava Application Environment (PJAE) provides a development environment for CDC devices rather than for desktop devices. However it is derived from the JDK 1.1, but makes a number of packages, classes and methods optional. Examples of optional packages include the `java.rmi` and `java.sql`.

Symbian has decided to implement all of the optional features in its implementation of PersonalJava, although it is down to the individual licensee to implement those features. This gives device manufacturers the choice when tussling with their ROM budget.

## 3.1. *PersonalJava Extensions*

PersonalJava provides extensions to the JDK 1.1 specification. The following provides a brief explanation of what these features are, for more details of these features please refer to the PersonalJava specification, as it is beyond the scope of this document.

### 3.1.1. **Double Buffering identification**

A new method `isDoubleBuffered()` has been added to the `java.awt.Component` class, which identifies whether drawing carried out in the `paint()` and `update()` methods have been automatically double buffered. The method returns `true` if it has.

### 3.1.2. **Specifying Component Input Behaviour**

PersonalJava provides a number of interfaces in the `com.sun.awt` package, which can be used by a UI component to specify its preferred input behavior.

i) **Action Input Preferred**

`ActionInputPreferred` indicates that a user can click on or activate a component by using a pointer device.

ii) **Keyboard Input Preferred**

`KeyboardInputPreferred` indicates that a component will be used primarily via the keyboard. For this to work the `Component.isFocusTraversable()` must be set to `true` for that component.

In the case of UIQ interfaces a pop up keyboard would be displayed in the screen to facilitate the keyboard entry from the user.

iii) **Positional Input Preferred**

`PositionalInputPreferred` indicates the component will be primarily used by the user selecting x,y co-ordinates within the component.

This is particularly relevant on the UIQ implementation of the PersonalJava platform.

iv) **No Input Preferred**

`NoInputPreferred` indicates the user may not navigate to a particular component. An example of this would be label components.

### 3.1.3. Timer API

The Timer API gives the developer the ability to create timers or timer based events. It saves developers from having to create their own timer threads and therefore saves on processor power.

The `PTimerSpec` class declares when a `PTimerWentOffEvent` occurs. These events are sent to listeners registered with the `PTimer` instance. The `PTimer` class manages the set of timer events specified by the `PTimerSpec`.

### 3.1.4. Unsupported Exception Handling

Due to the nature of the PersonalJava environment, there maybe some occasions when an application attempts to use an unimplemented PJAE feature. PersonalJava has the capability of throwing exceptions to handle this, such as `java.lang.NoClassDefFoundError`, `com.lang.UnsupportedOperationException` and `java.lang.NoSuchMethodError`.

## 3.2. JavaPhone API Vertical Extension of PJAE

The JavaPhone API extends the PersonalJava environment by providing the developer with access to telephony and underlying operating system functionality.

The following provides a brief explanation of the classes the JavaPhone API provides.

### 3.2.1. Direct Telephony Control

This addition to the environment is comes in the form of two sets of APIs, the JTAPI Core and the JTAPI Mobile.

The JTAPI Core, `javax.telephony`, provides the basic framework to model telephone calls and create, answer and disconnect a call. There are also some optional packages which provide extra features such as conferencing and transferring call, `javax.telephony.callcontrol`, and another packager, `javax.telephony.phone`, which provides details and control of a devices speakerphone, microphone, display buttons and ringer.

The JTAPI Mobile, `javax.telephony.mobile`, manages the network selection and identification, network and radio status and signal monitoring amongst other functions.

### 3.2.2. Datagram Messaging

The Network Datagram API, `javax.net.datagram`, provides a method of sending and receiving bearer independent messages, using address consisting of service name and location. This means essentially that developers can write transparent code regardless of the device or network.

### 3.2.3. Address Book Access

The address package, `java.pim.addressbook`, defines the objects required to access, add, delete and update contact information in the contact database, such as postal address, email address and phone numbers. The `AddressBook` package defines the base storage class for all types of items including contact items, calendar entries and to do entries. Typically these are types of information items held in the personal data interchange formats such as vCard and vCalendar.

### 3.2.4. Calendar Information Access

The Calendar package, `javax.pim.calendar`, provides the objects that enable the addition, removal, update and access to calendar event information in the Calendar database. It also provides storage for and access to schedule information such as dates and tasks entries and also provides the ability to make changes to that data.

### 3.2.5. User Information Access

The User profile, `java.pim.userprofile`, defines objects to get and set the current user contact information, which is returned in the same format as the `AddressBook` package.

### 3.2.6. Power Management

This optional package, `javax.power.monitor`, is a very important addition to the PersonalJava environment, gives the developer the ability to monitor and respond to changes in the power state of the mobile device.

### 3.2.7. Application Installation Mechanisms.

The JavaPhone API provides an interface to install and remove applications through the `java.ce.install` package. It is an optional package to for PJAE.

