OMTP

TECHNOLOGY ASSESSMENT

OMTP JAVA™ WITH FOCUS ON CDC: DEFINITION AND REQUIREMENTS

VERSION: Version 1_0, Release 1

STATUS: Approved

DATE OF LAST EDIT: 10 January 2006

OWNER: OMTP Java™ Workstream

CONTENTS

1 PREFACE

1.1 DOCUMENT PURPOSE

This document is the OMTP Java™ with Focus on CDC Requirements Specification. The purpose of the document is to identify requirements from the perspective of wireless network operators to develop an advanced mobile execution environment based on Java™ technology.

Java™ technology has established itself as a leading execution environment for downloadable applications. The industry has collectively identified the potential in the technology to go beyond its current state. In order to enable the delivery of further value-added services to consumers, the Java™ Micro Edition (Java™ ME) platform needs to grow in terms of APIs that the developers can use to write interesting applications. There is also a need to alleviate the fragmentation introduced due to implementation variations.

This document builds upon existing Java™ platform standards, and identifies technology gaps that are not yet covered by existing JCP (Java™ Community Process) standards. The OMTP Java™ Workstream will feed these gap requirements into JCP standardization activities (which may potentially initiate new JSRs) to ensure that all the operator requirements can be fulfilled by the Java™ platform. A central goal of this document is to perform a gap analysis for advanced mobile terminal platforms that are based on Java™ technology.

The document also identifies and defines key operator requirements for the Java™ platform in the user interface area. In addition, this document provides criteria to effectively evaluate two possible user interface technologies for the advanced Java™ ME platform.

1.2 INTENDED AUDIENCE

This requirement document is intended primarily for companies and individuals who are involved in the standardization and implementation of the Java™ Micro Edition platform (Java™ ME). The document is potentially useful also for tool vendors, content developers, or other parties who are generally interested in the future directions and evolution of Java™ technology for mobile devices.

1.3 CONVENTIONS

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC2119 [1].

- MUST: This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.

- MUST NOT: This phrase, or the phrase "SHALL NOT", mean that the definition is an absolute prohibition of the specification.
- SHOULD: This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
- SHOULD NOT: This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behaviour is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behaviour described with this label.
- MAY: This word, or the adjective "OPTIONAL", mean that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)

2 INTRODUCTION

2.1 OMTP JAVA™ WORKSTREAM GOALS

The OMTP Java Workstream has several goals that it is working toward. These goals support the overall effort to make the Java execution environment a compelling and complete platform which meets the needs of OMTP which is suitable for both operators and handset manufacturers. The goals that support this overall effort are as follows

- Define the base platform that will be used as the starting point for next-generation advanced mobile devices.
- Define platform requirements that will yield less fragmentation in the market.
- Identify gap technologies that are needed to enable Java™ applications to become more resident on handsets, e.g., in order to support built-in system applications written in the Java™ programming language.
- Drive gap requirements into JCP standardization activities and, if needed, initiate new JSRs to ensure that the operator requirements can be met by the Java™ platform.
- Define User Experience and customization requirements that meet the specified needs of operators.
- Provide evaluation criteria to guide the User Interface technology selection that will meet the operator requirements.

The OMTP Java™ Workstream will feed the requirements into JCP standardization activities, and may potentially initiate new JSRs in order to ensure that the operator requirements can be met by the Java™ platform.

2.2 RELATIONSHIP TO OTHER OMTP GROUPS/WORKSTREAMS

2.2.1 OMTP SOFTWARE GROUP

The OMTP Software group is identifying requirements that are independent of a specific technology. These requirements will be used by the Java™ Workstream as guidance in both the creation of requirements as well as defining specific gap technologies to help meet its goals. The Java™ Workstream, by nature, will establish more specific requirements since its focus is specifically on the development of the Java™ ME platform for the wireless industry.

2.2.2 OMTP USER EXPERIENCE GROUP

The User Experience group is identifying specific use cases and requirements for operators to provide a compelling user experience. The Software group will define generic (platform agnostic) requirements based on this work. Java™ Workstream will then utilize this set of relevant requirements and define the specific User Experience use cases and requirements for the Java™ Platform.

2.2.3 OMTP APPLICATION SECURITY

The Application Security group is identifying security requirements that will help enable the protection of the mobile devices and networks from rogue applications and external attacks. The Java™ Workstream will monitor the work of the Application Security group closely, and utilize the results in the future activities as necessary. The consideration, development, and requirement of a strong and flexible security framework in terminals including Java™ technology is key to their acceptance in the marketplace. Future revisions of this document must align with the work of Application Security group.

2.2.4 OTHER OMTP GROUPS

The output of the other groups will be evaluated for relevance to the Java™ platform. If relevant, this data will be incorporated into the future work of the Java™ Workstream.

3 OMTP JAVA™ PLATFORM VISION

This chapter summarizes the overall Java™ Platform vision and the high-level Java™ platform architecture defined by the OMTP Java™ Workstream. The results in this chapter are based on numerous conversations in the Java™ Workstream meetings, the gap analysis performed by a task force initiated by the Java™ Workstream, as well as the use cases developed during the OMTP activity.

3.1 JAVA™ PLATFORM VISION AND CORE COMPONENTS

In today's mobile devices, the role of the Java™ platform is primarily that of a "content player". The Java™ platform is used mainly for executing 3rd party applications – most typically games – that have been downloaded to the device over the wireless network. The Java™ platform has become enormously successful in this role. It has been estimated that over 800 million mobile devices already support the Java™ ME platform. Tens of thousands of commercial applications have already been developed, generating global revenues of over $1 billion annually.

Based on the requirements and use cases collected from operators, there is a desire to take the role of the Java™ platform in mobile devices further, and to enable the use of the Java™ platform not only as a content development platform but also as a platform for the creation of system software and resident applications. The Java™ platform capabilities should also be extended to enable more flexible user interface customization to meet the operator needs.

In order to support these extended capabilities and use cases, we envision a mobile Java™ platform built upon the more capable J2ME CDC (Connected Device Configuration) standard. An ongoing activity, MSA CDC (JSR 249), is currently defining a comprehensive platform built around CDC technology (see next subsection for a brief summary of the MSA CDC standardisation effort). In order to facilitate a transition from JavaTM ME CDLC based platforms to JavaTM ME CDC based platforms the OMTP JavaTM workstream has agreed upon a set of requirements and recommendations. These requirements and recommendations will be provided to the MSA CDC (JSR-249) expert group. The responsibility for this smooth transition between the platforms falls on the expert group of the MSA JSRs, the Java™ community, as well as the industry as a whole. f

Figure 1 illustrates our overall OMTP Java™ platform vision. We assume that the OMTP Java™ platform shall be built around the MSA CDC platform. This general architecture might also be considered as mechanism to transition existing CLDC based platforms to CDC. In addition, a number of new, additional APIs may have to be developed

in order to support the development of Java™-based system software and resident applications in case the existing standards do not provide the necessary capabilities.

Based on the use cases and requirements, we also assume that the user interface capabilities of the OMTP Java™ platform will have to be more capable than those provided by the libraries used in the majority of Java™-enabled mobile devices today. A key outcome of the OMTP Java™ Workstream activity will be the selection of a suitable user interface library for the advanced Java™ ME platform. The user interface evaluation criteria and evaluation results will be summarized later in this document.

