BANORTE

Anexo IV – Ejemplos de Integración Java y .Net

Diciembre 2019
Versión 2.3

Ejemplos de Integración

Integración basada en Java

Banorte le ofrecerá como input para este tipo de operaciones, una biblioteca dinámica (dll) BanortePinpadSeguro.dll y BanortePinpadSeguro.jar, los cuales contienen las llamadas a los métodos proporcionados en la API.

Será necesario que el usuario establezca el ambiente de ejecución, adecuado en su entorno de Java para ajustar las referencias a las bibliotecas. Para realizar esto usted puede:

1. Instalar el archivo BanortePinPad.jar (entregado por Banorte) en una ubicación para que pueda ser localizable por el JRE. Esto puede hacerse de varias maneras, dependiendo de las características específicas de la aplicación del cliente:
   - Incluyendo el archivo BanortePinPad.jar, en la variable de entorno CLASSPATH. Por ejemplo:

```
CLASSPATH=C:\AplicPuntoVenta\AplicPuntoVenta.jar;
C:\AplicPuntoVenta\Banorte\BanortePinPadSeguro.jar
```

   - Incluyendo el archivo BanortePinPad.jar como referencia en el manifiesto de la aplicación, si ésta se encuentra contenida en un archivo JAR autoejecutable. Por ejemplo:

```
Manifest-Version: 1.0
Main-Class: com.miempresa.puntoventa.Main Class-
Path: PuntoVenta.jar BanortePinPadSeguro.jar
```

   - Copiando el archivo BanortePinPad.jar al directorio de extensiones del JRE utilizado, típicamente lib/ext (no recomendado).
2. Copiar el archivo BanortePinPad.dll a alguna ubicación deseada, y poner visible dicha ubicación al JRE para que pueda cargar la biblioteca dinámica en tiempo de ejecución. Esto puede hacerse típicamente de varias maneras:
   - Especificando en la propiedad **java.library.path** la ubicación donde se copió el archivo BanortePinPad.dll. Esta propiedad deberá pasarse al JRE al momento de ejecutar la aplicación principal. Por ejemplo:

```
Java -cp
C:\AplicPuntoVenta\AplicPuntoVenta.jar;C:\AplicPuntoVenta\Ba
norte\BanortePinPadSeguro.jar
-D java.library.path=C:\AplicPuntoVenta\Banorte
```

   - Especificando la ubicación donde se copió el archivo BanortePinPadSeguro.dll en la variable de entorno LIBRARY_PATH. Por ejemplo:

```
LIBRARY_PATH=C:\AplicPuntoVenta\Banorte
```

- Copiando el archivo DLL a la ubicación por defecto del sistema operativo (Por ejemplo, C:\Windows\System32).
- Si el integrador desea hacer uso de alguna herramienta IDE (tal como Eclipse o NetBeans) para probar la aplicación con la API de Banorte, entonces será suficiente indicar en el proyecto creado en el IDE la ubicación del archivo BanortePinPadSeguro.jar en el CLASSPATH asignado a dicho proyecto.

El paquete de clases preparado para usuarios Java consta de una serie de clases e interfaces, cuyo detalle de utilización se describe en las siguientes subsecciones. La mayoría de los métodos ofrecidos por las clases pueden eventualmente lanzar la excepción ***BanorteException*** en caso de un problema; la aplicación del cliente deberá considerar esta situación para poder atrapar posibles excepciones y en su caso, tomar la acción correspondiente. La totalidad de clases e interfaces se encuentran dentro del paquete **com.banorte.pinpad**, el cual deberá ser referenciado por la aplicación del cliente por medio del estatuto **import**.

Para aquellos usuarios familiarizados con la documentación tipo **Javadoc**, se ofrece una ayuda en este formato, la cual se entrega junto con el archivo **BanortePinPadSeguro.jar**.

> **NOTA:** Para el uso del archivo **BanortePinPadSeguro.jar** y de la Dll es necesario contar con Java JDK 1.5 o superior.

Interfaz PIN Pad

La API para Java contiene una interfaz que define ciertos métodos que todo dispositivo PIN Pad provisto por Banorte deberá proporcionar. Dentro del mismo paquete existirá una clase por cada diferente tipo de dispositivo soportado por Banorte. En la versión actual de la API, se ofrecen las clases Vx810Segura y Vx820Segura.

Aún cuando la creación del objeto PIN Pad se haga sobre una versión en particular de dispositivo, se recomienda al diseñador de la aplicación del cliente tratar de usar la interfaz genérica, para simplificar el mantenimiento en caso de un eventual cambio de dispositivo.

Así, en vez de hacer esto:

```
Vx810 pinpad = new Vx810();
```

Se recomienda hacer esto:

```
Vx810Segura pinpad = new Vx810Segura();
```

De esta forma, la aplicación del cliente puede dinámicamente instanciar el PIN Pad a utilizar (probablemente con base a un archivo de configuración), y el código que haga uso de los servicios del PIN Pad será **independiente del modelo del dispositivo**.

Creación de objeto PIN Pad Java

Para comenzar a utilizar el PIN Pad, será necesario instanciar un objeto de tipo PIN Pad usando algunas de las clases concretas disponibles en el paquete. Como parámetro opcional, el usuario puede especificar una cadena de caracteres que indique el idioma deseado. Recuérdese que los textos de los parámetros de entrada y salida, así como los textos de los mensajes de error reportados por las excepciones varían en función del idioma seleccionado.

Ejemplos:

```
//Crea el objeto pinpad en Español
Vx810Segura pinpad = new Vx810Segura("ES")

//Crea el objeto pinpad en Inglés
Vx810Segura pinpad = new Vx810Segura("EN")
```

Este paso deberá hacerse una vez por cada PIN Pad conectada físicamente al equipo de punto de venta; la referencia devuelta será utilizada más adelante para solicitar servicios al objeto creado.

Inicialización de dispositivo

Una vez creado el objeto PIN Pad, el paso siguiente consiste en inicializar el dispositivo. Recuérdese que el PIN Pad se conecta por medio de un puerto serial, físico o virtual, por lo que será necesario definir los parámetros de configuración de dicho puerto.

El PIN Pad originalmente entregado de fábrica cuenta con la siguiente configuración:

Tabla 1. Configuración de PIN Pad

Parámetro	Valor
Velocidad	19200 bps
Paridad	Ninguna
Bits de datos	8
Bits de paro	1

Para inicializar el dispositivo, deberá ejecutarse una llamada al método ***prepareDevice***, pasando como parámetro de entrada un objeto de tipo **java.util.Map**. Este objeto deberá tener una entrada por cada combinación (parámetro, valor) que se requiera. Tanto el nombre del parámetro, como el valor deberán ser de tipo **java.lang.String**. La tabla con los parámetros necesarios se presenta en la subsección Parámetros de Inicialización.

Ejemplo:

```
//Se crea la tabla con los parámetros de inicialización
HashMap config = new HashMap(5);

config.put("PORT", "COM5");
config.put("BAUD_RATE", "19200");
config.put("PARITY", "N");
config.put("STOP_BITS", "1");
config.put("DATA_BITS", "8");

//Se inicializa la pinpad con los parámetros de la tabla
try {
pinpad.prepareDevice(config);
}
catch(BanorteException e) {
System.out.println("Falla al inicializar Pinpad: " + e.getMessage());
}
```

La llamada anterior deberá hacerse solamente una sola vez, durante el tiempo de vida de la aplicación.

Inicio de transacción Java

Por cada transacción que se desee ejecutar, será necesario primero hacer una llamada al método ***startTransaction*** del objeto PIN Pad. Esto se requiere para preparar el hardware del dispositivo para una nueva operación.

Este método no requiere parámetros.

