CONTENIDO

DESCRIPCIÓN

La especificación del MIDP 2.0 ha determinado la existencia de ocho tipos de paquetes, incluyendo los paquetes del núcleo, mientras que MIDP 1.0 define la existencia de cuatro tipos de paquetes, incluyendo también los paquetes del núcleo. El presente documento presenta una comparación entre estos dos perfiles, llegando a cada método y atributo que cambió de un perfil a otro. En la siguiente tabla se muestran los paquetes incluidos en cada uno de los perfiles indicando el tipo de paquete al que hacen referencia.

Tipo Del Paquete	Paquete MID2.0	Paquetes MID1.0
Interfaz de Usuario	javax.microedition.lcdui	javax.microedition.lcdui
Juegos	javax.microedition.lcdui.game	-----
Ciclo de vida	javax.microedition.midlet	javax.microedition.midlet
Interconexión a Redes	javax.microedition.io	javax.microedition.io
Seguridad	javax.microedition.pki	-----
Sonido	javax.microedition.media	-----
	javax.microedition.media.control	-----
Persistencia de datos	javax.microedition.rms	javax.microedition.rms
Básicos o del Núcleo	Java.io	Java.io
	java.lang	java.lang
	java.util	java.util

Paquetes y tipos de paquetes en MIDP 1.0 y MIDP2.0

1. Paquetes para el manejo de Interfaces de usuario

Tanto MIDP1.0 como MIDP2.0 definen el mismo paquete para la creación de interfaces de usuario(ver Tabla 2.2) en el dispositivo móvil, no obstante, MIDP 2.0 incluye soporte para interfaces de usuario mejoradas que permiten a los desarrolladores la creación de aplicaciones más atractivas, a través de nuevas clases e interfaces o la adición de métodos y atributos a las clases existentes, a saber:

a. Nuevas Interfaces

ItemCommandListener: la cual es utilizada para recibir notificaciones de comandos que han sido invocados a través de un objeto ***Item***, esto significa que con las mismas dos teclas de comandos que proveen los dispositivos móviles, ahora se pueden manejar múltiples comandos en la misma pantalla, dependiendo del Item que se esté seleccionando.

b. Nuevas Clases

CustomItem: es una clase que sirve para crear items personalizados a través de la herencia y sirve para introducir nuevos componentes visuales e interactivos, que no pertenecen al estándar, dentro de un ***Form.***

Spacer**:** un Item en blanco, no interactivo que tiene un tamaño mínimo fijable. El ancho mínimo es útil para ubicar espacios variables entre Items en una misma fila de un ***Form.*** La altura mínima es útil para forzar la altura mínima de una fila en particular.

c. Modificaciones y Adiciones a las clases existentes

- ***Clase Alert***

Nuevos Atributos

static Command DISSMIS_COMMAND: un comando entregado a un oyente para indicar que la alerta ha sido lanzada. Este comando está presente implícitamente cuando no hay otros comandos presentes.

Nuevos Métodos

public Gauge getIndicator(): obtiene el indicador de actividad de un objeto *Alert.*

public void RemoveCommand(Command cmd): el cual remueve un comando que ha sido adicionado, sin embargo, cuando una aplicación remueve el último comando de un *Alert, DISSMIS_COMMAND* es adicionado implícitamente.

public void setIndicator(Gauge indicator): un método que fija un indicador de progreso de actividad en un *Alert***.** El indicador de actividad es un objeto *Gauge* que debe cumplir con ciertas restricciones: no debe ser interactivo, no debe ser usado por otro contenedor(*Alert* o *Form*), no debe tener ningún *Command* o ningún *ItemCommandListener*, su etiqueta debe ser null, su ancho y altura deben estar ambos habilitados y el valor del *layout* debe ser *LAYOUT_DEFAULT.*

- **Clase Canvas**

Nuevos Métdos

setFullScreenMode(boolean mode): controla si el *Canvas* se encuentra en modo de pantalla completa o en modo normal.

sizeChanged(int w, int h): es un método que se llama cuando el área de dibujo del canvas ha sido cambiada, por ejemplo al cambiar de un modo normal a un modo full screen.

- **Clase ChoiceGroup**

Nuevos Métodos

public void deleteAll(): borra todos los elementos de un objeto ChoiseGroup.

public int getFitpolicy(): devuelve el modo en que se muestran las entradas de la lista en la pantalla activa. El valor de retorno es uno de los siguientes: *Choice.TEXT_WRAP_DEFAULT*, *Choice.TEXT_WRAP_On* y *Choice.TEXT_WRAP_OFF*, estás son constantes que han sido adicionadas a la interface *Choice.*

public Font getFont(int elementNum): devuelve la fuente del elemento elementNum (índice del elemento empezando en cero).

public void setFitPolicy(int fitPolicy): fija el modo en que se muestran las entradas de la lista en la pantalla activa. Este modo aplica a todos los elementos del objeto *Choice*. Los valores válidos para fitPolicy son: *Choice.TEXT_WRAP_DEFAULT*, *Choice.TEXT_WRAP_ON* y *Choice.TEXT_WRAP_OFF*, estás son constantes que han sido adicionadas a la interface *Choice.*

public void setFont(int elementNum, Font font): fija la fuente preferida de la aplicación para ser aplicada a un elemento específico de un Objeto *Choice*. El parámetro font puede ser un objeto Font válido o puede ser null, en el segundo caso, es usado el valor por defecto para ser aplicado a un elemento.