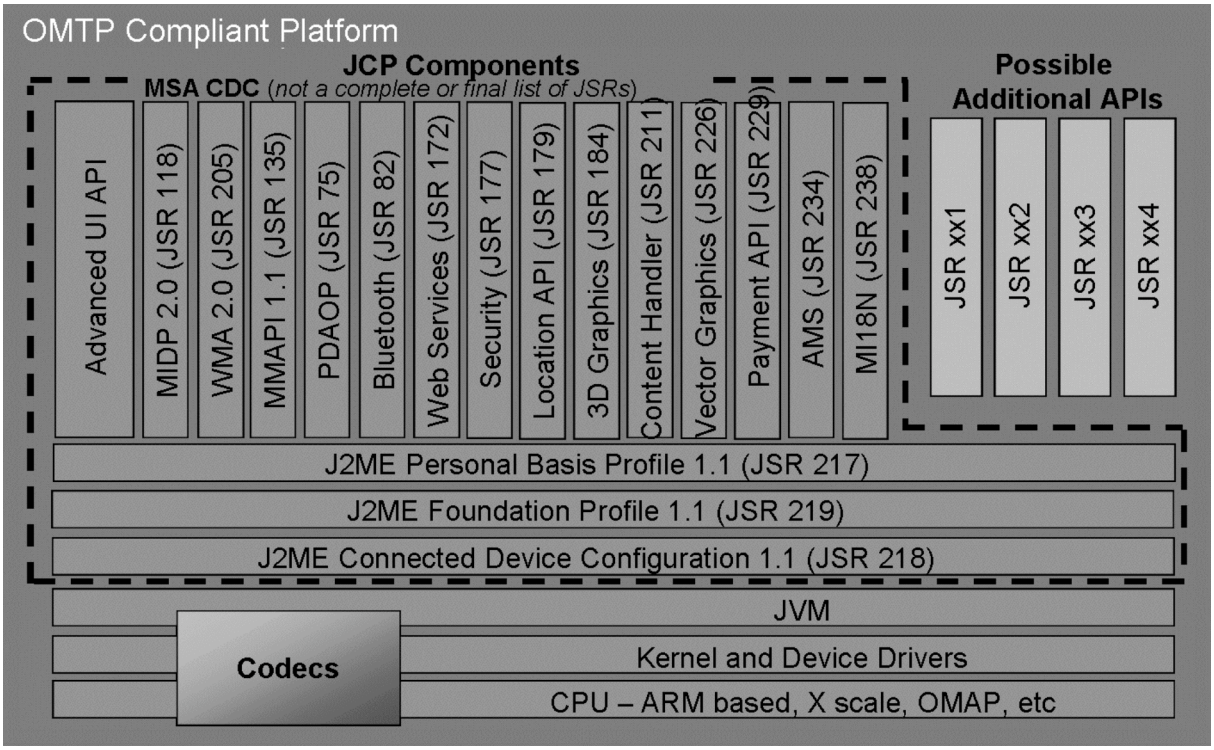

Figure 1: OMTP Java™ Platform Vision

3.1.1 MSA CDC (JSR 249)

The OMTP Java platform definition and requirements document provides requirements and recommendations to the MSA (Mobile Service Architecture) CDC Java platform, defined by the Java Community Process effort JSR 249. MSA CDC is a standardization effort that is currently defining a new, CDC-based Java™ platform targeted towards smartphones and other high-end mobile devices. MSA CDC consists of a Java™ virtual machine that is fully compliant with the Java™ Virtual Machine Specification and the J2SE virtual

machine, augmented with a large number of libraries that provide significantly more functionality than the more limited MSA CLDC platform.

The MSA CDC standardization effort is currently in progress, and the details of the platform are still subject to change. For current information on the MSA CDC platform, refer to the following JCP web site:

http://www.jcp.org/en/jsr/detail?id=249

3.1.2 Relationship to MSA CLDC (JSR 248)

Most mobile devices today are built upon the J2ME CLDC (Connected Limited Device Configuration) platform. The collective "umbrella" standard that defines the set of most widely supported Java™ APIs for the J2ME CLDC platform is known as the Java™ Technology for the Wireless Industry (JTWI) initiative (JSR 185). JTWI devices are already in widespread use in millions of devices, and there is a need for more advanced capabilities and features.

The MSA CLDC standardization effort (JSR 248) is currently defining the next generation mass-market mobile Java™ platform based on the existing JTWI standard. MSA CLDC is targeting high-volume, mass-market devices, and it will provide a subset of the functionality offered by the more capable MSA CDC platform.

For further information on the MSA CLDC platform, refer to the following
JCP web site:

http://www.jcp.org/en/jsr/detail?id=248

3.2 Backwards Compatibility

Backwards compatibility is a critical requirement for the successful deployment and evolution of the Java™ platform. The MSA CDC platform will be a pure superset of the MSA CLDC platform, i.e., all the content developed for the MSA CLDC platform will also run on MSA CDC devices. Correspondingly, MSA CLDC devices must be able to run existing JTWI and MIDP applications.

A key requirement in OMTP is to maintain the backwards compatibility requirement with MSA and earlier versions of the Java™ ME platform. As a general principle, OMTP shall not define features that would violate backwards compatibility with existing J2ME applications, devices or standards, unless such features are specified in order to fix serious incompatibilities between existing devices.

4 IDENTIFIED GAP AREAS/REQUIREMENTS

Although OMTP is not specifying that all applications must be written only in the Java™ programming language, there are currently a number of gaps in the available Java™ APIs that prevent the preparation of a complete set of phone applications or system software in the Java™ programming language. At present, many proprietary extensions are required, and the specification of the Java™ ME platform is not sufficiently taking into account system issues and the needs of resident system applications.

To support the OMTP Java workstream overall goal of defining a compelling and complete platform which meets the needs of OMTP it is also important to have a common understanding of the scope, features as well as the inherent limitations of such an execution environment; in terms of UI replacement/customization, execution and interrupt model (non real time), security and access model. Having this in mind will also ensure a successful adoption by developers and the market at large. Specifically it is envisioned that OMTP Java™ ME CDC will be particular well suited to provide UI front-ends to existing end-user applications exposing network services and device data such as phonebook, bookmarks, messages etc. in a user friendly and highly extensible and flexible way. It is fundamental that existing device software can be tapped into from the OMTP Java™ ME CDC in a fully standards compliant and secure way while making optimal use of state of the art device software that already have been certified for and execute in existing mobile networks.

One of the key goals of the OMTP Java™ Workstream was to identify gap areas that prevent the industry from moving towards mobile devices in which the majority of the resident applications and other system components are written in the Java™ programming language. The approach followed by the OMTP Java™ Workstream was to first analyse the relevant gap areas, and then to identify – for each of the gap areas – one or more existing JSR that can possibly fill or reduce the gap. Liaison statements were sent out to existing/ongoing JSRs to see if those activities could potentially cover the gaps as part of their ongoing work.

The gap areas are summarized in detail in the sections below.

4.1 RUNTIME MANAGEMENT AND ISOLATION

4.1.1 PROBLEM DESCRIPTION

In a phone system that is written mainly in the Java™ programming language, some Java™ applications will have the responsibility of performing high-level runtime management of other applications.

"Other applications" may be integral parts of the system, or extra applications downloaded and installed by the user.

In the absence of effective runtime management, it could be possible for an individual application to use an excessive amount of CPU time or other resources, making it impossible to start up new applications (such as the application to handle incoming phone calls.)

4.1.2 REQUIREMENTS

Given that the runtime management system may be written in the Java™ programming language, there must be a standardized Java™ mechanism for runtime management.

4.1.3 ACTIONS TAKEN

Liaison statements sent to

- JSR 121 (Isolation API)
- JSR 249 (MSA)
- JSR 271 (MIDP3)

4.1.4 RESULTS AND DECISIONS

Note that a new JSR (JSR 278 – Resource Management Framework) is going to be start soon and will focus specifically on resource management. This can potentially fill the gaps in the area of runtime resource management of simultaneously running Java™ applications.

