MIDP APIs for Wireless Applications

A Brief Tour for Software Developers

A White Paper

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
1 (800) 786.7638
1.512.434.1511

Please
Recycle

CHAPTER 1

Introduction

Writing wireless applications that run on all mobile phones and two-way pagers has become a lot easier with the recent release of the Java™ 2 Platform, Micro Edition (J2ME™) Mobile Information Device Profile (MIDP).

Applications that run on devices supporting MIDP are called MIDlets. Like applets, MIDlets are controlled by the software that runs them – in this case the cell phone or two-way pager device implementation that supports MIDP and the J2ME Connected Limited Device Configuration (CLDC).

Both CLDC and MIDP have been developed through the Java Community Process℠ program. More than 20 companies, representing handset manufacturers, wireless operators, and software providers, collaborated to create these specifications. CLDC outlines the basic set of libraries and Java™ virtual machine features that must be present in each implementation of a J2ME environment on highly constrained devices, such as cell phones and pagers. To form a complete environment for these devices, MIDP adds supplementary libraries that provide APIs not handled by the low-level CLDC, such as the user interface, database, and device-specific networking.

CHAPTER 2

A Complete Environment

MIDP is the first profile available for the J2ME mobile design center. The combination of CLDC and MIDP provides a complete environment for creating applications on cell phones and two-way pagers.

To provide you with an overview of the development process, this paper describes some of the most useful MIDP APIs and how one developer employed them to create a stock trading demo running on a cell phone.

You can use these same APIs, and many others that are available under the MIDP umbrella, to create a wide variety of applications — ranging from games to consumer applications, including the stock tracking MIDlet described below. Or, you may want to develop enterprise applications for use on devices issued to your company's sales force, field service representatives, or other corporate road warriors.

Despite the constraints associated with the limited memory, input, output, and screen size, the potential for creating ingenious, useful applications on these devices is almost unlimited.

A Word about MIDlets

A MIDlet, as we mentioned earlier, is a MIDP application. The application must extend the MIDlet class to allow the application management software to control the MIDlet, retrieve properties from the application descriptor, and notify and request state changes. All MIDlets extend the MIDlet class – the interface between the runtime environment (the application manager) and the MIDlet application code. The MIDlet class provides APIs for invoking, pausing, restarting, and terminating the MIDlet application.

The application management software can manage the activities of multiple MIDlets within a runtime environment. In addition, the MIDlet can initiate some state changes by itself, and notify the application management software of those changes.

Application Portability

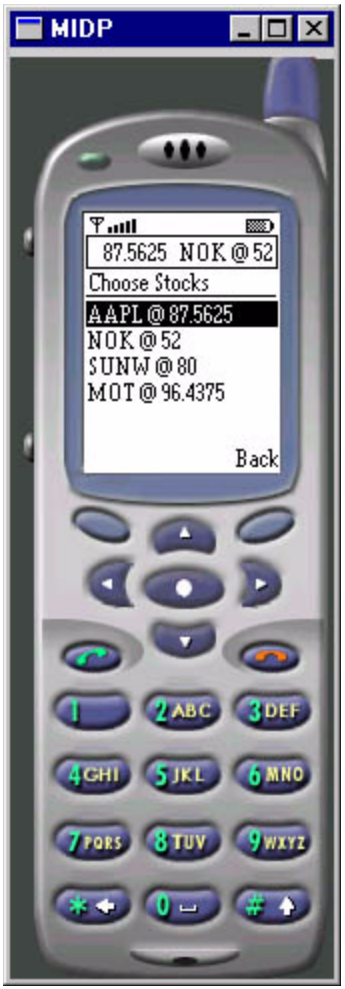
One key feature of high-level MIDP applications is their portability across various cell phones and pagers. The MIDP implementation insulates the application from differences among devices by handling issues such as screen layout and button mapping.

The MIDP APIs are logically composed of high-level and low-level APIs. The APIs are designed for applications or services where the handset functions as the client device. The user gains access to applications and services that run on the handset through a network service provider.