It supports the packaging of applications for development and distribution purposes, such as the creation of JAR files; unique identification of the application and identifying the main entry point class. It supplies an installation mechanism for installing the application in to the environment and also takes care of versioning control.

## 4. Introduction to the MIDP API's

The Mobile Information Device Profile (MIDP) is built upon the CLDC configuration architecture, although it does run CDC devices with the use of a CLDC adaptor layer. It is however the J2ME profile used for wireless devices such mobile phones and pagers.

Much MIDP has been derived from the Java Community Process, JSR 137, and is an evolving open application development environment. More information can be found at the following URL: <http://java.sun.com/products/midp/>

### 4.1. Background to Mobile Information Devices (MIDs)

Briefly devices for which this profile is aimed can be defined as the following:

- i) A display which is at least 96 by 54 pixels with a least a 1-bit display depth.
- ii) A one handed user input interface, two handed keyboard or a touch screen.
- iii) The memory is made up from 32kb of volatile space for the run time heap and 128kb of non-volatile space for MIDP components and 8kb of non-volatile memory for application created persistent data.
- iv) A two way intermittent network connection must be present.

## **4.2. The MIDP functionality**

The MIDP profile builds upon the CLDC configuration by addressing the following areas:

### **4.2.1. Application Life Cycle Management**

The MIDP profile provides the `javax.microedition.midlet` package, which contains the framework for the development of MIDlets. It provides the classes and methods for starting, stopping, pausing and destroying the applications with the host environment.

### **4.2.2. User Interface and Event Interfaces**

The `javax.microedition.lcdui` packages provide the classes and interfaces to create GUI components for applications.

### **4.2.3. Network Connectivity**

The CLDC configuration specifies that the devices must have a network connection. MIDP extends the `ContentConnection` interface of the Generic Connection framework by providing the `HttpConnection` interface. From here the developer can make connections to Http Web Servers and carry out standard `GET` and `POST` actions which enable developers to create data aware applications.

### **4.2.4. Persistent Storage**

The `javax.microedition.rms` package provides a means for Java developers to take advantage of the minimum 8kb persistent storage allocation in the memory configuration. It provides a record-based database management system and provides the framework with which to store data on the device.

## **5. Discussion to Assess the Differences between PJAE and MIDP**

We have so far introduced the two Java environments that appear on the Symbian mobile phone devices. On some of these devices both profile co-exist, where as other devices only ship with one, due to the configuration type of that device.

This does leave the developer with a certain dilemma. Which Java profile should they develop for? The following addresses some of the differences in the two environments:

### **5.1. PersonalJava allows for leverage of current developer skills**

PersonalJava is derived from the JDK 1.1 version of Java, with some additions and optional packages as has been explained previously in this document.

Current Java developer skills can be leveraged to create applications for mobile devices more quickly. Of course MIDP is not a totally different language at all, but some of the methods of implementation, such as the `Graphics` class.

The compilation of the programs for the PersonalJava environment is similar to that of the JDK 1.1 version. The JDK 1.1.8 is used to compile programs using the familiar `javac` command, where as applications for the MIDP

profile requires a verification process and the creation of a JAD file, which provides application versioning information to the person installing the MIDlet.

## **5.2. MIDP limitations**

The following investigates some of the more general limitations the MIDP profile has.

### **5.2.1. Telephony, Messaging Functions and Vibration Control.**

The MIDP profile does not cater for any ability by the application to be able to make phone calls or send SMS messages to 3<sup>rd</sup> parties. The only external communication a MIDP application can perform is HTTP networking.

The JTAPI classes give the PersonalJava environment the ability to make and receive voice calls, while the Datagram API gives the application functionality to send Short Messages (SMS).

However, some vendors have developed SMS APIs capable of implementing an SMS capability. These have been based on the Java Community Process specification and will be discussed later in this document.

Similarly vibration control cannot be attained with the standard release of the MIDP profile, although some devices do implement this.

### **5.2.2. Reflection**

CLDC applications do not have the ability to use the reflection APIs on their objects or classes. Therefore it follows that there is also no support for object serialization or RMI.

### **5.2.3. Floating Point Maths**

There is no support for floating point numbers. This is because most CLDC devices do not have floating point support in their hardware. Therefore the data types float and point are not present. Therefore some mathematical operations have to be calculated using Fixed Point maths solutions and the use of arbitrary decimal points.

### **5.2.4. Thread Groups**

Thread groups are not supported by the CLDC configuration. Although multi-threading is supported Daemon or thread groups are not. If this operation is required, then collection objects should be used to store the threads at application level.

### **5.2.5. Finalization**

You cannot control garbage collection manually with the use of the `Object.finalize()` method.

### **5.2.6. Error Handling**

The MID profile only supports three error classes, `java.lang.Error`, `java.lang.OutOfMemoryError` and `java.lang.VirtualMachineError`.

