Wireless Internet Platform for Interoperability 2.0.1

Part 1. Structure and Function of WIPI 2.0.1

September 2004

Korea Wireless Internet Standardization Forum

1. Introduction

1.1. Purpose of WIPI

This document seeks to define the Wireless Internet Platform for Interoperability (hereinafter referred to as "Platform for Interoperability") mounted on a mobile communication terminal, providing an environment that is used to execute an application program. The mobile platform (hereinafter referred to as "platform") satisfying this protocol ensures the interoperability of contents between platforms for the developer of application programs for the terminal, provides the developer of terminals with a platform porting facility, and offers diverse and rich contents to the end user.

1.2. Scope of WIPI

As a standard platform protocol, this document describes the following items:

○ Purpose, scope, requirements, and terms of this document

○ Conceptual structure of the platform as well as interface of terminals with other modules operating in a wireless environment (to be defined later)

○ HAL (Handset Adaptation Layer) Specification, an abstract layer used to support hardware independence in platform porting

○ Application Programming Interface (hereinafter referred to as "API") (WIPI supports Java and C languages)

○ Specifications for the major functions of the platform

1.3. Definitions of Terms

○ Wireless Platform

This is the execution environment for terminals that can execute application programs written based on the Wireless Internet Platform for Interoperability. It should include management functions for application programs and API.

○ Clet

This is an application program written in C language based on the Wireless Internet Platform for Interoperability. It should follow the life cycle of application programs.

○ Jlet

This is an application program written in Java language based on the Wireless Internet Platform for Interoperability. It should follow the life cycle of MSP (Mobile Standard Profile) application programs.

○ Basic Software for the Terminal

This is the basic software mounted on the platform. HAL functions as a link between the basic software of the terminal on the lower part and the platform.

○ Task

This is an independent implementation unit defined in the basic software of the terminal. Each task is independently executed using a separate program stack.

○ Executing Multiple Application Programs

This refers to a concurrent execution of Clet and Jlet on the Wireless Internet Platform, with each program maintaining independent memory space.

○ AOTC (Ahead-of-Time Compiler)

This is a compiler that converts the byte code generated through Java compilation into a machine code that is executable on the platform.

○ HAL (Handset Adaptation Layer)

This is an API layer used to abstract a handset for implementation by terminal manufacturers.

○ MSF (Mobile Standard Foundation)

This is a foundational API for mobile devices supporting I/O functions, network, security, and internationalization.

○ MSP (Mobile Standard Profile)

This is a profile for MSF-based mobile devices.

○ CLDC (Connected Limited Device Configuration)

This is a configuration for virtual machine-based mobile devices. CLDC consists of a virtual machine and several core APIs and supports I/O functions, network, security, and globalization.

○ MIDP (Mobile Information Device Profile)

This is a profile for CLDC-based terminals.

○ MIDlet

This is an application program written in Java language based on MIDP specifications. It follows the life cycle of MIDP application programs.

○ Expression

The terms used in this document should be clearly defined in Korean.

- Required: Use the term "should" (e.g., should do, should be, etc.).
- Optional: Use the term "may" or "can."
- Recommended: Use the term "recommend" or "may."

1.4. Minimum Recommended Specifications for the Terminal

The following specifications are recommended for terminals whose mounted platforms are defined in WIPI 2.0.1.

○ Display (screen size): 96 x 54 or larger

○ Color: Four or more gray tone colors or at least 256 natural colors

○ I/O device

○ Input device: Keypad

○ Sound equipment: Vibration and beep sound

○ Network: Transmission via wireless and serial

○ Non-volatile memory

○ At least 1 MB non-volatile memory that can be used by the platform library

○ At least 400 KB non-volatile memory that can be used by the application program manager or basic application program

○ At least 500 KB file system space that can be used by the application program

○ Volatile memory

○ At least 300 KB heap area that can be used by the application program

○ At least 20 KB area for the platform library

2. Conceptual Structure

Figure 2-1 shows the conceptual structure of the Wireless Internet Platform defined in WIPI 2.0.1. The basic software for the terminal includes basic communication functions as well as various device drives.

