

GERENCIA DE SISTEMAS**Integración POS Getnet con Puntos de Venta****Manual de Integración Paso a Paso**

Para:  
C# .Net  
Java  
Javascript  
Node.JS  
PHP  
C++  
Lenguaje C

Solicitante	Fecha
Cargo: Jefe de Proyectos Nombre: Alvaro Vera	02-11-2023



## CONTENIDO

1	HISTORIAL DE CAMBIOS .....	5
2	INTRODUCCIÓN .....	6
3	REQUERIMIENTOS MINIMOS.....	6
4	CONFIGURACION INICIAL DEL POS.....	7
5	ENTENDIMIENTO DE LOS COMPONENTES.....	11
5.1	HABILITAR PUERTOS COM.....	11
5.2	VALIDAR LA CONFIGURACIÓN DE PUERTOS COM .....	16
5.3	PRUEBA DE LOS SIMULADORES .....	17
5.4	ENTENDER EL FLUJO DE LA COMUNICACIÓN.....	19
5.5	COMANDOS A ENVIAR AL POS .....	20
5.6	PROBAR LA COMUNICACIÓN.....	21
5.7	ENTENDIMIENTO DE LAS SOLICITUDES.....	22
5.8	ENTENDIMIENTO DE LAS RESPUESTAS.....	23
5.9	LOGS.....	24
5.10	PARAMETROS DE VENTA .....	26
5.11	OTRAS RESPUESTAS POSIBLES.....	29
5.12	MENSAJES INTERMEDIOS .....	30
5.13	PRUEBAS ADICIONALES .....	32
5.14	PRUEBAS CON EL POS GETNET .....	32
5.15	PRUEBAS CON EL POS GETNET Y VYSOR .....	32
5.16	AGENTE POS .....	37
6	ESTRUCTURA DE LA MENSAJERIA .....	40
6.1	<b>VENTA</b> .....	40
6.1.1	BINES .....	41
6.2	<b>ULTIMO COMPROBANTE</b> .....	45
6.3	<b>ANULAR VENTA</b> .....	51
6.4	<b>CIERRE</b> .....	54
6.5	TOTALES DE VENTAS.....	58
6.6	<b>DETALLE DE VENTAS</b> .....	62
6.7	POLLING.....	65
6.8	CAMBIO A POS NORMAL.....	66
6.9	<b>TRANSAKCÓN DE DEVOLUCIÓN</b> .....	68
6.10	<b>DUPLICADO OTROS</b> .....	71
6.11	COMPRAS POR VENDEDOR .....	75
6.12	REPORTE DE PROPINAS .....	78
6.13	COMPRA PREDETERMINADA.....	80
6.14	REPORTE DE PARÁMETROS .....	82
6.15	REPORTE SIM .....	85
6.16	CANCELAR VENTA .....	88
6.17	<b>VENTA MULTICOMERCIO</b> .....	89
6.17.1	BINES .....	92
6.17.2	COMMERC DATA .....	92



---

6.17.3	COMMERCE PARAMS.....	93
<b>6.18</b>	<b>ANULACIÓN MULTICOMERCIO .....</b>	<b>99</b>
6.18.1	COMMERCE DATA .....	101
6.18.2	DE90 .....	101
<b>6.19</b>	<b>DEVOLUCIÓN MULTICOMERCIO .....</b>	<b>104</b>
6.19.1	COMMERCE DATA .....	105
<b>6.20</b>	<b>MAIN POS DATA .....</b>	<b>108</b>
<b>7</b>	<b>PROCESO DE INTEGRACIÓN PARA PROYECTOS WINFORM C# .NET .....</b>	<b>110</b>
7.1	LIBRERIA.....	110
7.2	AGREGAR LA LIBRERÍA EN LA SECCIÓN DE USING.....	111
7.3	CONTROLAR LA COMUNICACIÓN SERIAL Y TIMEOUTS .....	111
7.4	DEFINIR MANEJADORES DE LA COMUNICACIÓN HACIA LA CAJA .....	111
7.5	CONTROLAR EL CIERRE DE PUERTOS.....	112
7.6	CONTROLAR LOS TIMERS .....	112
7.7	CONOCER LOS PUERTOS DISPONIBLES .....	112
7.8	CONECTAR AL PUERTO SERIAL.....	113
7.9	ENVIAR SOLICITUDES AL POS .....	114
7.10	LEER LAS RESPUESTAS DEL POS .....	118
7.11	CASTEO DE OBJETOS RESPONSES .....	118
<b>8</b>	<b>PROCESO DE INTEGRACIÓN PARA PROYECTOS WEB C# .NET Y JAVASCRIPT .....</b>	<b>122</b>
8.1	LIBRERIA.....	122
8.2	ENTENDIENDO EL PROYECTO DE EJEMPLO .....	123
8.3	AGREGAR LA LIBRERÍA EN UN ARCHIVO JS .....	124
8.4	CREAR VARIABLES PARA ASOCIALOS CON LOS BOTÓNES DEL FRONTEND	124
8.5	CONTROLAR LOS LISTENERS .....	124
8.6	ENVIAR SOLICITUDES AL POS .....	125
8.7	LEER LAS RESPUESTAS DEL POS .....	128
8.8	PROCESAMIENTO DE LA RESPUESTA EN EL BACKEND .....	130
8.9	DESERIALIZACIÓN DE LOS DATOS .....	131
8.10	PROCESO ESPECIAL PARA EL NAVEGADOR FIREFOX .....	132
8.11	PROCEDIMIENTO DE INSTALACIÓN.....	132
8.12	FUNCIONAMIENTO DEL SERIAL COMMUNICATION SERVER .....	135
8.13	FUNCIONAMIENTO EN FIREFOX .....	136
<b>9</b>	<b>PROCESO DE INTEGRACIÓN PARA PROYECTOS JAVA .....</b>	<b>137</b>
9.1	DECLARACIÓN DE VARIABLES SEGÚN EL COMANDO .....	137
9.2	CONECTAR AL PUERTO SERIAL.....	138
9.3	ESCUCHAR LA COMUNICACIÓN POR EL PUERTO SERIAL.....	139
9.4	ENVIAR SOLICITUDES AL POS .....	140
9.5	CONTROL DE TIMEOUTS.....	142
9.6	CONTROL DE LA COMUNICACIÓN.....	142
<b>10</b>	<b>PROCESO DE INTEGRACIÓN PARA PROYECTOS NODE.JS .....</b>	<b>143</b>
10.1	INICIAR SIMULADOR DE POS Y CAJA .....	143
10.2	PROBAR FUNCIONAMIENTO DE COMANDOS .....	144
10.3	PROCESO DE INTEGRACIÓN.....	145



---

10.4 LIBRERIA.....	145
10.5 AGREGAR LA LIBRERÍA EN LA SECCIÓN REQUIRE.....	145
11 PROCESO DE INTEGRACIÓN PARA PROYECTOS PHP.....	146
11.1 DESDE JAVASCRIPT.....	146
11.2 DESDE PHP .....	146
11.3 COMANDOS .....	148
12 PROCESO DE INTEGRACIÓN PARA PROYECTOS EN C++ .....	152
12.1 INICIAR SIMULADORES DE POS Y CAJA.....	152
12.2 PROBAR FUNCIONAMIENTOS DE COMANDOS .....	153
12.3 PROCESO DE INTEGRACION PARA PROYECTOS WINFORM C++/CLI .NET FRAMEWORK.....	153
12.4 LIBRERÍA.....	153
12.5 AGREGAR LA LIBRERÍA EN LA SECCIÓN DE USING.....	154
12.6 CONTROLAR LA COMUNICACIÓN SERIAL Y TIMEOUTS .....	154
12.7 DEFINIR MANEJADORES DE LA COMUNICACIÓN HACIA LA CAJA .....	155
12.8 CONTROLAR EL CIERRE DE PUERTOS.....	155
12.9 CONTROLAR LOS TIMERS .....	156
12.10 CONOCER LOS PUERTOS DISPONIBLES .....	156
12.11 CONECTAR AL PUERTO SERIAL.....	157
12.12 ENVIAR SOLICITUDES AL POS .....	158
12.13 LEER LAS RESPUESTAS DEL POS .....	161
12.14 CASTEO DE OBJETOS RESPONSE .....	162
12.15 EJEMPLO.....	164
13 PROCESO DE INTEGRACIÓN PARA PROYECTOS EN LENGUAJE C .....	166
13.1 CONECTAR AL PUERTO SERIAL.....	167
13.2 ENVIAR SOLICITUDES AL POS .....	167
13.3 LEER LAS RESPUESTAS DEL POS .....	168
13.4 CASTEO DE RESPONSES.....	169
13.5 EJEMPLO.....	170
13.6 CONTROLAR EL CIERRE DE PUERTOS.....	171
14 EXCEPCIONES .....	172
15 ANEXOS .....	173
15.1 COMANDOS A ENVIAR AL POS .....	173
15.2 CODIGOS DE RESPUESTAS .....	174
15.3 ABREVIATURAS DE IDENTIFICADOR DE TARJETAS EN LA MENSAJERIA .....	183



## 1 HISTORIAL DE CAMBIOS

Nombre	Versión	Fecha	Detalle
Gerardo Villarroel	1.0	2023-11-02	Primera entrega.
Gerardo Villarroel	1.1	2023-11-15	<p>Se agrega la sección del proceso de integración para proyectos WEB.</p> <p>Se actualiza el tipo de dato "String" para la variable Last4Digits en los comandos SALE, LAST VOUCHER y DUPLICATE.</p> <p>Se normalizan las NOTAS.</p>
Sebastian Berrios	1.2	2023-12-06	Se agrega documentación sobre la pieza del Serial Communication Server para el navegador Firefox.
Sebastian Berrios	1.3	2023-12-28	Se agrega documentación sobre el SDK para Node.JS.
Victor Saldivia	1.4	2024-01-05	Se agrega documentación para el SDK para PHP.
Cesar Huidobro	1.5	2024-01-29	Se agrega documentación para el SDK para Java.
Gerardo Villarroel	1.6	2024-01-30	Se agrega el mensaje intermedio especial 96 Comando en Curso.
Sebastian Berrios	1.7	2024-01-31	Se agrega documentación para el SDK para C++.
Giovanni Vallejos	1.8	2024-02-01	Se agrega documentación para el SDK para Lenguaje C.
Gabriel Duerto	1.9	2024-07-24	Se agrega la configuración inicial del POS.
Gerardo Villarroel	1.10	2024-09-25	Se revisan algunos detalles de ortografía y se agregan pantallazos de la nueva versión del Simulador de Caja 1.18.
Gerardo Villarroel	1.11	2025-06-03	Se agregan comandos nuevos para MC y actualizaciones varias para nuevos parámetros y un nuevo Simulador de Caja.



## 2 INTRODUCCIÓN

El siguiente documento contempla una guía paso a paso para realizar la integración del POS Integrado Getnet con cualquier Software de Caja desarrollado en:

- C# .Net Framework (desde la versión 4 en adelante)
- Java 1.8
- Javascript
- Node.JS
- PHP 5.6
- C++
- Lenguaje C

## 3 REQUERIMIENTOS MÍNIMOS

Se asume que el encargado de la integración sea un profesional con experiencia en el desarrollo de software en el lenguaje C# .Net, Java, Javascript, Node.JS, PHP, C++ y Lenguaje C que pueda entender conceptos simples que permitan la integración de una componente .DLL, .JAR, .JS, .PHP, desde la cual se podrán utilizar métodos específicos que permitan la comunicación con el POS Integrado Getnet; de manera que su sistema de Caja pueda solicitar pagos con tarjetas de crédito y/o débito, esperar que el usuario final realice la transacción y recibir la respuesta de la operación por medio de un proceso de callback obteniendo un JSON con los valores requeridos y necesarios para ser posteriormente procesados por el sistema de caja.

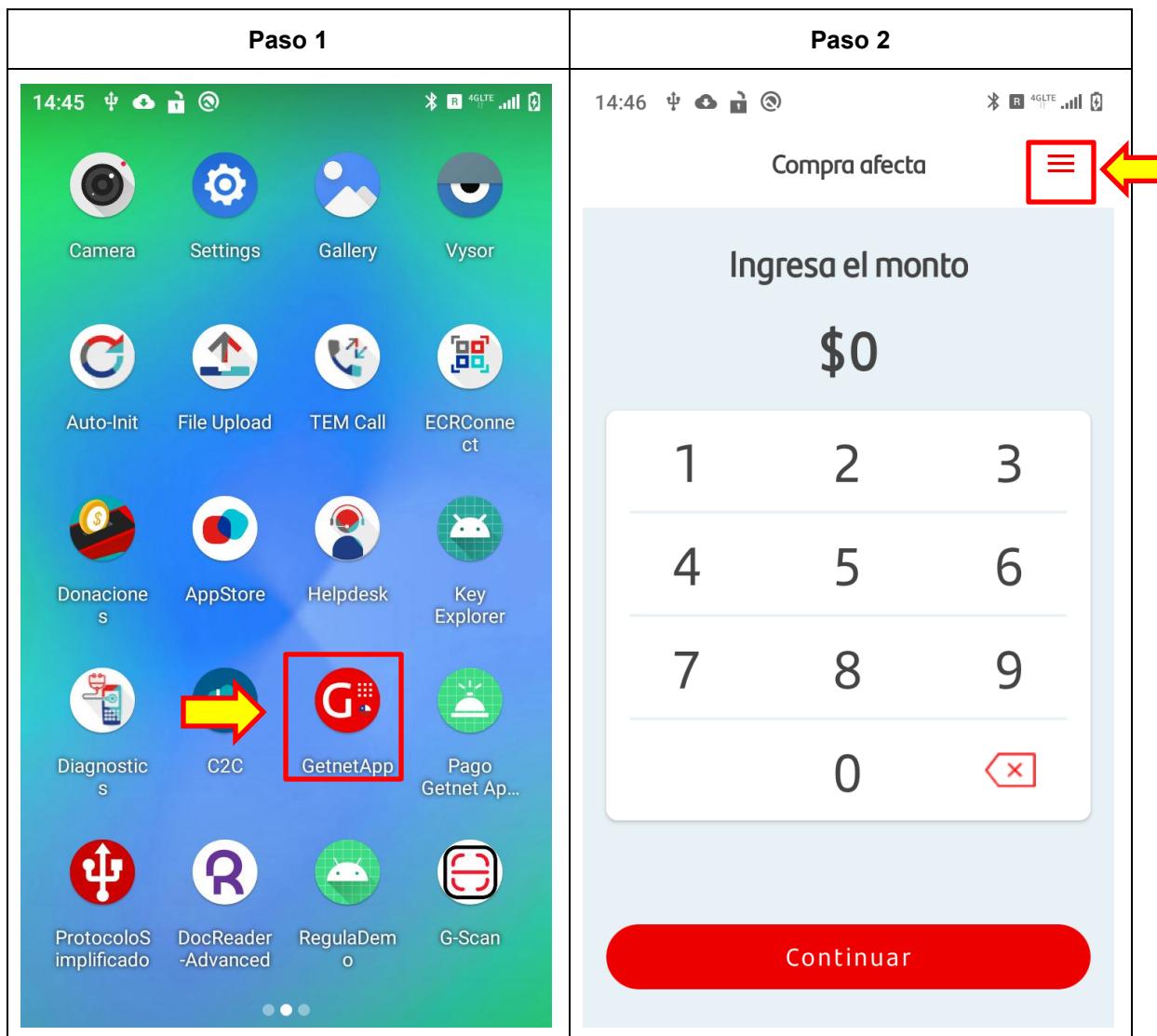
Lo que se requiere para iniciar el proceso de integración sería lo siguiente:

- Windows 10 o superior.
- Permisos de administrador de la máquina local de desarrollo.
- Microsoft Visual Studio Code (Para proyectos en Javascript, Node.JS y PHP)
  - Node.JS (idealmente la última versión)
  - Nodemon (opcional)
- Microsoft Visual Studio 2015 o superior (Con compatibilidad para .Net Framework 4 o superior).
- Android Studio (Para Cajas en Android nativo)
- Librería Getnet.POSIntegrado según el lenguaje a desarrollar.
- Getnet Agente POS.
- Getnet Simulador de Caja.
- Getnet Simulador de POS.
- Navegador Web (Recomendado Chrome).
- Dispositivo POS Getnet físico (no obligatorio).
- Cable USB tipo C.
- Acceso a Internet para acceder a este sitio <https://app.vysor.io/#/> (no obligatorio).