- **Clase Command**

Nuevos Métodos

Public command (String shortLabel, String longLabel, int commandType, int priority): es un constructor que crea un nuevo comando con las etiquetas, tipo y prioridad dadas. El parámetro shortLabel es obligatorio y debe ser diferente de null, el parámetro longLabel es opcional y puede ser null.

public String getLongLabel(): obtiene el *longLabe*l de un Objeto Command.

- **Clase Display**

Nuevos Atributos

***static int ALERT*:** tipo de imagen para la imagen de un elemento Alert. Su valor es 3.

static int CHOICE_GROUP_ELEMENT: tipo de imagen para la imagen de un elemento ChoiceGroup, tiene un valor de 2.

static int LIST_ELEMENT: tipo de imagen para la imagen de un elemento List, tiene un valor de 1.

static int COLOR_BACKGROUND: un especificador de color para ser usado con getColor, especifica el color de forndo de la pantalla. El color de fondo siempre estará en contraste con el color de primer plano. Su valor es 0.

static int COLOR_FOREGROUND: un especificador de color para ser usado con *getColor*, especifica el color de primer plano para caracteres de texto y gráficos simples en la pantalla. Texto estático y editable por el usuario podría aparecer con el color de primer plano. Su valor es de 1.

static int COLOR_BORDER: un especificador de color para ser usado con getColor. Identifica el color para los bordes de un objeto cuando este ha sido dibujado de manera no resaltada. El color de borde esta pensado para ser utilizado con el color de fondo y estará en contaste con este último, tiene un valor de 4.

static int COLOR_HIGHLIGHTED_BORDER: un especificador de color para ser usado con getColor. Identifica el color para los bordes de un objeto cuando este es dibujado en estado resaltado, tiene un valor de 5.

static int COLOR_HIGHLIGHTED_BACKGROUND: un especificador de color para ser usado con getColor, identifica el color del foco cuando este es mostrado como un rectángulo relleno. Siempre estará en contraste con el color de resaltado de primer plano. Tiene un valor de 2.

***static int COLOR_HIGHLIGHTED_FOREGROUND*:** un especificador de color para ser usado con getColor. Identifica el color de caracteres de texto y gráficos simples cuando

estos están resaltados. El color de resaltado de primer plano estará siempre en cpntraste con el color de resaltado del fondo, tiene un valor de 3.

Nuevos Métodos

public boolean flashBacklight(int duration): produce un efecto de flash en la pantalla durante la duración especificada en milisegundos, retorna true si el efecto puede ser controlado por la aplicación.

public int getBestImageHeight(int imageType): retorna la altura más adecuada para un tipo de imagen dado(*LIST_ELEMENT*, *CHOICE_GROUP_ELEMENT* o *ALERT*).

public int getBestImageWidth(int imageType): retorna el valor del ancho mas adecuado para un tipo de imagen dado(*LIST_ELEMENT*, *CHOICE_GROUP_ELEMENT* o *ALERT*).

public int getBorderStyle(boolean highlighted): devuelve el estilo del borde usado en la pantalla desplegada.

public int getColor(int colorSpecifier): devuelve uno de los colores del esquema de color de interfaces de usuario de alto nivel, de la forma 0x00RRGGBB basado en el especificador de color pasado como argumento.
public int numAlphaLevels(): devuelve el número de niveles de transparencia alfa soportados por esta implementación. El valor mínimo legal de retorno es 2, que indica soporte para transparencia completa u opacidad completa pero no una mezcla. Valores mayores que 2 indican soporte para las mezclas de opacidad y transparencia.

public void setCurrentItem(Item item): desplaza la pantalla hasta que sea posible ver y seleccionar el Item que se ha pasado como argumento.

public boolean vibrate(int duration): hace una petición a la operación de vibración del dispositivo por el número de milisegundos especificados. El valor de retorno indica si el vibrador puede ser controlado por la aplicación.

- **Clase Displayable**

Nuevos Métodos

public int getHeight(): obtiene la altura en pixels del área de despliegue disponible para la aplicación.

public Ticker getTicker(): obtiene el objeto *Ticker* usado por el *displayable* o null si no se ha fijado.

public String getTitle(): obtiene el título del objeto *Displayable* o null sino hay título.

public int getWidth(): obtiene el ancho en pixels del área de despliegue disponible para la aplicación.

public void setTicker(Ticker ticker): fija un objeto *Ticker* para ser usado en el *Displayable* y reemplaza el que existía previamente(si es que lo hay). Si se pasa null como parámetro se remueve el ticker actual,

public void setTitle(String s): fija el titulo del *Displayable*. Si se pasa null como parámetro se remueve el título actual.

protected void sizeChanged(int w, int h): este método es llamado por el software de gestión de aplicaciones(AMS – Application Managenement Software) cuando el área disponible para un objeto *Displayable* ha sido cambiada, como resultado de la adición o eliminación de componentes en la pantalla desplegada. El "área disponible" es el área de la pantalla que puede ser ocupada por el contenido de las aplicaciones, tales como *Items* en un *Form* o gráficos dentro de un Canvas, esto no incluye el espacio ocupado por un título, un ticker, etiquetas de comando, barras de desplazamiento, etc.