The high-level APIs are designed for applications where software portability across a range of handsets is desired. This is important if you are writing an application or service that a network service provider plans to deploy to a selected set of handsets. To achieve this portability, the APIs use a high level of abstraction. The trade-off is that the high-level APIs limit the amount of control the developer has over the human interface's look and feel. The underlying implementation of the user interface APIs, which is accomplished by the handset manufacturer, is responsible for adapting the human interface to the device's hardware and native user interface style.

CHAPTER 3

Tracking Stocks the MIDP Way



Despite its limitations, the cell phone proves to be an excellent device for consumers who want to keep track of their stock portfolio while on the go. They can keep a list of their stocks, track the stock prices over the Internet, and receive alerts when a stock hits a certain price. There is even a stock ticker API that displays real-time stock updates. The simple ticker scrolls across the top of the cell phone's screen, one stock at a time. For his demo, the developer used an online stock service to provide the stock text string with actual, real-time quotes.

The stock information is stored in the cell phone's database. Just how many stocks the user may include in the database depends on several factors. For example, different devices come with varying quantities of available memory – low-end cell phones provide only 10K of usable memory, while many midrange devices have 100K to 200K available.

Each stock quote consumes only about 20 bytes, so the user could potentially store information about a large number of stocks in the database. However, practicality intervenes – because of the difficulty of inputting stock information using the cell phone's keypad, and limited screen scrolling resources, most users will be content with tracking half a dozen stocks on their cell phones or pagers.

MIDP APIs

The APIs for this application center around two primary functions – the user interface and the record store for the database. Unlike working with the Java™ 2 Platform, Standard Edition (J2SE™), there is not much leeway in choosing APIs because of device limitations and the need to make these applications portable. However, as the developer pointed out, this can be an advantage — it makes the design task much easier. Even complex applications can be created using simple building blocks that can be relied on to work together.

For the user interfaces, the developer employed APIs for Forms, Items, and Commands. The underlying CLDC APIs were used to handle strings, objects, and integers. He used basic Java APIs for numbers, storing text, and vectors – fundamental functions used in every Java technology-based application. MIDP persistent storage classes were utilized to preserve the stock information in the database when the user turned off the phone. As the developer said, it was a very simple process.

Here are some of the primary APIs that were used to make this typical MIDP-based stock trading application.

MIDP APIs for the User Interface

The high-level portion of the user interface API is screen-based. That is, the API is designed so that interaction with the user is based around a succession of screens, each of which presents a reasonable amount of data to the user. Commands are presented to the user on a per-screen basis. They allow the application to determine which screen to display next, what computation to perform, what request to make of a network service, and so on.

Command

The Command object encapsulates the name and information related to the semantics of an action. It is primarily used for presenting a choice of actions to the user. The behavior that the Command activates is not encapsulated in the Command object. This means that it contains only information about “Command,” not the actual action that occurs when Command is activated. The resulting behavior is defined in a CommandListener associated with the screen.

Each Command contains three pieces of information: a label, a type, and a priority. The label is used for the visual representation of the command; the type and priority are used by the system to determine how the Command is mapped into a concrete user interface.

Commands may be implemented in any user interface construct that has semantics for activating a single action — a soft button, an item in a menu, or some other direct user interface construct. For example, a speech interface may present these Commands as voice tags. The way Command objects are presented, or "mapped," in the user interface, depends on the semantic information contained within the Command.

Mapping objects to concrete user interface constructs may also depend on the total number of Commands. For example, if an application asks for more abstract Commands than can be mapped onto the available physical buttons on a device, the device may use an alternate human interface such as a menu. The abstract Commands that cannot be mapped onto physical buttons are placed in a menu, and the label "Menu" is mapped to one of the programmable buttons.

Alert

An alert is a screen that informs the user about an exceptional condition or error. In the stock application, the alert lets the user know that a specified stock had reached a predetermined price.

The alert screen can handle both text and images, although in the stock demo, only text was used. There are two variations of alert screens: timed and modal. The timed variation allows the alert to pause for a certain period of time before proceeding to the next screen on its own. The model variation requires input from the user before it can proceed; the user must initiate a command (for example, press a button) for the screen to go away. If there is not enough space to display all of the timed alerts, and the user is forced to scroll the screen, the timed alerts can be turned into modal alerts.