Responses received from JSR 121 and JSR 271 are below:

> (From Grzegorz Czajkowski, Sun) **JSR 121**: Thanks for your comments. Indeed, they touch upon critical functionality, and before proposing this specification we the issue of resource management was on the table. However, bundling RM with isolation in one specification would inevitably delay completing the JSR 121, and there are people for whom there is significant value in Isolates alone. So the issue of resource management won't be addressed in the JSR 121.
>
> However, since the JSR 121 is rapidly approaching completion we (the 121 Expert Group) have started to talk about starting another JSR, this time focused on resource management of J2SE applications. Is it something you and/or your company would be interesting in participating in?
>
> (From Mike Milikich, Motorola) JSR 271 scope does not currently cover the OMTP Group requirements as stated for runtime management and application isolation. The JSR 271 EG is discussing fair application and thread scheduling mechanisms to prevent a single program from denial of service attacks, and we anticipate discussions about providing fair access to other resources within the scope of a MIDP application environment.

However, any runtime management API is not within the current scope proposed for JSR 271.

Given this scope for MIDP3, it may be possible to use privileged MIDlets to implement a platform wide AMS for Java as well as native applications. But JSR 271 will not specify this as an inherent part of the AMS, nor will any APIs be developed to include this functionality. JSR 271 will include the following:

1. JSR-271 will define a method of inter-MIDlet communication.
2. JSR-271 will address the correct operation and management of concurrent MIDlets, but will not include runtime management APIs for control of other applications. As in past MIDP specifications, runtime management functionality is within the domain of the AMS. Isolation of objects between MIDlet suites will be part of what is required to support concurrency.
3. Some additional consideration may be given to concurrency and synchronization utilities, perhaps in the form of JSR 166 or some subset therein.

4.2 FILE SYSTEM

4.2.1 PROBLEM DESCRIPTION

`Java.io` and the Generic Connection Framework **`file:`** mechanisms provide general access to file systems. However, they do not provide the detailed knowledge of how to use the file system in an integrated way on a mobile device. For example, there is no standard way of knowing where to save a sound file as a ring tone.

A possible resolution to this problem is to define properties for these locations. Where an application wishes to save a particular data type for system usage, it should lookup the property based on some criteria. The criteria should include mime type, user (if the OMTP requirements include support for multiple users, and if the device support multiple users), and usage. This mechanism would allow multiple applications to provide services for the same data.

4.2.2 REQUIREMENTS

There must be a standard way for a Java™ application to identify predefined file system locations.

4.2.3 ACTIONS TAKEN

Liaison statement sent to

- JSR 249 (MSA)

4.2.4 RESULTS AND DECISIONS

According to the MSA expert group, this area will be covered by the MSA Specifications. No further actions were deemed necessary by the OMTP Java™ Workstream during the last face to face meeting. However it should be noted that the solution in the Public Review draft of JSR-248 may be insufficient for the following reasons:

1. There should be support for multiple locations for storage of standard types of content. For instance many devices allow storage in phone memory as well as in memory cards.

2. The system property approach could be extended to define a path, but this makes additional work for every application using the properties.

3. Dealing with multiple locations for content and whether or not those locations are currently "reachable" imposes a lot of overhead on application code and fosters code duplication in applications. This overhead could be dealt with much more efficiently by the system, which could present to an application a unified view of what content is currently available regardless of location, whether removable storage is present or not, etc, together with events when the content available changes due for example to changes in network connectivity, removal of a storage card.

4. Lack of support of any metadata relating to the content (e.g. length of music track) and the characteristics of the places where it is stored (removable, persistent, temporary, secure etc) increases the burden on applications.. It should be possible for an untrusted application to get information *about* DRM protected content while not being able to directly access the actual content. Also, applications should be able to access metadata such as track length without having to be intimately familiar with all of the corresponding file formats.

5. JSR-75 leaves the implementation of the security model for the file system APIs up to the platform, so there is no standard way to define what kinds of content (music, ring tones, images, etc) , and what kind of access (read, write, create, delete) an application wants/should be allowed access to, or any way for this to be visible to the user.

4.3 APPLICATION META DATA

4.3.1 PROBLEM DESCRIPTION

Currently, there is no standardized application packaging solution for J2ME CDC applications. The CLDC/MIDP platform provides a solution, but it is not currently directly applicable to J2ME CDC.

Applications in are typically composed of the following elements:

- Application Binary: The application itself (the object code)
- Non-Code Resources: These include data required for the operation of the application
  - Images
  - Resource bundles (for I18N)
  - Data
  - Application Meta Data (the data NOT fundamental to the operation of the application, but perhaps useful in the deployment of the application.)

Examples of Application Meta Data:

- Icon references
- Display Name of Application
- Supported MIME Types
- Versioning Information
- Application library dependencies

4.3.2 REQUIREMENTS

The application packaging solution should meet the following requirements:

1. Meta Data SHOULD work with existing CDC application models
2. Meta Data SHOULD make use of JAR file manifest mechanism
3. Meta Data SHOULD provide pre-download properties (AKA JAD)
4. Meta Data SHOULD NOT be required to be provided with an application

There already exist a number of specifications that provide support for some of the features required to support Application Meta Data. Where

appropriate these facilities should be reused rather than inventing new specifications.

4.3.3 ACTIONS TAKEN

Liaison statement sent to

- JSR 249 (MSA)
- JSR 271 (MIDP3)

4.3.4 RESULTS AND DECISIONS

Response received from the JSR 271 expert group is below:

(From Mike Milikich, Motorola) JSR 271 addresses the four requirements mentioned above.

1. Meta Data SHOULD work with existing CDC application models MIDP3 specified meta data should have no conflicts when implemented on top of CDC, although CDC will not be required.

2. Meta Data SHOULD make use of JAR file manifest mechanism Guaranteed, for MIDP3 / MIDP2 backward compatibility.

3. Meta Data SHOULD provide pre-download properties (AKA JAD) Guaranteed, for MIDP3 / MIDP2 backward compatibility.

4. Meta Data SHOULD NOT be required to be provided with an application Guaranteed, for MIDP3 / MIDP2 backward compatibility.

Additionally, JSR-271 will revisit the split between descriptor and manifest, since currently it is somewhat unclear as to which attributes should reside in each. Furthermore, some MIDP2 required checks at installation necessitate the download of the JAR, negating the benefits of separating attributes into the descriptor.

4.4 PERSISTENT ALARMS

4.4.1 PROBLEM DESCRIPTION

The MIDP API provides the PushRegistry class but this does not provide the flexibility to define an arbitrary number of time-based alarms.

MIDP 2.0 has a suitable mechanism available, but it is currently limited to specifying only one timer-based application launch at a time. CHAPI could have been a place to do persistent alarms, but it's based around handling content rather than responding to timers.

4.4.2 REQUIREMENTS

There must be a mechanism for programmatically specifying a time when a particular application should be started. The number of timer-based application launches should not be limited to just one at a time.

4.4.3 ACTIONS TAKEN

Liaison statements sent to

- JSR 249 (MSA)
- JSR 271 (MIDP3)
- JSR 232 (Mobile Operational Management).