Ejemplo:

```
//Se inicia la transacción
try{
pinpad.startTransaction();
} catch (BanorteException e) {
System.out.println("Falla al inicializar transacción: " + e.getMessage());
}
```

Obtener la información del PIN Pad

Esta función realiza la petición al dispositivo para que provea la información de la versión del dispositivo. La llamada ***getInformation*** retorna el número de serie del dispositivo y la versión de la aplicación financiera instalada en el mismo. Esta llamada se ejecuta antes de realizar una carga de llave de cifrado o de una actualización de la llave. A continuación se muestra el procedimiento:

```
//Creamos el HashMap para obtener la información
HashMap informacion = new HashMap(5);
String numeroSerie = "";
```

```
try {
pinpad.getInformation(parametrosSalida);
numeroSerie = informacion.get("NUMERO_SERIE");
} catch (BanorteException e) {
System.out.println("Falla al obtener la información: " + e.getMessage());
}
```

Carga de llaves

Carga de llaves con Modo de Operación: Procesar Transacción

Esta función será utilizada exclusivamente por comercios en el que su punto de venta tiene acceso directo a Banorte mediante la Internet. Mediante la llamada ***updateMasterKey*** se realizará la carga de la llave de encriptación en el dispositivo. A continuación se muestra un ejemplo:

```
//Creamos el HashMap para realizar la carga de llave
HashMap cargarEntrada = new HashMap();

cargarEntrada.put("MERCHANT_ID", "7395007");
cargarEntrada.put("USER", "a7395007");
cargarEntrada.put("PASSWORD", "********");
cargarEntrada.put("CONTROL_NUMBER", "CARGALLAVE001");
cargarEntrada.put("RESPONSE_LANGUANGE", "EN");
cargarEntrada.put("BANORTE_URL",
"https://via.pagosbanorte.com/InterredesSeguro");
cargarEntrada.put("SERIAL_NUMBER", numeroSerie);

//Realizamos la carga de la llave
try {
pinpad.updateMasterKey(parametrosEntrada);
} catch (BanorteException e) {
System.out.println("Falla al cargar llave: " + e.getMessage());
}
```

Si la seguridad de la información del dispositivo pudiera llegar a estar comprometida, puede solicitarse la regeneración de una llave de encriptación y volver a ejecutar la llamada ***updateMasterKey***.

Carga de llave con Modo de Operación: Leer, Enviar y Notificar Transacción

Para los usuarios que desde su punto de venta no cuenten con acceso directo a Banorte será necesario realizar los siguientes pasos:

Obtención del selector

Es necesario solicitar el selector del dispositivo para obtener de Banorte la llave que el dispositivo específico requiere. Mediante la llamada ***getSelector*** se obtienes este dato. Por ejemplo:

```
//Se crea el HashMap para obtener el selector
HashMap selectorSalida = new HashMap();

//Se solicita el selector al pinpad
try {
pinpad.getSelector(selectorSalida);
String selector = selectorSalida("SELECTOR");
} catch (BanorteException e) {
System.out.println("Falla al obtener selector: " + e.getMessage());
}
```

Solicitud de la llave de encriptación

Como el punto de venta no tiene acceso directo a Banorte, la solicitud de la llave de encriptación se puede realizar utilizando la clase ***ConectorBanorte***, la cual contiene un único método estático llamado ***sendTransaction***. Este método utiliza dos **HashMap**, uno de parámetros de entrada y otro de parámetros de salida. Este método no realiza una conexión con el PIN Pad, sino que envía una transacción o un comando hacia Banorte, por lo tanto puede ser utilizado en una ubicación diferente al punto de venta, por ejemplo un servidor. A continuación se presenta un ejemplo:

```
//Se crean los HashMaps para solicitor la llave a Banorte
HashMap llaveEntrada = new HashMap();
HashMap llaveSalida = new HashMap();

//Llenamos los parámetros de la table
llaveEntrada.put("MERCHANT_ID", "7395007");
llaveEntrada.put("USER", "a7395007");
llaveEntrada.put("PASSWORD", "*********");
llaveEntrada.put("RESPONSE_LANGUAGE", "EN");
llaveEntrada.put("CMD_TRANS", "GET_KEY");
llaveEntrada.put("CONTROL_NUMBER", "SOLICITARLLAVE001");
llaveEntrada.put("BANORTE_URL",
"https://via.pagosbanorte.com/InterredesSeguro");
llaveEntrada.put("SELECTOR", selector);

//Utilizamos el método sendTransaction para solicitar la llave a Banorte
try {
ConectorBanorte.sendTransaction(llaveEntrada, llaveSalida);
} catch (BanorteException e) {
System.out.println("Falla al obtener llave: " + e.getMessage());
}

//Obtenemos los datos de la respuesta
String resultadoPayw = get.llaveSalida("PAYW_RESULT");
String codigoPayw = get.llaveSalida("PAYW_CODE");
String llaveMaestra = get.llaveSalida("TEXT");
```

Carga de llave de encriptación

Ya que se tiene la llave de encriptación se procede a cargar la llave en el dispositivo. Para realizar la carga en el dispositivo se realizan los siguientes pasos:

```
//Se valida que se haya obtenido correctamente la llave resutadoPayw = "A"
if (!"A".equals(resultadoPayw)) {
HashMap cancelarCarga = new HashMap()
cancelarCarga.put("SERIAL_NUMBER", numeroSerie);

//Se cancela carga de la llave
try {
pinpad.cancelarCargaLlave(parametrosEntrada);
} catch (BanorteException e) {
System.out.println("Falla al cancelar la carga: " + e.getMessage());
}
} else {

//Se crea el HashMap para cargar la llave en el pinpad
HashMap cargarLlave = new HashMap();

cargarLlave.put("SERIAL_NUMBER", numeroSerie);
cargarLlave.put("MASTER_KEY", llaveMaestra);

//Realizamos la carga de la llave en el pinpad
try {
pinpad.loadMasterKey(cargarLlave);
} catch (BanorteException e) {
System.out.println("Falla al cargar llave: " + e.getMessage());
}
}
```

Si el comercio cree que la llave de encriptación del dispositivo está comprometida deberá solicitar a su ejecutivo que genere una nueva llave de encriptación. Después el comercio realizará nuevamente los pasos para la carga de la llave, desde la obtención del selector hasta la carga de la llave.

Procesamiento de transacciones

Proceso autónomo de transacción (Procesar Transacción)

Para aquellos usuarios que prefieran dejar a cargo de la API el envío de la transacción hacia Banorte, ésta es la única llamada que requieren hacer para completar el proceso de leer la tarjeta, formar la transacción, enviarla a Banorte, recibir la respuesta y procesarla. La aplicación no recibirá el control hasta que se tenga respuesta del banco, o bien, haya expirado el tiempo máximo especificado para la transacción. En caso de recibir respuesta del banco, los parámetros de salida indicarán el resultado de la transacción; de lo contrario ocurrirá una excepción.

El método a ejecutar es ***processTransaction*** sobre el objeto PIN Pad; previamente deberá haberse ejecutado el método ***startTransaction***, ya que de lo contrario no se recibirá la respuesta esperada del dispositivo.

Este método espera dos parámetros, cada uno de ellos de tipo **java.util.Map**. El primer mapa define los parámetros de entrada, los cuales proveerán información para que la transacción

pueda procesarse; el segundo mapa deberá pasarse sin información, y será llenado por la API con el resultado de la transacción.

Los parámetros de entrada requeridos para este método se muestran en la tabla de la sección **Parámetros de entrada** del proceso de transacción. A continuación se muestra un ejemplo:

```
//Se crean las tablas para los parámetros de la transacción
HashMap parametrosEntrada = new HashMap(20);
HashMap parametrosSalida = new HashMap(20);

//Parámetros de entrada de la transacción
parametrosEntrada.put("MERCHANT_ID", "7395007");
parametrosEntrada.put("USER", "a7395007");
parametrosEntrada.put("PASSWORD", "*********");
parametrosEntrada.put("CMD_TRANS", "AUTH");
parametrosEntrada.put("TERMINAL_ID", "327779587");
parametrosEntrada.put("AMOUNT", "0.01");
parametrosEntrada.put("CONTROL_NUMBER", "VENTA0001");
parametrosEntrada.put("MODE", "PRD");
parametrosEntrada.put("RESPONSE_LANGUAGE", "EN");
parametrosEntrada.put("BANORTE_URL",
"https://via.pagosbanorte.com/InterredesSeguro");

//Se realiza el proceso de la transacción
try {
pinpad.processTransaction(parametrosEntrada, parametrosSalida);
} catch (BanorteException e) {
System.out.println("Falla al procesar la transacción: " + e.getMessage());
}

String declinadaOffline = (String) parametrosSalida.get("CHIP_DECLINED");

//Validamos si la transacción fue declina Offline
if (declinadaOffline.equals("1")) {
pinpad.displayText("Declinada Offline");
} else {

//Si no fue declinada Offline se revisa el resultado
String codigoBanorte = (String) parametrosSalida.get("PAYW_RESULT");

//Verificar si la transacción fue aprobada
if (codigoBanorte.equals("A")) {
String codigoAut = (String) parametrosSalida.get("AUTH_CODE");
System.out.println("Transacción aprobada con código: " + codigoAut);
pinpad.displayText("Aprobada: " + codigoAut);
} else {
System.out.println("Transacción declinada");
pinpad.displayText("Declinada");
}
}
```

Proceso de transacciones por módulos (Leer, Enviar y Notificar)

Lectura de tarjeta

Para aquellos usuarios que únicamente deseen utilizar la API para lectura de tarjetas y envíen la transacción a Banorte por un medio alterno, ésta llamada será la necesaria para obtener la información sobre tarjeta leída. A partir de los datos de salida devueltos por esta llamada, la aplicación será responsable de formar el mensaje hacia Banorte, dependiendo del medio que utilice para tal fin, y una vez obtenido el resultado, independientemente de si fue aprobada, declinada o sin respuesta. Para transacciones de chip deberá ejecutar una llamada al método ***notifyResult***, el cual se explica a continuación.