- **Clase Font**

Nuevos Atributos

public static final int FONT_INPUT_TEXT: un especificador de fuente usado por la implementación para dibujar texto entrado por un usuario. Tiene un valor de 1.

public static final int FONT_STATIC_TEXT: especificador de fuente por defecto usado para dibujar contenidos en objetos *Item* y *Screen*. Tiene un valor de 0.

Nuevos Métodos

public static Font getFont(int fontSpecifier): obtiene la fuente usada por la interfaz de usuario de alto nivel para el especificador de fuente pasado como parámetro(*FONT_INPUT_TEXT*, o *FONT_STATIC_TEXT*).

- **Clase Form**

Nuevos Métodos

public void deleteAll(): remueve todos los ítems del objeto *Form* que lo invoque. Este método no hace nada si el form ya esta vacío.

public int getHeight(): retorna la altura en pixels del área de despliegue disponible para ítems, este valor es el alto del *Form* que puede ser usado sin barras de desplazamiento. El valor depende de cómo el dispositivo utiliza la pantalla y podría verse afectado por la presencia o ausencia de un ticker, título o comandos.

public int getWidth(): retorna el ancho en pixels del área de despliegue disponible para items. El valor depende de cómo el dispositivo utiliza la pantalla y podría verse afectado por la presencia o ausencia de un ticker, titulo o comandos. Los ítems del *Form* se presentan para caber dentro de este ancho.

- **Clase Gauge**

Nuevos Atributos

public static final int CONTINUOUS_IDLE: representa el estado de marcha continua lenta de un objeto *Gauge* no interactivo con rango indefinido. En este estado el gauge muestra un gráfico indicando que no hay tareas en progreso. CONTINUOUS_IDLE tiene un valor de 0.

public static final int CONTINUOUS_RUNNING: representa el estado de marcha continua de un objeto *Gauge* no interactivo con rango indefinido. En este estado el gauge muestra una secuencia de animación de actualización continua que indica que hay tareas en progreso. Este valor tiene un significado especial solo para gauges no interactivos con un rango indefinido y es tratado como un valor ordinario para gauges interactivos y para gauges no interactivos con rangos definidos. CONTINUOUS_RUNNING tiene el valor de 2.

public static final int INCREMENTAL_IDLE: este valor representa el estado incremental lento de un Gauge no interactivo con rango indefinido. En el estado incremental lento el gauge muesta un gráfico indicando que no hay tareas en progreso. El valor de INCREMENTAL_IDLE es 1. Este valor tiene un significado especial solo para gauges no interactivos con un rango indefinido y es tratado como un valor ordinario para gauges interactivos y para gauges no interactivos con rango definido.

public static final int INCREMENTAL_UPDATING: este valor representa el estado de actualización incremental de un *Gauge* no interactivo con rango indefinido. En el estado de actualización incremental el gauge muesta un gráfico indicando que hay tareas en progreso, típicamente un marco de una secuencia animada. El gráfico debería ser actualizado en el siguiente marco de la secuencia solo cuando la aplicación llama a setValue(INCREMENTAL_UPDATING). Este valor tiene un significado especial solo para gauges no interactivos con un rango indefinido y es tratado como un valor ordinario para gauges interactivos y para gauges no interactivos con rango definido. El valor de INCREMENTAL_UPDATING es 3.

public static final int INDEFINITE: un valor especial usado para el valor máximo, con el fin de indicar que el *Gauge* tiene rango indefinido. Este valor puede ser usado como el parámetro maxValue en el constructor, el parámetro pasado a getMaxValue(), y como el valor de retorno de getMaxValue(). El valor de INDEFINITE es –1.

- **Clase Graphics**

Nuevos Métodos

public void copyArea(int x_src, int y_src, int width, int height, int x_dest, int y_dest, int anchor): copia el contenido de un área rectangular (x_src, y_src, width, height) a un área de destino. El efecto podría ser tal que el área de destino contenga una copia exacta del contenido del área fuente. Este evento debe ocurrir incluso si las áreas de fuente y destino se traslapan.

public void drawRegion(Image src, int x_src, int y_src, int width, int height, int transform, int x_dest,int y_dest,int anchor):copia una región de una imagen especifica a una ubicación de destino, permite aplicar efectos de transformación como rotación y reflejado.

public void drawRGB(int[] rgbData, int offset, int scanlength, int x, int y, int width, int height, boolean processAlpha): Renderiza una serie de valores de transparencias RGB (Red – Green - Blue) independientes del dispositivo.

public void fillTriangle(int x1,int y1,int x2,int y2,int x3,int y3)*:* rellena el triángulo especificado con el color actual, las líneas que conectan cada par de puntos son incluidas en el triangulo relleno.

public int getDisplayColor(int color): obtiene el color que será desplegado si un color específico es requerido. Este método permite al desarrollador revisar la manera en la

cual los valores RGB son mapeados al conjunto de colores distintos que el dispositivo puede desplegar realmente.