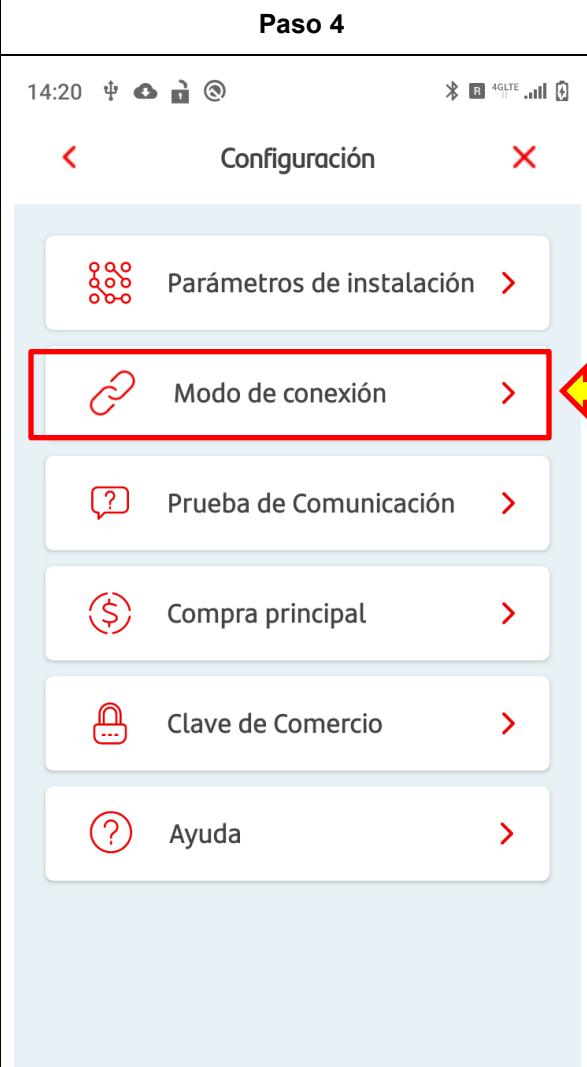


#### 4 CONFIGURACION INICIAL DEL POS

Esta configuración es indispensable para la conexión entre POS y Caja, cuenta de 8 sencillos pasos. En cada paso solo tendrá que hacer "Tap" a la pantalla en los recuadros rojos señalados.





Paso 3	Paso 4
 <p>14:19</p> <p>Getnet<sup>™</sup> By Santander</p> <p>Anulación Devolución</p> <p>Cierre de caja Reportes</p> <p>Afecta / Exenta Inicializar</p> <p>Volver a Compra</p>	 <p>14:20</p> <p>Configuración</p> <p>Parámetros de instalación</p> <p>Modo de conexión</p> <p>Prueba de Comunicación</p> <p>Compra principal</p> <p>Clave de Comercio</p> <p>Ayuda</p>



Paso 5	Paso 6
<p>14:22 ⌂ ⚡ 🔋</p> <p>Modo de conexión</p> <p>Modo POS Integrado SDK</p> <p>Modo POS Integrado Directo</p> <p>Modo Cloud to Cloud</p> <p>Modo PIN Pad</p> <p>Activar comunicación entre apps</p>	<p>14:23 ⌂ ⚡ 4GLTE 🔋</p> <p>Modo Integrado SDK</p> <p>Conexión via USB</p> <p>Conexión via Wifi</p>



Paso 7	Paso 8
<p>14:24 ☰ ⚡ 4G LTE ☰</p> <p>X</p> <p>ingresa clave de comercio</p> <p>.....</p> <p>1 2 3</p> <p>4 5 6</p> <p>7 8 9</p> <p>0</p> <p>Cancelar Aceptar</p> 	<p>14:24 ☰ ⚡ 4G LTE ☰</p> <p>Modo Integrado SDK →</p> <p>✓</p> <p>SDK</p> <p>Modo de conexión integrado SDK</p>



## 5 ENTENDIMIENTO DE LOS COMPONENTES

Previo a iniciar la integración vamos a entender cada componente para realizar una prueba inicial, de manera que tengamos acceso a la comunicación por serial COM en el PC con lo cual podamos realizar pruebas sin necesidad de conectar el POS Getnet físico todavía.

### 5.1 HABILITAR PUERTOS COM

Necesitamos habilitar los puertos COM en el PC. Para esto podemos ocupar una herramienta de virtualización que nos permita hacer esta prueba. Existen algunos programas que nos permiten realizar esta tarea. Recomendamos usar:

- HDD Virtual Serial Port Tools (Free Version, Non-Commercial Use Only)
- Virtual Serial Port Driver by Eltima Software (Esta permite el uso por 14 días)

Ambas aplicaciones nos permitirán crear puertos serial COM para realizar la integración. Puedes acceder a la siguiente URL <https://freevirtualserialports.com/> y descargar la versión según el tipo de sistema.

The screenshot shows the homepage of freevirtualserialports.com. At the top, there's a navigation bar with links like 'Home', 'Products & Services', and 'Contact Us'. Below the header, there's a large green banner with the title 'Create & Manage Virtual Serial Ports' and a subtext 'Create virtual serial ports and connect them using virtual null-modem cables, named pipes & more...'. To the left of the banner is an illustration of a laptop displaying two windows related to serial ports. To the right, there's a section titled 'Free Virtual Serial Ports Advantages' with five bullet points: RELIABLE, FLEXIBLE, COMPATIBLE, TRANSPARENT, and FREE. Each point has a brief description next to it. At the bottom of the banner, there are three download buttons: 'Download x86/x64', 'Download ARM64', and 'Download (ZIP)'. A 'GlobalSign' SSL certificate seal is visible in the bottom right corner of the page.

Para este ejemplo seleccionamos el botón **Download x86/x64**. Procedemos a instalar la aplicación tal como se haría con cualquier otro programa similar.



HHD Software Virtual Serial Port Tools Setup Pac... — □ X



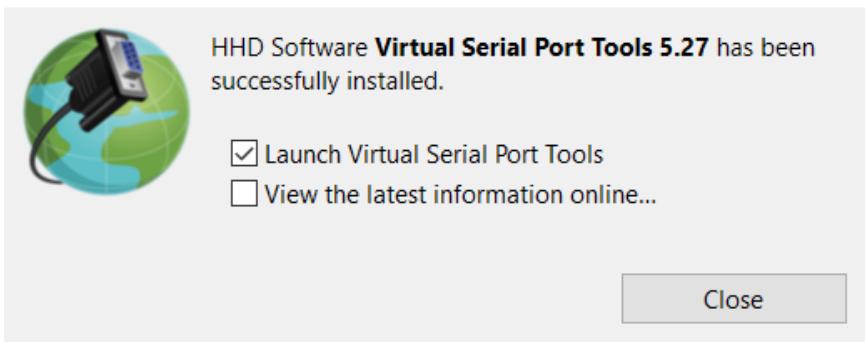
Seleccionamos **Install**.

HHD Software Virtual Serial Port Tools Setup Pac... — □ X

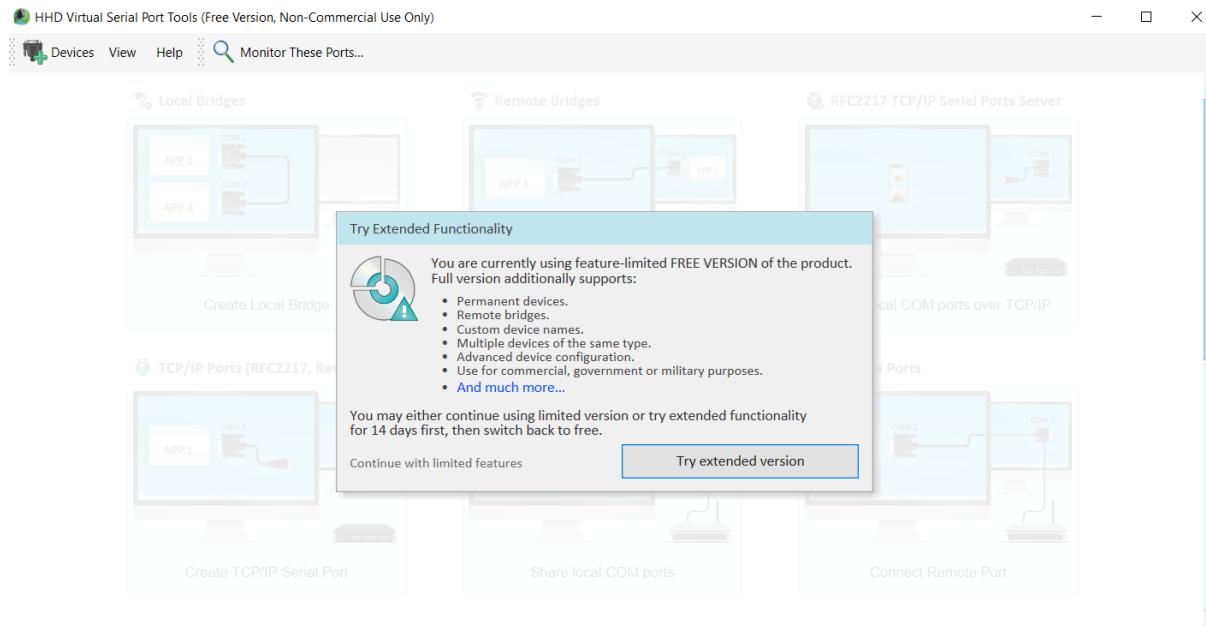


Luego esperamos que se instale la aplicación.

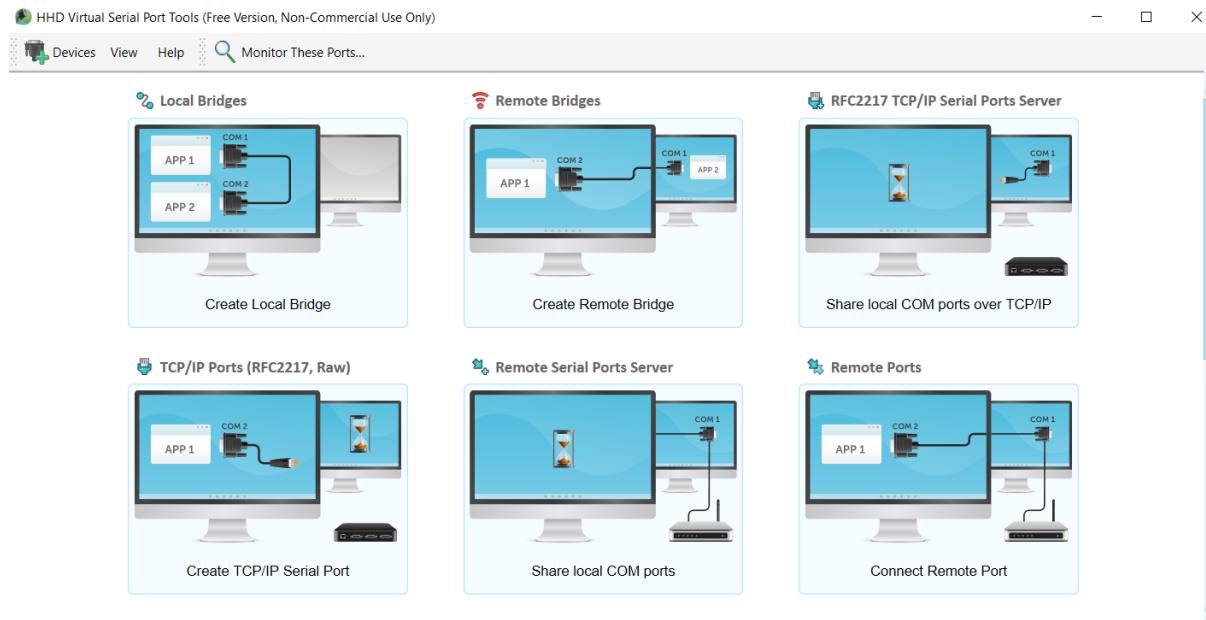
HHD Software Virtual Serial Port Tools Setup Pac... — □ X



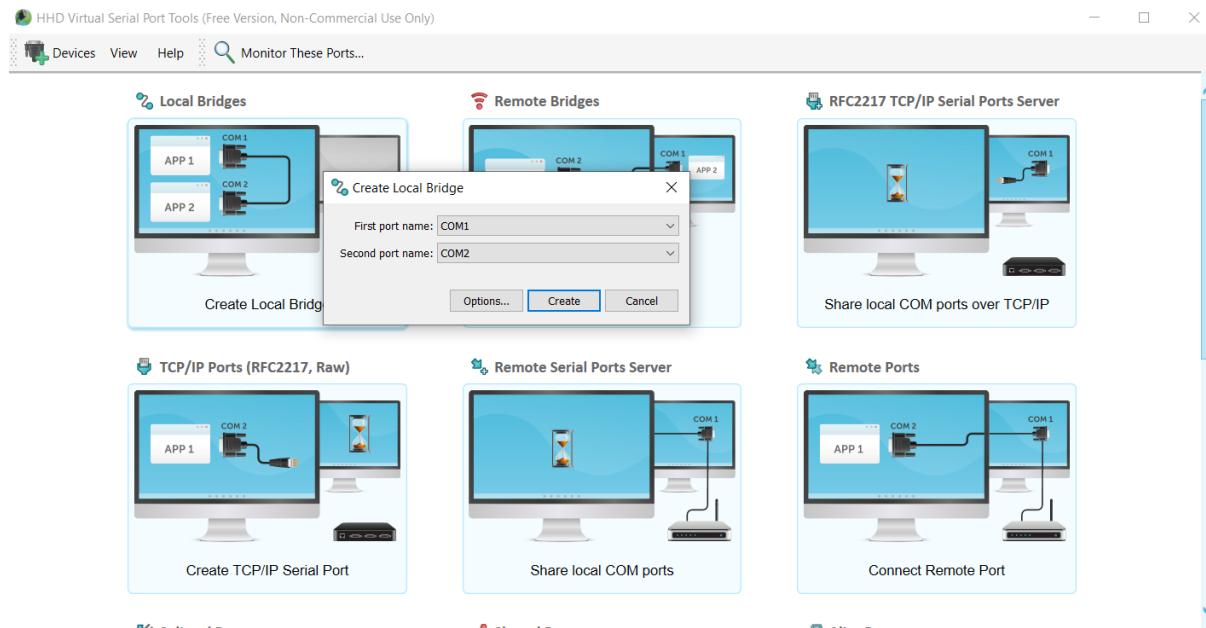
Quitamos el segundo ticket y presionamos **Close**.