4.4.4 RESULTS AND DECISIONS

Responses received from the JSR 232 and JSR 271 expert groups are below:

> (From Jon Bostrom, Nokia) JSR 232 provides a flexible event system by means of which it is possible to issue events of different types and register to be notified when given events occur. Each event has a topic and a set of properties defined in terms of key-value pairs. When registering for events one has to specify a topic and possibly a filter on the properties. In particular the system is able to deal with timer events. These events have org/osgi/timer as topic. The properties carried by each timer event describe the year, month, day, hour, minute and second the timer event occurred. The event system automatically issues timer events whenever someone is registered for receiving one. As an example, if someone registers for timer events specifying as filter
>
> (& (hour=12) (minute=0) (second=0))
>
> it will be notified everyday at noon.
>
> Moreover JSR232 allows to request the scheduling of an application when a specific event occurs if needed on a recurring basis. If the application is not running when the event occurs, the framework automatically starts it.
>
> The possibility of scheduling applications when given timer events occur seems to provide the power and flexibility that OMTP requires in terms of persistent alarms.
>
> (From Mike Milikich, Motorola) JSR 271 will provide such a mechanism; however, the proposed time format (yyyymm-dd hh:ss zone_offset) may be problematic for some MIDP class mobile devices. Timezone designations and / or the timezone offset from GMT are not consistently known on devices. MIDP3 will address this area.

Note that alarm based MIDlet launch is already provided within the MIDP2 APIs via dynamic registration. MIDP3 will investigate the addition of static registration of alarm based application launch. Note that since alarm based application launching may not be supported on all platforms, this feature will likely remain optional in MIDP3.

With respect to the comment about MIDP3 linking MIDlets to a top level UI window, this is already covered in MIDP2 using Display.getDisplay(MIDlet) to make the binding. If a device supports a window system, the implementation can use top level windows appropriately; the

Display is the top level window for a MIDlet.

4.5 System Information and Notifications

4.5.1 Problem Description

In a mobile terminal there is a lot of information that, though being made available by the underlying operating system, is not accessible by Java™ applications (at least not in a uniform way). Such information includes, for instance, information related to network accounts or information about installed and running applications. Accessing such system information and system attributes is critical in enabling the preparation of resident system applications entirely in the Java™ programming language.

4.5.2 Requirements

1. A Java™ application must be able to retrieve in a standard way all the information included in Table 1.

2. A Java™ application must be able to register in a standard way to be notified about changes of all information marked with *Yes* in the **Listener** column in Table 1.

3. A Java™ application must be able to modify in a standard way all information marked with *Yes* in the **Modify** column in Table 1.

Notes:

- The ***Type*** column in Table 1 contains a suggested type for the values of a given element and is included mainly for clarification purposes.

- The mechanism for accessing the attributes and settings needs to be aware of the underlying security practices, i.e., provide access to these attributes and settings only if the application has the necessary permissions to do so.

System Attribute	Description and Use case(s) for 2nd and 3rd party applications	Type	Listener	Modify
Identification				
IMEI	Enables an application to identify the handset: 1. Billing purposes 2. Personalization 3. Statistics 4. QoS measurement	String	No	No

SYSTEM ATTRIBUTE	DESCRIPTION AND USE CASE(S) FOR 2ND AND 3RD PARTY APPLICATIONS	TYPE	LISTENER	MODIFY
IMSI	For SIM-based devices only. Enables an application to identify the subscriber even if he migrates to another handset: 1. Billing purposes 2. Personalization 3. Statistics 4. QoS measurement	String	No	No
ICCID	For SIM-based devices only. For security, some carriers prefer that applications identify subscribers via ICCID instead of IMSI.	String	No	No
Brand	Enables an application to identify the handset brand: 1. Dynamic brand-specific customizations 2. Identification for Error Reporting	String	No	No
Model	Enables an application to identify the handset model: 1. Dynamic model-specific customizations 2. Identification for Error Reporting	String	No	No
OS	Enables an application to identify the handset OS: 1. Statistics (see comment about IMEI) 2. QoS measurements (see comment about IMEI) 2. On-demand download of native code	String	No	No
MOBILE NETWORK				
Cell-ID	Enables an application to obtain the ID of the current cell: 1. Activate diagnostic measurements in case a problem is detected in a cell	String	Yes: Cell-ID Changed	No
RSSI level	Allows the application to know the signal strength from the current cell: 1. Notification to the user 2. Enable diagnostic measurements in case a problem is detected in a cell (see 9)	Int	Yes: RSSI level changed	No
Neighboring cell RSSI	Enables an application to obtain the power level of neighboring cells: 1. Support finer localization mechanism implemented at the application level 2. Enable sophisticated cell relocation processes in case of cell congestion	Array of cell-ID, RSSI level couples	No	No
Available network accounts	Enable an application to retrieve network accounts currently defined in the terminal	Array of Objects containing the parameters below	Yes: account added/removed	Yes: It must be possible to create/delete network accounts
- Account name	The name of the network account	String		

SYSTEM ATTRIBUTE	DESCRIPTION AND USE CASE(S) FOR 2ND AND 3RD PARTY APPLICATIONS	TYPE	LISTENER	MODIFY
- Account APN	The name of the APN for the account	String		
- Bearer type (GPRS, EDGE, UMTS)	Enable an application to obtain the bearer type for a given account	String		
Default network account	Enable an application to obtain the name of the current default network account	String	Yes: default network account changed	Yes: it must be possible to set the default network account
Active PDP context(s)	Enables an application to obtain data about the current active PDP contexts: 1. QoS measurements	Array of Objects containing the parameters below:	Yes: PDP Context opened/closed	Yes: it must be possible to open a PDPContext using a given network account and close an active PDPCOntext
- Account-name	Enables an application to obtain the account name for the selected active PDP context	String		
- APN	Enables an application to obtain the APN name for the selected active PDP context: 1. QoS measurements	String		
- Bearer type (GPRS, EDGE, UMTS..)	Enables an application to obtain the bear type for the selected PDP context: 1. QoS measurements	String/int		
- Negotiated parameters (service class, bandwidth..)	Enables an application to obtain negotiated parameters for the selected PDP context: 1. Application can adjust behaviour to accommodate negotiated parameters.	Properties/ Enumeration		
- TX and Rx bytes	Enables an application to obtain the number of bytes transmitted and received for the selected PDP context: 1. Data for progress bars and/or meters	1. TX bytes (int) 2. Rx bytes (int)		
APPLICATIONS				
Installed applications	Enables an application to obtain the list of all applications currently installed in the device. (BOTH native and Java, both preinstalled or downloaded).	Array of objects containing the parameters below	Yes: appl. installed/ uninstalled /updated	Yes: it must be possible to install/ uninstall/ update at least Java applications
- Name		String		

SYSTEM ATTRIBUTE	DESCRIPTION AND USE CASE(S) FOR 2ND AND 3RD PARTY APPLICATIONS	TYPE	LISTENER	MODIFY
- Version		String		
- Vendor		String		
Running applications	Enables an application to obtain the list of all applications currently running in the device	Array of objects containing the parameters below	Yes: appl. started/ stopped	Yes: it must be possible to start/stop an instance of a given application
- Instance ID	Unique identifier of the running instance (in certain cases there may be two or more instances of the same application running at the same time on the same device)	String		
- Appl name	The name of the application that is running	String		
Foreground application	The running application that is currently in foreground	An object containing the parameters below	Yes: foreground application changed	Yes: it must be possible to bring a running application in foreground
- Instance ID	Unique identifier of the instance	String		
- Appl name	The name of the application that is running	String		
LOCAL NETWORK/ CONNECTIVITY				
WiFi on/off/unavailable	Enables an application to discover if WiFi functionality is available and if it is on or off: 1. Allows application to adjust behaviour based on WiFi Availability	int Fields: Unavailable Off On	Yes: WiFi state changed	No
Bluetooth on/off/unavailable	Enables an application to discover if Bluetooth functionality is available and if it is on or off: 1. Allows application to adjust behaviour based on Bluetooth Availability	int Fields: Unavailable Off On	Yes: Bluetooth state changed	No
IrDA on/off/unavailable	Enables an application to discover if IrDA functionality is available and if it is on or off: 1. Allows application to adjust behaviour based on IrDA Availability	int Fields: Unavailable Off On	Yes: IrDA state changed	No
Data Cable plugged/ unplugged/ unavailable	Enables an application to discover if a local cable connection is available and if it is on or off: 1. Allows application to adjust behaviour based on local cable connection Availability	int Fields: Disconnected Connected Unavailable	Yes: Cable state changed	No

