



March 2006

# Getting started with Java™ CDC development

---

P990

M600



Sony Ericsson P990



Sony Ericsson M600



Sony Ericsson

# Preface

This Special Interest Paper introduces you to Connected Device Configuration (CDC) based Java ME development with the Sony Ericsson P990 smartphone and M600 messaging device. No prior experience with Java ME is necessary, but you do need a firm understanding of the Java language and general familiarity with the features of the Java SE platform. Even if you're already familiar with Java ME or have programmed with PersonalJava, you'll need to pay close attention to the features and limitations of CDC-based environments on mobile phones.

The guide is divided into the following chapters:

- **Introduction** explains the background to the Java ME Platform and its two main configurations, setting the scene for the rest of the paper.
- **Java ME Concepts** summarizes the important concepts introduced by Java ME, including configurations, profiles and optional packages.
- **Design Considerations when Developing Applications for the Sony Ericsson P990 and M600** provides specific details about supported JSRs on the Sony Ericsson P990 and M600 phones and presents some initial design considerations.
- **Understanding the CDC** looks more closely at the CDC and the profiles, including the Personal Profile, built on top of the CDC.
- **Migrating from PersonalJava** compares the PersonalJava environment (a legacy Java system for constrained devices) to the Personal Profile.
- **Developing for the CDC** describes the basic development process for writing and testing CDC applications.
- **Accessing the File System** demonstrates how to read and write files.
- **Accessing the PIM System** demonstrates how to read and write contacts, addresses, and other PIM-based information.

After reading this guide, you'll be able to write simple CDC-based applications for Sony Ericsson's P990 and M600 phones.

This Special Interest Paper is published by:  
Sony Ericsson Mobile Communications AB,  
Developer Program,  
SE-164 94 Kista, Sweden

[www.sonyericsson.com/developer](http://www.sonyericsson.com/developer)

© Sony Ericsson Mobile Communications AB,  
2006. All rights reserved. You are hereby granted  
a license to download and/or print a copy of this  
document.

Any rights not expressly granted herein are  
reserved.

First edition (March 2006)  
Publication number: EN/LZT 108 8717 R1A

This document is published by Sony Ericsson  
Mobile Communications AB, without any  
warranty\*. Improvements and changes to this text  
necessitated by typographical errors, inaccuracies  
of current information or improvements to  
programs and/or equipment, may be made by  
Sony Ericsson Mobile Communications AB at any  
time and without notice. Such changes will,  
however, be incorporated into new editions of this  
document. Printed versions are to be regarded as  
temporary reference copies only.

\*All implied warranties, including without limitation  
the implied warranties of merchantability or fitness  
for a particular purpose, are excluded. In no event  
shall Sony Ericsson or its licensors be liable for  
incidental or consequential damages of any  
nature, including but not limited to lost profits or  
commercial loss, arising out of the use of the  
information in this document.

# Sony Ericsson Developer World

---

On [www.SonyEricsson.com/developer](http://www.SonyEricsson.com/developer), developers will find documentation and tools such as phone White Papers, Developers Guidelines for different technologies, SDKs and relevant APIs. The website also contains discussion forums monitored by the Sony Ericsson Developer Support team, an extensive Knowledge Base, Tips & Tricks, example code and news.

Sony Ericsson also offers technical support services to professional developers. For more information about these professional services, visit the Sony Ericsson Developer World website.

## Document history

---

Change history		
2006-03-03	Version R1A	Document published on Developer World

# Contents

<b>Introduction</b> .....	<b>5</b>
Java ME is more than small devices .....	5
About the Sony Ericsson P990 and M600 .....	6
<b>Java ME concepts</b> .....	<b>7</b>
The Java Community Process .....	7
Configurations .....	7
Profiles .....	9
Optional packages .....	9
Platforms .....	10
<b>Design considerations when developing applications for the Sony Ericsson P990 and M600</b> .....	<b>11</b>
Java ME Support .....	11
MIDP or Personal Profile? .....	11
<b>Understanding the CDC</b> .....	<b>13</b>
The CLDC .....	13
The Generic Connection Framework .....	14
The Mobile Information Device Profile .....	15
The CDC .....	15
The Foundation Profile .....	16
The Personal Basis Profile .....	17
The Personal Profile .....	17
The User Interface .....	18
The Application model .....	18
<b>Migrating from PersonalJava</b> .....	<b>19</b>
PersonalJava .....	19
Common Classes .....	19
Porting issues due to missing classes and methods .....	20
<b>Developing for the CDC</b> .....	<b>21</b>
Installing the CDC extensions .....	21
Defining the CDC root .....	21
Compilation .....	22
Packaging .....	23
Building with Ant .....	23
Testing the application .....	25
Enabling console output .....	25
<b>Accessing the file system</b> .....	<b>26</b>
Standard file access classes .....	26
The Generic Connection Framework .....	26
The File Connection Optional Package .....	27
<b>Accessing the PIM system</b> .....	<b>28</b>
Overview of the PIM Optional Package .....	28
Security Restrictions .....	28
Accessing PIM data .....	29
<b>Conclusion</b> .....	<b>31</b>
<b>References</b> .....	<b>32</b>

# Introduction

To many developers, Java ME (Java Micro Edition) programming is all about creating small applications that work within environments that are very limited when compared to conventional Java SE (Java Standard Edition) desktop and server systems. Real limits on application size and new APIs (application programming interfaces) challenge developers used to the (nearly) unlimited resources and flexibility of standard Java systems.

Convinced that they simply cannot work in such a constrained environment, many Java developers are loath to learn more about Java ME. Porting their Java SE applications to Java ME seems impractical, if not impossible. But the growth of the mobile handset market - the primary market for Java ME technologies - means that Java ME-based phones are an important and emerging market that cannot be ignored. Market demands and the benefits of end-to-end Java programming will require many developers to learn more about Java ME.

**Java 2 Micro Edition is now Java Micro Edition:** Java Micro Edition (Java ME) was formerly known as Java 2 Micro Edition (J2ME). This document uses the current terminology, but when searching for additional information about Java ME be sure to include the older form, especially the J2ME abbreviation, in your searches.

## Java ME is more than small devices

---

What you may not realize, however, is that Java ME is not a *single* platform tailored to a narrow set of limited-capability devices. Java ME in fact supports different classes of devices with its concept of *configurations*, which define the core features of a supported Java environment.