Choice

Choice defines an API for user interface components that implement a selection from a predefined number of choices.

Each element of a Choice is composed of a text string and an optional image. If the application provides an image, the implementation may choose to ignore the image if it exceeds the device's display capacity. If the implementation displays the image, it is displayed adjacent to the text string and the pair is treated as a unit.

Images within any particular Choice object should all be of the same size, because the implementation is allowed to allocate the same amount of vertical space for every element. If an element is too long to be displayed, the implementation will enable the user to see the whole element. If this is done by wrapping an element to multiple lines, the second and subsequent lines indicate to the user that they are part of the same element, not a new element.

After a Choice object has been created, elements may be inserted, appended, and deleted. In addition, the implementation may get and set each element's text string and image parts.

ChoiceGroup

A ChoiceGroup is a group of selectable elements intended to be placed within a Form. The group may be created with a mode that requires either single or multiple choices to be made. The implementation is responsible for providing the graphical representation of these modes and must provide visually different graphics for various modes. For example, it might use "radio buttons" for the single-choice mode and "checkboxes" for the multiple-choice mode.

In the stock demo, radio buttons were provided to allow the user to choose how often to update the stock information in the cell phone's database – continuously, every 15 minutes, one half hour, three hours, and so on.

Form

When the developer wanted to add more than one item to a screen, he used the Form API.

For example, if the consumer wants to find out how well his initial investment is paying off, he can use the text box on this screen to enter the original number of shares purchased, their price at the time, and their current price. A Command box then causes the application to calculate the payoff for selling those shares and display the results.

A Form is defined as a screen that contains an arbitrary mixture of items: images, read-only text fields, editable text fields, editable date fields, gauges, and choice groups. In general, any subclass of the Item class may be contained within a Form. The implementation handles layout, traversal, and scrolling. None of the components contained in the Form has any internal scrolling; all contents scroll together. This differs from the behavior of other classes, the List for example, where only the interior scrolls.

If the developer had wanted to draw images or animations on the screen, he would have used the MIDlet Canvas object instead of a Form. In this case, he made the application very simple – displaying text only on a Form.

List

Lists are the lifeblood of the stock demo and every other application – this is the API most frequently used by MIDP developers. In the stock demo, there are lists of stocks, lists of alerts – everything visible on the screen is either a List or a Form.

The List class is defined as a screen containing a list of choices. Most of the behavior is common with the ChoiceGroup class, and the common API is defined in the interface Choice.

When a List is present on the display, the user can interact with it indefinitely (for instance, traversing from element to element and possibly scrolling). These traversing and scrolling operations do not cause application-visible events. The system notifies the application when a Command is invoked.

List, like any Choice, utilizes a dedicated “select” or “go” functionality of the devices. Typically, the select functionality is distinct from the soft buttons, but some devices may use soft buttons for the select. In any case, the application does not have the means to set a label for a select key.

StringItem

This simple class extends Item by allowing a string to be put on a form. A StringItem is display only; the user cannot edit the contents. Both the label and the textual content of a StringItem may be modified by the application.

TextBox

The TextBox class is a screen that allows the user to enter and edit text. A TextBox has a maximum size or capacity – the maximum number of characters that can be stored in the object at any time.

This limit is enforced when the TextBox instance is constructed, when the user is editing text within the TextBox, and when the application program calls methods on the TextBox that modify its contents. The maximum size is the maximum stored capacity, and is unrelated to the number of characters that may be displayed at any given time.

The number of characters displayed and their arrangement on the display, is determined by the device. The text contained within a TextBox may be more than can be displayed at one time. If this is the case, the implementation will let the user view and edit any part of the text by scrolling. This scrolling occurs transparently to the application.

TextField

Like the TextBox, a TextField is an item that may be placed within a Form, and has a maximum size determined by the maximum number of objects that can be stored in the object at any time. In a Form, multiple TextFields can be used to input data such as the stock’s original purchase price and the number of shares held by the user.