Para pedir al PIN Pad que se prepare para leer una tarjeta, la aplicación del cliente deberá ejecutar la llamada al método ***readCard***. Este método tiene dos versiones, la primera espera dos argumentos: un mapa de parámetros de entrada y otro de parámetros de salida. La segunda versión espera como único argumento un mapa de parámetros de salida.

La primera versión del método (con parámetros de entrada y de salida) debe usarse siempre que la lectura de tarjeta sea con el fin de procesar alguna transacción que eventualmente pudiera ser con tarjeta de chip, para que se hagan las inicializaciones necesarias en el lector correspondiente. La aplicación del cliente debe especificar como mínimo el monto de la transacción que se ejecutará. Recuérdese que si la transacción es de chip, la aplicación del cliente debe notificar el resultado posteriormente con una llamada al método ***notifyResult***.

Obsérvese que si se usa esta versión del método, entonces como mínimo deberá de pasarse el monto de la transacción en el mapa de parámetros de entrada; de lo contrario el método lanzará una excepción.

La segunda versión del método (sin parámetros de entrada) puede usarse cuando la aplicación del cliente trabaje con tarjetas propias de banda magnética y no hay la posibilidad de que se presenten transacciones de chip.

En ambas versiones, el mapa de parámetros de salida se deberá proporcionar inicialmente vacío; el método internamente llenará el mapa con información para la aplicación acerca de las características de la lectura realizada.

Como en otros métodos, el tipo de los mapas de parámetros será **java.util.Map**, y tanto el nombre del parámetro como su valor serán de tipo **java.lang.String**.

En el momento de la ejecución de este método, el PIN Pad desplegará un mensaje en su pantalla invitando al cliente a insertar o deslizar su tarjeta. En el caso de tarjeta de chip, ésta no deberá retirarse del PIN Pad hasta que la aplicación haya completado la llamada a ***notifyResult***.

Los posibles parámetros de entrada que se pueden pasar a este método se documentan en el Anexo V.

```
//Se crean los HashMaps para los parámetros de la lectura de la tarjeta
HashMap leerEntrada = new HashMap(20);
HashMap leerSalida = new HashMap(20);
```

```
//Se crean los HashMaps para los parámetros para enviar la transacción
HashMap parametrosEntrada = new HashMap(20);
HashMap parametrosSalida = new HashMap(20);

//Ingresamos los parámetros de entrada para la lectura de la tarjeta
String amount = "1";
String PagoMovil = "0";
leerEntrada.put("AMOUNT", amount);
leerEntrada.put("PAGO_MOVIL", PagoMovil);

//Se solicita la lectura de la tarjeta
try{
pinpad.readCard(leerEntrada, leerSalida);
}catch(BanorteException e){
System.out.println("Error al leer la tarjeta" + e.getMessage());
}

//Recuperamos datos de la Lectura de la Tarjeta
String Track1 = (String) leerSalida.get("TRACK1");
String Track2 = (String) leerSalida.get("TRACK2");

//Determinar el Tipo de Entrada (Banda/Chip/Contactless)
String PosEntryMode = (String) leerSalida.get("MODO_ENTRADA");
String declinadaChip = "";
String EmvTags = "";

//Revisamos si es de Chip para así poder obtener los EMVTAGS
if (PosEntryMode.equals("CHIP")||(PosEntryMode.equals("CONTACTLESSCHIP"))){
EmvTags = (String) leerSalida.get("EMV_TAGS");
parametrosEntrada.put("EMV_TAGS", EmvTags);
declinadaChip = (String)leerSalida.get("CHIP_DECLINED");
}

//Primeramente validamos si no fue un Declinado Offline, validando el valor
de la variable de retorno DECLINADA_CHIP
if (declinadaChip.equals("1"){

//Aquí termina la transacción
System.out.println("La transacción fue declinada offline");
pinpad.desplegarTexto("Declinada Offline");
}else{

//Se envía transacción a Banorte mediante el método readCard
}
```

NOTA: Al enviar el parámetro **PAGO_MOVIL** con valor de “1” en el método ***readCard*** (***leerTarjeta***) el PIN Pad permitirá los 3 métodos de lectura de tarjeta: banda, chip y manual (Pago Móvil). Si se envía con valor de “0” únicamente permitirá la lectura de banda, de chip y de Contactless.

Envío de Transacción

Para aquellos usuarios que por su arquitectura no estén en condiciones de enviar directamente la transacción a Banorte desde su punto de venta por medio de la llamada a ***processTransaction***, pero que puedan hacerlo desde otro punto y no requieran de utilizar ningún componente de software adicional, la API provee una clase denominada ***ConectorBanorte***, el cual posee un único método de tipo estático ***sendTransaction***, que tiene la capacidad de enviar una transacción hacia Banorte a partir de un mapa de parámetros de entrada y entregar información sobre el resultado en un mapa de parámetros de salida inicialmente vacío.

Como siempre, los mapas de parámetros tanto de entrada como de salida, deberán ser objetos que implementen la interfaz **java.util.Map**, y tanto los nombres como los valores de cada parámetro serán objetos de tipo **java.lang.String**.

Es importante enfatizar, que este método no tiene relación alguna con el manejo de la transacción a nivel PIN Pad, sino que constituye una vía de comunicación directa a Banorte para aquellos usuarios que por sus características puedan encontrarlo de utilidad. La aplicación del cliente deberá usar las otras llamadas de la API (particularmente ***readCard*** y ***notifyResult***) en los equipos que tengan conectado el PIN Pad para hacerse cargo de la manipulación de éste.

También es importante señalar que, debido a que la comunicación se hace directamente con el procesador central de pagos de Banorte, es necesario especificar el idioma en que se manejarán los parámetros. Los dos idiomas soportados por Payworks son español e inglés.

La aplicación del cliente deberá ser responsable de asegurarse de poner la totalidad de parámetros necesarios en el mapa de entrada. Por ejemplo, si utiliza el método ***readCard*** para procesar las lecturas de tarjetas de clientes y una de ellas resultare ser de chip, deberá asegurarse de incluir los parámetros **EMV_TAG** y **ENTRY_MODE** que la llamada a ***readCard*** le haya regresado en el mapa de parámetros de entrada que se pase al método ***sendTransaction***.

Al igual que en el caso del método ***processTransaction*** invocado sobre el objeto PIN Pad, el método ***sendTransaction*** invocado estáticamente sobre la clase ***ConectorBanorte*** requieren visibilidad hacia Banorte desde el equipo en donde se hace la ejecución.

Obsérvese que la gran diferencia entre el método ***processTransaction*** invocado sobre el objeto PIN Pad y el método ***sendTransaction*** invocado estáticamente sobre la clase ***ConectorBanorte***, es que el primero incluye el manejo de la lectura de la tarjeta en el PIN Pad y la notificación del resultado de la transacción, mientras que el segundo es exclusivamente para hacer llegar una transacción hacia el procesador central de pagos de Banorte y se asume que la aplicación del cliente se está haciendo cargo aparte del manejo del PIN Pad por medio de las llamadas a ***readCard*** y ***notifyResult***.

Un ejemplo de invocación de este método sería el siguiente:

```
//Llenamos la tabla para enviar la transacción
parametrosEntrada.put("MERCHANT_ID", "7395007");
parametrosEntrada.put("USER", "a7395007");
parametrosEntrada.put("PASSWORD", "*********");
parametrosEntrada.put("CMD_TRANS", "AUTH");
```

```
parametrosEntrada.put("TERMINAL_ID", "327779587");
parametrosEntrada.put("CONTROL_NUMBER", "TRANSACCIONPRO249");
parametrosEntrada.put("MODE", "PRD");
parametrosEntrada.put("AMOUNT", amount);
parametrosEntrada.put("TRACK2", Track2);
parametrosEntrada.put("ENTRY_MODE", PosEntryMode);
parametrosEntrada.put("RESPONSE_LANGUAGE", "EN");
parametrosEntrada.put("BANORTE_URL",
"https://via.pagosbanorte.com/InterredesSeguro");

//Validamos si el Track1 está presente
if Track1 != "" {
   parametrosEntrada.put("TRACK1", EmvTags);
}

//Método Enviar Transacción
try{
ConectorBanorte.sendTransaction(parametrosEntrada,parametrosSalida);
}catch(BanorteException e){
System.out.println("Error al enviar la operación a Banorte: " +
e.getMessage());
}
```

Al enviar la transacción hacia Payworks, el parámetro **PAGO_MOVIL** deberá ir en "1" para transacciones realizadas de forma manual (Pago Móvil). Si el método de entrada es diferente a **MANUAL** el parámetro **PAGO_MOVIL** no deberá ser enviado o enviado con valor de "0".