SYSTEM ATTRIBUTE	DESCRIPTION AND USE CASE(S) FOR 2ND AND 3RD PARTY APPLICATIONS	TYPE	LISTENER	MODIFY
TELEPHONY				
Number of Missed Calls	Enables an application to obtain the number of missed calls: 1. Allows an application to notify the user how many calls they have missed.	int	Yes: Number of missed calls changed	No
Number of unread messages	Enables an application to obtain the number of unread messages: 1. Allows an application to notify the user how many new messages they have.	int	Yes: Number of unread messages changed	No
MMI				
Battery level	Enables the application to inquire the current battery level of the device.	int	No	No
Main Display on/off	Enables an application to know if the main display is switched on or off 1. Allows application to adjust behaviour (i.e. stop drawing to the screen)? 2. Enables the application to request the display is switched on?	int Fields: Off On	Yes: Main display state changed	No
Sub Display on/off/unavailable	Enables an application to know if the sub display is switched on or off 1. Allows application to adjust behaviour (i.e. stop drawing to the screen)? 2. Enables the application to request the display is switched on?	int Fields: Off On	Yes: Sub-display state changed	No
Display focus	Enables an application to know which display is currently in focus: 1. Application can optimize drawing accordingly	int Fields Main Sub	Yes: Display focus changed	No
Keylock on/off/unavailable	Enables an application to know if the keypad is disabled 1. Behaviour change - (e.g. Demo mode v interactive mode) 1. Notification to the user.	int Fields: Off On	Yes: Keylock status changed	No

Table 1. System information

4.5.3 ACTIONS TAKEN

Liaison statements sent to

- JSR 249 (MSA)
- JSR 232 (Mobile Operational Management).
- JSR 253 (Mobile Telephony API)

- JSR 256 (Sensor API)

4.5.4 RESULTS AND DECISIONS

The responses received from the JSR 232 and JSR 253 efforts are below:

(From Jon Bostrom, Nokia) JSR 232 seems to have the mechanisms in place that you are requesting in terms of information retrieval, notification and modification. These include

- A flexible and extensible event system. This event system provides the capability to define new event types in a predictable format. JSR 232 is currently working to define a core set of events and will review your requests in that area to account and then notify you. JSR 232 will also publish, as part of the spec, the format for the events which will allow other parties to use that definition to create new event types that meet their specific needs. OMTP will therefore be able to encourage other bodies to use the same format to define events that will eventually not be included in the JSR 232 core set.
- A generic API for accessing the DMT (Device Management Tree) and a set of predefined management objects for operational management some of which can be used to retrieve and modify system information. Concerning this point however the relation between the JSR 232 and the JSR 246 (Device Management API for CLDC), that is focusing on similar aspects for CLDC devices, still has to be clarified.

(From ekaterina.chtcherbina@siemens.com) JSR 253 will not provide mechanisms to retrieve the number of missed calls and unread messages. However, although the primary goal of the JSR 253 is to handle calls and supplementary services, a very basic support of getting network information is also provided. Therefore, some of features mentioned in OMTP requirements for Mobile Network are also covered. Thus, for example, application can require which bearers are currently available (e.g. UMTS, GSM, CDMA or any other which could be related to telephony). If bearer becomes unavailable then application will receive a notification if an appropriate listener is registered. The same way, if quality of coverage changes (e.g. full coverage, limited coverage, out of coverage), then application will get a notification. Application can query the default bearer for telephony, e.g. if no specific bearer is chosen what would be the default one. Application can also receive the name of servicing network (provider name) as a string and notification if servicing network has been changed. However, application cannot change any network settings using JSR 253.

4.6 DIGITAL RIGHTS MANAGEMENT

4.6.1 PROBLEM

Digital Rights Management (DRM) refers to the mechanisms used on a computing device to protect digital content from unauthorized use. For instance, DRM technology can be used to control unauthorized copying of digital music or video content. There a number of mechanisms that have been developed and are in use today by various service providers and electronics manufacturers. Despite the growing use of this technology, there has not yet been defined a Java™ programming interface that allows a Java™ application to in any way control protected content.

4.6.2 REQUIREMENTS

The requirements for DRM have not been discussed formally in this workstream so this section should be considered as an outline for a more formal discussion on this topic.

A Java™ programming interface for DRM should include support for the basic functions of DRM, querying and changing the policy of content, querying and changing the state of content, and perhaps other functions. Any API should be DRM standard agnostic such that the API could be successfully implemented on any of the commonly used DRM schemes in used today.

4.6.3 ACTIONS TAKEN

This document has been edited to include this new section.

4.6.4 RESULTS

This topic should be discussed within the workgroup to determine if there is enough interest to include it more formally in this document.

5 NEXT STEPS

Based on the responses received to the liaison statements, it seems that the gap areas identified by the OMTP Java™ Workstream will not all be covered sufficiently by the forthcoming Java™ platform standards and that there is a need to initiate new JCP efforts or other standardization activities at least in the area of System Information and Notification. The information exchange with the ongoing standardization activities will still continue in order to ensure that all the details mentioned in the previous chapter will be covered adequately.

The list below summarizes the possible action items for the remaining OMTP Java™ Workstream activities:

- Compose a revised liaison statement to JSR 253 in order to ensure complete feature coverage in the area of system information.
- Compose a new liaison statement to JSR 246 (Device Management) in order to ensure the effective exchange of information between JSR 246 and OMTP Java™ Workstream.
- Compose a new liaison statement to JSR 278 (Resource Management) in order to ensure that this new JSR will take the OMTP requirements into account during its work.

6 APPENDIX 1: UI TECHNOLOGIES - EVALUATION CRITERIA & RESULTS

6.1 OVERVIEW

This chapter has been prepared to identify the criteria to effectively evaluate two possible user interface technologies for the advanced Java™ ME platform: JSR 209 and eSWT. The results presented here are based on the UI Evaluation Task Force initiated by the OMTP Java™ Workstream.

Note: The UI Technologies evaluation included in this revision of the OMTP Definition and Requirements document includes analyses of JSR-209 and eSWT. It has been noted that other Java™ ME UI toolkit specifications may allow implementation on the CDC specification. Further analysis of these technologies should be considered when their specifications become publicly available. The following JSR's were mentioned in face to face meetings of the OMTP Java™ Workstream JSR-258 (Mobile User Interface Customization API) and JSR-271 (Mobile Information Device Profile 3) although this list should not be considered exclusive of other technologies. The OMTP Java™ Workstream will determine which technologies they find relevant.

6.2 EVALUATION BASED ON OMTP UI CUSTOMIZATION USE CASES

The OMTP Board has approved a collection of use cases related to the customization of mobile terminals by an operator. They relate to some aspects of the evaluation of UI Technologies. The following use cases are included in this document for informative purposes.