TextField operates the same as TextBox with regard to the maximum size or capacity. The application may also require the implementation to limit the user's choice of inputs – for example, numeric only. In the stock application, both alpha and numeric symbols were accepted.

Ticker

This class is particularly suited for the developer's application. Ticker implements a ticker string, a piece of text that runs continuously across the top portion of the display. The direction and speed of scrolling are determined by the implementation.

When animated, the ticker tape string scrolls continuously. That is, when the string finishes scrolling off the display, the ticker starts over at the beginning of the string. There is no API provided for starting and stopping the ticker. However, the implementation is allowed to pause the scrolling for power consumption purposes – for example, if the user doesn't interact with the device for a certain period of time. The implementation will resume scrolling the ticker when the user interacts with the device again.

These are the basic classes that the developer used to create the user interface for the wireless stock application. There are a number of other useful APIs among the MIDP MIDlets that may be suited for your particular application. A full version of the MIDP specification can be found on java.sun.com/products/midp/.

MIDP APIs for Handling the Database

To organize and manipulate the cells phone's database, the developer used another major class of APIs. Here's a brief description:

RecordStore

A RecordStore consists of a collection of records, which remain persistent across multiple invocations of the MIDlet.

The platform is responsible for making its best effort to maintain the integrity of the MIDlet's record stores throughout the normal use of the platform, including reboots, battery changes, and so on.

Record stores are created in platform-dependent locations, which are not exposed to the MIDlets. Each MIDlet suite is restricted to its own space for RecordStores. MIDlets within a suite are not permitted access to RecordStores created by MIDlets in another suite. MIDlets within a MIDlet suite are allowed to create multiple RecordStores, as long as they each have a different name. When a MIDlet suite is removed from a platform, all the RecordStores associated with its MIDlets will also be removed.

These APIs allow only the MIDlet suite's own RecordStores to be manipulated, and do not provide any mechanism for record sharing between MIDlets in different MIDlet suites. MIDlets within the same MIDlet suite can access each other's RecordStores directly.

RecordStore names are case sensitive and may consist of any combination of up to 32 Unicode characters. RecordStore names must be unique within the scope of a given MIDlet suite. In other words, MIDlets within a MIDlet suite are not allowed to create more than one RecordStore with the same name. However, each MIDlet in different MIDlet suites is allowed to have a RecordStore with the same name as a MIDlet in another MIDlet suite. In that case, the RecordStores are still distinct and separate.

No locking operations are provided in this API. Record store implementations ensure that all individual RecordStore operations are atomic, synchronous, and serialized, so no corruption will occur with multiple accesses. However, if a MIDlet uses multiple threads to access a RecordStore, it is the MIDlet's responsibility to coordinate this access, or unintended consequences may result (in other words, the program could die). Similarly, if a platform performs transparent synchronization of a RecordStore, it is the platform's responsibility to enforce exclusive access to the RecordStore between the MIDlet and synchronization engine.

RecordComparator

This interface defines a comparator that compares two records in a RecordStore to see if they match or what their relative sort order is. This is utilized to help locate the record that the user wants to update, add, or delete.

RecordFilter

This interface defines a filter that is used to extract sets of records that match a criteria. For example, the RecordFilter can be instructed to find all stocks with a certain price, or stocks that are named “Sun,” in order to create user-specified alerts. The device will tell the user when the stock reaches a certain price point, and filter out the rest of the data regarding it and other stocks in the database.

CHAPTER 4

A New World of Wireless Applications

Mobile service providers are upgrading their networks to support value-added data services in addition to the voice services available today. By enabling data communications over the mobile phone network, services such as stock trading, instant messaging, and e-commerce will be available to consumers wherever they are.

With the creation of the Mobile Information Device Profile and its many APIs, developers now have access to simple but powerful tools.They enable the creation of new applications that will make wireless a way of life for more and more people as the Internet economy blossoms.

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303

1 (800) 786.7638
1.512.434.1511

http://java.sun.com/products/midp/

February 2001