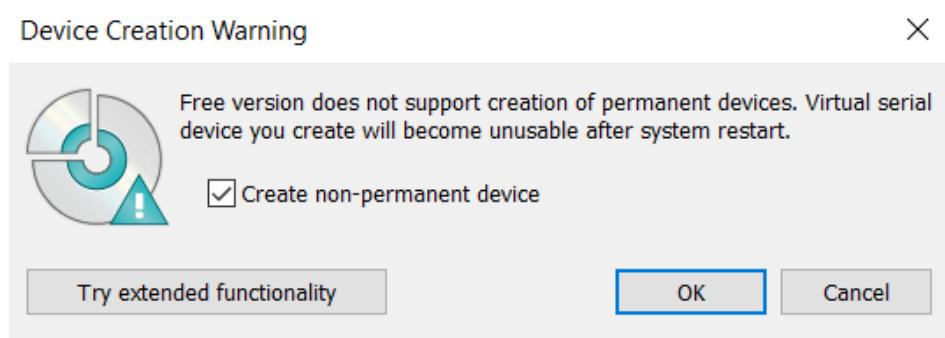
Al abrir la aplicación por primera vez nos aparece este recuadro y debemos seleccionar la opción que dice **Continue with limited features**. A la izquierda abajo.



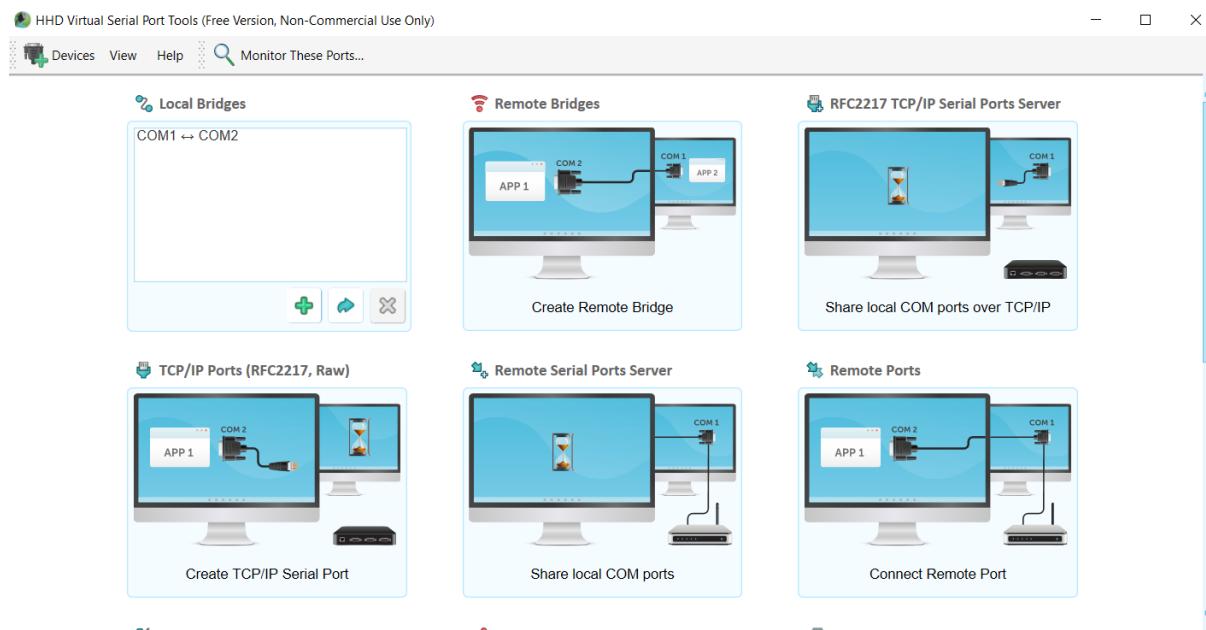
Ya tenemos lista la aplicación. En nuestro caso solo utilizaremos la primera opción arriba a la izquierda que dice **Local Bridges**. Seleccionamos **Create Local Bridge**.



Seleccionamos **Create**.



Seleccionamos el ticket **Create non-permanent device** y luego el botón **OK**.

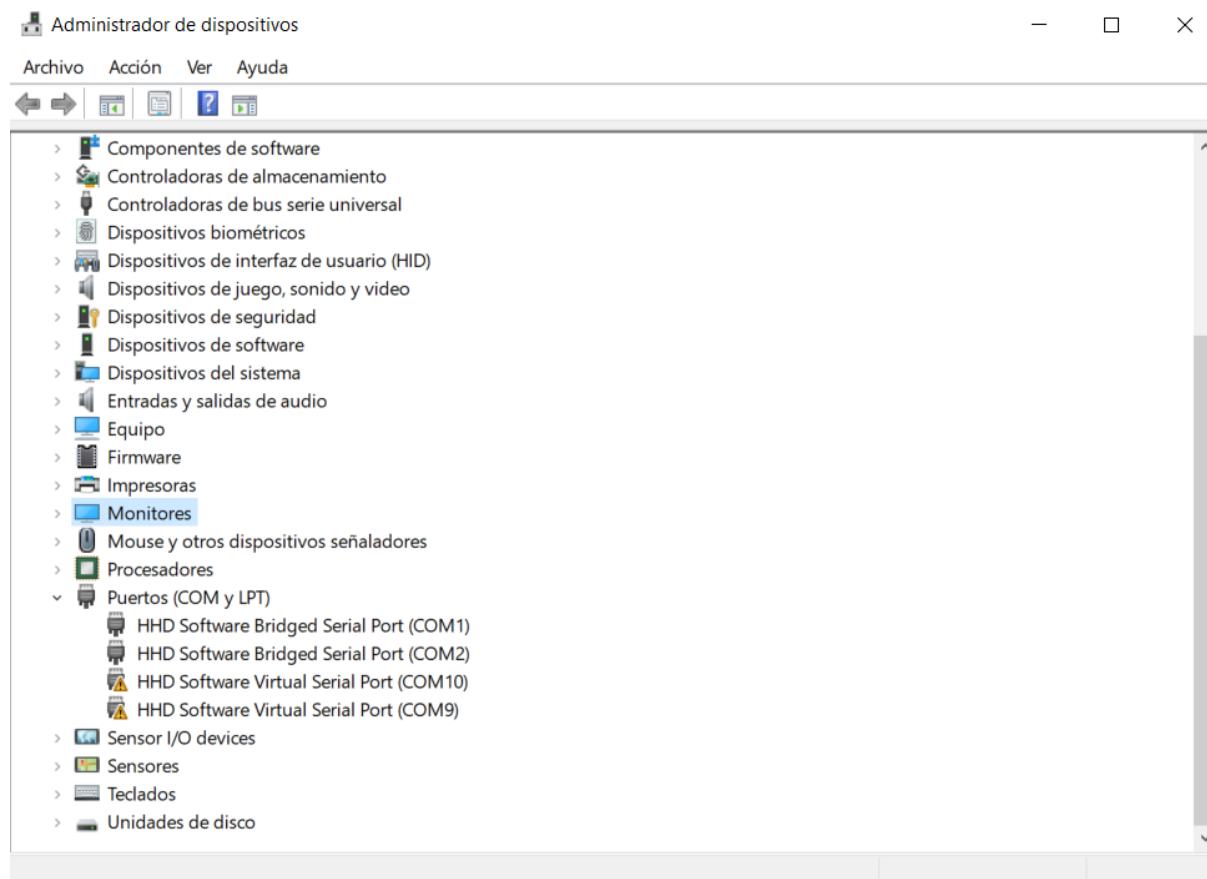


Y con esto ya quedaría lista la configuración.



## 5.2 VALIDAR LA CONFIGURACIÓN DE PUERTOS COM

Si queremos validar que funcionó, podemos abrir el administrador de dispositivos de Windows para confirmar la creación de los puertos serial.



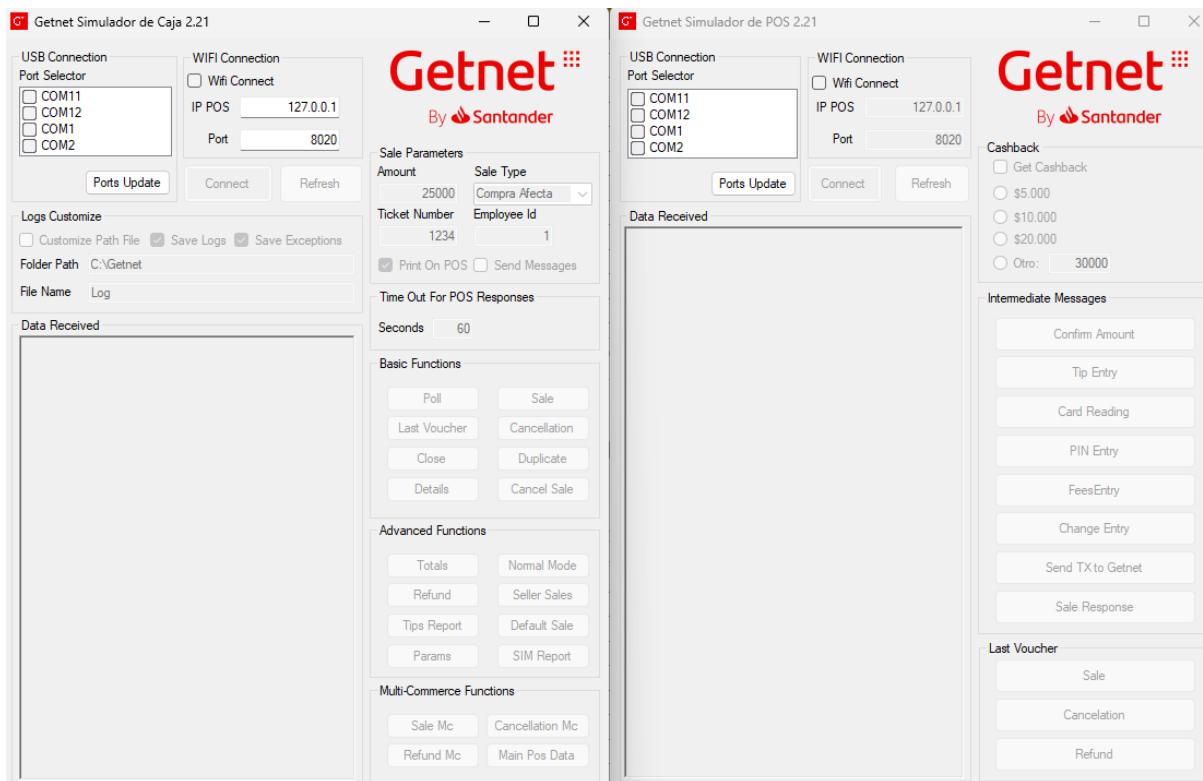
Acá podemos ver que se crearon el COM1 y el COM2.

**NOTA:** En el ejemplo aparecen otros 2 adicionales (COM9 y COM10) con un signo de exclamación. Son solo pruebas de nuestro ambiente de desarrollo. No considerar.



### 5.3 PRUEBA DE LOS SIMULADORES

Ahora podemos abrir los simuladores tanto de caja como de POS para realizar pruebas y entender el funcionamiento de la integración y como hacer las pruebas previas antes de siquiera iniciar los desarrollos.



Al abrir los simuladores tenemos 2 maneras de probar la comunicación:

- Por cable USB seleccionando los puertos Serial COM disponibles (Uno para cada simulador).
- O por Wifi pulsando el check que dice **Wifi Connect**.

Ambas opciones nos permiten probar la comunicación de los Simuladores y entender cómo podemos probar lo que vamos a integrar, pero en nuestro caso lo haremos por **USB Connection**. Entonces, seleccionamos COM1 para el Simulador de Caja (a la izquierda) y COM2 para el simulador de POS a la derecha y seleccionamos el botón **Connect**.

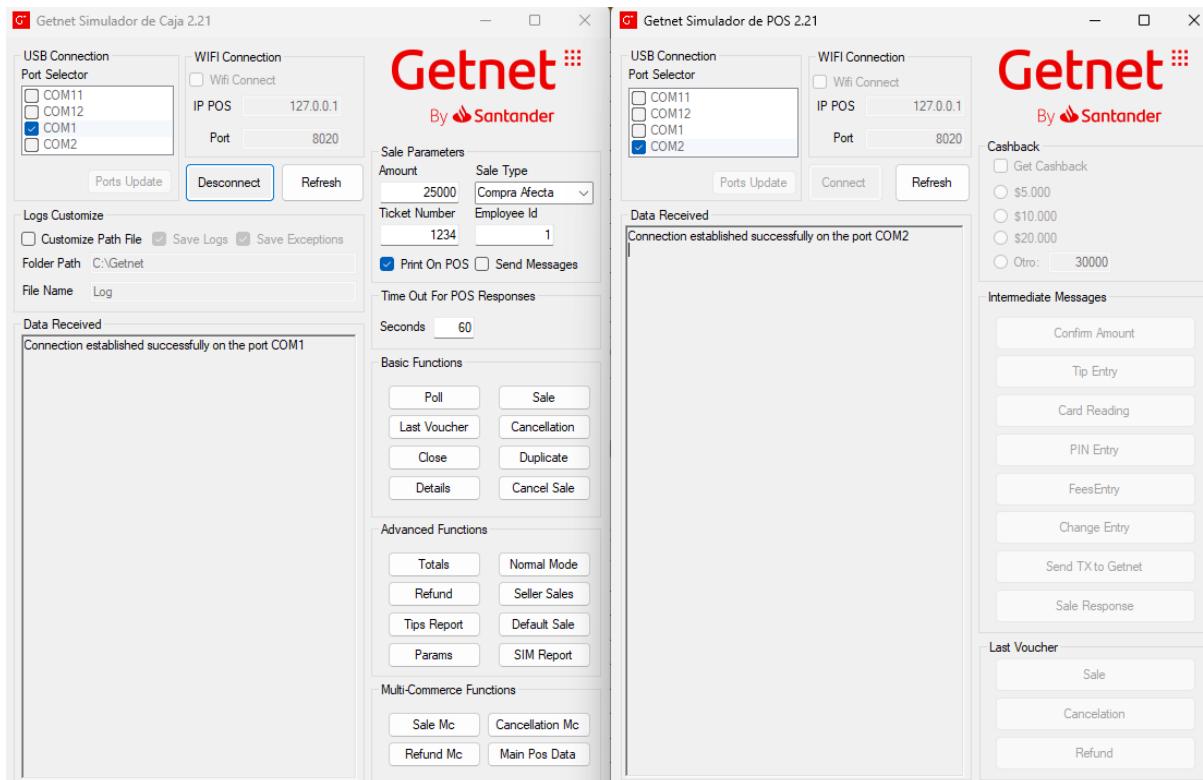
**NOTA:** Si quisiéramos conectarnos por Wifi, la comunicación opera de la misma forma que por USB pero el que hace de Servidor es el Simulador de POS por lo tanto primero debemos conectar este por Wifi y luego el Simulador de Caja (que hace de Cliente). Si te equivocas recibirás un mensaje de error en el Simulador de Caja.