Respecto a transacciones con American Express, la aplicación del comercio debe solicitar el CVV2 antes de enviar la transacción cuando en la transacción la lectura sea por banda magnética, debido a que éste es un parámetro de entrada del método y se envía en la transacción.

Notificación de resultado

La ejecución de este método es requerida para aquellos usuarios que se hagan cargo del envío de la transacción, cuando ésta sea de chip.

El método ***readCard*** retorna a la aplicación del cliente información en los parámetros de salida acerca del tipo de lectura obtenida (chip, banda, contactlesschip o contactlessbanda). En el caso de chip, la aplicación deberá ejecutar una llamada a este método una vez que tenga el resultado de la transacción. La llamada deberá hacerse, independientemente si la transacción fue aprobada o declinada, e inclusive si no hubo respuesta.

En caso de que el banco emisor de la tarjeta haya decidido enviar información de autenticación al chip de la tarjeta, ésta será recibida por la aplicación del cliente al momento de recibir respuesta de la transacción enviada a Banorte y deberá pasarse como parámetro de entrada al método ***notifyResult***.

Finalmente, sólo para el caso de transacciones aprobadas, deberá también proporcionarse como parámetro de entrada el código de autorización recibido. Para mayor detalle revisar los parámetros de notificación de resultado.

Un ejemplo para invocar este método se describe a continuación:

```
// Validamos si la tarjeta es CHIP
if (PosEntryMode.equals("CHIP")){

//Se crean los HashMaps para los parámetros de Notificar Resultado
HashMap parametrosNotifyEntrada = new HashMap(20);
HashMap parametrosNotifySalida = new HashMap(20);

//Obtenemos datos de salida del procesamiento de la transacción
String resultadoPayw = (String) parametrosSalida.get("PAYW_RESULT");
String codigoAut = (String)parametrosSalida.get("AUTH_CODE");
String datosEMV = (String)parametrosSalida.get("EMV_DATA");

//Validamos si hubo respuesta de la transacción
if(resultadoPayw != null){

//Validamos si existe información en DATOS_EMV
if (datosEMV != null){
parametrosNotifyEntrada.put("EMV_DATA", datosEMV);
}

if (resultadoPayw.equals("A")){
parametrosNotifyEntrada.put("RESULT", "APPROVED");
parametrosNotifyEntrada.put("AUTH_CODE", codigoAut);
}else
if (resultadoPayw.equals("D")){
parametrosNotifyEntrada.put("RESULT", "DECLINED");
}else{
parametrosNotifyEntrada.put("RESULT", "NO_RESPONSE");
}
 }
} else{
parametrosNotifyEntrada.put("RESULT", "NO_RESPONSE");
}

//Método notificar resultado
try {
pinpad.notifyResult(parametrosNotifyEntrada, parametrosNotifySalida);
}
catch (BanorteException e) {
System.out.println("Error al notificar el resultado:" +e.getMessage());
}

//Obtenemos el resultado de notificar la transacción
String resultadoEMV = (String) parametrosNotifySalida.get("EMV_RESULT");

//Validamos resultadoEMV
if (resultadoEMV != null){

if (resultadoEMV.equals("D") && resultadoPayw.equals("A")){
pinpad.displayText("DECLINADA EMV");

//Se genera el reverso por Declinado EMV
```

```
//Se crean los hashMaps para el reverso
HashMap reversaEntrada = new HashMap(20);
HashMap reversaSalida = new HashMap(20);

//Se obtiene la referencia de la transacción anterior
String reference = (String) parametrosSalida.get("REFERENCE");

//Se llena la table con los parámetros del reverso
reversaEntrada.put("MERCHANT_ID", "7395007");
reversaEntrada.put("USER", "a7395007");
reversaEntrada.put("PASSWORD", "********");
reversaEntrada.put("CMD_TRANS", "REVERSAL");
reversaEntrada.put("TERMINAL_ID", "327779587");
reversaEntrada.put("REFERENCE", reference);
reversaEntrada.put("MODE", "PRD");
reversaEntrada.put("RESPONSE_LANGUAGE", "EN");
reversaEntrada.put("BANORTE_URL",
"https://via.pagosbanorte.com/InterredesSeguro");

//Como fue un declinado EMV se envía el parámetro CAUSE con valor fijo 17
reversaEntrada.put("CAUSE", "17");

//Método Enviar Transacción para enviar el reverso
try{
ConectorBanorte.sendTransaction(parametrosEntrada,parametrosSalida);
}catch(BanorteException e){
System.out.println("Error al enviar la operación a Banorte: " +
e.getMessage());
}
}else if (resultadoEMV.equals("A") && resultadoPayw.equals("A")){
pinpad.displayText("APROBADA");
}else{
pinpad.displayText("DECLINADA");
}
}
}
}else{

//Si la transacción fue realizada con banda magnética
String resultado = (String)parametrosSalida.get("PAYW_RESULT");

//Únicamente se valida el resultado de Payworks
if (resultado.equals("A")){
String codAut = (String) parametrosSalida.get("AUTH_CODE");
pinpad.displayText("APROBADA: " + codAut);
}else{
pinpad.displayText("DECLINADA");
}
}
```

Finalizar transacción

Una vez completada una transacción, la aplicación deberá ejecutar el método ***endTransaction*** para indicar este evento al dispositivo. Obsérvese que la ejecución de este método, junto con el

de inicio de transacción deberá hacerse **POR CADA TRANSACCION** realizada con el dispositivo. Este método no requiere parámetros.

Ejemplo:

```
try {
pinpad.endTransaction();
} catch(BanorteException e) {
System.out.println("Falla al terminar la transacción: " +
e.getMessage());
}
```

Despliegue de texto

El método ***displayText*** es útil y puede ser empleado por la aplicación del cliente para personalizar los mensajes que aparecen en la pantalla del PIN Pad.

Recibe como parámetro único un objeto de tipo **java.lang.String** que contiene el texto que se desea desplegar. Se recomienda no utilizar acentos ni caracteres especiales, ya que no es seguro que éstos sean soportados por todas las versiones de firmware en los dispositivos.

Ejemplo:

Se desea desplegar en pantalla el mensaje "APROBADA: xxxxxx", donde xxxxxx es el código de autorización recibido de la transacción, o bien DECLINADA en caso de rechazo.

```
if (codigoBanorte.equals("A")){
String codigoAut = (String)parametrosSalida.get("AUTH_CODE");
pinpad.displayText("Aprobada: " + codigoAut);
}else {
pinpad.displayText(" DECLINADO ");
}
```

Liberación de dispositivo

El método ***releaseDevice*** debe ejecutarse al terminar la operación del PIN Pad, para asegurarse de liberar el puerto serial y los recursos asignados por el sistema operativo. Este método no requiere parámetros.

Ejemplo:

```
try {
pinpad.releaseDevice();
} catch(BanorteException e) {
System.out.println("Falla al liberar dispositivo: " + e.getMessage());
}
```

Obtención de versión de la API

El método ***getVersion*** es un método utilitario suministrado para que la aplicación del cliente pueda verificar con qué versión la API se encuentra trabajando y así llevar un control efectivo en caso de que posteriormente sean liberadas versiones adicionales con nuevas capacidades. Este es un método estático que deberá ser invocado sobre la clase API, incluida dentro del mismo paquete **com.banorte.pinpad**.

El método deberá ser invocado con un objeto de tipo **java.util.Map** inicialmente vacío; que el método retornará lleno con información sobre su versión. La tabla ¡Error! No se encuentra el origen de la referencia. muestra los nombres de dichos parámetros, a fin de que la aplicación del usuario pueda reconocerlos.

Ejemplo:

```
//Se crea el HashMap para obtener la versión
HashMap parametrosSalida = new HashMap();

try {
API.getVersion(parametrosSalida);
String version = parametrosSalida.get("VERSION");
System.out.println("Versión de API: " + version);
} catch(BanorteException e) {
System.out.println("Incapaz de obtener versión de API: " +
e.getMessage());
}
```

Excepciones

Los métodos de la API en general pueden lanzar una excepción si ocurrió un problema durante la ejecución del método. La clase que representa esta excepción se llama ***BanorteException*** y está dentro del mismo paquete **com.banorte.pinpad** que se incluye dentro del archivo .JAR entregado al cliente.