Most mobile phones on the market today support the configuration known as the *Connected Limited Device Configuration* (CLDC), which is *extremely* limited when compared to Java SE. This has led to the perception that Java ME is a tiny, inadequate environment for serious application development, although the breadth of applications available today for CLDC platforms proves that this is merely a perception, not a reality. Still, the perception persists.

But Java ME also defines a second configuration, the *Connected Device Configuration* (CDC) that incorporates more of the features and richness of Java SE to provide Java ME developers with a much more conventional and much less limited Java programming environment - an environment much closer to Java SE than the CLDC and therefore the one recommended for enterprise application development for mobile phones whenever possible. In fact, the CDC can be viewed as the natural evolution of PersonalJava, an earlier constrained Java programming environment that was a precursor to Java ME and available on the Sony Ericsson P800/P900/P910 smartphones.

# About the Sony Ericsson P990 and M600

---

Our exploration of the CDC focuses specifically on the Sony Ericsson P990 smartphone and the M600 messaging device, but the concepts described here apply to all CDC environments. Separate Developers Guidelines, [Java Platform, Micro Edition - CDC in Sony Ericsson phones](#), provide detailed instructions for downloading and installing the Sony Ericsson CDC Platform Extension Package for the UIQ 3 software development kit (SDK), including software-based P990 and M600 emulators, so that you can test and develop CDC applications from the comfort of your own desktop computer.

# Java ME concepts

Java ME is not a single technology, but rather a group of related Java technologies designed specifically to make Java programming possible on hardware and software platforms where Java SE will not work for a variety of reasons. Successful programming requires a firm understanding of the core Java ME concepts and terminology. It's easy to be overwhelmed by the sheer number of Java ME technologies that are available today or that are in the process of being defined; this chapter summarizes the various technologies.

## The Java Community Process

---

Java ME, like all Java platforms, is developed through the [Java Community Process](#), or JCP for short. A community-driven process that includes members from all parts of the Java development community, the various expert groups of the JCP define and approve the different standards that define the Java ME platform. Now in its second term as an elected member of the Java ME Executive Committee, Sony Ericsson actively participates in a number of Java Expert Groups.

All publicly-released Java ME standards developed through the JCP are available for download from [www.jcp.org](http://www.jcp.org). Once in development, a Java standard is designated as a Java Specification Request, or JSR for short, and given an identifying number. The Connected Device Configuration 1.0, for example, is known as JSR-36. The different standards are often referred to by these designations.

Defining a standard through the JCP is a formal, and usually lengthy, process. When the final version of a JSR is approved, a reference implementation (RI) and a technology compatibility kit (TCK) are also made available to aid vendors in implementing and testing the standard.

The JCP site includes a [list of all Java ME-related JSRs](#). There are more than 75 such JSRs, with new ones constantly being proposed. In this chapter we only touch on the JSRs relevant for the CDC implementation on Sony Ericsson's P990 and M600 phones.

## Configurations

---

At the heart of the Java ME platform is the concept of a *configuration*. A configuration defines a *minimal* Java runtime environment suitable for a certain *class* or *family* of devices. In particular, a configuration defines:

- The capabilities of the Java virtual machine (VM);
- The native code that interfaces with the underlying system;
- A set of Java runtime classes for running applications; and
- Minimal device requirements for memory and input/output mechanisms.

For the most part, a configuration can be considered to be a subset of a Java SE environment.

**Java ME is primarily based on Java 1.3 and Java 1.4:** For the most part, features new to Java 1.5 (also known as Java 5) and higher are *not* yet supported by the Java ME platform. Some Java ME specifications have been updated with Java 1.4 support: for example, CDC 1.0 is based on Java 1.3 but CDC 1.1 is based on Java 1.4. If portability between Java SE and Java ME environments is a concern, you must avoid using more advanced features like generics and enumerations that were introduced post-1.4 until the standards are updated to support them.

It's important to understand that while a configuration does define a *complete* Java runtime environment, that environment on a mobile phone is limited, sometimes extremely so, in many important areas. The library of classes available to the programmer as compared to a Java SE environment is greatly curtailed. In particular, all graphical interface classes are excluded and are left for the profiles to define. A configuration is a base system, but almost all Java ME applications make use of one or more profiles to provide the additional classes required to build a complex, interactive application.

Currently, Java ME defines two configurations:

- the **Connected Limited Device Configuration (CLDC)** for less-capable or otherwise constrained devices, and
- the **Connected Device Configuration (CDC)** for devices with the ability to support a more conventional Java environment similar to Java SE.

The CDC and the CLDC are closely related because the CDC is a superset of the CLDC. What this means is that code written for the CLDC can also run in the CDC **provided that any profiles, classes and vendor-specific extensions that the code uses are also available**. The reality, of course, is that code written for a CLDC-based environment is likely to use features not supported in a CDC-based environment and vice-versa. But some degree of portability is possible if code is written carefully with the CLDC's constraints in mind.

Configurations, like all JSRs, are revised from time to time to incorporate new features. There are currently two versions of the CLDC available, 1.0 ([JSR-30](#)) and 1.1 ([JSR-139](#)), and two versions of the CDC, 1.0 ([JSR-36](#)) and 1.1 ([JSR-218](#)). Differences between these versions, and between the CDC and the CLDC, are discussed in the next chapter.

Until recently, most Java ME-enabled devices have been CLDC-based, not CDC-based. The Sony Ericsson P990 and M600 are among the first mobile phones with built-in support for both configurations.

**PersonalJava:** A Java platform called PersonalJava, available for some time now on certain higher-end mobile phones like the Sony Ericsson P800/P900/P910, predates Java ME and incorporates many of the features found in the CDC and in CDC-based profiles. It has since been superseded by Java ME. Porting from PersonalJava to Java ME is discussed later in this document.

It should be noted that because configurations are complete Java environments that are normally under a vendor's complete control, CDC- or CLDC-based devices may be augmented with proprietary extensions defined through the JCP. Although using such extensions in an application lets the application take advantage of the unique features of a phone, which makes the application more attractive for promotion and distribution by the handset manufacturer, it also hampers the application's portability. When possible, use the classes and interfaces defined in community-supported profiles and optional packages instead.

Read the Sony Ericsson Developers Guidelines for Java ME CDC and Java ME CLDC for details on what JSRs Sony Ericsson phones support and specific guidance on how to develop Java ME applications beyond what is presented here.