USE CASE	DEFINITION
4A: ADD BASIC BRANDING ELEMENTS	• Brand *basic* elements of a terminal UI in a consistent manner across all terminals, OTA updates possible • E.g., background, menu items, wallpapers, colour scheme, start-up/shutdown sequences, ring tones, logo, screensaver, etc.
4B: SIMPLE LOOK AND FEEL CUSTOMISATION	• Customise simple look and feel aspects of a terminal UI in a consistent manner across all terminals, OTA updates possible • E.g., splash screens, sounds, status indicators, animations, soft key area, etc. plus level 4A elements
4C: DEEP LOOK AND FEEL CUSTOMISATION	• Define the look and feel of all UI components and their layout in a consistent manner across all terminals and all applications, OTA updates possible • E.g., scroll bar, text entry boxes, buttons, list boxes, notifications, etc. plus level 4B

4D: Basic Menu Customisation	• Customise the ordering and labelling of the device menu structure and define device shortcuts and soft-keys, OTA updates possible • E.g., emphasise and prioritise terminal features, lock specific menu items, define idle screen shortcuts, etc
4E: Simple Application Integration	• Define the structure of the menu of a terminal UI across all terminals, including the addition of links to operator applications and services. OTA possible • E.g., add an offline operator menu structure, access operator application from idle screen, plus 4D
4F: Application Interworking Customisation	• Customise the structure of Application Interworking Workflows across all terminals to define a seamless integration. OTA possible • E.g. define application interworking workflows (AIW), plus 4E

The following table represents an evaluation of JSR-209 and eSWT based on the use case definitions described above. This evaluation is based on the features actually specified in the specifications listed not in any of the underlying technologies, which may be used to support their implementations.

Use Case	JSR-209	eSWT
4A: Add Basic Branding Elements	No direct support, could be supported by an implementation of Pluggable Look and Feel APIs.	eSWT relies on the native toolkit for appearance of the visual elements of a terminal's interface. As such, there is no functionality included in the eSWT specification that addresses this use case.
4B: Simple Look and Feel Customisation	JSR-209 is technology that could be used to implement the entire UI of the terminal but as an enabling technology does not specifically address this use case.	eSWT is technology that could be used to implement the entire UI of the terminal but as an enabling technology does not specifically address this use case.
4C: Deep Look and Feel Customisation	Support via the Pluggable Look and Feel APIs.	This use case is specifically NOT addressed by eSWT.
4D: Basic Menu Customisation	JSR-209 provides the facilities to manipulate and customize menus as described in the use case, however these features and how they relate to the operation of the terminal are out of scope for JSR-209.	eSWT provides the facilities to manipulate and customize menus as described in the use case, however these features and how they relate to the operation of the terminal are out of scope for eSWT.
4E: Simple Application Integration	These features are out of scope for any user interface toolkit although terminal software could be implemented with JSR-209 to support this use case.	These features are out of scope for any user interface toolkit although terminal software could be implemented with eSWT to support this use case.

4F: APPLICATION INTERWORKING CUSTOMISATION	These features are out of scope for any user interface toolkit although terminal software could be implemented with JSR-209 to support this use case.	These features are out of scope for any user interface toolkit although terminal software could be implemented with eSWT to support this use case.

Notes: Some of the features described in requirements 4a, 4b, 4d, 4e, and 4f may be supported in JSR-258 however no publicly available specification was available to evaluate it. JSR-258 specifically mentions support for JSR-209 but it may also be applicable to eSWT.

6.3 *EVALUATION CRITERIA*

The table below summarizes the evaluation criteria that were developed by the OMTP Java™ Workstream in order to compare Java™ UI toolkits in an objective fashion.

CATEGORY	CRITERION	DESCRIPTION	JUSTIFICATION
INTRINSIC CAPABILITY	Functionality	This criterion covers the richness and completeness of functionality presented to the application programmer and UI writer. Completeness is assessed in the context of the projected typical UI requirements of target devices within the OMTP target timescale.	One of the reasons for requiring a new UI/graphics platform API to supplement/supersede lcdui is the requirement for greater functionality.
	Flexibility	This criterion covers the intrinsic flexibility of the framework and API, both from the point of view of the application programmer and UI writer. Flexibility is taken to mean the ability of the framework to accommodate a wide range of differing requirements without requiring extension on the framework or undue upheaval	One of the reasons for requiring a new UI/graphics platform API to supplement/supersede lcdui is the requirement for greater flexibility.
ENGINEERING COMPATIBILITY	Footprint requirements	The footprint requirements of the solution, taking into account both code size and runtime memory requirements (native and Java if necessary).	In order to be deployable to as many devices as possible, smaller footprint solutions are preferred.
	Processing requirements	The CPU processing requirements of the solution. This is assessed in the context of the projected typical UI requirements of target devices within the OMTP target timescale.	In order to be deployable to as many devices as possible, solutions that are less processor-intensive are preferred.

CATEGORY	CRITERION	DESCRIPTION	JUSTIFICATION
	Implementation feasibility/cost	The implementation expense and time taken of creating and validating a solution, taking into account the creation of the libraries themselves and any necessary underlying extensions to the native platform or UI.	In order to be practicably deployable to as many devices as possible, solutions that are deployable with sooner and with lower engineering cost are preferred.
	Availability of implementations for different platforms	Are implementations of the solution available currently or in line with the OMTP Roadmap? For how many platforms relevant for mobile terminals (and others) these implementations have been implemented on / ported to?	Availability of implementations on different platforms relevant for mobile terminals (and others) gives assurance that the solution is feasible and implementable on those platforms with reasonable effort.
	Availability of Compatibility/ Conformance Tests	Do implementations include a regimented and complete set of conformance tests based on a framework familiar to platform developers.	Compatibility testing ensures that implementations from different vendors interoperate. The suite of tests used to test a platform should be comprehensive (adequately test the breadth of the API) and integrate well with other tests required by the platform.
RELEVANCE TO OMTP OBJECTIVES	Specification fragmentation	The degree to which the technology/standard fragments the device API/profile landscape (or risks future fragmentation). Specific relevant sub-criteria are: compatibility with CLDC implementations compatibility with lcdui implementations ability for the API to scale across multiple device tiers compatibility with other OMTP groups As well as “API fragmentation” (where different devices support different APIs) the framework should be assessed in terms of “variant fragmentation” (where multiple implementations of an application are required for different devices or device variants, even when those variants ostensibly implement the same API and profiles).	A key aim of the OMTP is to enhance the consistency of user experience and enhance application programmability by reducing fragmentation in platform APIs and profiles and device variants.
	Specification Maturity	The degree to which the underlying specification has been available in the market and allowed developer feedback to improve it’s general quality.	There is a risk associated with the introduction of any new technology. As much as possible we should rely on technologies that have allowed application developers a significant amount of “seat time” to identify areas for improvement in the specification.

CATEGORY	CRITERION	DESCRIPTION	JUSTIFICATION
	Application portability	The degree to which client and/or application software can be written that can successfully migrate from one implementation of the standard to another. This is strongly determined by the completeness and level of ambiguity in the specification.	The standard must allow for independent implementations. A specification proven to support multiple independent implementations is preferred.
	Form factor portability	This is the degree to which the framework supports the creation of applications that can be portable across multiple disparate form factors and input systems. This is determined by a wide range of issues, including: versatility and generality of navigation and traversal model to support different input systems; availability of appropriately high level UI abstractions in addition to only components/widgets and low level interaction system suitability of available UI components to phone input systems, especially without pointer device flexibility of the UI framework to allow developers to easily create portable custom components	The standard must preserve the application writer's ability to target multiple form factors and input systems. Ideally the form factor portability should be superior to that of lcdui. Poor form factor portability will contribute to fragmentation.
	Support for modification of LAF	This is the degree to which the framework supports the modification of the look and feel of a platform UI, and the ability for applications to be portable across disparate LAF implementations/behaviours	A key requirement of OMTP is to maximise the uniformity of technology implementations whilst permitting operator-specific customisations to be deployed. It is important that those customisations do not contribute to variant fragmentation.
	Suitability for mixed terminal s/w architectures – Consistency of the LAF across the whole device	This is the suitability of the framework to coexist with other UI technology elements on devices that support multiple content execution environments (e.g. native as well as Java). In the event that there are multiple environments, frameworks should be assessed according to the degree to which consistency of LAF is achieved across those environments.	OMTP's desire is to maximise the consistency of user experience and it is believed that this should be achievable on platforms that support mixed terminal s/w architectures