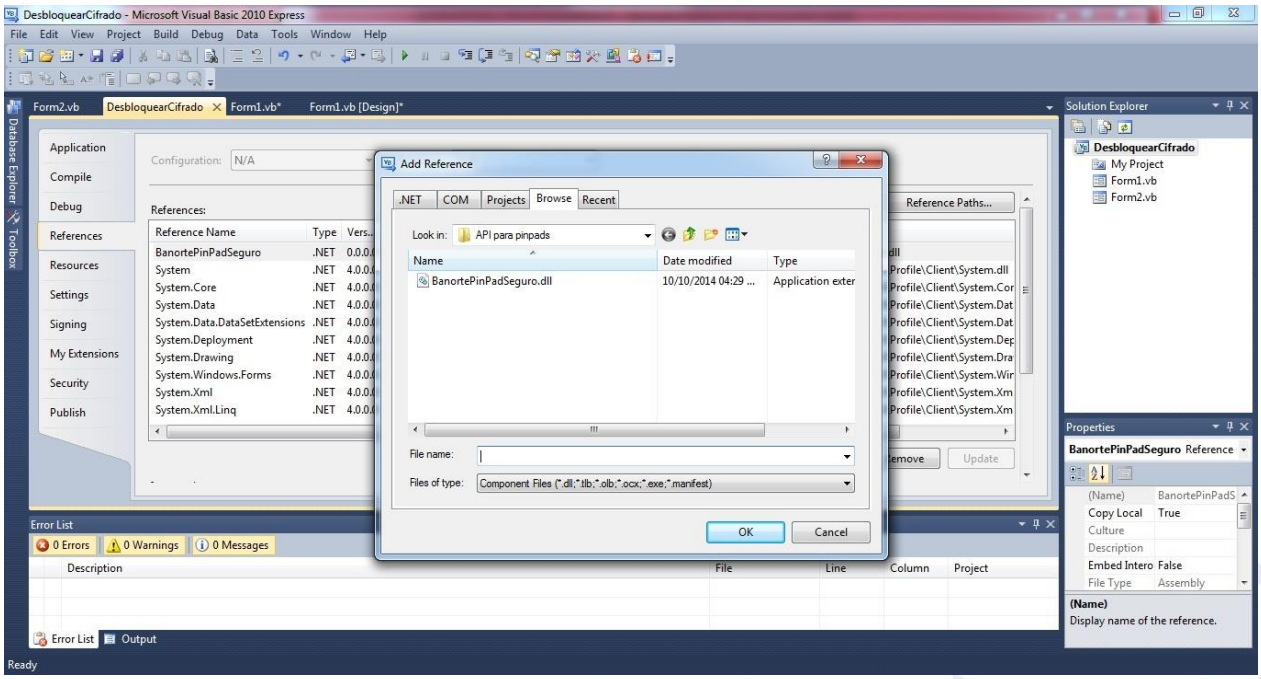
La clase ***BanorteException*** deriva directamente de **java.lang.Exception**, y por tanto, el mensaje de error representado por la excepción puede obtenerse, como con cualquier excepción Java, por medio del método ***getMessage()***. Sin embargo, para comodidad del usuario, se incluye un método adicional, denominado ***getCodigo()***, el cual provee un código de error que la aplicación del cliente puede analizar para determinar exactamente la causa del error reportado y tomar la acción correspondiente. En el apéndice especial se encuentran documentados los posibles códigos de error y sus textos asociados. También se proporciona una breve ayuda sobre cuáles podrían ser las causas y posibles soluciones.

Integración con aplicaciones .NET

Para los usuarios que utilizan .NET que desean integrar con la API Banorte, será necesario realizar lo siguiente:

Tendremos que generar una referencia a **BanortePinPadSeguro.dll**, esto dando botón derecho al proyecto y Agregar referencia (Ver Figura 1):

Figura 1. Agregar referencia VB 2010



La DLL diseñada y detallada en esta sección muestra cada uno de los métodos a usar y sus respectivos ejemplos para una mayor comprensión. La mayoría de los métodos ofrecidos por las clases pueden eventualmente lanzar la excepción ***BanorteException*** en caso de un problema; la aplicación del cliente deberá considerar esta situación para poder atrapar posibles excepciones y en su caso tomar la acción correspondiente. La totalidad de clases e interfaces se encuentran dentro de **Banorte.PinPad**.

> **NOTA:** Debido a que el archivo **BanortePinpadSeguro.dll** fue compilado utilizando el .NET Framework 4.5 y visual c++ 2013, es necesario que el desarrollo de la aplicación de punto de venta sea realizado en Visual Studio 2013.

Interfaz PIN Pad

La API para .NET contiene una interfaz que define ciertos métodos que todo dispositivo PIN Pad provisto por Banorte deberá proporcionar. Dentro del mismo paquete existirá una clase por cada diferente tipo de dispositivo soportado por Banorte.

Aún cuando la creación del objeto PIN Pad se haga sobre una versión en particular de dispositivo, se recomienda al diseñador de la aplicación del cliente tratar de usar la interfaz como sigue:

```
'DECLARA UN OBJETO PINPAD
Dim pinpad As New Banorte.PinPad.Vx820Segura("EN")
```

De esta forma, la aplicación del cliente puede dinámicamente instanciar el PIN Pad a utilizar (probablemente con base a un archivo de configuración), y el código que haga uso de los servicios del PIN Pad será independiente del dispositivo específico.

Creación de objeto PIN Pad

Para comenzar a utilizar el PIN Pad, será necesario instanciar un objeto de tipo PIN Pad usando algunas de las clases concretas disponibles en la dll. Como parámetro opcional, el usuario puede especificar una cadena de caracteres que indique el idioma deseado. En este caso los idiomas disponibles son español (ES) e inglés (EN). Recuérdese que los textos de los parámetros de entrada y salida, así como los textos de los mensajes de error reportados por las excepciones varían en función del idioma seleccionado.

```
'DECLARA UN OBJETO PINPAD EN EL IDIOMA INGLÉS
Dim pinpad As New Banorte.PinPad.Vx820Segura("EN")

'DECLARA UN OBJETO PINPAD EN EL IDIOMA ESPAÑOL
Dim pinpad As New Banorte.PinPad.Vx820Segura("ES")
```

Este paso deberá hacerse una vez por cada PIN Pad conectada físicamente al equipo de punto de venta; la referencia devuelta será utilizada más adelante para solicitar servicios al objeto creado.

Inicialización de dispositivo

Una vez creado el objeto PIN Pad, el paso siguiente consiste en inicializar el dispositivo. Recuérdese que el PIN Pad se conecta por medio de un puerto serial, físico o virtual, por lo que será necesario definir los parámetros de configuración de dicho puerto. El PIN Pad cuenta con la siguiente configuración:

Tabla 2. Configuración de PIN Pad

Parámetro	Valor
Velocidad	19200 bps

Paridad	Ninguna
Bits de datos	8
Bits de paro	1

Para inicializar el dispositivo, deberá ejecutarse una llamada al método ***prepareDevice***, pasando como parámetro de entrada un objeto de tipo **Hashtable**. Este objeto deberá tener una entrada por cada combinación (parámetro, valor) que se requiera. Tanto el nombre del parámetro como el valor deberán ser de tipo **String**. La tabla con los parámetros necesarios se presenta en la subsección Parámetros de Inicialización.

```
'CREAR TABLA DE PARÁMETROS DE CONFIGURACIÓN DEL DISPOSITIVO
Dim config As New Hashtable()
config.Add("PORT", "COM1")
config.Add("BAUD_RATE", "19200")
config.Add("PARITY", "N")
config.Add("STOP_BITS", "1")
config.Add("DATA_BITS", "8")
Try
pinpad.prepareDevice(config)
Catch ex As Exception
MsgBox("Falla al iniciar la transacción: " + ex.Message)
End Try
```

La llamada anterior deberá hacerse solamente una sola vez durante el tiempo de vida de la aplicación.