Para el ejemplo se usa la IP de la tarjeta de red local 127.0.0.1 y el puerto 8020, pero al momento de hacer pruebas con el POS se debe usar la IP local del PC y el puerto 8020 (dado que es el puerto usado por el POS para la comunicación).



En el POS hay que cambiar la configuración para Wifi (seguir los pasos antes indicados en el punto 4, pero en el Modo de conexión seleccionar Wifi en el paso 6).

Ahora ambos simuladores se habilitan (según corresponda) para poder realizar las pruebas. Dentro del recuadro **Data Received** obtenemos un mensaje **Connection established successfully on the port COM1 o COM2** respectivamente.



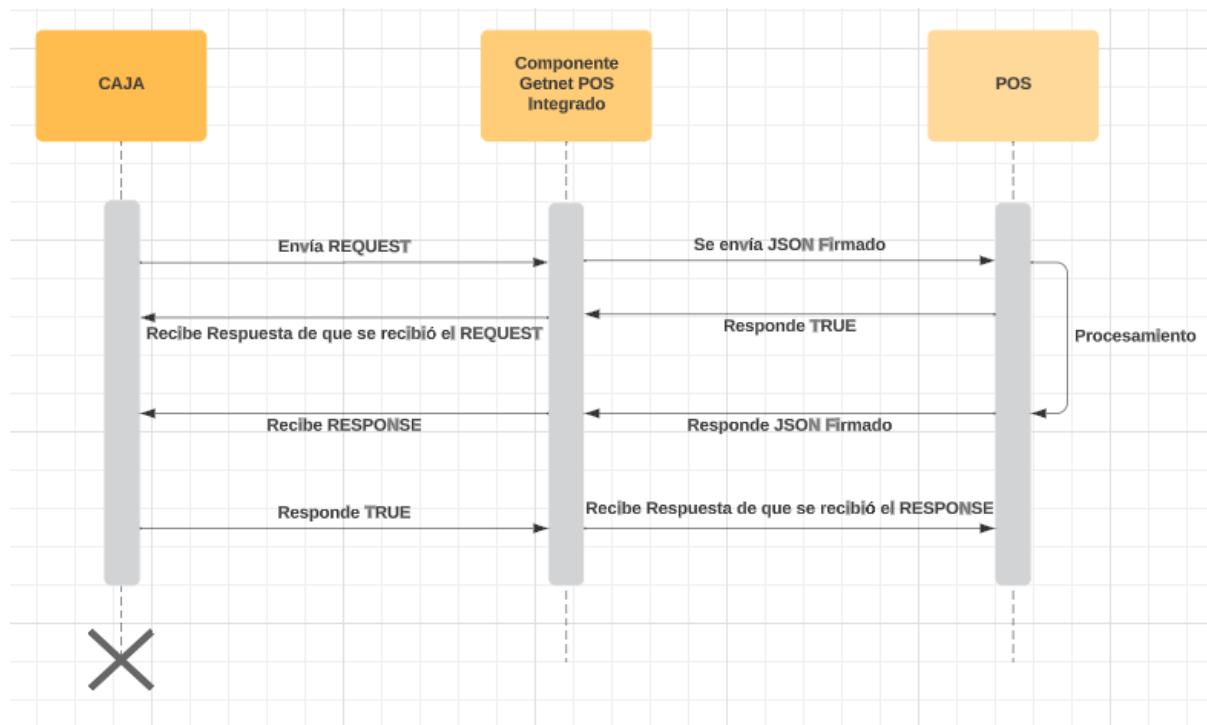
Esto significa que ya podemos transmitir datos por el puerto serial o lo que para nosotros sería el cable USB. A pesar de que solo es una Simulación, todo nos permitirá que funcione tal como se requiere para la integración con el POS Getnet.



## 5.4 ENTENDER EL FLUJO DE LA COMUNICACIÓN

Ahora estamos listos para iniciar nuestras pruebas.

- a. El simulador de Caja es una muestra fiel de lo que tenemos que integrar en la Caja.
- b. Contamos con 16 comandos disponibles para realizar acciones en el POS.
- c. Cada comando permite enviarle al POS un **REQUEST** en formato JSON a lo que el POS responderá con un TRUE si recibió la solicitud.
- d. Luego de procesar el **REQUEST** el POS responderá con un **RESPONSE** según el comando enviado con un JSON del lado de la caja.
- e. Una vez que la caja recibe el **RESPONSE**, también le responderá al POS con un TRUE indicándole que recibió la respuesta esperada y con esto se finaliza la comunicación.



En este diagrama podemos validar el flujo de comunicación. De manera que del lado del software de Caja solo bastará con llamar un método disponible en la componente Getnet POS Integrado y esta es la encargada de transformar los datos a un JSON, firmar el mensaje, convertirlo en bytes y enviarlo por el canal de comunicación de la manera que lo requiere el POS Getnet.



## 5.5 COMANDOS A ENVIAR AL POS

En el simulador de Caja podemos ver una serie de botones con los nombres de cada acción posible para enviarle un comando al POS. Los comandos que se pueden enviar son los siguientes:

1. **POLLING**: Permite validar la conexión entre el POS y la Caja.
2. **SALE**: Permite enviar una solicitud de venta para solicitar el pago con tarjeta de débito o crédito.
3. **LAST VOUCHER**: Le solicita al POS la última transacción realizada, la cual puede ser una venta (**SALE**) una anulación (**CANCELLATION**) o una devolución (**REFUND**).
4. **CANCELLATION**: Permite hacer una anulación de la venta siempre y cuando no se haya enviado un **CLOSE** para hacer el cierre de las ventas diarias. Esto suele hacerse durante un día normal ya que la copia de la operación aun esta almacenada en el POS.
5. **CLOSE**: Permite hacer un cierre de las operaciones en el POS. Se recomienda hacer esta acción al cerrar la caja en un proceso diario.
6. **TOTAL**: Le solicita al POS el total de las ventas realizadas.
7. **DETAILS**: Le solicita al POS el detalle de las ventas realizadas separadas en un arreglo de **SALES**.
8. **NORMAL MODE**: Permite que el POS pueda usarse de manera normal sin conexión con la Caja. Un posible caso de uso puede ser cuando la Caja tenga un problema y es necesario seguir operando con el POS para recibir pagos.
9. **REFUND**: Permite hacer una anulación de la venta una vez que ya se ha hecho un **CLOSE** en el POS. Esto suele hacerse en días posteriores en los que un cliente solicita anular una venta, pero ésta ya no está almacenada en el POS sino que hay que consultarla al autorizador.
10. **DUPLICATE**: Solicitud al POS una copia de una transacción específica.
11. **SELLER SALES**: Solicitud el total de ventas por vendedor.
12. **TIPS REPORT**: Solicitud el detalle de las propinas por vendedor.
13. **DEFAULT SALE**: Establece el tipo de venta por defecto.

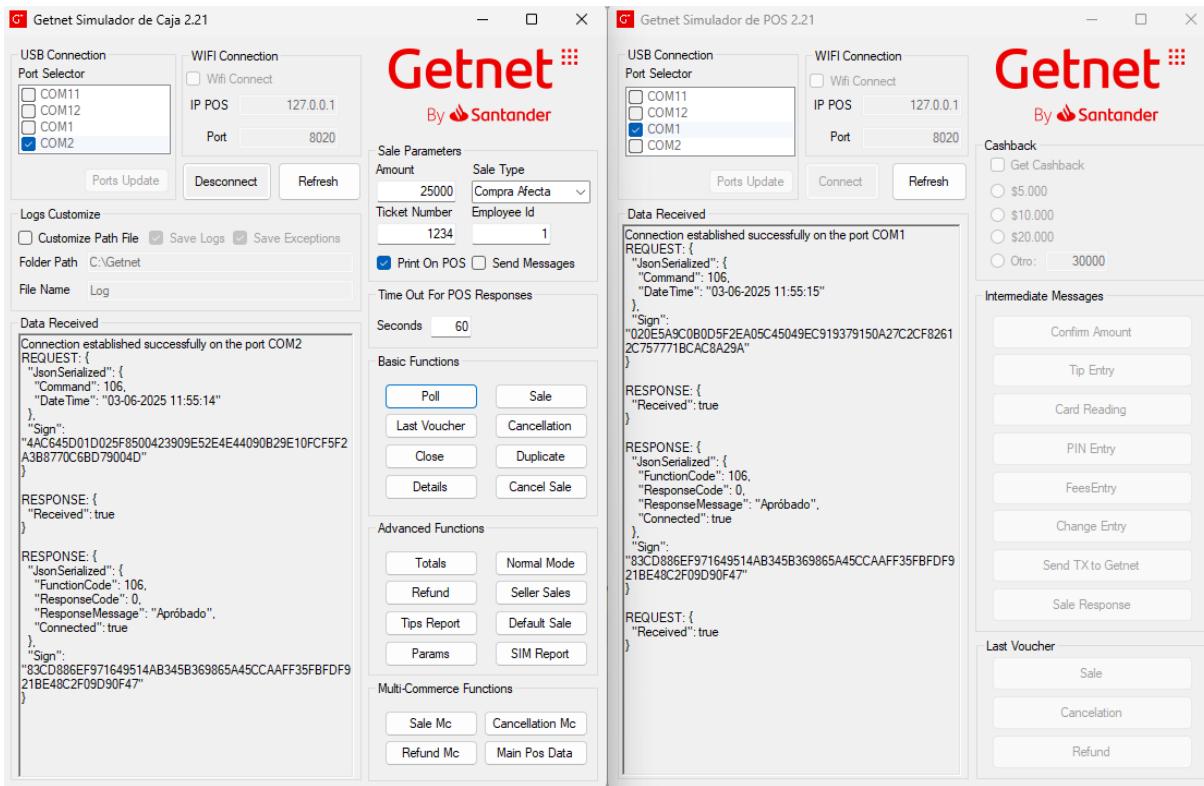
Comando SaleType	Tipo
0	Compra
1	Compra Afecta
2	Factura Afecta
3	Compra Exenta
4	Factura Exenta
5	Recaudación Afecta
6	Recaudación Exenta

14. **PARAMS**: Solicitud al POS los parámetros del dispositivo.
15. **SIM REPORT**: Solicitud al POS los parámetros del chip telefónico.
16. **CANCEL SALE**: Permite anular una venta en proceso desde la Caja. Siempre y cuando el usuario final no haya ingresado el PIN y enviado la operación al autorizador.



## 5.6 PROBAR LA COMUNICACIÓN

El primer comando a probar en el Simulador de Caja para validar la comunicación entre ambos simuladores es el de **POLL**. En ese momento se valida todo lo que se mostró anteriormente en el flujo de comunicación (punto 4.4) y se puede ver el **REQUEST** y **RESPONSE** respectivos además de los TRUE tras cada envío que se mostrarán en el recuadro **Data Received** en cada simulador respectivamente.



El simulador de POS está creado para que responda con JSON de ejemplo de forma automatizada para casi todos los comandos. A Excepción de los comandos **SALE** y **LAST VOUCHER** que requieren una interacción del usuario del lado del Simulador de POS.



## 5.7 ENTENDIMIENTO DE LAS SOLICITUDES

Todas las solicitudes **REQUEST** desde el Simulador de Caja tienen una estructura que se compone de 2 nodos:

- **JsonSerialized:** Que contiene la data propia de la solicitud (en formato JSON) y se explica en detalle en la Estructura de la Mensajería (punto 5 en adelante).
- **Sign:** Firma del mensaje. El cual permite dar una integridad del mensaje para validar que no pueda ser modificado en tránsito.

### **REQUEST:**

```
{"JsonSerialized":"{\\\"Command\\\":106,\\\"DateTime\\\":\\\"8/11/2023  
11:40:42\\\"}","Sign":"CDF49E3DA16B1F4024667067A0122026CF733B111E8A53C019E33E3FAB6E3  
09E"}
```

En este caso se está enviando un comando **POLL** para validar la conexión entre el POS y la Caja.



## 5.8 ENTENDIMIENTO DE LAS RESPUESTAS

Todos los **RESPONSES** desde el Simulador de POS tienen una estructura definida en la Estructura de la Mensajería (punto 5 en adelante). Cada respuesta tiene 3 nodos que siempre se repiten y el resto cambia según el comando solicitado.

Siempre recibiremos los siguientes nodos:

- **FunctionCode:** Es el número de comando a enviar al POS. Esto está definido en el anexo 9.1 Comandos a enviar al POS.
- **ResponseCode:** Es un número que representa un tipo de error, definido en el anexo 9.2 Códigos de respuestas.
- **ResponseMessage:** Es una glosa en palabras que representa el significado del error.

### **RESPONSE:**

```
{"JsonSerialized":"{\"FunctionCode\":106,\"ResponseCode\":0,\"ResponseMessage\":\"Apróbad\"o\"," "Connected":true}","Sign":"83CD886EF971649514AB345B369865A45CCAFF35FBFDF921BE48C2F09D90F47"}
```

En este caso se está recibiendo una estructura para un comando **POLL** para validar la conexión entre el POS y la Caja.

- El **ResponseCode** es igual a 0 lo que significa que no hay errores en la solicitud.
- El **ResponseMessage** dice Aprobado.
- Y lo que nos interesa en este caso es el último nodo que dice **Connected** = true.

**NOTA:** De aquí en adelante ya podemos interiorizarnos mejor con la comunicación consultando la Estructura de la Mensajería (punto 8 en adelante).



## 5.9 LOGS

La componente Getnet POS Integrado, genera un log automatizado tras el envío de cada comando entre los Simuladores. Toda esta información queda almacenada en una carpeta llamada **logs** en la raíz del proyecto donde se encuentra el ejecutable de los Simuladores.