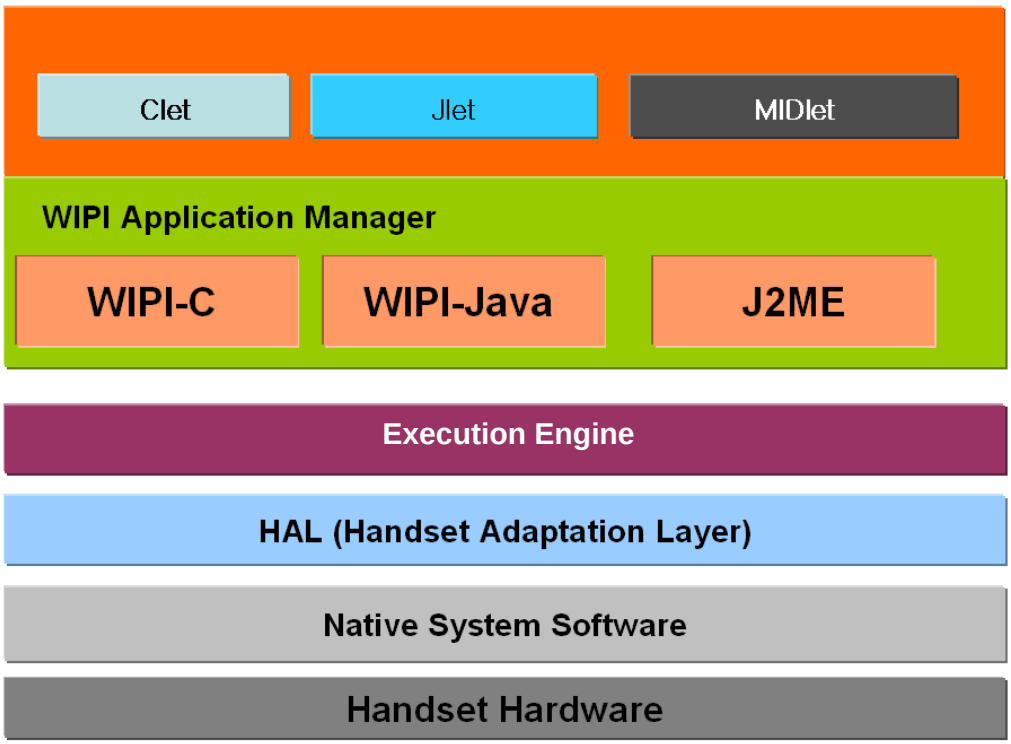

Figure 2-1. Conceptual structure.

2.1. HAL

HAL is a layer used to support hardware independence in platform porting. Through this layer, abstraction for the terminal is performed. The platform is also configured independently from the hardware.

2.2. Required API

This is a collection of required APIs supported by the platform that is used by the developer of application programs. C and Java APIs are supported.

3. Specifications for Major Functions

This chapter defines the specifications for the major functions of the Wireless Internet Platform.

3.1. Specifications for the Machine Code of the Application Program

The platform should download and execute application programs having a machine code type. For a Java application program, the machine code can be created from Java byte code through AOTC.

3.2. Executing Multiple Application Programs

The platform should support an environment where several application programs can be loaded to the memory for concurrent execution. It should also be able to execute several application programs concurrently, assign execution priority to each application program, and execute the application program with the highest execution priority every moment.

In case of several application programs, each application program should be executed independently. To support communication among independent application programs, a method of transmitting shared memory and events should be provided.

In addition, the platform should be able to manage the life cycle of an application program written in a programming language supported by the platform.

3.3. Supported Programming Languages

To enable an application program developer to write application programs using the required APIs of the platform in Java or C language, the platform should support such languages.

3.3.1. Support for C Language

The platform should support APIs for C language as defined in the basic API and the context of ANSI C language.

3.3.2. Support for Java Language

The platform should support APIs for Java language as defined in the required API and the context of JAVA Programming Language (2nd edition).

3.4. Platform Security

3.4.1. Definition of Security

Platform security refers to measures in terms of both policy and related technology against the actions of an application program when it affects the operation of other entities (other application programs, platforms where these application programs operate, terminals where such platforms are mounted, or network/server system with which those terminals are interlinked) through platform resources (APIs, storage space, shared memory, etc.) or accesses information owned by other entities. The platform should be able to control the access of these entities to platform resources in accordance with the security specifications of this section.

3.4.2. Security Level

Basically, platform security techniques should be based on the access right to the platform through the security level defined for each resource and information on the security level of each application program. For the type or number of security level for the platform, more than one type or number should be defined, with the content primarily dealing with implementation. For the interoperability of the pre-WIPI version and MIDP 2.0.1 specifications, however, support for the following basic security levels is recommended:

- **PUBLIC**

Among the security levels existing on the platform, Public denotes the least reliable level or one with the most restrictions. When an application program of this level is executed, the platform should not allow access since it may cause security problems such as those affecting the terminal system or allowing access to personal information.

- **CP**

At this level, some or all restrictions placed on resources at the Public level are released because the platform trusts the application program either fully or to some extent.

- **SYSTEM**

At this level, which is a subset of the CP security level, access to all resources is automatically allowed because the platform regards the application program as completely trustworthy.

Table 3-4-1. Security relationship between WIPI and MIDP.