Inicio de transacción

Por cada transacción que se desee ejecutar, será necesario antes hacer una llamada al método ***startTransaction*** del objeto PIN Pad. Esto se requiere para preparar el hardware del dispositivo para una nueva operación. Este método no requiere parámetros.

```
Try
pinpad.startTransaction()
Catch ex As Exception
MsgBox("Falla al iniciar la transacción: " + ex.Message)
End Try
```

Obtener la información del PIN Pad

Esta función realiza la petición al dispositivo para que provea la información contenida en el dispositivo. La llamada ***getInformation*** retorna el número de serie del dispositivo y la versión de la aplicación financiera instalada en el mismo. Esta llamada se realiza antes de aplicar una carga de llave de cifrado o de una actualización de la llave. A continuación se muestra el procedimiento:

```
'CREAMOS EL HASHTABLE PARA OBTENER LA INFORMACIÓN
Dim salidaInformacion As New Hashtable()
Dim numeroSerie As String

'OBTENEMOS LA INFORMACIÓN DEL DISPOSITIVO
Try
Pinpad.getInformation(salidaInformacion)
numeroSerie = salidaInformacion.Item("SERIAL_NUMBER")
Catch ex As Exception
   MsgBox("Falla al obtener la información del pinpad: " + ex.Message)
End Try
```

Carga de llaves

Carga de llaves con modo de operación Process Transaction

Esta función será utilizada exclusivamente por comercios en el que su punto de venta tiene acceso directo a Banorte mediante la Internet. Mediante la llamada ***updateMasterKey*** se realizará la carga de la llave de encriptación en el dispositivo. A continuación se muestra un ejemplo:

```
'CREAMO EL HASTABLE PARA LA CARGA DE LA LLAVE
Dim entradaCarga As New Hashtable()

'LLENAMOS EL HASHTABLE
entradaCarga.Add("USER", "a7395007")
entradaCarga.Add("PASSWORD", "********")
entradaCarga.Add("MERCHANT_ID", "7395007")
entradaCarga.Add("CONTROL_NUMBER", "CARGA0001")
entradaCarga.Add("RESPONSE_LANGUAGE", "EN")
entradaCarga.Add("SERIAL_NUMBER", numeroSerie)
entradaCarga.Add("BANORTE_URL",
"https://via.pagosbanorte.com/InterredesSeguro")

'REALIZAMOS LA CARGA DE LA LLAVE
Try
Pinpad.updateMasterKey(entradaCargar)
Catch ex As Exception
   MsgBox("Falla al cargar la llave en el pinpad: " + ex.Message)
End Try
```

Si la seguridad de la información del dispositivo estuviera comprometida puede solicitarse la regeneración de una llave de encriptación y volver a ejecutar la llamada ***updateMasterKey***.

Carga de llave con modo de operación Leer, Enviar y Notificar Transaccion

Para los usuarios que desde su punto de venta no cuenten con acceso directo a Banorte será necesario realizar los siguientes pasos:

Obtención del selector

Es necesario solicitar el selector del dispositivo para obtener de Banorte la llave que el dispositivo especifico requiere. Mediante la llamada ***getSelector*** se obtienes este dato. Por ejemplo:

```
'SE CREA EL HASHTABLE PARA LA OBTENCIÓN DEL SELECTOR
Dim salidaSelector As New Hashtable()
Dim selector As String

'SE SOLICITA EL SELECTOR
Try
Pinpad.getSelector(salidaSelector)
selector = salidaSelector("SELECTOR")
Catch ex As Exception
   MsgBox("Falla al obtener el selector: " + ex.Message)
End Try
```

Solicitud de la llave de encriptación

Como el punto de venta no tiene acceso directo a Banorte, la solicitud de la llave de encriptación se puede realizar utilizando la clase ***ConectorBanorte***, la cual contiene un único método estático llamado ***sendTransaction***. Este método utiliza dos **hashtables**, uno de parámetros de entrada y otro de parámetros de salida. Este método no realiza una conexión con el PIN Pad, sino que envía una transacción o un comando hacia Banorte, por lo tanto puede ser utilizado en una ubicación diferente al punto de venta, por ejemplo un servidor. A continuación se presenta un ejemplo:

```
'SE CREAN LOS HASHTABLES PARA SOLICITAR LA LLAVE
Dim entradaEnviar As New Hashtable()
Dim salidaEnviar As New Hashtable()

entradaEnviar.Add("CMD_TRANS", "GET_KEY")
entradaEnviar.Add("USER", "a7395007")
entradaEnviar.Add("PASSWORD", "**********")
entradaEnviar.Add("MERCHANT_ID", "7395007")
entradaEnviar.Add("CONTROL_NUMBER", "CARGALLAVE0001")
entradaEnviar.Add("SELECTOR", selector)
entradaEnviar.Add("RESPONSE_LANGUAGE", "EN")
entradaEnviar.Add("BANORTE_URL",
"https://via.pagosbanorte.com/InterredesSeguro")

'SE ENVÍA EL COMANDO A BANORTE PARA LA SOLICITUD DE LA LLAVE
Try
Banorte.ConectorBanorte.sendTransaction(entradaEnviar, salidaEnviar)
Catch ex As Exception
   MsgBox("Falla al enviar la transacción: " + ex.Message)
End Try

Dim resultadoPayw, paywCode As String

resultadoPayw = salidaEnviar.Item("PAYW_RESULT")
paywCode = salidaEnviar.Item("PAYW_CODE")
```

Carga de llave de encriptación

Ya que se tiene la llave de encriptación se procede a cargar la llave en el dispositivo. Para realizar la carga en el dispositivo se realizan los siguientes pasos:

```
'SE VALIDA QUE SE HAYA OBTENIDO CORRECTAMENTE LA LLAVE
If resultadoPayw <> "A" Then
Dim entradaCancelar As New Hashtable()
entradaCancelar.Add("SERIAL_NUMBER", numeroSerie)

Try
pinpad.cancelLoadKey(entradaCancelar)
   Catch ex As Exception
MsgBox("Falla al cancelar la carga de llave: " + ex.Message)
End Try
Else
Dim entradaCarga As New Hashtable()
Dim llaveMaestra As String
llaveMaestra = salidaEnviar.Item("TEXT")

entradaCarga.Add("SERIAL_NUMBER", numeroSerie)
entradaCarga.Add("MASTER_KEY", llaveMaestra)

Try
pinpad.loadMasterKey(entradaCarga)
Catch ex As Exception
MsgBox("Falla al realizar la carga de llave: " + ex.Message)
End Try
End If
```

Si el comercio cree que la llave de encriptación del dispositivo está comprometida deberá solicitar a su ejecutivo que genere una nueva llave de encriptación. Después, el comercio realizará nuevamente los pasos para la carga de la llave, desde la obtención del selector hasta la carga de la llave.

Procesamiento de transacciones

Proceso Autónomo de transacción

Para aquellos usuarios que prefieran dejar a cargo de la API el envío de la transacción hacia Banorte, ésta es la única llamada que requieren hacer para completar el proceso de leer la tarjeta, formar la transacción, enviarla a Banorte, recibir la respuesta y procesarla. La aplicación no recibirá el control hasta que se tenga respuesta del banco, o bien, haya expirado el tiempo máximo especificado para la transacción. En caso de recibir respuesta del banco, los parámetros de salida indicarán el resultado de la transacción; de lo contrario ocurrirá una excepción.

El método a ejecutar es ***processTransaction*** sobre el objeto PIN Pad; previamente deberá haberse ejecutado el método ***startTransaction***, ya que de lo contrario no se recibirá la respuesta esperada del dispositivo.

Este método espera dos parámetros, cada uno de ellos de tipo **Hashmap**. El primer mapa define los parámetros de entrada, los cuales proveerán información para que la transacción pueda procesarse; el segundo mapa deberá pasarse sin información, y será llenado por la API con el resultado de la transacción.

Los parámetros de entrada requeridos para este método se muestran en la tabla de la sección Parámetros para formar la transacción.

Ejemplo:

```
'CREAMOS LOS HASHTABLE DE PARAMETROS PARA LA TRANSACCION
Dim parametrosEntrada As New Hashtable()
Dim parametrosSalida As New Hashtable()

'PARAMETROS DE ENTRADA PARA LA TRANSACCION
parametrosEntrada.Add("MERCHANT_ID", "7395007")
parametrosEntrada.Add("USER", "a7395007")
parametrosEntrada.Add("PASSWORD", "*********")
parametrosEntrada.Add("CMD_TRANS", "AUTH")
parametrosEntrada.Add("TERMINAL_ID", "327782962")
parametrosEntrada.Add("CONTROL_NUMBER", "TRANSACCION01")
parametrosEntrada.Add("MODE", "PRD")
parametrosEntrada.Add("AMOUNT", "10.00")
parametrosEntrada.Add("RESPONSE_LANGUAGE", "EN")
parametrosEntrada.Add("BANORTE_URL",
"https://via.pagosbanorte.com/InterredesSeguro")

'METODO PROCESAR TRANSACCION
Try
pinpad.processTransaction(parametrosEntrada, parametrosSalida)
Dim codigoBanorte, codigoAut, declinadaOffline As String

'OBTENEMOS LOS DATOS PARA IDENTIFICAR EL RESULTADO
codigoBanorte = parametrosSalida.Item("PAYW_RESULT")
declinadaOffline = parametrosSalida.Item("CHIP_DECLINED")

'REVISAMOS SI FUE UN DECLINADO OFFLINE
If declinadaOffline = "1" Then
Console.WriteLine("Declinada Offline")
Else
If codigoBanorte = "A" Then
codigoAut = parametrosSalida.Item("AUTH_CODE")
Console.WriteLine("Transacción aprobada: " + codigoAut)
Else
Console.WriteLine("Transacción declinada")
End If
End If
Catch ex As Exception
Console.WriteLine("Falla al procesar transacción: " + ex.Message)
End Try
```

Proceso de transacciones por módulos (Leer, Enviar y Notificar)

Lectura de tarjeta

Para aquellos usuarios que únicamente deseen utilizar la API para lectura de tarjetas y envíen la transacción a Banorte por un medio alterno, ésta llamada será la necesaria para obtener la información sobre tarjeta leída. A partir de los datos de salida devueltos por esta llamada, la aplicación será responsable de formar el mensaje hacia Banorte dependiendo del medio que utilice para tal fin, y una vez obtenido el resultado, independientemente de si fue aprobada, declinada o sin respuesta. Para transacciones de chip deberá ejecutar una llamada al método ***notifyResult***, el cual se explica más adelante.