Nombre	Fecha de modificación	Tipo	Tamaño
logs	6/11/2023 15:09	Carpeta de archivos	
Getnet.POSIntegrado.dll	8/11/2023 11:39	Extensión de la ap...	58 KB
Getnet.POSIntegrado.pdb	8/11/2023 11:39	Archivo PDB	240 KB
GetnetSimuladorDeCaja.exe	8/11/2023 11:39	Aplicación	33 KB
GetnetSimuladorDeCaja.exe.config	12/4/2023 12:08	XML Configuration...	1 KB
GetnetSimuladorDeCaja.pdb	8/11/2023 11:39	Archivo PDB	48 KB
GetnetSimuladorDeCaja.vhost.exe	8/11/2023 11:37	Aplicación	23 KB
GetnetSimuladorDeCaja.vhost.exe.config	12/4/2023 12:08	XML Configuration...	1 KB
GetnetUsbSimuladorDeCaja.exe	8/11/2023 11:35	Aplicación	33 KB
GetnetUsbSimuladorDeCaja.exe.config	12/4/2023 12:08	XML Configuration...	1 KB
GetnetUsbSimuladorDeCaja.pdb	8/11/2023 11:35	Archivo PDB	48 KB
GetnetUsbSimuladorDeCaja.vhost.exe.c...	12/4/2023 12:08	XML Configuration...	1 KB
GetnetUsbSimuladorDeCaja.vhost.exe.m...	7/12/2019 06:10	Archivo MANIFEST	1 KB

Ejemplo del Simulador de Caja

Nombre	Fecha de modificación	Tipo	Tamaño
2023-5	23/5/2023 17:32	Carpeta de archivos	
2023-6	28/6/2023 21:13	Carpeta de archivos	
2023-7	29/7/2023 23:45	Carpeta de archivos	
2023-9	11/9/2023 11:24	Carpeta de archivos	
2023-10	26/10/2023 09:28	Carpeta de archivos	
2023-11	8/11/2023 11:03	Carpeta de archivos	

Ejemplo del contenido de la carpeta logs. Se almacena separado por mes y año.



Nombre	Fecha de modificación	Tipo	Tamaño
Logs-11-05-2023.txt	11/5/2023 12:21	Documento de tex...	21 KB
Logs-12-05-2023.txt	12/5/2023 18:41	Documento de tex...	3 KB
Logs-18-05-2023.txt	18/5/2023 20:25	Documento de tex...	14 KB
Logs-23-05-2023.txt	23/5/2023 17:32	Documento de tex...	1 KB

Ejemplo del contenido de los logs. Se almacenan por día.

The screenshot shows a Windows Notepad window titled "Logs-08-11-2023.txt: Bloc de notas". The content of the log file is as follows:

```
|-- 8/11/2023 12:49:44 ---  
REQUEST: {"JsonSerialized": "{\"Command\":106, \"DateTime\":\"8/11/2023  
12:49:43\"}", "Sign":"991A98797FAD5E194F8AAC5A8F10A6893C35EAC35BAF58A040E3C5934CCC5554"}  
---  
--- 8/11/2023 12:49:44 ---  
RESPONSE: {"Received":true}  
---  
--- 8/11/2023 12:49:44 ---  
REQUEST: {"JsonSerialized": "{\"FunctionCode\":106, \"ResponseCode\":0, \"ResponseMessage\":\"  
\\\"Apr\u00f3bado\\\", \\\"Connected  
\\\" :true\"}, \"Sign\":\"83CD886EF971649514AB345B369865A45CCAFF35FBFDF921BE48C2F09D90F47\"}  
---
```

At the bottom of the Notepad window, there are status bar settings: Línea 1, columna 1 | 100% | Windows (CRLF) | UTF-8

Contenido del archivo de Logs. Se guarda siempre lo mismo que se muestra en el Simulador en la sección de **Data Received**.

**NOTA:** Desde la nueva versión 1.18 del Simulador de Caja, también está la opción de enviar la ruta de la carpeta donde se quiere almacenar el Log y el nombre personalizado del archivo (no es necesario agregar la extensión del archivo). En el proyecto de ejemplo viene cómo hacer esta configuración.

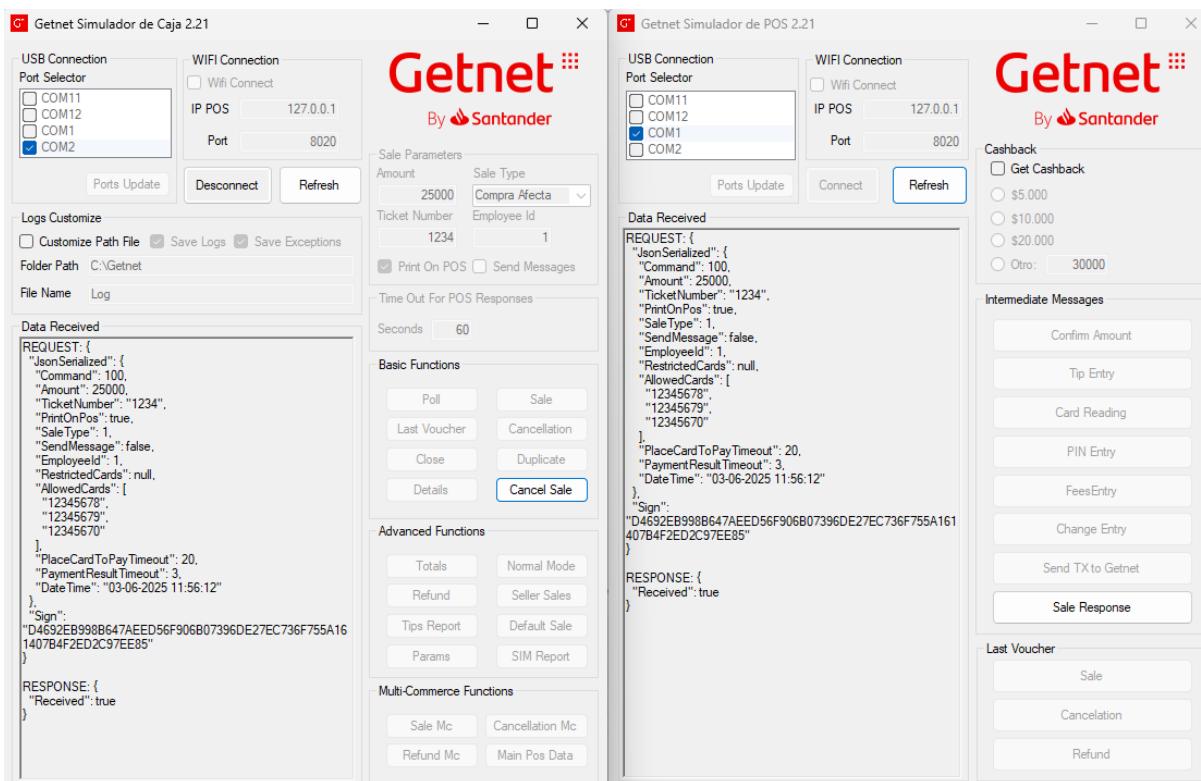


## 5.10 PARAMETROS DE VENTA

En el simulador de Caja hay una sección (arriba a la derecha) que contiene una serie de parámetros que son los mismos que solicita el método de integración (en código). Básicamente son los siguientes:

- f. **AMOUNT:** Monto de la venta.
- g. **TICKET NUMBER:** Número de voucher o boleta.
- h. **SALE TYPE:** Tipo de venta como: compra, compra afecta, compra exenta, etc.
- i. **EMPLOYEE ID:** Número de identificador para el vendedor.
- j. **PRINT ON POS:** Enviar el resultado para que lo imprima el POS.
- k. **SEND MESSAGES:** Enviar mensajes intermedios. Esto quiere decir que el POS le dirá a la Caja el estado del proceso de la venta como, por ejemplo: Lectura de tarjetas, confirmación de importe, selección de cuotas, entrada de PIN, enviando transacción a Getnet, entrada de propina, entrada de vuelto.
- l. **SECONDS:** Se refiere al tiempo en segundos para esperar a que el POS responda. Se consideran 2 timers. El de la conexión (que espera recibir TRUE) y el de la respuesta que espera al recibir el JSON final.

Todos estos parámetros se utilizan para definir una venta y luego se puede presionar el botón **SALE**.



Al presionar el botón **SALE** el Simulador de Caja deshabilita todos los componentes y solo deja habilitado el botón **CANCEL SALE** de manera que no haya posibilidad de enviar otros comandos mientras se procesa una venta.



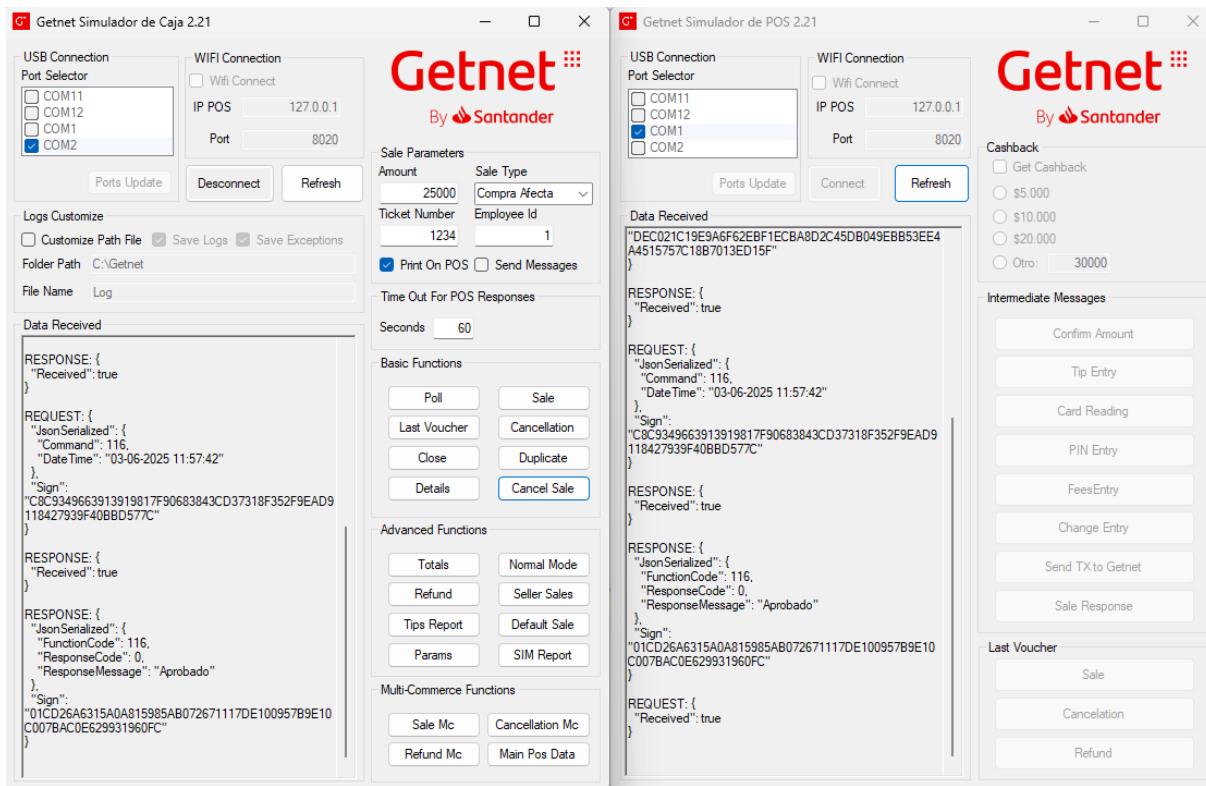
Por otro lado, en el Simulador de POS se habilita el check de **CASHBACK** (arriba a la derecha) por si el usuario requiere vuelto y se habilita el botón **SALE DATA** para responder a la venta. De esta manera se simula que un usuario final puede tomarse su tiempo al momento de realizar una venta en un POS físico.

The screenshot shows two windows side-by-side:

- Getnet Simulador de Caja 2.21**: This window is for managing ports. It shows a list of available ports (COM11, COM12, COM1, COM2) with **COM2** selected. It also displays WiFi connection details (IP POS: 127.0.0.1, Port: 8020). Buttons for **Ports Update**, **Disconnect**, and **Refresh** are present.
- Getnet Simulador de POS 2.21**: This window is for performing sales. It has a **Sale Parameters** section where **Amount** is set to 25000 and **Sale Type** is set to "Compra Afecta". Other fields include **Ticket Number** (1234), **Employee Id** (1), and checkboxes for **Print On POS** and **Send Messages**. A **Time Out For POS Responses** field is set to 60 seconds. Below these are sections for **Basic Functions** (Poll, Sale, Last Voucher, Cancellation, Close, Duplicate, Details, Cancel Sale) and **Advanced Functions** (Totals, Refund, Seller Sales, Tips Report, Default Sale, Params, SIM Report). A **Multi-Commerce Functions** section includes buttons for Sale Mc, Cancellation Mc, Refund Mc, and Main Pos Data.

The right-hand window also displays a **Data Received** panel containing a large JSON object representing the transaction details. The JSON includes fields like TerminalId, Ticket, AuthorizationCode, Amount, SharesNumber, SharesAmount, Last4Digits, OperationId, CardType, AccountingDate, AccountNumber, CardBrand, RealDate, EmployeeId, Tip, SaleType, PosMode, Cashback, TransToken, ExpiryDate, EntryMode, AID, CommerceRut, CommerceName, BranchName, BranchAddress, BranchDistrict, Bin, Sign, and various commerce-specific details.

Al presionar el botón de **SALE DATA** en el Simulador de POS se finaliza el flujo y el Simulador de Caja habilita nuevamente todos los componentes. Por otro lado, el Simulador de POS queda bloqueado para volver a su estado en espera de comandos. De forma adicional recibimos el JSON con la respuesta de la venta.



Otra opción sería presionar el botón **CANCEL SALE** en el Simulador de Caja, lo cual también finaliza el flujo, pero bajo otro escenario.

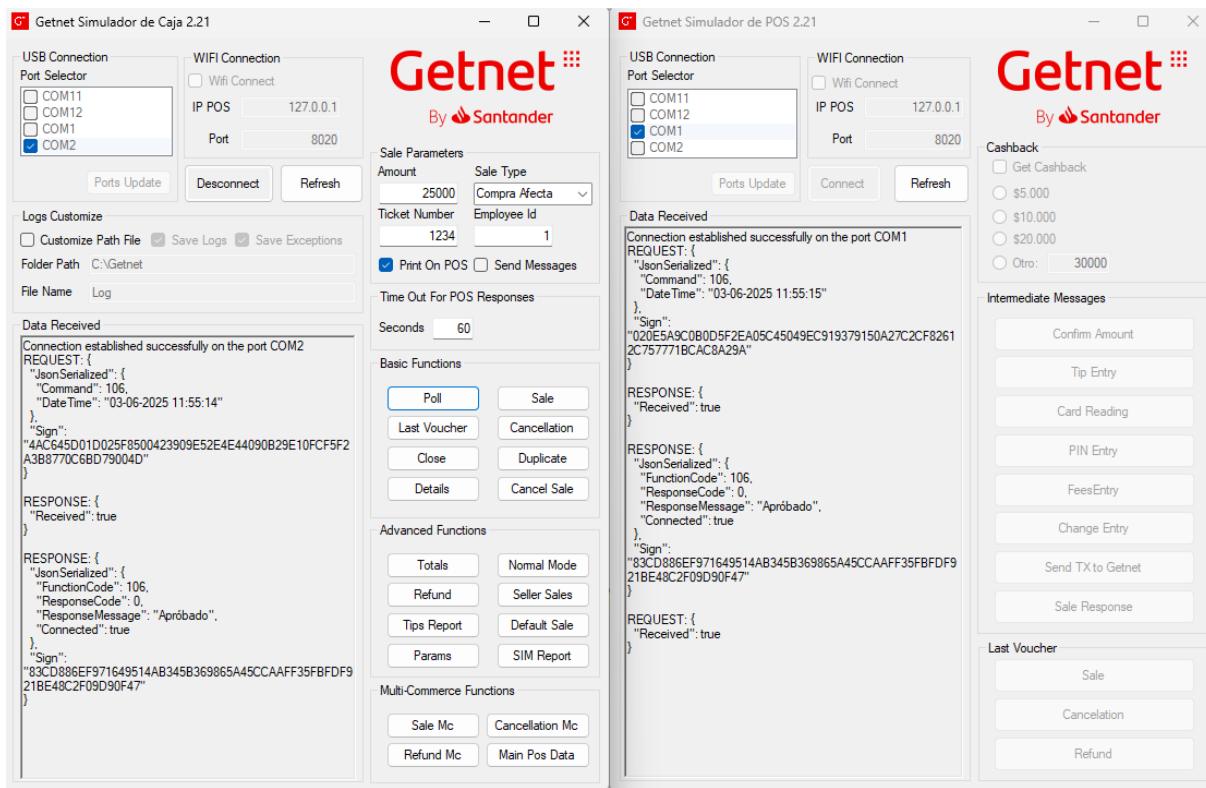


## 5.11 OTRAS RESPUESTAS POSIBLES

De forma adicional será necesario enviar una confirmación de recepción de mensajes por ambos extremos. Tanto el Simulador de Caja como el Simulador de POS deberán responder con un booleano tras la recepción de los mensajes. Si el mensaje se puede mapear, quiere decir que se recibió completo y se responderá true, de lo contrario habría que responder con un false.