- **Clase Image**

Nuevos Métodos

public static Image createImage(Image image,int x,int y, int with, int height,int transform): crea una imagen inmutable usando como datos los pixels de una determinada imagen fuente y transformándola de la manera especificada. Esta transformación es la definida en la clase *Sprite*:

Sprite.TRANS_NONE: hace que la región especificada de una imagen sea copiada sin cambios.

Sprite.TRANS_ROT90: produce una rotación en la imagen copiada de 90 grados en sentido de las manecillas del reloj.

Sprite.TRANS_ROT180: produce una rotación de 180 grados en sentido de las manecillas del reloj a la región de la imagen especificada.

Sprite.TRANS_ROT270: produce una rotación de 270 grados en sentido de las manecillas del reloj a la región de la imagen especificada.

public static Image createImage (java.io.InputStream stream) throws java.io.IOException: crea una imagen inmutable a través de la decodificación de datos de imagen obtenidos desde un *InputStream*, este método se bloquea hasta que la imagen ha sido leída y decodificada

public static Image createRGBImage (int[] rgb, int width, int height, boolean processAlpha): crea una imagen inmutable a partir de una secuencia de valores ARGB, especificados como 0xAARRGGBB.

public void getRGB(int[] rgbData, int offset, int scanlength, int x, int y, int width,int height): obtiene datos de pixels ARGB de la región especificada de una imagen y los almacena en el arreglo de enteros proveído. El valor de cada píxel es

almacenado en formato 0xAARRGGBB, donde el byte de mas alto orden contiene el canal alfa y los demás bytes contienen los componentes de color para rojo, verde y azul respectivamente.

- **Clase ImageItem**

Nuevos Atributos

Los nuevos atributos que posee esta clase en el perfil MID2.0 son heredados directamente de la clase *Item*.

Nuevos Métodos

public ImageItem(String label, Image image, int layout, String altText, int appearanceMode): es un nuevo constructor que crea un nuevo objeto *ImageItem* con la etiqueta, imagen, directiva de layout, apariencia y texto alternativo dados. El parámetro appearanceMode está definido en la clase Item y puede ser: Item. PLAIN Item.HYPERLINK, o Item.BUTTON

public int getAppearanceMode(): devuelve el modo de apariencia del *ImageItem*.

- **Clase Item**

Nuevos atributos

public static final int BUTTON: un valor de modo de apariencia, que indica que el *Item* aparecerá como un botón. Un valor de 2 es asignado a BUTTON.

***public static final int HYPERLINK*:** un valor de modo de apariencia, que indica que el *Item* aparecerá como un hipervínculo. Un valor de 1 es asignado a HIPERLINK.

public static final int LAYOUT_2: una directiva de disposición en pantalla que indica que las nuevas reglas de disposición de MIDP2.0 producen efecto en este *Item*. Un valor de 0x4000 es asignado a LAYOUT_2.

***public static final int LAYOUT_BOTTOM*:** una directiva de disposición que indica que este *Item* puede tener un alineamiento inferior. Un valor de 0x20 es asignado a LAYOUT_BOTTOM.

public static final int LAYOUT_CENTER: una directiva de disposición que indica que este *Item* puede tener un alineamiento centrado horizontalmente. Un valor de 3 es asignado a LAYOUT_CENTER.

public static final int LAYOUT_DEFAULT: una directiva de disposición que indica que este *Item* debe seguir por defecto, las políticas de disposición de su contenedor. Un valor de 0 es asignado a LAYOUT_DEFAULT.

public static final int LAYOUT_EXPAND: una directiva de disposición que indica que el ancho de este *Item* puede ser incrementado hasta ocupar el espacio disponible. Un valor de 0x800 es asignado a LAYOUT_EXPAND.

public static final int LAYOUT_LEFT: una directiva de disposición que indica que este *Item* puede tener un alineamiento a la izquierda. Un valor de 1 es asignado a LAYOUT_LEFT.

public static final int LAYOUT_NEWLINE_AFTER: una directiva de disposición que indica que este *Item* es el último de su fila y de haber un nuevo *Item* en el contenedor, debería ser puesto en una nueva fila. Un valor de 0x200 es asignado a LAYOUT_NEWLINE_AFTER.

public static final int LAYOUT_NEWLINE_BEFORE: una directiva de disposición que indica que este *Item* debería ser ubicado al principio de una nueva línea o fila. Un valor de 0x100 es asignado a LAYOUT_NEWLINE_BEFORE.

public static final int LAYOUT_RIGHT: una directiva de disposición que indica que este *Item* debería tener un alineamiento a la derecha. Un valor de 2 es asignado a LAYOUT_RIGHT.

public static final int LAYOUT_SHRINK: una directiva de disposición que indica que el ancho de este *Item* puede ser reducido a su mínimo valor. Un valor de 0x400 es asignado a LAYOUT_SHRINK.

public static final int LAYOUT_TOP: una directiva de disposición que indica que este *Item* debería estar alineado en la parte superior. Un valor de 0x10 es asignado a LAYOUT_TOP.

public static final int LAYOUT_VCENTER: una directiva de disposición que indica que este *Item* debe estar centrado verticalmente. Un valor de 0x30 es asignado a LAYOUT_VCENTER.

public static final int LAYOUT_VEXPAND: una directiva de disposición que indica que el alto de este *Item* puede ser incrementado hasta ocupar el espacio disponible. Un valor de 0x2000 es asignado a LAYOUT_VEXPAND.

public static final int LAYOUT_VSHRINK: una directiva de disposición que indica que la altura de este *Item* puede ser reducida a su mínimo valor. Un valor de 0x1000 es aignado a LAYOUT_VSHRINK.

public static final int PLAIN: un valor de modo de apariencia que indica que el *Item* tendrá una apariencia normal. Un valor de 0 es asignado a PLAIN.