Para pedir al PIN Pad que se prepare para leer una tarjeta, la aplicación del cliente deberá ejecutar la llamada al método ***readCard***. Este método tiene dos versiones. La primera espera dos argumentos: un **hashtable** de parámetros de entrada y otro de parámetros de salida. La segunda versión espera como único argumento un **hashtable** de parámetros de salida.

La primera versión del método (con parámetros de entrada y de salida) debe usarse siempre que la lectura de tarjeta sea con el fin de procesar alguna transacción que eventualmente pudiera ser con tarjeta de chip, para que se hagan las inicializaciones necesarias en el lector correspondiente. La aplicación del cliente debe especificar como mínimo el monto de la transacción que se ejecutará. Recuérdese que si la transacción es de chip, la aplicación del cliente debe notificar el resultado posteriormente con una llamada al método ***notifyResult***.

Obsérvese que si se usa esta versión del método entonces como mínimo deberá de pasarse el monto de la transacción en el **hashtable** de parámetros de entrada; de lo contrario el método lanzará una excepción.

La segunda versión del método (sin parámetros de entrada) puede usarse cuando la aplicación del cliente trabaje con tarjetas propias de banda magnética y no haya la posibilidad de que se presenten transacciones de chip.

En ambas versiones, el **hashtable** de parámetros de salida se deberá proporcionar inicialmente vacío; el método internamente llenará el **hashtable** con información para la aplicación acerca de las características de la lectura realizada.

En el momento de la ejecución de este método, el PIN Pad desplegará un mensaje en su pantalla invitando al cliente a insertar o deslizar su tarjeta. En el caso de tarjeta de chip, ésta no deberá retirarse del PIN Pad hasta que la aplicación haya completado la llamada a ***notifyResult***.

Los posibles parámetros de entrada que se pueden pasar a este método se documentan en el Anexo V.

Ejemplo:

```
'CREAMOS LOS HASHTABLES PARA LA LECTURA DE LA TARJETA
Dim lecturaEntrada As New Hashtable()
Dim lecturaSalida As New Hashtable()
```

```
'CREAMOS LOS HASHTABLES PARA LOS PARAMETROS PARA ENVIAR LA TRANSACCION
Dim parametrosEntrada As New Hashtable()
Dim parametrosSalida As New Hashtable()

'INGRESAMOS COMO PARAMETRO DE ENTRADA EL MONTO DE LA TRANSACCION
lecturaEntrada.Add("AMOUNT", "10.00")
lecturaEntrada.Add("PAGO_MOVIL", "0")

'METODO DE LEER TARJETA
Try
pinpad.readCard(lecturaEntrada, lecturaSalida)
Catch ex As Exception
MsgBox("Falla al Leer la Tarjeta: " + ex.Message)
End Try
'OBTENEMOS LOS DATOS DE LA LECTURA DE LA TARJETA
Dim track2, track1, posEntryMode, tagsEMV As String

tagsEMV = lecturaSalida.Item("EMV_TAGS")
track1 = lecturaSalida.Item("TRACK1")
track2 = lecturaSalida.Item("TRACK2")
posEntryMode = lecturaSalida.Item("ENTRY_MODE")

'SI LA TARJETA ES CHIP ENVIAMOS LOS TAGS EMV
If PosEntryMode = "CHIP" or  PosEntryMode = "CONTACTLESSCHIP" Then
parametrosEntrada.Add("EMV_TAGS ", tagsEMV)
End If

'OBTENEMOS EL VALOR DE CHIP_DECLINED Y VER SI FUE UN DECLINADO OFFLINE
Dim declinadoOffline As String
declinadoOffline = lecturaSalida.Item("CHIP_DECLINED")

'VALIDAMOS UN DECLINADO OFFLINE
If declinadoOffline = "1" Then

'AQUI TERMINA LA OPERACION
Console.WriteLine("DECLINADO OFFLINE")
Else

'MÉTODO ENVIAR TRANSACCIÓN
End If
```

NOTA: Al enviar el parámetro **PAGO_MOVIL** con valor de "1" en el método ***readCard*** (***leerTarjeta***) el PIN Pad permitirá únicamente la lectura de una tarjeta en forma manual para PagoMóvil. Si se envía con valor de "0" únicamente permitirá la lectura de banda, de chip y Contactleschip.

Envío de transacción

Para aquellos usuarios que por su arquitectura no estén en condiciones de enviar directamente la transacción a Banorte desde su punto de venta por medio de la llamada a ***processTransaction***, pero que puedan hacerlo desde otro punto y no requieran de utilizar ningún componente de software adicional, la API provee una clase denominada ***ConectorBanorte***, el cual posee un

único método de tipo estático ***sendTransaction***, que tiene la capacidad de enviar una transacción hacia Banorte a partir de un **hashtable** de parámetros de entrada y entregar información sobre el resultado en un **hashtable** de parámetros de salida inicialmente vacío.

Es importante enfatizar, que este método no tiene relación alguna con el manejo de la transacción a nivel PIN Pad, sino que constituye una vía de comunicación directa a Banorte para aquellos usuarios que por sus características puedan encontrarlo de utilidad. La aplicación del cliente deberá usar las otras llamadas de la API (particularmente ***readCard*** y ***notifyResult***) en los equipos que tengan conectado el PIN Pad para hacerse cargo de la manipulación de éste.

También es importante señalar que, debido a que la comunicación se hace directamente con el procesador central de pagos de Banorte, los parámetros de entrada deberán especificarse en su versión en inglés, y los retornados en la salida igualmente se entregarán en su versión en inglés.

La aplicación del cliente deberá ser responsable de asegurarse de poner la totalidad de parámetros necesarios en el **hashtable** de entrada. Por ejemplo, si utiliza el método readCard para procesar las lecturas de tarjetas de clientes y una de ellas resultare ser de chip, deberá asegurarse de incluir los parámetros **EMV_TAGS** y **MODO_ENTRADA** que la llamada a readCard le haya regresado en el **hashtable** de parámetros de entrada que se pase al método ***sendTransaction***.

Al igual que en el caso del método ***processTransaction*** invocado sobre el objeto PIN Pad, el método ***sendTransaction*** invocado estáticamente sobre la clase ***ConectorBanorte*** requieren visibilidad hacia Banorte desde el equipo en donde se hace la ejecución.

Obsérvese que la gran diferencia entre el método ***processTransaction*** invocado sobre el objeto PIN Pad y el método ***enviarTransaccion*** invocado estáticamente sobre la clase ***ConectorBanorte***, es que el primero INCLUYE el manejo de la lectura de la tarjeta en el PIN Pad y la notificación del resultado de la transacción, mientras que el segundo es exclusivamente para hacer llegar una transacción hacia el procesador central de pagos de Banorte y se asume que la aplicación del cliente se está haciendo cargo aparte del manejo del PIN Pad por medio de las llamadas a ***readCard*** y ***notifyResult***. Ejemplo:

```
'LLENAMOS LOS PARAMETROS DEL HASHTABLE
parametrosEntrada.Add("CMD_TRANS", "AUTH")
parametrosEntrada.Add("MODE", "PRD")
parametrosEntrada.Add("MERCHANT_ID", "7395007")
parametrosEntrada.Add("USER", "a7395007")
parametrosEntrada.Add("PASSWORD", " a7395007")
parametrosEntrada.Add("TERMINAL_ID", "327782962")
parametrosEntrada.Add("CONTROL_NUMBER", "VENTA201410150001")
parametrosEntrada.Add("TRACK2", track2)
parametrosEntrada.Add("RESPONSE_LANGUAGE", "EN")
parametrosEntrada.Add("BANORTE_URL",
"https://via.pagosbanorte.com/InterredesSeguro")
parametrosEntrada.Add("AMOUNT", "10.00")
parametrosEntrada.Add("ENTRY_MODE", posEntryMode)

If posEntryMode = "CHIP" or  posEntryMode = "CONTACTLESSCHIP"  Then
parametrosEntrada.Add("EMV_TAGS", tagsEMV)
```

```
ElseIf track1 <> "" Then
parametrosEntrada.Add("TRACK1", track1)
   End If
End If

'TOMAMOS DATOS DEL HASHTABLE Y LA ENVIA A PROCESAR A BANORTE
Try
Banorte.ConectorBanorte.sendTransaction(parametrosEntrada,
parametrosSalida)
Catch ex As Exception
MsgBox("Falla al procesar la transacción: " + ex.Message)
End Try
```

Notificación de resultado

La ejecución de este método es requerida para aquellos usuarios que se hagan cargo del envío de la transacción, cuando ésta sea de chip.