WIPI 2.0.1	MIDP
Public	Not trusted
CP	Trusted
System	Trusted

3.4.3. Target Resources for the Security Policy

The security level includes at least the following groups having a resource type that should be controllable by group:

Storage: Private directory/Application-shared directory/System-shared directory

Network: Connection-oriented, Datagram, HTTP

Secured network connection: HTTPS

Serial device: RS-232C, USB

Personal information: List of addresses, photos, and other personal information

3.4.4. Access Right to Resources by Security Level

There should be a table that defines the access right to the resources of the platform by security level. Likewise, the security level should be specified according to the application program. The table showing the access right to resources by security level can be located either in the terminal platform or in the server hosting the application program. Table 3-4-2 shows an example of such access rights. Since the classification of resources and that of security level in Table 3-4-2 are only examples, there is no need to follow them strictly.

Table 3-4-2. Access right to resources by security level.

	Graphics	**Sound**	**Personal Info**	**System Property**
Public	Allowed	Allowed	Not allowed	Not allowed
CP	Allowed	Allowed	Requires user permission	Not allowed
System	Allowed	Allowed	Allowed	Allowed

When the application program tries to access certain resources, the platform should be able to exercise the following controls based on the security level of the application program and access right by security level based on Table 3-4-2:

- **Allowed**

  Access to resources shall be allowed automatically without intervention by the user.

- **Requires User Permission (User)**

  When the application program tries to access a resource specified as "Requires User Permission," the platform should notify the user accurately and respond in accordance with the user's choice. The corresponding action on the user's choice should be pre-determined.

- **Not Allowed(Deny)**

Access to resources shall be denied automatically without intervention by the user.

- **Requires User Permission**

  When the application program tries to access a resource specified as "Requires User Permission," the platform should notify the user accurately and enable the user to choose one of the following options. When the platform offers a user interface that can query and change the access right to resources by application program, the user can change the access right using the user interface regardless of the operation status of the application program. The changed items should be available whenever the application program accesses the corresponding resource during the valid period by condition.

- **Always Not Alloweds(Deny)**

  Access shall be denied until the application program is uninstalled.

- **Not Allowed Until the Application Program is Terminated**

  Access shall be denied until the application program is terminated.

- **Not Allowed (Deny) at This Time**

  Access shall be denied at the present function call. Next time, however, the application program shall contact the user again.

- **Allowed at This Time**

  Only the present function call will be allowed. Next time, however, the application program shall contact the user again.

- **Allowed Until the Application Program is Terminated**

  Access shall be allowed until the application program is terminated.

- **Always Allowed**

  Access shall be allowed until the application program is uninstalled. When the platform offers a user interface that can query and change the security level for resources by application program, the user may change the policy for resources, i.e., always allowed.

3.4.5. Regulation on Security Implementation

Operations on security should be implemented based on Table 3-4-2 (Access right to resources by security level) and security level set by the application program. The details of the regulation are as follows:

○ The platform should control access to resources based on security information defined by the application program and security access condition by platform resource.

○ The security information of the application program should be supplied together with the application program.

○ The security information of the application program can be certified using certification technology.

○ In case access to resources by the application program is not allowed, the application program should be notified accordingly. For Java, security exception should be triggered, whereas an error message should be returned in the case of C.

3.5. Support for Adding/Overriding API

The platform can add/override APIs over the wireless network, and Add/Override should be carried out using DLLs.

3.5.1. DLL (Dynamic Linking Library)

DLL is a method by which the platform adds new APIs and overrides the existing APIs.