Nuevos Métodos

public void addCommand(Command cmd): adiciona al *Item* un comando sensitivo al contexto. La implementación presentará el comando solo cuando el *Item* esté activo, esto es, seleccionado en pantalla. Es ilegal llamar a este método si el *Item* está contenido dentro de un *Alert*.

public int getLayout(): obtiene las directivas de disposición utilizadas para ubicar el *Item*.

public int getMinimumHeight(): obtiene la mínima altura de este *Item*, esta es la altura a la cual el *Item* puede funcionar y desplegar su contenido, aunque no de manera óptima.

public int getMinimumWidth():obtiene el ancho mínimo de este *Item*, este es un ancho con el cual el *Item* puede funcionar y desplegar su contenido, aunque no de manera óptima.

public int getPreferredHeight(): obtiene la altura ideal para este *Item*. Si la aplicación tiene fijada la altura a un valor específico, este método devuelve este valor. En otro caso, el valor devuelto es computado con base en el contenido del *Item*.

public int getPreferredWidth():obtiene el ancho ideal para este *Item*. Si la aplicación tiene fijado el ancho a un valor específico, este método devuelve este valor. En otro caso, el valor devuelto es computado con base en el contenido del *Item*.

public void notifyStateChanged(): hace que el *Form* que contiene al *Item* notifique al itemStateListener del *Item*. La aplicación llama a este método para informar al oyente del *Item* que su estado ha sido cambiado en respuesta a una acción.

public void removeCommand(Command cmd): remueve un comando de un *Item*. Si el comando no está en el *Item*, este método no tiene efecto.

public void setDefaultCommand(Command cmd): fija el comando por defecto para este *Item*. Si el *Item* tenía previamente un comando por defecto, este ya no será el comando por defecto pero seguirá asociado al *Item*. Es ilegal llamar a este método si el *Item* se encuentra dentro de un *Alert*.

public void setItemCommandListener(ItemCommandListener l): fija el oyente de commandos para este *Item*, reemplaza cualquier otro oyente de comandos previo. Si se pasa null como parámetro, se removerá cualquier oyente existente. Es ilegal llamar a este método si el *Item* esta dentro de un *Alert*.

public void setLayout(int layout): fija las directivas de disposición para este *Item*.

public void setPreferredSize(int width,int height): fija el ancho y el alto preferido para este *Item*. Valores para el ancho y alto menores que -1 son ilegales. Es ilegal llamar a este método si el *Item* está dentro de un *Alert*.

- **Clase List**

Nuevos Métodos

public void deleteAll(): borra todos los elementos de esta lista.

public int getFitPolicy(): devuelve el modo en que se muestran las entradas de la lista en la pantalla activa. El valor de retorno es uno de los siguientes: Choice.TEXT_WRAP_DEFAULT, Choice.TEXT_WRAP_ON y Choice.TEXT_WRAP_OFF, estás son constantes que han sido adicionadas a la interface Choice.

public Font getFont(int elementNum): obtiene la fuente fijada por la aplicación para renderizar el elemento específico de este *Choice*. El valor devuelto es la fuente que ha sido fijada por la aplicación

public void removeCommand(Command cmd): remueve un comando que haya sido adicionado a esta lista.

public void setFitPolicy(int fitPolicy): fija el modo en que se muestran los elementos de la lista en el espacio de pantalla disponible, este método tiene efecto sobre todos los elementos de la lista. Valores válidos de parámetros son: Choice.TEXT_WRAP_DEFAULT,Choice_TEXT_WRAP_ON,Choice.TEXT_WRAP_OFF.

public void setFont(int elementNum, Font font): fija la fuente en la aplicación para renderizar un elemento específico de la lista. El primer parámetro puede estar en el rango[0...size()-1], inclusive.

public void setSelectCommand(Command command): fija el comando a ser usado en la acción de selección de una lista implícita. Por defecto, una selección implícita de una lista resultará en un List.SELECT_COMMAND para ser usado. Este método no tiene efecto si la lista es no implícita.

- **Clase StringItem**

public StringItem(String label,String text,int appearanceMode): crea un nuevo objeto *StringItem* con la etiqueta, contenido textual y modo de apariencia dados(Item.PLAIN, Item.HYPERLINK, or Item.BUTTON).

public int getAppearanceMode(): obtiene el modo de apariencia del *StringItem*, uno de los siguientes valores: Item.PLAIN,Item.HYPERLINK.Item.BUTTON.

public Font getFont(): obtiene la fuente usada por la aplicación para renderizar este *StringItem*. El valor retornado es la fuente que ha sido fijada por la aplicación.

public void setFont(Font font): fija la fuente que será usada por la aplicación para renderizar este *StringItem*.