Ejemplo:

```
{ "Received": true }
```



Como se puede apreciar en el recuadro **Data Received** cada aplicación responde con una confirmación luego de recibir un mensaje.

**NOTA:** Esto no es necesario desarrollarlo en el Software de Caja, solo es una muestra de como funciona la componente Getnet POS Integrado en el proceso de comunicación.

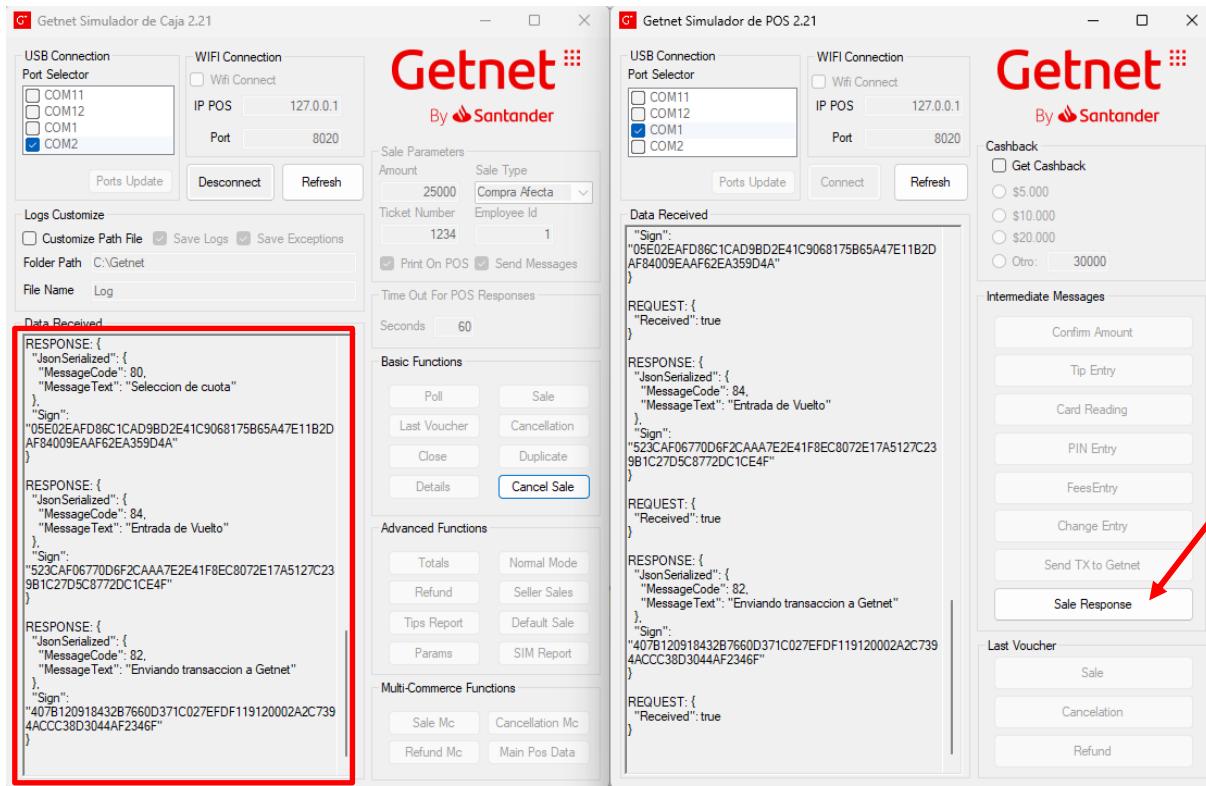


## 5.12 MENSAJES INTERMEDIOS

Para el caso del método de venta con el ticket en “Send Messages”, el Simulador de POS habilita los botones de los mensajes intermedios uno por uno y en la Caja solo se habilita el **CANCEL SALE**.

The image contains four screenshots of the Getnet Simulators, arranged in a 2x2 grid. Each screenshot shows two windows: one for the Cashier (Caja) and one for the POS (POS).

- Screenshot 1 (Top Left):** Shows the Cashier window with a red arrow pointing to the "Cancel Sale" button. The POS window shows the "Data Received" section with a JSON request for a sale.
- Screenshot 2 (Top Right):** Shows the Cashier window with a red arrow pointing to the "Confirm Amount" button. The POS window shows the "Intermediate Messages" section with a "Confirm Amount" button highlighted.
- Screenshot 3 (Bottom Left):** Shows the Cashier window with a red box highlighting the "Response" section, which contains a JSON object with a "MessageCode": 79 and "MessageText": "Confirmacion de Importe".
- Screenshot 4 (Bottom Right):** Shows the Cashier window with a red arrow pointing to the "Tip Entry" button. The POS window shows the "Intermediate Messages" section with a "Tip Entry" button highlighted.



Para finalizar el flujo solo basta con presionar los botones de la sección **INTERMEDIATE MESSAGES** en el Simulador de POS hasta llegar al botón **SALE DATA** al igual que ya lo habíamos indicado anteriormente.



## 5.13 PRUEBAS ADICIONALES

A partir de aquí, ya es posible realizar pruebas con el resto de los comandos disponibles para entender su funcionamiento, de manera que usa tu curiosidad y presiona todo, no hay manera de que algo falle, está desarrollado para pulsar todo y hacer las pruebas que estimes conveniente.

## 5.14 PRUEBAS CON EL POS GETNET

Si ya cuentas con un dispositivo físico puedes hacer lo siguiente:

- a. Abrir el Simulador de Caja
- b. Conectar el POS Getnet por USB y hacer las pruebas.

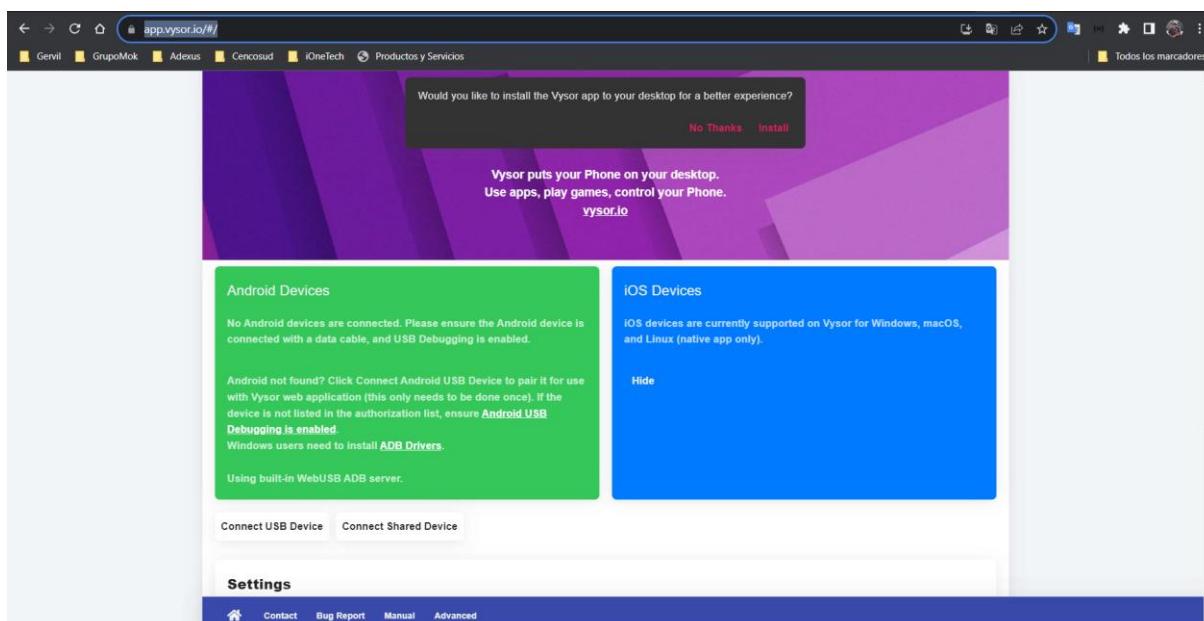
Hay que hacer notar que para el caso de realizar pruebas con el POS Getnet ya no sería necesario usar los puertos Serial COM virtualizados, dado que el POS Getnet despliega sus propios puertos y Windows le otorga un número diferente a cada uno.

**NOTA:** De todas maneras, si usted se equivoca asignando los puertos lo que recibirá será un comportamiento errático del software, dado que ambos dispositivos la Caja y el POS Getnet estarían usando el mismo puerto de comunicaciones.

## 5.15 PRUEBAS CON EL POS GETNET Y VYSOR

Esta la posibilidad de conectar el Vysor que es una alternativa para compartir lo que se ve en el POS Getnet en tu pantalla del PC para hacer pruebas en reuniones y compartir con otros.

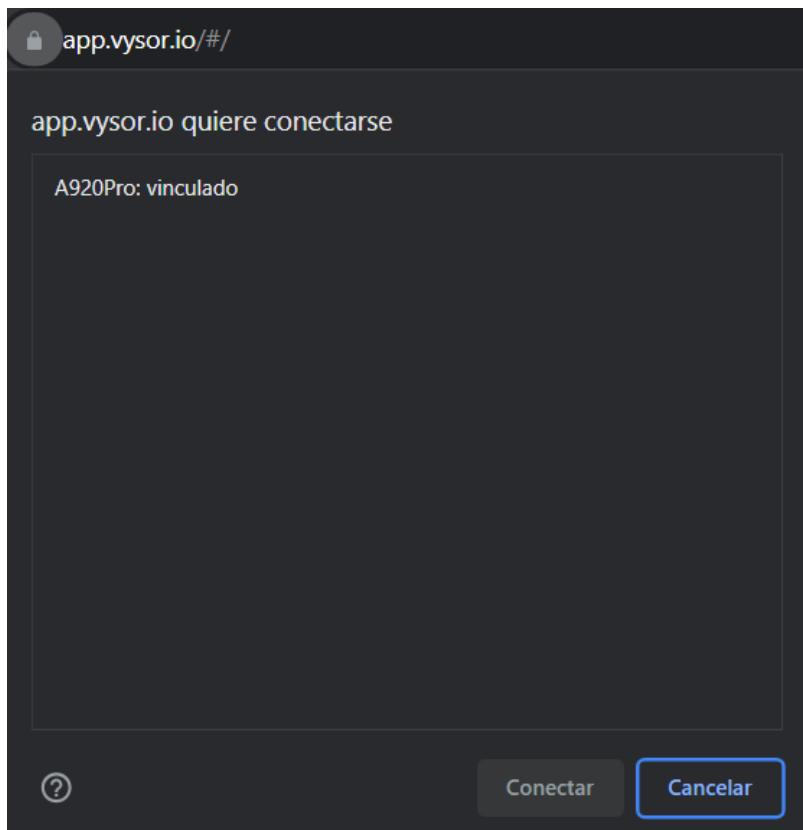
**PASO 1:** Abrir esta URL <https://app.vysor.io/#/>



Aquí puedes instalar la APP en el PC o bien seguir usando el navegador.

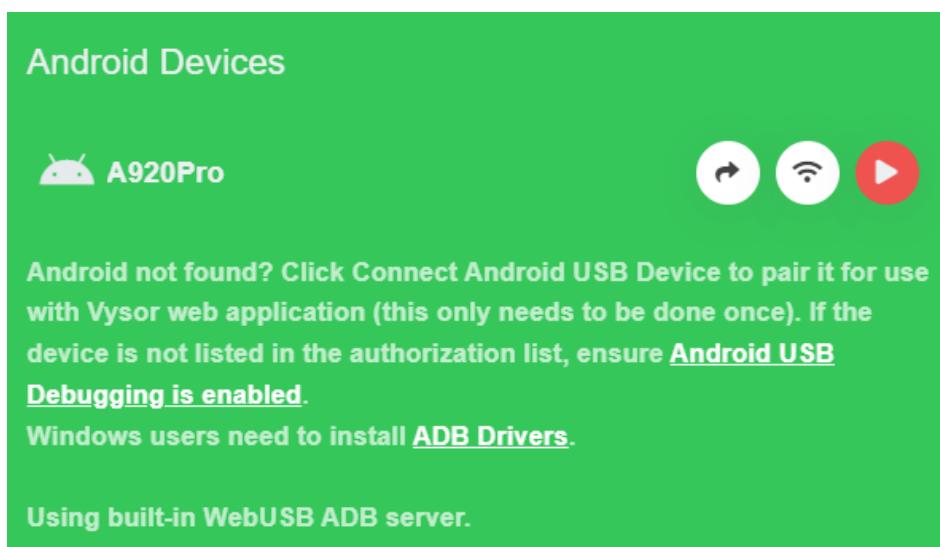


**PASO 2:** Conectar el POS Getnet al PC por USB y presionar el botón **Connect USB Device**.



Seleccionar el dispositivo y luego **Conectar**.

**PASO 3:** Se reconoce el dispositivo y ahora solo **PLAY**.





The image shows two side-by-side screenshots. On the left is the 'Getnet Simulador de Caja 2.21' application window. It displays a 'USB Connection' section with a 'Port Selector' containing 'COM11', 'COM12', 'COM1' (which is checked), and 'COM2'. Below it are 'Logs Customize' options like 'Customize Path File', 'Save Logs', and 'Save Exceptions', along with a 'Folder Path' set to 'C:\Getnet' and a 'File Name' set to 'Log'. A 'Data Received' panel shows JSON logs for a connection attempt on port COM1 and a successful response from the POS. On the right is the 'A920Pro' POS terminal screen. It features the 'Getnet' logo and 'By Santander'. The main message says 'Esperando instrucción de Caja' (Waiting for cash register instruction) with a right-pointing arrow icon. Below this is a graphic of a tablet POS terminal and a smartphone connected by a circular signal. The bottom of the screen has a blue bar with icons for back, home, and menu.

Aquí se puede apreciar que al hacer un **POLL** estamos conectados correctamente.