# Profiles

---

While a configuration provides broad, if limited, support for a class or family of devices, a *profile* extends a configuration with additional APIs that expand the capabilities of the platform for application developers while simultaneously narrowing the type of supported devices. A profile can impose additional hardware and software requirements on devices that exceed or complement those of the underlying configuration. If a profile defines APIs for graphical user interface creation, for example, it will also specify the minimum screen size and pixel depth expected of devices that support the profile. At a bare minimum, a profile will define the additional memory requirements needed to support a complete implementation of the profile.

A number of profiles have been developed:

- the **Mobile Information Device Profile (MIDP)**, available today in versions 1.0 ([JSR-37](#)) and 2.0 ([JSR-118](#)), with a third ([JSR-271](#)) currently under development, for mobile information applications and voice communication devices (mobile phones);
- the **Information Module Profile (IMP)**, which subsets the MIDP ([JSR-195](#) and [JSR-228](#)), for embedded devices without graphic display capabilities;\*
- the **Foundation Profile (FP)**, versions 1.0 ([JSR-46](#)) and 1.1 ([JSR-219](#)), for CDC-based devices with additional network connectivity;
- the **Personal Basis Profile (PBP)**, versions 1.0 ([JSR-129](#)) and 1.1 ([JSR-217](#)), for network-connected devices with a basic level of graphic capability;
- the **Personal Profile (PP)**, versions 1.0 ([JSR-62](#)) and 1.1 ([JSR-216](#)), for devices with more complicated graphic capabilities; and
- the **Digital Set Top Box Profile (DSTBP)** ([JSR-242](#)), for programmable cable television "set top" boxes.\*

\*Not relevant/applicable to mobile phones.

The MIDP, IMP and DSTBP specifications are CLDC-based profiles, but the FP, PBP and PP specifications build on the features of the CDC.

The first Java ME profile developed and released by the JCP was the MIDP. Most Java-enabled mobile phones in use today are CLDC-based and support either MIDP 1.0 or MIDP 2.0. Nothing precludes phones from support CLDC and CDC environments, however. For example, the Sony Ericsson P990 and M600 include MIDP 2.0 support in addition to supporting the FP 1.0, PBP 1.0 and PP 1.0 standards.

The CDC-based profiles are discussed in more detail later in this document.

## Optional packages

---

The final core Java ME concept is that of the *optional package*. An optional package extends a configuration or profile with additional support for narrowly-defined features that may only be present in certain classes or families of devices. Optional packages are normally defined at the configuration level and can then be used with any profile supported by that configuration. A CLDC-based optional package, for example, can be implemented on *any* Java ME phone with the appropriate underlying support. Here is a sampling of the most important optional packages in use today:

- the **PDA Optional Packages (PDA)** ([JSR-75](#)) adds support for file and contact/calendar/event management to via two optional packages, the **File Connection Optional Package (FCOP)** and the **PIM Optional Package (PIMOP)**;
- the **JDBC Optional Package for CDC/Foundation Profile (JDBCOP)** ([JSR-169](#)) adds a subset of JDBC 3.0 (for database connectivity) to CDC-based phones;
- the **Mobile Media API (MMAPI)** ([JSR-135](#)) adds support for the recording and playback of various multimedia formats;
- the **Wireless Messaging API (WMA)** ([JSR-120](#) and [JSR-205](#)) adds the ability to send and receive SMS and other wireless message types; and
- the **Java APIs for Bluetooth (BTAPI)** ([JSR-82](#)) adds access to Bluetooth-based communications.

Most JSRs now being defined for the Java ME platform are being defined as optional packages, so the list is quite extensive, but few phones support them all.

## Platforms

---

A number of *umbrella* specifications have been defined under the JCP to group commonly-used profiles and optional packages together to clarify their relationships and to impose further restrictions on their use. Such a grouping is often referred to as a Java ME *platform*.

The primary platform defined so far is **Java Technology for the Wireless Industry (JTWI)** ([JSR-185](#)), for wireless handsets based on the CLDC 1.0 and MIDP 2.0 specifications that include support for JSRs 120 and 135.

Two other platforms, the **Mobile Service Architecture for CLDC** ([JSR-248](#)) and the **Mobile Service Architecture for CDC** ([JSR-249](#)) extend JSR-185 by making a large set of optional packages mandatory and further clarifying interrelationships between the various pieces of the MSA platforms.

Most mobile phones in production today, including the Sony Ericsson P990 and M600, support the JTWI platform. Going forward, it is expected that most wireless devices in the future will support either of the two MSA (Mobile Service Architecture) specifications. Sony Ericsson is actively involved in both of these Expert Groups.

# Design considerations when developing applications for the Sony Ericsson P990 and M600

This chapter outlines some design considerations to be aware of when creating applications for the Sony Ericsson P990 and M600 phones.

## Java ME Support

---

The Sony Ericsson P990 and M600 phones support both Java ME configurations. CLDC support is based on CLDC 1.1 and MIDP 2.0. A number of optional packages are supported, see the Sony Ericsson [P990 product description](#) and [M600 product description](#) pages for more details.

CDC support in the P990 and M600 consists of the following:

- CDC 1.0 (JSR-36)
- Foundation Profile 1.0 (JSR-46)
- Personal Basis Profile 1.0 (JSR-129)
- Personal Profile 1.0 (JSR-62)
- JDBC Optional Package for CDC/Foundation Profile (JSR-169)
- File Connection Optional Package (JSR-75)
- PIM Optional Package (JSR-75)

Note that the CDC and CLDC implementations on the P990 and M600 currently support different sets of optional packages.

## MIDP or Personal Profile?

---

Since the Sony Ericsson P990 and M600 phones support both Java ME configurations, your first decision as an application developer is to decide which Java environment to use, the MIDP environment or the Personal Profile (PP) environment. A number of factors must be considered, so here are some guidelines to help you decide.

- **Does the application need to run when the phone's flip is closed?**

If so, choose the MIDP, as PP applications on the Sony Ericsson P990 can only run when the flip is open.

- **Which optional packages are needed?**

Access to the WMA, for example, is limited to MIDP applications, while JDBC support is only available to PP applications.

- **Is portability across a wider set of Sony Ericsson phones or phones from other vendors a concern?**

If so, a MIDP application will be more portable, as few other phones today support the PP.

- **Is an application being ported from a PersonalJava or a Java SE environment?**

Then the PP is the better choice.