CATEGORY	CRITERION	DESCRIPTION	JUSTIFICATION
	Suitability for mixed terminal s/w architectures – Customisation of the LAF across the whole device	This is the suitability of the framework to coexist with other UI technology elements on devices that support multiple content execution environments (e.g. native as well as Java). How does the solution allow the customisation of the whole device LAF in a consistent way?	OMTP's desire is to maximise the consistency of user experience and it is believed that this should be achievable on platforms that support mixed terminal s/w architectures. Also the customisation technology must allow customising the whole device in a consistent way.
	Portability of LAF	This is the ease with which LAF implementations or customisations are migrated from one implementation of the framework to another.	OMTP's aim is to reduce the cost of achieving operator-specific customisations, and this is reduced if the same customisations can be redeployed across implementations from multiple manufacturers.
	Consistency of LAF across devices ABC	This is the consistency of LAF implementations or customisations across different devices	OMTP's aim is to achieve consistent implementations of an operator LAF across devices, including devices from different manufacturers
	Familiarity to developer community	This is the degree to which the framework is already familiar to the developer community. Some of the criteria to be considered are: Size of the developer community Consistency with the Java programming paradigms Number of commercial applications Availability of tools	A framework is preferred if it is more readily accessible to the developer community.
PROCESS	Standardisation process	This is the extent to which the standardisation process for the framework is compatible with OMTP objectives	OMTP must be satisfied that the process for definition, maintenance and enforcement of the standard is workable.
	Maturity/ timescale	This is the extent to which the remaining standardisation work is compatible with OMTP timescales.	OMTP must be satisfied that the standard can be adopted and achieve its objectives within the required timescales.
	Licensing	What kind of licensing terms these implementations have?	OMTP should ensure that the technology is available on predictable and reasonable licensing terms.

6.4 EVALUATION RESULTS

The table below summarizes the actual JSR 209 vs. eSWT evaluation results based on the criteria presented in the section above.

CATEGORY	CRITERION	JSR 209	ESWT
INTRINSIC CAPABILITY	Functionality	**List of JSR 209 UI Components:** JButton JCheckBox JCheckBoxMenuItem JColorChooser JComboBox JComponent JDesktopPane JDialog JEditorPane JFileChooser JFormattedTextField JFormattedTextField.AbstractFormatter JFormattedTextField.AbstractFormatterFactory JFrame JInternalFrame JInternalFrame.JDesktopIcon JLabel JLayeredPane JList JMenu JMenuBar JMenuItem JOptionPane JPanel JPasswordField JPopupMenu JProgressBar JRadioButton JRadioButtonMenuItem JRootPane JScrollBar JScrollPane JSlider JSpinner JSpinner.DateEditor JSpinner.DefaultEditor JSpinner.ListEditor JSpinner.NumberEditor JTabbedPane JTable JTextArea JTextField JTextPane JToggleButton JToggleButton.ToggleButtonModel JToolTip JViewport JWindow	**List of eSWT UI Components:** Button Canvas ColorDialog Combo Composite Control Dialog DirectoryDialog FileDialog FontDialog Item Label List Menu MenuItem MessageBox ProgressBar Scrollable ScrollBar Shell Slider Table TableColumn TableItem Text Tree TreeItem

CATEGORY	CRITERION	JSR 209	ESWT
	Functionality	**2D Rendering APIs:** Paint PaintContext Shape Stroke GradientPaint RenderingHints RenderingHints.Key TexturePaint AffineTransform Arc2D Arc2D.Float Area CubicCurve2D CubicCurve2D.Float Dimension2D Ellipse2D Ellipse2D.Float FlatteningPathIterator GeneralPath Line2D Line2D.Float Point2D Point2D.Float QuadCurve2D Rectangle2D RectangularShape RoundRectangle2D BufferedImageOp RasterOp RenderedImage AffineTransformOp BufferedImage ColorModel ConvolveOp DataBuffer DataBufferInt Kernel LookupOp LookupTable PackedColorModel Raster RescaleOp SampleModel SinglePixelPackedSampleModel WritableRaster IIOParamController Classes IIOParam ImageIO ImageWriter ImageReader ImageReadParam ImageTypeSpecifier ImageWriteParam	**2D Rendering API:** Color Device Font FontData FontMetrics GC Image ImageData ImageLoader PaletteData Point Rectangle Resource RGB

CATEGORY	CRITERION	JSR 209	ESWT
	Flexibility	JSR 209 and the Swing and Java 2D frameworks that they are based on have been designed from the beginning to provide a very high degree of platform independence and advanced features for developers. In particular the following aspects of the specification should be noted: The Swing specification provides a framework allowing platform independent implementations of a UI toolkit The Swing specification (and JSR 209) allows a high level of platform customization in the form of pluggable look and feel infrastructure. This will allow both the behaviour and appearance of the UI to be customized on a device (allowing a consistent look and feel across devices from different manufacturers as required). The pluggable look and feel framework can also be used to integrate with a native toolkit to provide a native look and feel (using native rendering libraries) to provide fidelity with native applications (as required) The Java 2D rendering API allows Java devices to take advantage of the most advanced rendering capabilities available on devices today.	The goal of the SWT project was to provide a UI toolkit for the Eclipse IDE that was tightly coupled with the native toolkit. This philosophy allows the implementer of the toolkit to very tightly bind an SWT implementation to a native toolkit including support for platform specific features. (http://www.eclipse.org/rcp/) Only the customization provided by the native toolkit is supported by SWT. An implementation can only include the look and feel provided natively by the device.

CATEGORY	CRITERION	JSR 209	ESWT
ENGINEERING COMPATIBILITY	Footprint requirements	Varies based on whether Swing implementation is built in the Java programming language or a native toolkit. Varies based on whether Swing implementation is built in the Java programming language or a native toolkit. Native toolkit implementations can be the same or smaller than an equivalent SWT implementation. Java Swing implementation would include functionality included in eJFace (and not part of core eSWT).	Varies depending on the native toolkit, target size for the binding layer is approximately 150k Most of eSWT covered by existing native code Additional widgets may be implemented by native or Java. Target size is 300-500k (Taken from v 0.9.2 of eSWT HLA): · Core eSWT– 300k · Expanded eSWT – 200k · Mobile Extensions – 200k Note that these numbers do not include eJFace which would be required for functionality parity with JSR 209.
	Processing requirements	Optimized for mobile phones.	Optimized for mobile phones.
	Availability of implementations for different platforms	Not yet announced.	MS Windows, MS Windows Mobile, and Nokia Series 80.
	Availability of Compatibility/ Conformance Tests	Conformance tests will be based on the J2SE conformance tests. These tests have been developed and released on multiple releases of J2SE and adapted to JSR 209. They will be released when JSR 209 is released.	Compatibility tests will be available from the Eclipse project. They have been developed specifically for eSWT and will be released for the first time with eSWT.