The image shows two screenshots side-by-side. On the left is the 'Getnet Simulador de Caja 2.21' application window. It displays a JSON request for a sale (Sale Type: Compra Afecta) with an amount of 25000, ticket number 1234, and employee ID 1. The request includes card numbers (12345678, 12345679, 12345670), a payment result timeout of 3 seconds, and a date of 03-06-2025 12:09:49. On the right is a mobile payment interface for an A920Pro device. It shows a red smartphone icon, a credit card icon, and a large '\$25.000' amount. Below this, it says 'Opera tarjeta'. At the bottom, there are logos for VISA, MasterCard, American Express, Magna, Apple Pay, Google Pay, and a note 'DEBUG only Net for COMMERCIAL'. A button at the bottom says 'Paga con QR y tu App Santander'.

En este caso se envió un **SALE** y el Simulador de Caja se comporta tal como antes con el Simulador de POS, recibimos un TRUE del POS Getnet, indicando que está a la espera de que el usuario final interactúe con el dispositivo y realice el pago, de manera que se pueden realizar las mismas pruebas, pero en este caso obtendríamos respuestas más realistas.



Como se puede apreciar en el Simulador de Caja obtenemos la respuesta y el POS vuelve al estado de espera de solicitudes. Hay varios casos posibles para finalizar la venta:

- Realizar el pago con tarjeta de crédito y/o débito.
- Cancelar la venta desde el POS Getnet.
- Cancelar la venta desde el Simulador de Caja.
- No hacer nada y esperar que pase el tiempo. El POS Getnet anulará la venta de forma automática.

En todos los casos, siempre recibiremos la respuesta esperada.

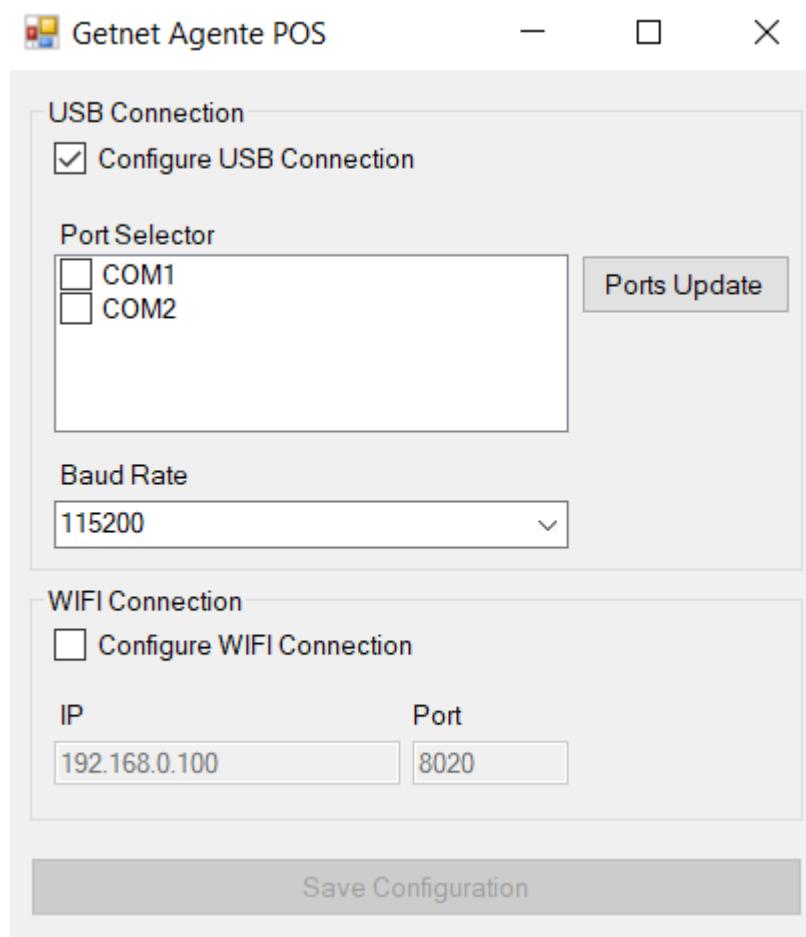


## 5.16 AGENTE POS

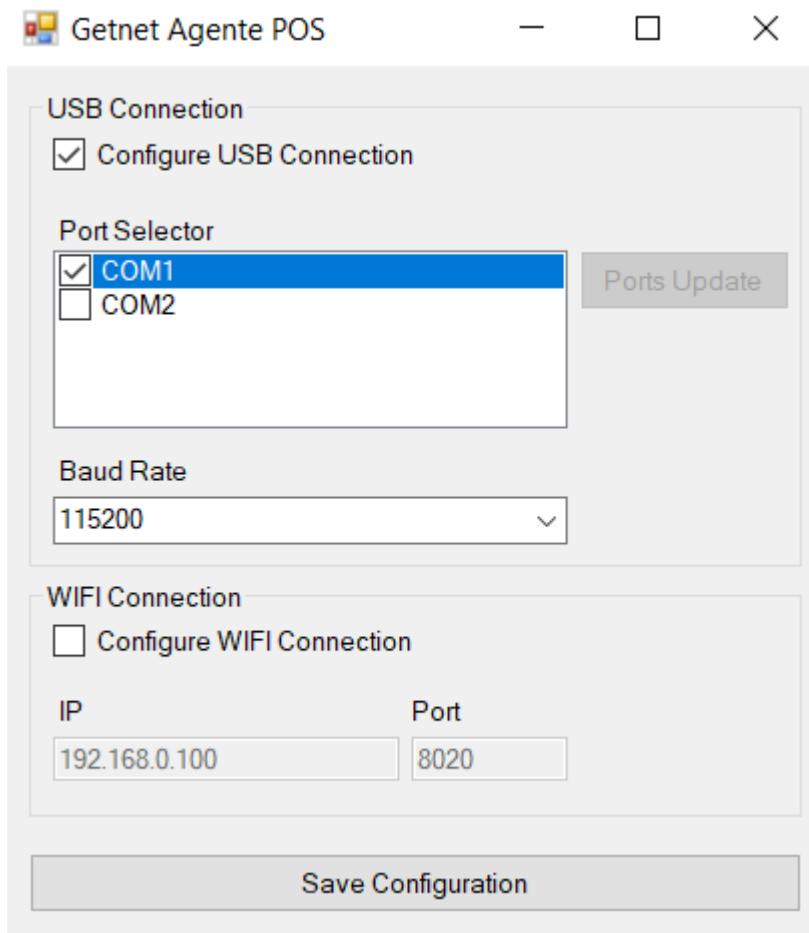
Esta aplicación cumple una misión super importante, dado que permite configurar el puerto serial por el que se configurará el software de Caja en **PRODUCCIÓN**. Si bien se puede establecer el puerto por código, debemos recordar que es Windows quien asigna los nombres de los puertos y esto es aleatorio, por lo que no podemos saber a priori cual será el nombre del puerto en desarrollo y en producción.

Con esta aplicación podemos definir la configuración predeterminada para el software de Caja y así la componente de Getnet POS Integrado lo revisará internamente para tomar esa configuración y comunicarse con el POS Getnet sin problemas.

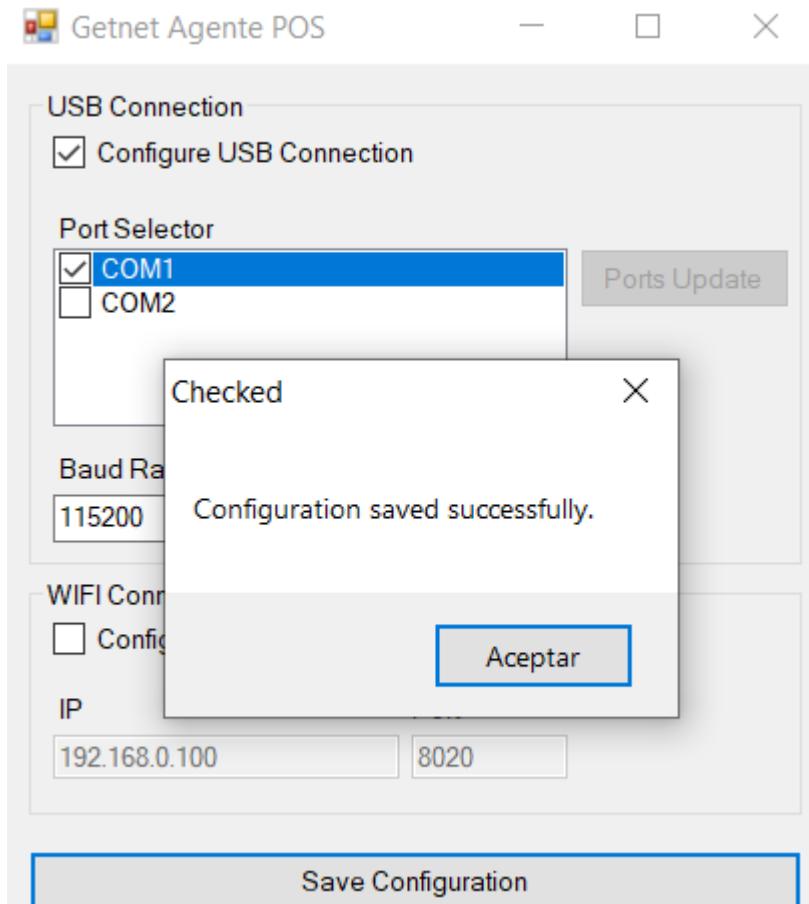
**NOTA:** Esta aplicación requiere de permisos de administrador del equipo local. Una vez configurado se guarda un archivo en la siguiente ruta del PC: C:\Program Files\Getnet\pos.config



Al abrir la aplicación se debe seleccionar el tipo de conexión entre USB y WIFI. Para este caso lo haremos por USB.



Al seleccionar el puerto serial COM1, se habilita el botón **Save Configuration**.



Al presionar el botón **Save Configuration** se despliega una alerta indicando que se guardó la configuración de forma exitosa. Y con esto estamos listos para correr en producción. Ya se puede cerrar la aplicación y dar por finalizado el paso de configuración de la Caja.



## 6 ESTRUCTURA DE LA MENSAJERIA

A continuación, se describe la estructura propuesta para cada uno de los métodos necesarios para dicha integración.

**NOTA:** Para todos los **RESPONSES NULL** o vacíos desde el POS las fechas y horas por defecto serán las del momento en que se devuelve la respuesta.

**NOTA:** Se usará el formato de codificación de caracteres **UTF8** para todos los mensajes.

### 6.1 VENTA

Comando enviado por el PDV para solicitarle al POS la ejecución de una venta. Los siguientes parámetros deben ser enviados en la solicitud:

#### Request:

Nombre Parámetro	Tipo	Longitud	Descripción
Command*	int	-	Comando a enviar para solicitar una acción en el POS.
Amount	long	-	Valor en pesos para realizar la transacción.
TicketNumber	string	24	Este dato lo genera el PDV en el comprobante que se genera para la venta. Equivale al NB (Número de Boleta)
PrintOnPos	bool	-	Verdadero: solicita la impresión en papel en el POS. Falso: No solicita la impresión al POS. Ambas operaciones envían una respuesta al PDV. <b>[campo opcional]</b>
SaleType	int	-	Define el tipo de venta en caso de ser distinta a la predeterminada. <b>[campo opcional]</b>
SendMessage	bool	-	Permite enviar mensajes intermedios a la Caja.
EmployeeId	int	-	Número que representa al vendedor. <b>[campo opcional]</b>
SecondsTimeOut	int	-	Cantidad de segundos de timeout para esperar por la respuesta del POS. <b>[campo opcional]</b> <b>[no es requerido en el request que va al POS]</b>
RestrictedCards	Bines	-	Los bines indicados NO pueden realizar compras. Los otros se deben aceptar.
AllowedCards	Bines	-	Los bines indicados SI pueden realizar compras. Los otros se deben rechazar.
PlaceCardToPayTimeout	int	-	Cantidad de segundos de espera para la pantalla de Opere Tarjeta. <b>[campo opcional]</b>
PaymentResultTimeout	int	-	Cantidad de segundos de espera para la pantalla de Resultado de la Venta, sea aprobada o rechazada. <b>[campo opcional]</b>



DateTime	string	-	Fecha actual en el momento de enviar el comando. Se usa para que la firma del mensaje sea siempre diferente.
----------	--------	---	--

(\*) La data en gris corresponde a parámetros que se envía de forma interna en la API, no se solicitan en el método expuesto al desarrollador del PDV.

Comando SaleType	Tipo
1	Compra Afecta
2	Factura Afecta
3	Compra Exenta
4	Factura Exenta
5	Recaudación Afecta
6	Recaudación Exenta

### 6.1.1 BINES

Modelo de datos para los restrictedCards y allowedCards de SaleMC.

Nombre Parámetro	Tipo	Longitud	Descripción
RestrictedCards	String[]	8	Arreglo de bines numéricos.
AllowedCards	String[]	8	Arreglo de bines numéricos.

#### Request Object:

```
{
    "Command": 100,
    "Amount": 15000,
    "TicketNumber": "1234567890123456789012345",
    "PrintOnPos": false,
    "SaleType": 1,
    "SendMessage": false,
    "EmployeeId": 1,
    "RestrictedCards": ["12345678", "87654321", "23456789"],
    "AllowedCards": ["12345678", "87654321", "23456789"],
    "PlaceCardToPayTimeout": 5,
    "PaymentResultTimeout": 2,
```



```

    "DateTime": "2023-12-28 22:35:12"
}

```

## Mensajes Intermedios

Los mensajes intermedios enviados por el POS Getnet y que deben ser visualizados por la Caja deben corresponder según los siguientes códigos:

- 78: Lectura de tarjetas.
- 79: Confirmación de Importe.
- 80: Selección de cuota.
- 81: Entrada de PIN.
- 82: Enviando transacción a Getnet.
- 83: Entrada de Propina.
- 84: Entrada de Vuelto.