- **Clase TextBox**

Nuevos Métodos

public void setInitialInputMode(String characterSubset):fija el modo de entrada que será utilizado por la aplicación cuando el usuario inicie a editar este *TextBox*, si se pasa null como parámetro, entonces la implementación escogerá un modo de entrada por defecto.

- **Clase TextField**

Nuevos Atributos

public static final int DECIMAL: permite al usuario ingresar valores numéricos con fracciones decimales opcionales, por ejemplo "-123", "0.123", o ".5". La implementación puede desplegar un punto “.” o una coma “,” como separador de la parte decimal, dependiendo de las convenciones utilizadas en el dispositivo. La constante 5 es asignada a DECIMAL.

***public static final int INITIAL_CAPS_SENTENCE**:* indica que la letra inicial de cada oración debe empezar con mayúscula. Un valor de 0x200000 es asignado a INICIAL_CAPS_SENTENCE

public static final int INITIAL_CAPS_WORD: indica que la letra inicial de cada palabra debe estar en mayúscula. Un valor de 0x100000 es asignado a INITIAL_CAPS_WORD.

public static final int NON_PREDICTIVE: indica que el texto ingresado no consiste de palabras que podrían ser encontradas en diccionarios usados típicamente por esquemas de entrada predictivos. Un valor de 0x80000 es asignado a NON_PREDICTIVE.

public static final int SENSITIVE: indica que el texto ingresado es un conjunto de datos que la implementación nunca debería almacenar en un diccionario o tabla para un uso predictivo, auto completado o cualquier otro esquema acelerado de entrada. El número de una tarjeta de crédito es una ejemplo de dato sensitivo. Un valor de 0x40000 es asignado a SENSITIVE.

public static final int UNEDITABLE: indica que la edición esta actualmente deshabilitada, es decir, no se puede cambiar el contenido de este objeto. Un valor de 0x20000 es asignado a UNEDITABLE.

Nuevos Métodos

public void setInitialInputMode(String characterSubset):fija el modo de entrada que será utilizado por la aplicación cuando el usuario inicie a editar este *TextField*, si se pasa null como parámetro, entonces la implementación escogerá un modo de entrada por defecto.

2. Paquetes para la creación de Juegos

Adicionalmente a las modificaciones y adiciones presentadas en las clases anteriores para el manejo de interfaces gráficas de usuario, el perfil MID2.0, ha incluido el paquete

javax.microedition.lcdui.game, el cual provee una serie de clases que posibilitan el desarrollo de juegos con rico contenido gráfico en dispositivos inalámbricos.

Los dispositivos inalámbricos tienen mínimas capacidades de procesamiento, por lo cual la mayor parte de este API intenta mejorar su funcionamiento reduciendo al mínimo la cantidad de trabajo hecho en java, esta característica trae además el beneficio adicional de reducir el tamaño de la aplicación. El API está estructurado para proveer considerable libertad cuando se implemente sobre él, permitiendo el uso extensivo de código nativo, aceleración de hardware y formatos de datos de imagen específicos del dispositivo cuando sea necesario.

El API usa las clases para gráficos estándar de bajo nivel de MIDP (*Graphics*, *Image*,etc) para poder utilizar las clases del API de juegos de alto nivel en conjunto con primitivas gráficas, por ejemplo es posible un fondo complejo utilizando el API de juegos y dibujar algo encima de él utilizando primitivas gráficas como *drawLine*, etc.

El API esta compuesto por las siguientes clases:

GameCanvas: es una subclase de *lcdiu.Canvas* que provee la funcionalidad de pantalla básica para un juego. Además de los métodos propios de *Canvas*, también provee características tales como la habilidad para conocer el estado actual de las teclas en un juego e imágenes sincronizadas; estas características simplifican el desarrollo del juego y mejoran su rendimiento.

Layer: representa un elemento visual en un juego así como un Stprite o un TiledLayer. Esta clase abstracta provee atributos básicos como localización, tamaño y visibilidad.

LayerManager: para juegos que emplean varios Layers, esta clase simplifica el desarrollo del juego brindando al desarrollador un mejor control de las diferentes vistas del juego.

Sprite: es una capa animada básica que puede desplegar uno o mas contenedores gráficos. Los contenedores son todos de igual tamaño y proveen la imagen simple de un

objeto. Además hace posible animar estos contenedores de manera secuencial o arbitraria y provee varios métodos de transformación de imágenes (estirado y rotación), así como métodos para detección de colisión que simplifican la implementación de la lógica del juego.

TiledLayer: esta clase permite al desarrollador crear grandes áreas de contenido gráfico sin que una imagen grande sea requerida. Se compone de una rejilla de celdas y cada celda puede desplegar una de varias porciones provistas por un simple objeto de imagen.

3. Paquetes para el manejo del ciclo de vida de las aplicaciones

Define las aplicaciones MIDP y las interacciones entre la aplicación y el entorno sobre el cual se está ejecutando.

MIDP define un modelo de aplicación para permitir que los recursos limitados de los dispositivos sean compartidos por múltiples aplicaciones MIDP. Este modelo define que es un MIDlet, como es empaquetado, que ambiente de ejecución está disponible para ese MIDlet y cómo debe comportarse de tal forma que puedan ser administrados los recursos del dispositivo.