### **5.2.7. Weak References**

Weak referencing is not permitted on the CLDC platform.

### **5.2.8. User-Defined Class Loaders**

For security reasons, a Java virtual machine supporting the CLDC must have a built in classloader which cannot be overridden or replaced by a user.

## **5.3. Java Native Interface (JNI)**

One of the major differences between PJAE and MIDP is the capability of PersonalJava to access the native code in the underlying operating system.

Of course leaving the potential lack of portability aside, this can indeed be a very important asset of the PersonalJava environment.

JNI gives the developer the opportunity to gain control of the Symbian device drivers. Although many of the lower level functions such as access to Calendar and Telephony are made available with the JavaPhone APIs, there may be a requirement to control sound, printing functions or controlling an IrDa port. Also native data types such as Word, Agenda and Contacts can be more easily manipulated by native code. Also control over the user interface can be achieved through JNI.

## **5.4. JavaPhone lacks uptake, more memory requirements, limited uptake.**

So far the uptake of PJAE has been very limited. The mobile devices with PJAE included are high end devices with the power and memory capacity to be able to support the environment. This provides a slim opportunity for developers to create far-reaching applications. However it has to be pointed out that the devices on which PJAE is usually found are more likely to be used by enterprise users and as such the opportunity to create lucrative applications exists.

## **5.5. Differences in Installation on Target Devices.**

MIDP presents the developer with the ability to provide applications over the air. This means that the potential sales target can be to any device with a WAP connection.

Conversely applications created for the PJAE have to be installed using a far more complicated process, OTA provisioning is more complicated. In some ways it is similar to the MIDP process in that a `.jar` file is created, but additionally a package file (`.pkg`) has to be created which lists the objects comprising the application. A `.sis` file is then created to provide a convenient installation file for the end user.

## **5.6. PersonalJava is older version of the JRE than J2ME.**

PersonalJava is based upon the JDK 1.1, which of course is an older version of Java than J2ME. Therefore, long term development may be hampered as classes and methods become deprecated and changes have to be made to keep up with the evolutionary cycle of the Java Run Time Engine.

## **5.7. Commercial viability of both environments.**

Commercially, there are differences between the two Java environments available for mobile devices.

Some of these points have been raised previously in this document, but in the commercial context we might be concerned with revenue generation, distribution and the uptake of the application environment on devices.

MIDP has been taken up by a large number of devices and device manufacturers. MIDP is a standard in the Series 60 platform where device manufacturers such as Panasonic, Siemens, Samsung and Sendo have become licensees of the platform. This together with the ability from MIDP applications to be installed over the air certainly solidifies the case for MIDP.

Conversely, PersonalJava, being a CDC based profile, may be a more suitable development for business type applications. The capabilities of the environment to go native, the potential to connect and manipulate database data, may make this profile a better bet.

## **5.8. Application Development Scenarios.**

We have examined the commercial scenarios, which may suit either profile, but what are the possibilities for real applications?

The uptake by games and entertainment developers with the MIDP profile has been high. The CLDC configuration is more suited to the mass-market device, which means the market segment for game developers is significant. Also the nature of game usage sits comfortably with the smaller development time required to create such applications.

MIDP is also well suited to alert type applications, where end users can track stock market prices or read news headlines with a Java MIDP news readers.

The PJAE lends itself more suitably to business and enterprise applications. Applications such as contact management and meeting organization applications that require access to device resident applications. The ability to use the telephony features to communicate with other devices and organize meetings is very useful.

Also the ability of devices to store large amounts of information, due to the large resident memory in CDC devices. Therefore portable catalogues can be stored for field sales operatives to refer to whilst visiting customers.

## **6. Future Developments**

The future of these two Java environments will of course change as the environments for Java evolve.

PersonalJava, although currently being shipped in the form described in this document, is due for a face-lift. It is forming the basis of the Personal Profile. The Personal Profile APIs includes PersonalAWT, networking and Applets and Xlets, which are downloadable applications. JDBC and RMI support will be optional. The profile has been developed under the Java Community Process JSR 62

MIDP is also being updated to MIDP 2.0, which will provide an enhanced user interface; multimedia and game functionality; end to end security and greater networked connectivity to mobile phones and entry level PDAs.

Other individual APIs are also being developed under the Java Community Process. The Multi Media API, JSR 135, extends the J2ME platform by providing audio, video and other time-based multimedia support to constrained devices. It also gains access to native multimedia services on such devices.



The Wireless Messaging API, JSR 120, gives the developer the ability to send SMS and to establish a Cell Broadcast Service (CBS). The Bluetooth API, JSR 82, enables the Bluetooth protocols to be accessed and gives the ability of applications becoming interoperable with other devices over this service.

Other new additions to the J2ME environment can be found on Sun's J2ME site at <http://java.sun.com/j2me/>. Of course availability of these APIs will be dependent upon the device manufacturers adoption of such capabilities.