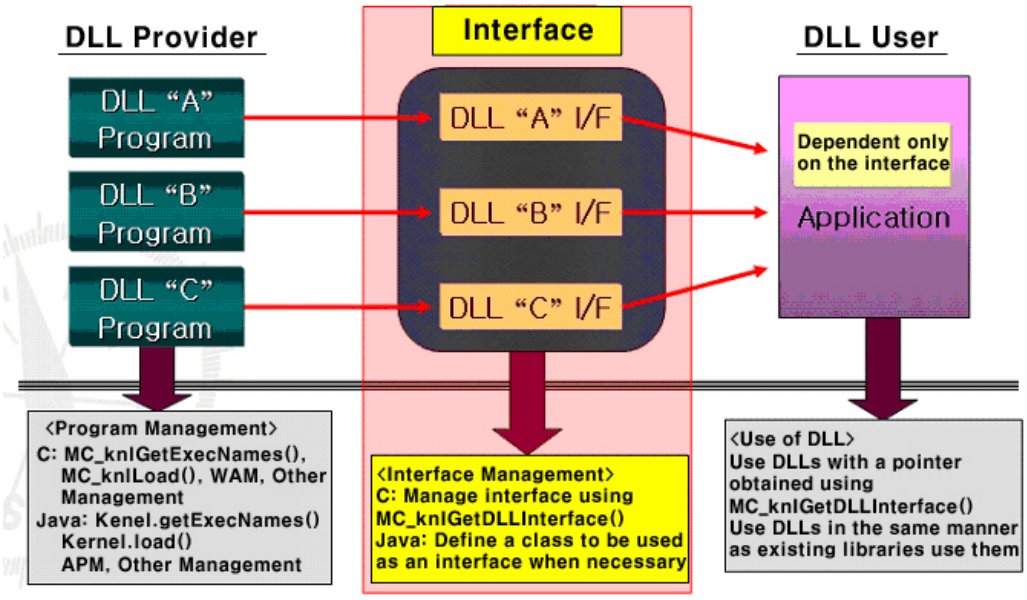

Figure 3-5-1. DLL service and use scenario.

DLL can be divided into implementation and interface. To manage the outcome of the DLL implementation, the platform should possess version management and Install/Delete functions required in the addition and overriding of APIs. The DLL user (application program developer) can use the specific functions (libraries) of DLL through the interface.

The platform should equally apply the API security level policy to added or overridden APIs.

DLL/Interface can either be mounted as a basic embedded item in the terminal or downloaded according to the specification of an application manager or the user.

3.6. Managing Memory

The platform should manage the heap memory used by the application program as follows:

3.6.1. Automatic Release of Memory

When an application program is terminated, all memories related to the program should be returned to the platform. Therefore, memories used dynamically such as event should all be returned automatically.

3.6.2. Memory Compaction

When assigning memories or releasing assigned memories that are used dynamically in the platform, memory compaction can be carried out to reduce memory fragmentation.

3.6.3. Java Garbage Collection

The platform should support garbage collection within the context of Java language.

3.6.4. Java Stack

The platform should be able to assign stacks or release the assigned stacks by Java application program and change the stack size dynamically by application program. When a Java application program requests for an assignment of stack in excess of the memory limit, the platform should notify the application program of the exception. After the exception occurred, the platform should operate normally.

3.6.5. Support for Shared Memory

The memory used by application programs should be independent from each other, and the platform should support memory that can be shared among application programs. When all application programs sharing the platform are terminated, the platform should automatically release the shared memory.

3.7. Managing the Application Program

The platform should provide the following functions:

3.7.1. Management Function

When executing an application program, startup should be determined based on the setup of limitations on the date and number of executions.

The Install/Uninstall function of the application programs should be provided. Likewise, the suspension function of an application program may be offered. The suspension function deletes only the execution program; thus leaving the corresponding data files intact. As such, existing data can be utilized when the program is installed again. In addition, the platform should enable the Add/Override API function and the forced termination function of an application program.

3.7.2. Downloading the Application Program

The platform should support the downloading of application programs. In case of error during the download, the application program should be restored to its default state. The download function through the serial interface is optional.

3.8. Multi-lingual Support

3.8.1. Support for Unicode

The platform should support Unicode for Java application programs and the input and output strings should be converted into corresponding character codes to suit local characteristics. In the case of Korea, Unicode strings should be converted into EUC_KR character sets and vice versa.

3.8.2. Support for Locale

The platform should identify C application programs as character sets that are referred to and supported based on local information. In the case of Korea, the EUC_KR character set should be used.

3.8.3. Expanded Unicode

The EUC_KR character set includes unique graphic characters that do not match in the Unicode. To support such graphic characters, the expanded Unicode that allows private use of the (0xE000-0xF8FFF) area in the Unicode specification can be utilized.

3.9. Support for CLDC/MIDP

WIPI 2.0.1 has adopted CLDC/MIDP as the required specification. As such, it should comply with the following conditions:

3.9.1. Support for CLDC

The criteria for CLDC specification should be Sun Microsystems' CLDC Specification 1.1 (http://jcp.org/aboutJava/communityprocess/final/jsr139/index.html). Since the platform is based on the execution of binary codes, the platform engine should accommodate virtual machine functions as defined in the CLDC specification. In CLDC specification, however, the byte code and code verification process should be specified in a manner that enables the interpretation of the meaning of the platform including AOTC (the definition of "In-device" should be interpreted to include both the platform and AOTC in Phase 2: In-device verification and 5.2.1.1 Verification process of CLDC Specification.) In case CLDC is supported, "2.1 CLDC Class, Part 4" of the Specification should be replaced to be compatible with CLDC Core API.

3.9.2. Support for MIDP

The criteria for MIDP specification should be Sun Microsystems' Specification 2.0 (http://jcp.org/aboutJava/communityprocess/final/jsr118/index.html).

3.9.3. Interoperability with the Required API

CLDC/MIDP should be used together with the required Java APIs. The mixed use of APIs with identical functions should be prohibited, although APIs supported only on one side can be used complementarily.

4. References