En el paquete javax.microedition.midlet solo se encuentra la clase *MIDlet*. Un *MIDlet* es una aplicación MIDP, la aplicación debe extender esta clase para permitir al software de administración de aplicaciones controlar al *MIDlet* y permitirle recuperar características del descriptor de la aplicación así como notificar y solicitar cambios de estado. Los métodos de esta clase permiten crear, iniciar, pausar y destruir una aplicación.

Nuevos Métodos

public final int checkPermission(String permission): obtiene el estado de un permiso específico. Si no hay una API en el dispositivo que defina los permisos específicos requeridos, entonces estos son reportados como denegados. Si el estado del permiso no es conocido porque puede que se requiera alguna interacción con el

usuario, entonces el estado es reportado como desconocido. Este método devuelve 0 si el permiso es denegado, 1 si se da el permiso y –1 si es desconocido.

public final boolean platformRequest(String URL) throws ConnectionNotFoundException: solicita que el dispositivo maneje (por ejemplo, despliegue o instale) un recurso de la URL indicada. Si se pasa una cadena vacía como parámetro (no null), este método cancela cualquier solicitud hecha anteriormente. Este método devuelve true si la MIDletSuite debe teminar su ejecución antes que el contenido pueda ser descargado(por ejemplo, en la actualización de la aplicación MID que está corriendo) y false si la aplicación MID puede seguir ejecutándose(por ejemplo al descargar una nueva aplicación).

4. Paquete para persistencia de datos

El MIDP provee un mecanismo para que los MIDlets almacenen datos en forma persistente y posteriormente estos se puedan recuperar. Este mecanismo de almacenamiento es modelado como una pequeña base de datos orientada a registro y se conoce con el nombre de Record Managment System (RMS).

Un record store consiste de una colección de registros que permanecen persistentes durante múltiples invocaciones de un MIDlet. La plataforma es la responsable de mantener la integridad de los RecordStores de un MIDlet, durante el uso normal de la plataforma, incluyendo reinicio, cambios de batería, etc.

Los MIDlets dentro de un MIDlet suite pueden crear múltiples RecordStores, mientras cada uno de ellos tenga distinto nombre. Cuando un MIDlet suite es removido de una plataforma, todos los RecordStores asociados con estos MIDlets deben también ser removidos. Los MIDlets dentro de un MIDlet suite pueden acceder a los RecordStores directamente. Nuevas APIs en MIDP 2.0 permiten compartir explícitamente los RecordStores si el MIDlet crea el RecordStore seleccionando tal permiso. Los controles de acceso se definen cuando un RecordStores que se compartirá es creado. Los controles de acceso se hacen cumplir cuando los RecordStores son abiertos. Los modos

de acceso permiten uso privado o compartido con cualquiera de los MIDlets suite. Los nombres de los RecordStores son sensibles a las mayúsculas y minúsculas y pueden estar formados por combinaciones de hasta 32 caracteres Unicode. Los nombres de los RecordStores deben se únicos dentro del alcance de un MIDlet suite; en otras palabras, MIDlets dentro de un MIDlet suite no deben permitir crear más de un RecordStore con el mismo nombre; sin embargo, un MIDlet en un MIDlet suite puede tener un RecordStore con el mismo nombre que un MIDlet en otro MIDlet suite. En este caso, los RecordStores son considerados distintos y separados.

La implementación de un registro asegura que todas las operaciones individuales de un RecordStore son atómicas, síncronas y serializables de tal manera que no exista corrupción con múltiples accesos. Sin embargo, si un MIDlet usa múltiples hilos para acceder a un record store, este MIDlet es el responsable de coordinar estos accesos, o se pueden tener consecuencias no deseadas.

La única clase dentro del paqute javax.microedition.rms es *RecordStore*, Las modificaciones y adiciones hechas a esta clase en el MIDP2.0 se ven a continuación:

Nuevos atributos

public static final int AUTHMODE_ANY: Autorización para permitir el acceso a cualquier MIDletSuite. AUTHMODE_ANY tiene un valor de 1.

public static final int AUTHMODE_PRIVATE: Autorización para permitir el acceso únicamente a la MIDletSuite actual. AUTHMODE_PRIVATE tiene un valor de 0.

Nuevos métodos

public static RecordStore openRecordStore(String RecordStoreName, boolean createIfNecesary, int authmode, boolean writable) throws RecordStoreException,RecordStoreFullException, RecordStoreNotFoundException: abre y posiblemente crea un almacén de registros que puede ser compartido con otros

Midlet suites. El modo de autorización es fijado cuando el almacén de registros es creado de la siguiente forma:

- AUTHMODE_PRIVATE solo permite que el MIDlet suite que ha creado el *RecordStore* tenga acceso a este. Este comportamiento es idéntico al de openRecordStore(recordStoreName, createIfNecessary) en el MIDP1.0.

- AUTHMODE_ANY: permite a cualquier MIDlet en el dispositivo acceder al *RecordStore*.