There are important differences and tradeoffs in using one environment over the other. A good strategy is to write your code as portably as possible.

# Understanding the CDC

We continue our look at CDC-based development by looking closely at the Connected Device Configuration itself and the profiles that depend on it. The CDC is best understood, however, by first looking at the CLDC.

## The CLDC

---

The first configuration to be defined and placed into production, the CLDC is at the heart of most mobile phones in circulation today. The CLDC 1.0 specification defines a very stripped-down form of a Java environment that excludes a number of standard Java features, including:

- Support for floating-point types and operations;
- object finalization, weak references;
- JNI, reflection, serialization; and
- thread groups and class loaders.

The CLDC 1.1 specification reintroduced floating-point operations and weak references but left most of the restrictions unchanged.

The CLDC includes support for a small subset of Java 1.3 system classes from only three core Java packages: `java.lang`, `java.io` and `java.util`. The subset is so tiny that even the collections classes have been excluded, leaving the developer to build data structures based only on `java.util.Enumeration`, `java.util.Hashtable` and `java.util.Vector`. Needless to say, porting existing Java code to the CLDC is non-trivial because of the lack of a comprehensive class library.

**No additions or modifications to Java SE classes:** When a Java SE class is included in a Java ME configuration, profile, or optional package, only private and package level members (methods or data) may be modified or added to the class. Public or protected members may only be removed, not modified.

The CLDC also introduces a new wrinkle to the development process: standard Java bytecode verification was deemed too computationally expensive to perform exclusively on-device. As such, the verification step - performed each time a class is loaded by the virtual machine - was split into off-device and on-device steps. The off-device step, known as *preverification*, does the actual verification of the classes and inserts attributes into the class files. The on-device step then looks for and validates those attributes to ensure that the class file has not been tampered with since its contents were verified. Preverification is done using a tool called a *preverifier*; class files must be processed by the preverifier before being packaged for deployment.

**Preverification is not necessary for CDC development:** The virtual machine in a CDC-based environment performs standard bytecode verification when classes are loaded, hence a preverification step is not required. However, libraries of classes designed for use with either the CDC or the CLDC must be preverified, otherwise the classes will fail to load, often for no apparent reason, on a CLDC-based device.

The CLDC requires a very small footprint (the CLDC 1.0 VM and support classes fit in 128K of memory) and was designed to run on extremely limited devices compared to a full-blown Java SE environment. Early Java ME mobile phones supporting the CLDC 1.0 and MIDP 1.0 specifications often had a 100K, or less, limit on the total size of an application. These constraints have been relaxed considerably in more recent phones - and even more so in CDC-based phones like the Sony Ericsson P990 and M600 - but it's still a challenge to write Java ME applications that run in an environment that is very limited compared to a typical desktop-based system.

## The Generic Connection Framework

---

The CLDC (and hence the CDC) includes a set of new classes and interfaces not found in Java SE called the Generic Connection Framework, or GCF for short. Strictly speaking, then, the CLDC is not a true subset of Java SE. However, the Generic Connection Framework Optional Package for Java SE ([JSR-197](#)) can be used to add the GCF to a Java SE implementation and make it easier to port Java ME code, especially from a CDC-based phone like the Sony Ericsson P990 or M600, back to a Java SE environment. All GCF classes and interfaces are found in the `javax.microedition.io` package.

The GCF greatly simplifies the Java input/output (I/O) model. In Java SE, I/O operations are spread across different classes in different packages. File I/O, for example, is mostly performed using `java.io.File`, `java.io.FileReader` and `java.io.FileWriter`, while the `java.net` package is used for socket-based I/O. Given that many CLDC-based phones lack file systems and/or socket-based connectivity, it didn't make sense to enlarge the CLDC footprint with the Java SE I/O model.

The GCF defines a single factory class, `Connector`, whose static methods are used to obtain a *connection* to an input or output source. Connections are represented by the `Connection` interface and its extensions. Connection sources are represented as URLs:

```
import javax.microedition.io.*;

.....

StreamConnection conn = null;
String url = "socket://foo.myco.com:7836";

try {
    conn = (StreamConnection) Connector.open( url );
    InputStream in = conn.openInputStream();
    OutputStream out = conn.openOutputStream();
    .... // read and write data
}
catch( IOException e ){
    .... // handle errors
}

if( conn != null ){
    try { conn.close(); } catch( IOException e ){}
}
```

Remember that object finalization is missing from the CLDC, which makes it imperative that you close I/O connections explicitly and as soon as possible in your applications, otherwise resource leakage can lead to application failure. Even CDC-based applications should follow this advice, since many phones have absolute limits on the number of connections that an application (or the system as a whole) can have open at any given time.

The GCF itself does not *mandate* support for any specific I/O connection type, leaving it up to the configurations and profiles to define what's required. The CLDC does not require any I/O connectivity, but the CDC does expose file and datagram support through the GCF. Specific support for HTTP, the most common I/O transport for mobile phones, is actually defined at the profile level.

## The Mobile Information Device Profile

---

No discussion of the CLDC and the GCF is complete without mentioning the Mobile Information Device Profile (MIDP). As mentioned previously, MIDP is the leading platform for Java ME development today. From a programming perspective, the MIDP specifications add a number of important features to the basic environment provided by the CLDC:

- a new model for application deployment and startup;
- a simpler set of graphical user interface classes **not** based on Java SE's Abstract Windowing Toolkit (AWT) classes;
- HTTP-based connectivity via the new GCF-based `HttpConnection` interface; and
- special classes for writing games, including multimedia playback.

The MIDP thus makes it possible to write interactive networked applications that can run on a variety of mobile devices.

A MIDP application is referred to as a *MIDlet*. Its lifecycle is similar to that of a Java applet. One or more MIDlets packaged together in a JAR file are referred to as a *MIDlet suite*. Applications are always deployed as MIDlet suites. Support is defined specifically for HTTP-based wireless deployment of MIDlet suites, also known as *over-the-air (OTA) provisioning*.