CATEGORY	CRITERION	JSR 209	ESWT
RELEVANCE TO OMTP OBJECTIVES	Specification fragmentation	JSR 209 is intended to leverage the millions of developers and tools currently targeting J2SE and J2EE. The Swing and Java 2D APIs which JSR 209 is based on represent the largest segment of the Java Developer Community. Providing an API familiar to these developers will provide operators with increased availability of compelling content for the mobile platform. JSR 209 is targeted at mid-Tier devices and is intended to provide advanced graphics and UI support to developers. There is no intention in allowing direct compatibility with LCDUI implementation, however integration of JSR 209 and LCDUI implementations has been demonstrated. The JSR 209 can only be supported on CDC platforms.	ESWT is based on the SWT APIs which have had some limited notoriety in the industry. Its design is focused around providing a Java binding to a native toolkit. ESWT is only supported on CDC platforms. (Note: From eSWT specification overview, “eSWT provides a core UI API that Java applications can rely on and can be used in any application models or architectures that have support for full Java VM specification.” (CLDC is not a full Java VM)
	Specification Maturity	The JSR 209 specification is based on the J2SE Swing API’s which have been developed and refined since 1997. They have been developed to insulate the developer from the details of the underlying operating environment and allow the production of portable applications.	SWT was first introduced in 2001 as part of the Eclipse IDE project.

CATEGORY	CRITERION	JSR 209	ESWT
	Application portability	The Swing platform was developed to provide a high degree of application portability. For instance, focus management is consistent across implementations of Swing on different platforms.	Although application portability is important to the SWT platform, the authors of SWT make clear that developers can not depend on the platform to provide a dependable level of cross platform consistency and this is noted as part of the design of the SWT framework. “developers need to understand that applications can potentially behave differently to match the operating system behaviour” (from http://dev.eclipse.org/viewcvs/index.cgi/platform-swt-home/main.html?rev=1.11) The SWT design allows implementations to compensate for different behaviour in the underlying toolkit. For instance, some toolkits may automatically set the focus of a UI component on a particular platform and generate an event to indicate this to the application. *Nokia does not agree with this wording, however it represents content from the Eclipse project website.*
	Support for modification of LAF	The JSR 209 EG has reconsidered including support for PLAF APIs as per a request from the OMTP Java WS. A first revision of the PLAF APIs appeared in the Public Review version of the JSR 209 specification recently approved by the Java Community Process.	ESWT depends on the underlying UI toolkit to define the look and feel of the UI.
	Suitability for mixed terminal s/w architectures – Consistency of the LAF across the whole device	The look and feel of the JSR 209 toolkit is subject to the implementation of the look an feel.	ESWT only allows for the native look and feel to be supported.
	Suitability for mixed terminal s/w architectures – Customisation of the LAF across the whole device	As supported by a look and feel implementation.	Only as supported by the native toolkit.

CATEGORY	CRITERION	JSR 209	ESWT
	Portability of LAF	A Look and Feel can implemented in the Java Programming language which would allow a consistent look and feel across devices from different manufacturers. The Swing architecture allows for a LaF to be defined separately from the toolkit implementation. Although not currently part of the public JSR 209 API, a look and feel could be constructed that represented a particular brand or company and then be deployed on any appropriately implemented JSR 209 platform regardless of the underlying operating system or UI toolkit. The "Metal" look and feel which is included in J2SE is a toolkit independent look and feel deployed with all J2SE platforms (Windows, Mac, Linux, Solaris, etc). In summary: 1. The UI can be independent of the native widget-set, therefore the Java LaF may be different form the native LaF on the terminal (this is an implementation detail). 2. The UI will be consistent across all Java applications, but not necessarily between native and Java applications (this is an implementation detail). 3. JSR 209 Public Review Draft includes support for pluggable look and feel.	If custom widgets are used, they will provide the same LaF regardless of the device being used (as long as each device uses the same native toolkit). eSWT APIs will be tightly specified to ensure consistent behaviour across different platforms implementations (however the SWT architecture allows for platform differences to be exposed to applications). Native side and all Java UI toolkits in a technology agnostic way through JSR 258. Results into operator LaF on the whole device and into the possibility to combine native and Java applications in an optimised way. JSR 258 when completed may provide some level of UI skinning of a native toolkit.

CATEGORY	CRITERION	JSR 209	ESWT
	Consistency of LAF across devices	JSR 209 is based on the "Swing" user interface toolkit. The Swing toolkit was designed to allow the implementation of a User Interface toolkit entirely in the Java Programming Language. JSR 209 has tightened the Swing portion of the specification in a way that allows JSR 209 to be implemented both on a native toolkit or in the Java Programming Language. In the case where JSR 209 is implemented in the Java Programming Language, the implementation of the toolkit Look and Feel is entirely defined by the implementation of the specification and not the underlying toolkit.(Java UI implementation) If JSR 209 is implemented with a native look and feel, the native toolkit provides the look and feel capabilities. Therefore independence from the underlying widgets can only be guaranteed by JSR 209 when a Java Look and Feel is used.	SWT's architecture provides a thin layer around the native toolkit and retains only the look of the native toolkit. From eclipse.org "The Standard Widget Toolkit (SWT) is a widget toolkit for Java developers that provides a portable API and tight integration with the underlying native OS GUI platform." In other words, the underlying native toolkit will provides the Look and Feel capabilities so they can not be independent. If standard widgets are used, eSWT applications will inherit the LaF that is appropriate for the device, thereby providing a user experience that blends with native applications. However, an eSWT may choose to create custom widgets that provide a specific LaF independent of the underlying toolkit. eSWT can be implemented with pure-Java in the same fashion as Swing. So if a eSWT implementation is done so that LAF is wanted to be different than th native implementation then it's of course possible and eSWT can have whatever LAF needed. Often, however, this is not a desirable implementation as it gives user non-unified user experience of a device with the rest of the device UIs.
	Familiarity to developer community	The Swing UI toolkit from which JSR 209 is based has been used to develop hundreds of commercial applications many of which are listed here: http://java.sun.com/products/jfc/tsc/sightings/index.html	Few commercial applications have been developed for SWT.
PROCESS	Standardisation process	The wireless industry standard process for the development of Java programming interfaces, the Java Community Process. The current JSR 209 specification effort includes software platform providers, OEMs, and operators.	The open source community through the Eclipse project.

CATEGORY	CRITERION	JSR 209	ESWT
	Maturity/ timescale	Swing was first released in 1997 as part of JDK 1.2 and refined in JDK 1.3, and JDK 1.4. JSR 209 was initially approved by the Java Community Process in March 2003. Development continues with input from the Java Community and will complete before the end of the year.	SWT was introduced as an open source project in 2001 and the eSWT project began sometime in 2004.
	Licensing	Information on licensing can be found on the JCP website: http://www.jcp.org/en/jsr/detail?id=209	Eclipse Projects are typically licensed under the “Common Public License”

Note: As stated in the *Overview* of this section, the OMTP Java™ workstream agreed only to consider technologies described in publicly available specifications. When discussed in the last face to face meeting, the JSR 258 specification was not yet publicly available. The planned features of JSR 258 may possibly enhance the ability to customize mobile terminals, and as stated in the JSR text will be compatible with JSR 209 and potentially eSWT as well.

7 ABBREVIATIONS

ABBREVIATION	DESCRIPTION
3GPP	3rd Generation Partnership Project
OMTP	Open Mobile Terminal Platform
OTA	Over The Air
WAP	Wireless Application Protocol
JCP	Java™ Community Process
JSR	Java™ Specification Request
CDC	Connected Device Configuration
CLDC	Connected Limited Device Configuration
MIDP	Mobile Information Device Profile
JAVA™ ME	Java™ Micro Edition
JAVA™ SE	Java™ Standard Edition
MSA	Mobile Service Architecture

8 REFERENCED DOCUMENTS

NO.	DOCUMENT	AUTHOR	DATE
1	RFC2119 (http://rfc.net/rfc2119.html)	RFC	March 1997