El parámetro writable debe ser true si otros MIDlets deben tener la capacidad de escribir en el RecordStore

public static RecordStore openRecordStore(String recordStoreName,String vendorName, String suiteName)throws RecordStoreException, RecordStoreNotFoundException: Abre un *RecordStore* asociado con el nombre del MIDlet suite, el cual es identificado por el MIDlet-vendor y el MIDlet-name. El acceso es posible solo si el modo de autorización del *RecordStore* permite el acceso a la MIDlet suite actual. El acceso se ve limitado por el modo de autorización fijado cuando el almacén de registros fue creado.

public void setMode(int authmode,boolean writable)throws RecordStoreException: cambia el modo de acceso para este *RecordStore*. Solo la MIDlet suite que creó el *RecordStore* tiene los permisos para cambiar su modo de autorización.

5. Paquetes para interconexión con redes

MIDP 2.0 ha adicionado soporte para conexiones seguras a través del protocolo HTTPS, el cual es básicamente HTTP sobre la capa de sockets seguros (SSL). SSL es un protocolo que cifra los datos para ser enviados a través de la red y provee autenticación para sockets finales.

Aunque HTTP y HTTPS son las únicas interfaces requeridas que una implementación MIDP debería soportar, varias interfaces adicionales de red han sido definidas en el MIDP2.0:

CommConnection: define una conexión lógica con el puerto serie.

HttpsConnection: define los métodos y constantes necesarias para establecer conexiones de red seguras.

SecureConnection: define el socket de conexión seguro.

SecurityInfo: define métodos para acceder a la información a través de una conexión de red segura.

ServerSocketConnection: define la conexión a través de un socket servidor.

SocketConnection: define la conexión a través de un socket.

UDPDatagramConnection: define una conexión basada en datagramas en la cual se conoce la dirección de los puntos finales.

Adicionalmente una de las nuevas características más potentes es lo que se ha denominado “Push Registry”, que activa aplicaciones dormidas cuando está disponible cierta información. Un MIDlet puede registrar un *Listener* de tal forma que cuando el MIDlet no está activo, el software de gestión de aplicaciones de la plataforma MIDP (AMS) escucha peticiones entrantes. Cuando una de estas peticiones es recibida, el AMS lanza al MIDlet asociado usando el método startApp(). A través de este sistema, por ejemplo, un proceso servidor puede activar una aplicación en el dispositivo para notificar un nuevo evento, aunque la aplicaciones no esté activa en ese preciso momento, esto se hace a través de la clase *PushRegistry*.

6. Paquetes para seguridad

MIDP 2.0 mejora de manera considerable el modelo de seguridad del MIDP 1.0 a través del paquete ***javax.microedition.pki***. Este paquete provee el manejo de certificados que son usados para la autenticación de la información en conexiones seguras.

En MIDP 2.0 es posible realizar firmado de aplicaciones y manejar dominios privilegiados. Con el firmado de aplicaciones podemos comprobar la identidad del proveedor de la aplicación y por lo tanto confiar en la misma. A través de los privilegios de dominio los proveedores de aplicaciones y los vendedores de dispositivos, pueden definir qué APIs son consideradas como restringidas. Estas nuevas características protegen al dispositivo frente accesos no autorizados por parte de las aplicaciones. Los protocolos seguros estándar como HTTPS, TLS/SSL y WTLS permiten la transmisión segura de datos a través de la encriptación de los mismos.

7. Paquetes para el manejo de audio

El rango de dispositivos J2ME va desde teléfonos celulares con una simple generación de tonos hasta PDAs con avanzadas capacidades de audio y video. Para trabajar con las diversas configuraciones y capacidades de procesamiento de multimedia, se necesita una API con un alto grado de abstracción. La meta del trabajo de MMAPI(Mobile Media API) Expert Group ha sido encaminada a un amplio rango de áreas de aplicaciones, y el resultado del trabajo ha sido ofrecer dos conjuntos de APIs: *Mobile Media API (JSR 135)* y *MIDP2.0 Media API.*

La primera API está enfocada para dispositivos J2ME con avanzadas capacidades de audio y multimedia. La última API es un subconjunto directamente compatible con el API de multimedia y esta enfocada para dispositivos con mayor cantidad de restricciones.

La MIDP2.0 Media API, al ser un subconjunto de la MMAPI, difiere respecto a esta de la siguiente forma: es únicamente para el manejo de audio, es decir no incluye ninguno de los controles específicos para proveer el manejo de video o gráficos y, no incluye el

paquete javax.microedition.media.protocol, utilizado para el manejo de protocolos personalizados.

La MIDP2.0 Media API cuenta con una sola clase, ***Manager***, la cual sirve como punto de acceso para obtener los diferentes recursos del sistema. Las demás funcionalidades son provistas a través de interfaces de los paquetes ***javax.microedition.media*** y ***javax.microedition.media.control***.

BIBLIOGRAFÍA

GALVEZ ROJAS SERGIO. ORTEGA DIAZ LUCAS. JAVA A TOPE: J2ME(JAVA 2 MICRO EDITION). Universidad de Málaga 2003.

Motorola, Inc. and Sun Microsystems, Inc. Mobile Information Device Profile, v2.0 (JSR-118). Noviembre de 2002

Sun Microsystems Inc. Mobile Information Device Profile, v1.0 (JSR-37). Diciembre de 2000.