## The CDC

---

The CDC is a much more comprehensive Java environment than the CLDC. No language features are excluded - a standard Java VM is at the heart of the CDC. No preverification step is required. A much larger footprint - a minimum of 512K is required for the CDC 1.0 - allows for more Java SE classes to be included. The list of supported packages is longer:

- `java.io`
- `java.lang`
- `java.lang.ref`
- `java.lang.reflect`
- `java.lang.math`

- `java.net`
- `java.security`
- `java.security.cert`
- `java.text`
- `java.util`
- `java.util.jar`
- `java.util.zip`

As with the CLDC, however, the CDC does not include every class from a given package. The `java.net.Socket` class is missing from the CDC, for example, though `java.net.DatagramSocket` is included. (Stream-based socket support is reintroduced in the Foundation Profile.)

Porting an application between Java SE and the CDC is simpler because the CDC includes the basic collections classes that are missing from the CLDC.

In terms of I/O support, the CDC includes both the GCF and the Java SE classes for file and datagram manipulation. For example, files read or written directly via `java.io.FileReader` and `java.io.FileWriter` or by using the GCF with a URL based on the `file:` protocol. If portability with CLDC-based environments is desirable, use the GCF whenever possible.

## The Foundation Profile

---

The Foundation Profile (FP) extends the CDC with additional classes drawn from the Java SE libraries, such as `java.io.StringReader` and `java.io.StringWriter`. It also includes support for standard sockets and HTTP-based connectivity. Socket support is based on the `java.net.Socket` class from Java SE, which is also exposed via the GCF using the `socket:` protocol. HTTP support is also available two ways: via `java.net.URLConnection` or via the GCF using the `http:` protocol. For the latter, the FP includes the `HttpConnection` interface defined by the MIDP specification.

**HTTP support is compatible with MIDP:** HTTP is the standard communication transport for MIDP applications to communicate with external servers, so much code has been written in support of that communication path. That code runs unchanged in FP environments assuming it doesn't make use of any other MIDP-specific features.

Note that the Foundation Profile is designed to serve primarily as the foundation for creating other CDC-based profiles. Classes that were kept out of the CDC to keep it as small as possible but that are considered generally useful for more advanced profiles are included in the FP. Both the Personal Basis Profile and the Personal Profile are based on the Foundation Profile, for example. The Sony Ericsson P990 and M600, like most CDC-based devices, includes the FP.

# The Personal Basis Profile

---

The Personal Basis Profile (PBP) extends the Foundation Profile with new features and with more classes drawn from the Java SE class library. In particular, the PBP allows for the creation of interactive applications in a CDC-based environment.

Features new to the PBP include:

- the Xlet application model for controlling application lifecycle, based on a similar model defined in the Java TV APIs;
- a local interprocess communication model based on Remote Method Invocation (RMI) called *Inter-Xlet communication* (IXC); and
- a subset of the Java SE AWT for creating user interfaces based solely on lightweight AWT components.

**Lightweight vs. heavyweight components:** Abstract Windowing Toolkit (AWT) was originally based on a *peering* model where AWT components (Java objects) were each peered with an equivalent component in the native GUI environment. The native component did the actual drawing and user interaction, with the Java component acting as its proxy in the application. Such a peered component is referred to as a *heavyweight* component. The capabilities and the look of heavyweight components are limited by the underlying platform, so the concept of a *lightweight* component was added to AWT. A lightweight component is a component with no native peer. Lightweight components draw themselves and directly handle user interaction. The Swing toolkit (part of Java SE but *not* part of any Java ME profile) is the best-known example of lightweight component usage.

Most of the new classes in the PBP are drawn from the following Java SE packages:

- `java.awt`
- `java.awt.color`
- `java.awt.event`
- `java.awt.image`
- `java.beans`
- `java.rmi`
- `java.rmi.registry`

Note that full RMI support is *not* part of the PBP - only the RMI classes necessary to support IXC are included.

The Sony Ericsson P990 and M600 include the PBP.

# The Personal Profile

---

The Personal Profile (PP) extends the PBP in three major areas:

- applet support;
- heavyweight AWT components; and
- clipboard support.

The new Java SE packages included by the PP are `java.applet` and `java.awt.datatransfer`.

The PP is also included with the Sony Ericsson P990 and M600 phones.

## The User Interface

---

Most Java-based user interface development today is done using the Swing toolkit. Unfortunately, Swing is *not* included in the Personal Profile. PP applications developed for the Sony Ericsson P990 and M600 must use the original AWT heavyweight components or a third-party toolkit based on AWT.

**Third-party toolkits can use lightweight components:** All user interface toolkits that use lightweight components, including Swing, actually depend on specific support in AWT for creating top-level windows and for drawing and managing sub-components of those windows, which is exactly what lightweight components are. The fact that Swing is excluded from the Personal Profile does not mean that lightweight components cannot be used to build applications, but only that a toolkit must be included with the application.

When developing your user interfaces, try not to hardcode device-specific information. Use the `Toolkit` instance at runtime to determine screen dimensions, for example:

```
Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
```

Position components on frames and dialogs using layouts like `BorderLayout` and `GridBagLayout` – these will require extra effort on your part but will ensure that your application works seamlessly on future versions of CDC-based Sony Ericsson phones where screen dimensions may be significantly different.

## The Application model

---

The Personal Profile supports *three* different ways to start and manage applications:

- The traditional **main( String[] args )** entry point.
- The **applet** model.
- The **Xlet** model.

The traditional model is the simplest to understand, but it provides no facility for the system to manage the application other than by forcing it to shut down. The applet and Xlet models, on the other hand, provide callbacks (much like those used in the MIDP) that let the system notify the application that it needs to pause itself or to cleanly shut itself down. The Xlet model is designed primarily for use on set-top boxes.

On the Sony Ericsson P990 and M600 phones, applications are written using the traditional model.

# Migrating from PersonalJava

A Java platform known as PersonalJava predates Java ME. A number of mobile devices, including smartphones from Sony Ericsson, shipped with a PersonalJava environment. PersonalJava has been discontinued and replaced by the Personal Profile, though the two are not identical. This chapter describes how to migrate code from PersonalJava to the Personal Profile.

## PersonalJava

---

PersonalJava was really the first attempt at adapting the Java platform to meet the needs of more constrained devices. Unlike the CLDC or even the CDC, however, PersonalJava has a much larger footprint and has been generally used only on high-end devices. The Sony Ericsson P800, P802, P900, P908 and P910 series of smartphones all support PersonalJava, for example. The Sony Ericsson P990, however, replaces PersonalJava with the Personal Profile.

PersonalJava is mostly based on Java 1.1, with support for a small set of additional classes drawn from Java 1.2. All Java ME platforms, in contrast, are based on Java 1.3 or higher.