El método ***readCard*** retorna a la aplicación del cliente información en los parámetros de salida acerca del tipo de lectura obtenida (chip, banda o banda forzada). En el caso de chip, la aplicación deberá ejecutar una llamada a este método una vez que tenga el resultado de la transacción. La llamada deberá hacerse independientemente, si la transacción fue aprobada o declinada, e inclusive si no hubo respuesta.

En caso de que el banco emisor de la tarjeta haya decidido enviar información de autenticación al chip de la tarjeta, ésta será recibida por la aplicación del cliente al momento de recibir respuesta de la transacción enviada a Banorte y deberá pasarse como parámetro de entrada al método ***notifyResult***.

Finalmente, sólo para el caso de transacciones aprobadas, deberá también proporcionarse como parámetro de entrada, el código de autorización recibido a los parámetros de entrada que deben pasarse al método ***notifyResult***.

En el ejemplo de código mostrado en esta sección, se indica la lógica completa para una aplicación que se hace cargo de la transacción; el fragmento correspondiente a la llamada a ***notifyResult*** se pone aquí para comodidad del lector.

```
Dim codigoBanorte, codigoAut As String
codigoBanorte = parametrosSalida("PAYW_RESULT")
codigoAut = parametrosSalida("AUTH_CODE")

'REVISAREMOS SI LA TRANSACCION FUE CHIP O BANDA
If posEntryMode = "CHIP" Then

   'CREAMOS EL HASHTABLE PARA NOTIFICAR EL RESULTADO
   Dim parametrosEntradaNotificacion As New Hashtable()
   Dim parametrosSalidaNotificacion As New Hashtable()

'NOTIFICAMOS EL RESULTADO DEBIDO A QUE ES CHIP
If codigoBanorte <> "" Then
Dim datosEMV As String
```

```
datosEMV = parametrosSalida("EMV_DATA")

If datosEMV <> "" Then
parametrosEntradaNotificacion.Add("EMV_DATA", datosEMV)
End If

If codigoBanorte = "A" Then
parametrosEntradaNotificacion.Add("RESULT", "APPROVED")
parametrosEntradaNotificacion.Add("AUTH_CODE", codigoAut)
ElseIf condigoBanorte = "D"
parametrosEntradaNotificacion.Add("RESULT", "DECLINED")
Else
   parametrosEntradaNotificacion.Add("RESULT", "NO_RESPONSE")
End If
Else
parametrosEntradaNotificacion.Add("RESULT", "NO_RESPONSE")
End If

'METODO NOTIFICAR RESULTADO
Try
pinpad.notifyResult(parametrosEntradaNotificacion,
parametrosSalidaNotificacion)
Catch ex As Exception
MsgBox("Falla al notificar el resultado: " + ex.Message)
End Try

Dim EMVResultado As String
EMVResultado = parametrosSalidaNotificacion("EMV_RESULT")

If EMVResultado = "A" And codigoBanorte = "A" Then
Console.WriteLine("APROBADA")
ElseIf EMVResultado = "D" And codigoBanorte = "A" Then
Console.WriteLine("DECLINADA EMV")
Console.WriteLine("***** AQUI SE ENVIA LA REVERSA *****")

'GENERAMOS LOS HASTABLE PARA EL REVERSO DE LA TRANSACCIÓN ANTERIOR
Dim entradaReversa As New Hashtable()
Dim salidaReversa As New Hashtable()
Dim referencia As String
referencia = parametrosSalida.Item("REFERENCE")

'LLENAMOS LOS PARAMETROS DEL HASHTABLE
entradaReversa.Add("CMD_TRANS", "REVERSAL")
entradaReversa.Add("MODE", "PRD")
entradaReversa.Add("MERCHANT_ID", "7395007")
entradaReversa.Add("USER", "a7395007")
entradaReversa.Add("PASSWORD", " a7395007")
entradaReversa.Add("TERMINAL_ID", "327782962")
entradaReversa.Add("REFERENCE", referencia)
entradaReversa.Add("RESPONSE_LANGUAGE", "EN")
entradaReversa.Add("BANORTE_URL",
"https://via.pagosbanorte.com/InterredesSeguro")

'EN TODAS LAS REVERSAS CUANDO EL EMV_RESULT ES IGUAL A "D" SE DEBE ENVIAR EL
PARÁMETRO "CAUSA" CON VALOR FIJO DE "17"
```

```
entradaReversa.Add("CAUSA", "17")

'TOMAMOS LOS DATOS DEL HASHTABLE Y SE ENVIA EL REVERSO A BANORTE
Try
Banorte.ConectorBanorte.sendTransaction(entradaReversa,
salidaReversa)
Catch ex As Exception
MsgBox("Falla al procesar el reverso: " + ex.Message)
End Try
Else
Console.WriteLine("DECLINADA")
End If
Else

'SOLO REPORTAMOS EL RESULTADO DEBIDO A QUE ES BANDA
If codigoBanorte = "A" Then
Console.WriteLine("APROBADA: " + codigoAut)
Else
Console.WriteLine("DECLINADA")
End If
End If
```

Finalizar transacción

Una vez completada una transacción, la aplicación deberá ejecutar este método para indicar este evento al dispositivo. Obsérvese que la ejecución de este método, junto con el de inicio de transacción deberá hacerse **POR CADA TRANSACCION** realizada con el dispositivo. Este método no requiere parámetros.

```
Try
pinpad.endTransaction()
Catch ex As Exception
MsgBox("Falla: " + ex.Message)
End Try
```

Despliegue de Texto

El método ***displayText*** es un método utilitario que puede ser empleado por la aplicación del cliente para personalizar los mensajes que aparecen en la pantalla del PIN Pad. Recibe como parámetro único un objeto de tipo **String** que contiene el texto que se desea desplegar. Se recomienda no utilizar acentos ni caracteres especiales, ya que no es seguro que éstos sean soportados por todas las versiones de firmware en los dispositivos.

Este llamado se puede operar como en el siguiente ejemplo:

```
If (CodigoBanorte = "A") Then
pinpad.displayText("Aprobada: " + CodigoAut)
Else
pinpad.displayText("Declinada")
End If
```

Liberación de dispositivo

El método ***releaseDevice*** debe ejecutarse al terminar la operación del PIN Pad, para asegurarse de liberar el puerto serial y los recursos asignados por el sistema operativo. Este método no requiere parámetros.

Ejemplo:

```
Try
pinpad.releaseDevice()
Catch ex As Exception
Console.WriteLine("Error al liberar dispositivo: " + ex.Message)
End Try
```

Obtención de versión de la API

El método ***getVersion*** es un método utilitario suministrado para que la aplicación del cliente pueda verificar con qué versión de la API se encuentra trabajando y así llevar un control efectivo en caso de que posteriormente sean liberadas versiones adicionales con nuevas capacidades.

El método deberá ser invocado con un objeto de tipo **Hashtable** inicialmente vacío; que el método retornará lleno con información sobre su versión. A continuación se muestra los nombres de dichos parámetros, a fin de que la aplicación del usuario pueda reconocerlos.

Ejemplo:

```
Dim salidaVersion As New Hashtable()
Dim version As String

Try
API.getVersion(salidaVersion)
version = salidaVersion.Item("VERSION")
Console.WriteLine("Versión de API: "+ version)
Catch ex As Exception
Console.WriteLine("Falla al obtener la versión: " + ex.Message)
End Try
```

Excepciones

Los métodos de la API en general pueden lanzar una excepción si ocurrió un problema durante la ejecución del método. La clase que representa esta excepción se llama ***BanorteException*** y se encuentra de igual manera dentro de ***BanortePinPad.dll*** (**Banorte.BanorteException**).

Para comodidad del usuario, se incluye un método adicional, denominado **getCodigo()**, el cual provee un código de error que la aplicación del cliente puede analizar para determinar exactamente la causa del error reportado y tomar la acción correspondiente. En el apéndice B se encuentran documentados los posibles códigos de error y sus textos asociados. También se proporciona una breve ayuda sobre cuáles podrían ser las causas y posibles soluciones.