**PersonalJava applications and the Personal Profile:** In general, most PersonalJava applications will run unchanged in a Personal Profile environment. Those that don't will normally require only minor changes to adjust for missing classes and methods.

## Common Classes

---

The overlap between PersonalJava and the Personal Profile is considerable. The classes in the following PersonalJava packages are also found in the Personal Profile:

- `java.applet`
- `java.awt.datatransfer`
- `java.awt.event`
- `java.awt.image`
- `java.io`
- `java.lang`
- `java.reflect`
- `java.net`
- `java.security`
- `java.security.cert`

- `java.security.interfaces`
- `java.security.spec`
- `java.text`
- `java.util`
- `java.util.jar`
- `java.util.zip`

In the `java.awt` package, `PersonalJava` includes two classes that are missing from the Personal Profile - `PrintGraphics` and `PrintJob` - but are otherwise identical.

## Porting issues due to missing classes and methods

---

Unfortunately, a number of `PersonalJava` classes are missing from the Personal Profile. Some of the classes are merely found elsewhere: RMI classes (except for those required for inter-Xlet communication) are now found in the RMI Optional Package ([JSR-66](#)) (not yet supported on Sony Ericsson phones) while JDBC classes (for database connectivity) are in the JDBC Optional Package for CDC/FP ([JSR-169](#)).

Other classes have no Java ME equivalent, such as the design-time classes from the `java.beans` package and certain `PersonalJava` APIs in subpackages of `com.sun`. Applications that use these classes will need to be rewritten.

Another area that may cause portability issues are deprecated methods. All methods deprecated in Java 1.1.8, with the exception of those in the AWT classes, are no longer found in the Personal Profile. Applications that use these methods will also need to be rewritten.

# Developing for the CDC

Developing applications for CDC-based Java platforms is very similar to developing for the Java SE platform. The same development tools can be used for both tasks, with only minor adjustments to the build process. The real differences arise when it's time to test an application.

## Installing the CDC extensions

---

This guide assumes that you've followed the instructions in the [Java Platform, Micro Edition - CDC in Sony Ericsson phones](#) Developers Guidelines to correctly install the CDC development environment for the Sony Ericsson P990 and M600 phones. You must install the UIQ 3 SDK, available from <http://developer.uiq.com/downloads/sdk>, and then the Sony Ericsson CDC extension packs for the P990 and M600, available separately from <http://www.sonyericsson.com/developer>. After installation, the P990CDC or M600 extension pack must be activated and environment variables configured as described in the Developers Guidelines.

## Defining the CDC root

---

After installing the CDC development environment, be sure to define the `EPOCROOT` and `SDKDRIVE` environment variables correctly. `EPOCROOT` must refer to the installation path of the UIQ 3 SDK, but it must NOT contain the drive letter and MUST start and end with a backslash:

```
set EPOCROOT=\symbian\UIQ3SDK\
```

`SDKDRIVE` refers to the drive on which the SDK is installed:

```
set SDKDRIVE=c
```

To simplify development, define a new environment variable called `CDCROOT` as follows:

```
rem Sets CDCROOT in two steps for readability
set CDCROOT=%SDKDRIVE%:%EPOCROOT%epoc32\release\wincw
set CDCROOT=%CDCROOT%\udeb\z\Resource\ive\lib\jclFoundation10
```

Note that the `sdkext` command used to activate the P990CDC or M600 extension pack will have copied the necessary files into the `%EPOCROOT%\epoc32` tree.

**Alternate shell users:** If you use a replacement shell like 4NT, be aware that the `sdkext` command may not copy the necessary files. Re-run the `sdkext` command using the standard Windows shell.

## Compilation

---

The standard Java compiler, `javac`, is used to compile Java source code into Java bytecode format. The `-bootclasspath` option must be used to override the standard Java SE classes with their CDC equivalents. This ensures that your application only uses supported classes – any attempt to use an unsupported class will be reported by the compiler as an error.

The following files should always be included in the boot classpath when compiling applications for the Sony Ericsson P990 or M600:

- `%CDCROOT%\classes.zip` – the CDC and Foundation Profile classes
- `%CDCROOT%\locale.zip` – resources for various locales
- `%CDCROOT%\ppro10\ppro-ui.jar` – the Personal Basis Profile and the Personal Profile classes
- `%CDCROOT%\ppro10\ppro-extras.jar` – additional PBP and PP classes

These files from the `ext` directory can also be included as necessary:

- `%CDCROOT%\ext\fc.jar` – the File Connection Optional Package (part of the PDA Optional Packages, JSR-75)
- `%CDCROOT%\ext\pim.jar` – the PIM Optional Package (part of the PDA Optional Packages, JSR-75)
- `%CDCROOT%\ext\jdbc.jar` – the JDBC Optional Package (JSR-169)

When compiling, the `-target 1.2` and `-source 1.2` options must be set to the 1.2 format to ensure that the correct class file format is used.

Here's a sample command to compile a Java source file for the Personal Profile:

```
javac -bootclasspath %CDCROOT%\classes.zip;%CDCROOT%\locale.zip;%CDC-  
ROOT%\ppro10\ppro-ui.jar;%CDCROOT%\ppro10\ppro-extras.jar -source 1.2 -target  
1.2 MyClass.java
```

Be sure to adjust the classpath appropriately for your application.

**Newer Java features are not supported:** Language features introduced in Java 1.3 and higher, such as assertions (1.4) and generics (1.5), are not available when the `-source 1.2` option is used.

## Packaging

---

After the classes have been compiled, they need to be packaged together into a standard JAR file for testing and deployment:

```
jar cf MyApp.jar MyClass.class
```

Then you create a `MyApp.j9` file in the same directory as `MyApp.jar`. The minimal `j9` file for the application is:

```
-cp c:\MyAppDir\MyApp.jar MyClass
```

Now run the `create-ppro-app` tool to create a Symbian executable for the application:

```
create-ppro-app win32 MyApp 10000000 c:\MyAppDir\MyApp.j9
```

The application can now be tested with the emulator.

## Building with Ant

---

The steps required to build an application can be combined together into a single build process using the [Apache Ant](#) build tool. Here is a sample build file that automates the creation of the "MyApp" application:

```
<?xml version="1.0"?>
<project name="MyApp" default="all">

  <property environment="env"/>

  <property name="app" value="MyApp"/>
  <property name="app.jar" value="${app}.jar"/>
  <property name="app.j9" value="${app}.j9"/>
  <property name="uid" value="10000000"/>

  <property name="epoc"
    value="${env.SDKDRIVE}:${env.EPOCROOT}epoc32"/>
  <property name="cdcroot"
value="${epoc}/release/winscw/udeb/z/Resource/ive/lib/jclFoundation10"/>

  <property name="build.dir" location="build"/>

  <property name="compile.debug" value="true"/>
  <property name="compile.deprecation" value="false"/>
  <property name="compile.optimize" value="false"/>
  <property name="compile.version" value="debug"/>
  <property name="constant.debug" value="true"/>
```

```

<target name="all" depends="compile,dist"
    description="Clean build and dist, then compile">
</target>

<target name="clean"
    description="Delete old build and dist directories">
    <delete dir="${build.dir}"/>
    <delete file="${app.jar}"/>
</target>

<target name="compile" depends="prepare"
    description="Compile Java sources">
    <path id="cdc.classpath">
        <pathelement path="${cdccroot}/classes.zip"/>
        <pathelement path="${cdccroot}/locale.zip"/>
        <pathelement path="${cdccroot}/ppro10/ppro-ui.jar"/>
        <pathelement path="${cdccroot}/ppro10/ppro-extras.jar"/>
        <pathelement path="${cdccroot}/ext/fc.jar"/>
        <pathelement path="${cdccroot}/ext/pim.jar"/>
        <pathelement path="${cdccroot}/ext/jdbc.jar"/>
    </path>

    <javac destdir="${build.dir}"
        srcdir="."
        source="1.2"
        target="1.2"
        debug="${compile.debug}"
        deprecation="${compile.deprecation}"
        optimize="${compile.optimize}">
        <bootclasspath refid="cdc.classpath"/>
    </javac>
</target>

<target name="dist" depends="compile"
    description="Create binary distribution">
    <jar jarfile="${app.jar}"
        basedir="${build.dir}"/>
    <exec executable="cmd.exe">
        <arg line="/c create-ppro-app win32 ${app} ${uid}
${basedir}\${app.j9}"/>
    </exec>
</target>

<target name="prepare">
    <mkdir dir="${build.dir}"/>
</target>

</project>

```

Note that the `SDKDRIVE` and `EPOCROOT` environment variables must be set before invoking this file. Three primary targets are defined: `clean` to delete temporary files, `compile` to compile the classes, and `dist` to create the jar file and build the Symbian OS executable.

## Testing the application

---

The simplest way to test the application is with the emulator. Before launching the emulator for the first time, however, be sure to run the `SDKConfig` tool to configure support for the P990 or the M600, as appropriate. (Note that the P990 emulator must be set to the `P990 CDC Flip Open` style, because CDC applications on the P990 can only run in flip open mode.) You should also modify the `epoc.ini` file found in `%EPOCROOT%\epoc32\data` to include the following line:

```
_epoc_drive_d c:\MyAppDir
```

This maps the emulator's `D:` drive to the `C:\MyAppDir` directory on your computer. You must also change the `.j9` file to refer to the `D:` drive:

```
-cp d:\MyApp.jar MyClass
```

This lets you test the application without having to copy its files into the emulator's directory tree. Once these changes are made, invoke the emulator by using the `SDKConfig` tool or directly from the command line:

```
%EPOCROOT%\epoc32\release\wincsw\udeb\epoc.exe
```

Your application should now be in the emulator's application list.

## Enabling console output

---

The ability to print to the standard output and error streams is very valuable to the developer. Later examples in this document make use of these streams. In order to enable console output to be trapped and redirected to a file, create the following directories:

```
mkdir %epocroot%\epoc32\wincsw\c\logs
mkdir %epocroot%\epoc32\wincsw\c\logs\j9vm
```

When a CDC application runs and outputs to the `System.out` stream, a file named `pid_stdout.txt` is created in the `j9vm` directory, where `pid` refers to the process ID for the application. Similarly, output to `System.err` is redirected to `pid_stderr.txt`.

# Accessing the file system

The file system on the Sony Ericsson P990 and M600 phones can be accessed in two ways. Which way you choose depends more on portability considerations than anything else.

## Standard file access classes

---

The Personal Profile supports the standard Java SE file access classes from the `java.io` package, including:

- `java.io.File`
- `java.io.FileReader`
- `java.io.FileWriter`
- `java.io.FileDialog`

Code written with these classes will work unchanged in a PP environment, subject of course to the limits and restrictions on the underlying filesystem.

## The Generic Connection Framework

---

All CDC implementations minimally support the reading of files via the Generic Connection Framework (GCF):

```
import java.io.*;
import javax.microedition.io.*;

InputConnection ic = null;

try {
    ic = (InputConnection) Connector.open( "file:///d:/data.txt" );
    // process the file here
}
catch( IOException e ){
    // error
}
catch( IllegalArgumentException e ){
    // bad format
}

if( ic != null ){
    try { ic.close(); } catch( IOException e ){} }
}
```

```
}
```

The Sony Ericsson P990 and M600 also support the writing of files through this interface by using the second form of the `Connector.open` method:

```
OutputConnection oc = (OutputConnection)
    Connector.open( "file:///d:/data.txt", Connector.WRITE );
```

GCF operations other than reading or writing files can be done using the File Connection Optional Package.

## The File Connection Optional Package

---

The CDC's support for GCF-based file operations is very weak. No means are provided to create files or directories or to perform other common tasks on files except through the standard Java SE file classes. Such capabilities are provided, however, by the File Connection Optional Package (FCOP).

The FCOP defines a new interface, `FileConnection`, for file manipulation. Part of the new `javax.io.microedition.file` package, `FileConnection` extends the GCF's `StreamConnection` interface - used for two-way, stream-based connections - to provide support for checking that a file or directory exists, deleting files, rename files, and so forth. The same syntax is used to obtain the connection:

```
FileConnection fc = (FileConnection)
    Connector.open( "file:///d:/data.txt" );
```

The `FileConnection` instance is analogous to a `java.io.File` instance in many ways:

```
FileConnection fc = .... // a file connection

if( fc.exists() && !fc.isDirectory() ){
    DataInputStream in = fc.openDataInputStream();
    ..... // process
    in.close();
}

fc.close();
```

Because the FCOP is used in mobile phones, however, it also provides classes for detecting changes to the filesystem, such as the removal of a memory card. On the Sony Ericsson P990 and M600, this can be used to detect the removal and reinsertion of the Memory Stick that maps to the `D:` drive.

Note that the P990 and M600 support the FCOP in both the MIDP and Personal Profile environments.

# Accessing the PIM system

The PIM (Personal Information Management) data on a Sony Ericsson P990 and M600 can only be accessed through the PIM Optional Package.

## Overview of the PIM Optional Package

---

The PIM Optional Package (PIMOP) provides access to three kinds of PIM data: addresses, calendar entries, and action or "to do" items. On the Sony Ericsson P990 and M600, these map directly to the data managed by the Contacts, Agenda and Todo applications. For security reasons, information in these applications can *only* be accessed through the PIMOP. Like the File Connection Optional Package, the PIMOP is available in both the MIDP and Personal Profile environments.

PIMOP classes and interfaces are defined in the `javax.microedition.pim` package. An instance of the `PIM` factory class is required to access PIM data:

```
PIM pim = PIM.getInstance();
```

Most PIM operations throw `PIMException` on error, but we'll omit the error checking in the samples shown here.

The PIMOP is designed to work across a wide variety of PIM-capable devices, not all of which store the same information. Specific details on which types of data are supported by the PIMOP on the Sony Ericsson P990 and M600 can be found in the "Programming Issues" section of the [Java Platform, Micro Edition - CDC in Sony Ericsson Smartphones](#) Developers Guidelines.

## Security Restrictions

---

Access to PIM data is tightly controlled in order to protect the user's confidential information. On a real device, only signed applications can access the data – signing details are described in the Developers Guidelines.

When building an application that uses the PIMOP, the `ReadUserData` and `WriteUserData` must be added to the list of capabilities required by the application during the `create-pro-app` step:

```
create-pro-app win32 MyApp E0000000 c:\MyAppDir\MyApp.j9 ReadUserData WriteUserData
```

If you don't do this, your application will terminate abruptly when it first accesses PIM data, even within the emulator.

You can disable the emulator's checking of security by turning off platform security within the `epoc.ini` file:

```
PlatSecEnforcement OFF
```

It's recommended that you test with security restrictions enabled, however.

## Accessing PIM data

---

To access any PIM information, you must first open the appropriate database. Each database is identified by a constant in the `PIM` class: `CONTACT_LIST` (addresses), `EVENT_LIST` (calendar) and `TODO_LIST`. Each database type has a corresponding list type: `ContactList`, `EventList` and `ToDoList`. Each of these extends the `PIMList` class. Here's an example of opening the contact database:

```
ContactList clist = (ContactList)
    PIM.getInstance().openPIMList( PIM.CONTACT_LIST, PIM.READ_WRITE );
```

Note that the Sony Ericsson P990 and M600 only support the `PIM.READ_WRITE` mode to access these databases.

After opening the database, various operations can be performed. To list the data currently in the database, use the `items` method, which returns an enumeration:

```
Enumeration e = clist.items();
while( e.hasMoreElements() ){
    Contact contact = (Contact) e.nextElement();
    String[] name = contact.getStringArray( Contact.NAME, 0 );

    System.out.println( "Given: " + name[ Contact.NAME_GIVEN ] );
    System.out.println( "Family: " + name[ Contact.NAME_FAMILY ] );
}
```

**PIMOP written to CLDC standard:** The PIMOP, like the FCOP, can be used with the CLDC as well as the CDC, so it can only refer to classes such as `java.util.Enumeration`, not the more commonly used `java.util.Iterator`.

There are separate classes for each type of database entry: `Contact`, `Event` and `ToDo`, each of which extends the `PIMItem` class. New items are created using factory methods on the list objects:

```
Contact contact = clist.createContact();
```

Here's an example of how to set the name of a contact:

```
String[] name = new String[ clist.stringArraySize( Contact.NAME ) ];
name[ Contact.NAME_GIVEN ] = "Melissa";
name[ Contact.NAME_FAMILY ] = "Gilfer";

contact.addStringArray( Contact.NAME, PIMItem.ATTR_NONE, name );
contact.commit();
```

The exact details vary depending on the fields being set, of course. When done reading or writing data, be sure to close the database:

```
clist.close();
```

# Conclusion

At this point you should have a good understanding of CDC programming in general and CDC development for the Sony Ericsson P990 and M600 in particular. In combination with the PDA Optional Packages, the Personal Profile provides a rich programming environment that will be comfortable to most Java developers, especially those who have never done any Java ME programming.

# References

**CDC Platform Extension Package for the UIQ 3 SDK** (Sony Ericsson):

[www.sonyericsson.com/developer/symbian](http://www.sonyericsson.com/developer/symbian)

**Connected Device Configuration 1.0** (Sun Microsystems):

[www.jcp.org/en/jsr/detail?id=36](http://www.jcp.org/en/jsr/detail?id=36)

**Connected Limited Device Configuration 1.0** (Sun Microsystems):

[www.jcp.org/en/jsr/detail?id=30](http://www.jcp.org/en/jsr/detail?id=30)

**Foundation Profile 1.0** (Sun Microsystems):

[www.jcp.org/en/jsr/detail?id=46](http://www.jcp.org/en/jsr/detail?id=46)

**Java Community Process** (Sun Microsystems):

[www.jcp.org](http://www.jcp.org)

**Java Platform, Micro Edition - CDC in Sony Ericsson Phones** (Sony Ericsson):

[www.sonyericsson.com/developer/symbian](http://www.sonyericsson.com/developer/symbian)

**JDBC Optional Package for CDC/Foundation Profile** (Sun Microsystems):

[www.jcp.org/en/jsr/detail?id=169](http://www.jcp.org/en/jsr/detail?id=169)

**Personal Basis Profile 1.0** (Sun Microsystems):

[www.jcp.org/en/jsr/detail?id=129](http://www.jcp.org/en/jsr/detail?id=129)

**Personal Profile 1.0** (Sun Microsystems):

[www.jcp.org/en/jsr/detail?id=62](http://www.jcp.org/en/jsr/detail?id=62)

**PDA Optional Packages** (PalmSource/IBM):

[www.jcp.org/en/jsr/detail?id=75](http://www.jcp.org/en/jsr/detail?id=75)

**UIQ 3 SDK** (Symbian):

[developer.uiq.com/downloads/sdk](http://developer.uiq.com/downloads/sdk)