## Response Object:

```

{
    "MessageCode": 78,
    "MessageText": "Lectura de tarjeta"
}

```

## Response:

Nombre Nodo	Tipo	Longitud	Descripción
FunctionCode	int	-	Tipo de función del POS.
ResponseCode	int	-	Código de la respuesta. ( <a href="#">tabla de códigos de respuesta en los Anexos</a> )
ResponseMessage	string	30	Mensaje de texto que representa la respuesta de la operación puede ser Aprobado o Rechazado.
CommerceCode	long	-	Código único de sucursal, aprobado por Getnet para generar transacciones.
TerminalId	string	20	Código del terminal POS.
Ticket	string	24	Este dato lo genera el PDV en el comprobante que se genera para la venta (número de boleta). [campo opcional]
AuthorizationCode	string	20	Código de autorización de la transacción.
Amount	long	-	Valor en pesos de la venta a transaccionar.
SharesNumber	Int	-	Numero de cuotas.
SharesAmount	Int	-	Valor de cada cuota <b>[campo opcional]</b>



Last4Digits	string	4	Últimos cuatro dígitos de la tarjeta del cliente.
OperationId	int	-	Corresponde al número de comprobante que genera el POS.
CardType	string	2	Tipo de Tarjeta.
AccountingDate	string	-	Fecha y hora de la transacción.
AccountNumber	string	30	Número de cuenta. <b>[campo opcional]</b>
CardBrand	string	2	Marca de la tarjeta utilizada en la transacción.
RealDate	string		Fecha actual.
EmployeeId	int	-	Número que representa al vendedor. <b>[campo opcional]</b>
Tip	long	-	Propina de la venta. <b>[campo opcional]</b>
SaleType	int	-	Dato que identifica el tipo de venta afecta/exenta
PosMode	int	1	Permite conocer si la venta se realizó en el Modo Pos Integrado o Pos Normal. 0 = no integrado   1 = integrado
Cashback	long	-	Valor del vuelto solicitado.
TransToken	string	30	Dato de largo máximo que corresponde al TID + ID Sucursal + OperationID + yyyyMMddhhmmss
ExpiryDate	string	4	Corresponde al formato numérico de la fecha de expiración de la tarjeta YYMM.
EntryMode	string	2	Corresponde a uno de los valores de la tabla 10.4 de los anexos.
AID	string	-	Es el Application Identifier para EMV.
CommerceRut	string	10	Número único para el pago de impuestos - Ejemplo: 12345678-9.
CommerceName	string	25	Nombre del Comercio.
BranchName	string	25	Nombre de la Sucursal.
BranchAddress	string	40	Dirección de la Sucursal.
BranchDistrict	string	25	Comuna de la Sucursal.
Bin	string	8	Número BIN de la tarjeta que realizó la compra.
PaymentId	string	-	Identificación única de la transacción SEP.

**Response Object:**

```
{  
    "FunctionCode": 100,  
    "ResponseCode": 0,  
    "ResponseMessage": "Aprobado",  
    "CommerceCode": 550062700310,  
    "TerminalId": "ABC1234C",  
    "Ticket": "123456789012345678901234",  
    "AuthorizationCode": "XZ123456",  
    "Amount": 15000,  
    "SharesNumber": 3,  
    "SharesAmount": 5000,  
    "Last4Digits": 6677,  
    "OperationId": 60,  
    "CardType": "CR",  
    "AccountNumber": "300000000000",  
    "CardBrand": "AX",  
    "RealDate": "2023-12-28 22:35:12",  
    "EmployeeId": 1,  
    "Tip": 1500,  
    "SaleType": 1,  
    "PosMode": 1,  
    "Cashback": 1000,  
    "TransToken": "8000001231060620240923213405",  
    "ExpiryDate": "0828",  
    "EntryMode": "23",  
    "Aid": "315041592E5359532E4444463031",  
    "CommerceRut": "12345678-9",  
    "CommerceName": "Getnet SPA",  
    "BranchName": "Work Café Agustinas",  
    "BranchAddress": "Agustinas 920",  
    "BranchDistrict": "Santiago",  
    "Bin": "12345678",  
    "PaymentId": "2c341d28-491b-4cf8-aec7-eeb60136b7a5"  
}
```



## 6.2 ÚLTIMO COMPROBANTE

Comando enviado por el PDV para solicitarle al POS el resultado de la última venta transacción realizada que pueden ser tres: Una Venta, una Anulación de Venta y una Transacción de Devolución.

### Request:

Nombre Parámetro	Tipo	Longitud	Descripción
Command*	int	-	Comando a enviar para solicitar una acción en el POS.
PrintOnPos	bool	-	Verdadero: solicita la impresión en papel en el POS. Falso: No solicita la impresión al POS. Ambas operaciones envían una respuesta al PDV. <b>[campo opcional]</b>
SecondsTimeOut	int	-	Cantidad de segundos de timeout para esperar por la respuesta del POS. <b>[campo opcional]</b> <b>[no es requerido en el request que va al POS]</b>
DateTime	string	-	Fecha actual en el momento de enviar el comando. Se usa para que la firma del mensaje sea siempre diferente.

(\*) La data en gris corresponde a parámetros que se envía de forma interna en la API, no se solicitan en el método expuesto al desarrollador del PDV.

### Request Object:

```
{
    "Command": 101,
    "PrintOnPos": false,
    "DateTime": "2023-12-28 22:35:12"
}
```

### Response 1: Sale

Nombre Nodo	Tipo	Longitud	Descripción
FunctionCode	int	-	Tipo de función del POS.
ResponseCode	int	-	Código de la respuesta. ( <b>tabla de códigos de respuesta en los Anexos</b> )
ResponseMessage	string	30	Mensaje de texto que representa la respuesta de la operación puede ser Aprobado o Rechazado.
CommerceCode	long	-	Código único de sucursal, aprobado por Getnet para generar transacciones.
TerminalId	string	20	Código del terminal POS.



Ticket	string	24	Este dato lo genera el PDV en el comprobante que se genera para la venta (número de boleta). [campo opcional]
AuthorizationCode	string	20	Código de autorización de la transacción.
Amount	long	-	Valor en pesos de la venta a transaccionar.
SharesNumber	Int	-	Numero de cuotas.
SharesAmount	Int	-	Valor de cada cuota <b>[campo opcional]</b>
Last4Digits	string	4	Últimos cuatro dígitos de la tarjeta del cliente.
OperationId	int	-	Corresponde al número de comprobante que genera el POS.
CardType	string	2	Tipo de Tarjeta.
AccountingDate	string	-	Fecha y hora de la transacción.
AccountNumber	string	30	Número de cuenta. <b>[campo opcional]</b>
CardBrand	string	2	Marca de la tarjeta utilizada en la transacción.
RealDate	string		Fecha actual.
EmployeeId	int	-	Número que representa al vendedor. <b>[campo opcional]</b>
Tip	long	-	Propina de la venta. <b>[campo opcional]</b>
SaleType	int	-	Dato que identifica el tipo de venta afecta/exenta
PosMode	int	1	Permite conocer si la venta se realizó en el Modo Pos Integrado o Pos Normal. 0 = no integrado   1 = integrado
Cashback	long	-	Valor del vuelto solicitado.
TransToken	string	30	Dato de largo máximo que corresponde al TID + ID Sucursal + OperationID + yyyyMMddhhmmss
ExpiryDate	string	4	Corresponde al formato numérico de la fecha de expiración de la tarjeta YYMM.
EntryMode	string	2	Corresponde a uno de los valores de la tabla 10.4 de los anexos.
AID	string	-	Es el Application Identifier para EMV.
CommerceRut	string	10	Número único para el pago de impuestos - Ejemplo: 12345678-9.
CommerceName	string	25	Nombre del Comercio.
BranchName	string	25	Nombre de la Sucursal.



BranchAddress	string	40	Dirección de la Sucursal.
BranchDistrict	string	25	Comuna de la Sucursal.
Bin	string	8	Número BIN de la tarjeta que realizó la compra.
PaymentId	string	-	Identificación única de la transacción SEP.

**Response Object 1: Sale**

```
{  
    "FunctionCode": 100,  
    "ResponseCode": 0,  
    "ResponseMessage": "Aprobado",  
    "CommerceCode": 550062700310,  
    "TerminalId": "ABC1234C",  
    "Ticket": "1234567890123456789012344",  
    "AuthorizationCode": "XZ123456",  
    "Amount": 15000,  
    "SharesNumber": 3,  
    "SharesAmount": 5000,  
    "Last4Digits": 6677,  
    "OperationId": 60,  
    "CardType": "CR",  
    "AccountingDate": "2023-12-28 22:35:12",  
    "AccountNumber": "300000000000",  
    "CardBrand": "AX",  
    "RealDate": "2023-12-28 22:35:12",  
    "EmployeeId": 1,  
    "Tip": 1500,  
    "SaleType": 1,  
    "PosMode": 1,  
    "Cashback": 1000,  
    "TransToken": "8000001231060620240923213405",  
    "ExpiryDate": "0828",  
    "EntryMode": "23",  
    "Aid": "315041592E5359532E4444463031",  
    "CommerceRut": "12345678-9",  
    "CommerceName": "Getnet SPA",  
    "BranchName": "Work Café Agustinas",  
    "BranchAddress": "Agustinas 920",  
    "BranchDistrict": "Santiago",  
    "Bin": "12345678",  
}
```



```

    "PaymentId": "2c341d28-491b-4cf8-aec7-eeb60136b7a5"
}

```

### Response 2: Refund

Nombre Nodo	Tipo	Longitud	Descripción
FunctionCode	int	-	Tipo de función del POS.
ResponseCode	int	-	Código de la respuesta.
ResponseMessage	string	30	Mensaje de texto que representa la respuesta de la operación puede ser Aprobado o Rechazado.
CommerceCode	long	-	Código único del comercio, aprobado por Getnet para generar transacciones.
TerminalId	string	20	Código del terminal POS.
AuthorizationCode	string	20	Código de autorización de la transacción.
OperationId	int	-	Número de la operación.
Success	bool	-	Resultado de la operación.
TransToken	string	30	Dato de largo máximo que corresponde al TID + ID Sucursal + OperationID + yyyyMMddhhmmss
ExpiryDate	string	4	Corresponde al formato numérico de la fecha de expiración de la tarjeta YYMM.
IssuerId	string	2	Corresponde a uno de los valores de la tabla 10.5 de los anexos.
Last4Digits	string	4	Últimos cuatro dígitos de la tarjeta del cliente.
CommerceRut	string	10	Número único para el pago de impuestos - Ejemplo: 12345678-9.
CommerceName	string	25	Nombre del Comercio.
BranchName	string	25	Nombre de la Sucursal.
BranchAddress	string	40	Dirección de la Sucursal.
BranchDistrict	string	25	Comuna de la Sucursal.

### Response Object 2: Refund

```

{
    "FunctionCode": 102,
    "ResponseCode": 0,
    "ResponseMessage": "Aprobado",
}

```



```
"CommerceCode": 597029414300,  
"TerminalId": "ABCD1234",  
"AuthorizationCode": "ABCD1234",  
"OperationID": 123456,  
"Success": true,  
"TransToken": "8000001231060620240923213405",  
"ExpiryDate": "0828",  
"IssuerId": "01",  
"Last4Digits": "1234",  
"CommerceRut": "12345678-9",  
"CommerceName": "Getnet SPA",  
"BranchName": "Work Café Agustinas",  
"BranchAddress": "Agustinas 920",  
"BranchDistrict": "Santiago"  
}
```

### Response 3: Return

Nombre Nodo	Tipo	Longitud	Descripción
FunctionCode	int	-	Tipo de función del POS.
ResponseCode	int	-	Código de la respuesta.
ResponseMessage	string	30	Mensaje de texto que representa la respuesta de la operación puede ser Aprobado o Rechazado.
CommerceCode	long	-	Código único del comercio, aprobado por Getnet para generar transacciones.
TerminalId	string	20	Código del terminal POS.
AuthorizationCode	string	20	Código de autorización de la transacción.
OperationId	int	-	Número de la operación.
Success	bool	-	Resultado de la operación.
DateTime	string	-	Fecha y hora de la transacción.
TransToken	string	30	Dato de largo máximo que corresponde al TID + ID Sucursal + OperationID + yyyyMMddhhmmss
ExpiryDate	string	4	Corresponde al formato numérico de la fecha de expiración de la tarjeta YYMM.
IssuerId	string	2	Corresponde a uno de los valores de la tabla 10.5 de los anexos.
Last4Digits	string	4	Últimos cuatro dígitos de la tarjeta del cliente.



CommerceRut	string	10	Número único para el pago de impuestos - Ejemplo: 12345678-9.
CommerceName	string	25	Nombre del Comercio.
BranchName	string	25	Nombre de la Sucursal.
BranchAddress	string	40	Dirección de la Sucursal.
BranchDistrict	string	25	Comuna de la Sucursal.

**Response Object 3: Return**

```
{  
    "FunctionCode": 108,  
    "ResponseCode": 0,  
    "ResponseMessage": "Aprobado",  
    "CommerceCode": 597029414300,  
    "TerminalId": "ABCD1234",  
    "AuthorizationCode": "ABCD1234",  
    "OperationID": 123456,  
    "Success": true,  
    "DateTime": "2023-12-28 22:35:12",  
    "TransToken": "8000001231060620240923213405",  
    "ExpiryDate": "0828",  
    "IssuerId": "01",  
    "Last4Digits": "1234",  
    "CommerceRut": "12345678-9",  
    "CommerceName": "Getnet SPA",  
    "BranchName": "Work Café Agustinas",  
    "BranchAddress": "Agustinas 920",  
    "BranchDistrict": "Santiago"  
}
```



### **6.3 ANULAR VENTA**

Comando enviado por el PDV para solicitarle al POS la anulación de una venta realizada. Corresponde a la anulación de cualquier transacción efectuada. El POS imprime el comprobante de anulación y posteriormente envía los datos del resultado a la caja.

**Nota:**

No es posible anular una transacción efectuada después de haber efectuado un “Cierre de Terminal”.

**Request:**

Nombre Parámetro	Tipo	Longitud	Descripción
Command*	int	-	Comando a enviar para solicitar una acción en el POS.
OperationId	int	-	Número de la comprobante.
PrintOnPos	bool	-	Verdadero: solicita la impresión en papel en el POS. Falso: No solicita la impresión al POS. Ambas operaciones envían una respuesta al PDV. <b>[campo opcional]</b>
SecondsTimeOut	int	-	Cantidad de segundos de timeout para esperar por la respuesta del POS. <b>[campo opcional]</b> <b>[no es requerido en el request que va al POS]</b>
PlaceCardToPayTimeout	int	-	Cantidad de segundos de espera para la pantalla de Operar Tarjeta. <b>[campo opcional]</b>
PaymentResultTimeout	int	-	Cantidad de segundos de espera para la pantalla de Resultado de la Venta, sea aprobada o rechazada. <b>[campo opcional]</b>
DateTime	string	-	Fecha actual en el momento de enviar el comando. Se usa para que la firma del mensaje sea siempre diferente.

(\*) La data en gris corresponde a parámetros que se envía de forma interna en la API, no se solicitan en el método expuesto al desarrollador del PDV.

**Request Object:**

```
{
    "Command": 102,
    "OperationId": 123456,
    "PrintOnPos": false,
    "PlaceCardToPayTimeout": 5,
    "PaymentResultTimeout": 2,
    "DateTime": "2023-12-28 22:35:12"
}
```

**Response:**

Nombre Nodo	Tipo	Longitud	Descripción
FunctionCode	int	-	Tipo de función del POS.
ResponseCode	int	-	Código de la respuesta.
ResponseMessage	string	30	Mensaje de texto que representa la respuesta de la operación puede ser Aprobado o Rechazado.
CommerceCode	long	-	Código único del comercio, aprobado por Getnet para generar transacciones.
TerminalId	string	20	Código del terminal POS.
AuthorizationCode	string	20	Código de autorización de la transacción.
OperationId	int	-	Número de la operación.
Success	bool	-	Resultado de la operación.
TransToken	string	30	Dato de largo máximo que corresponde al TID + ID Sucursal + OperationID + yyyyMMddhhmmss
ExpiryDate	string	4	Corresponde al formato numérico de la fecha de expiración de la tarjeta YYMM.
IssuerId	string	2	Corresponde a uno de los valores de la tabla 10.5 de los anexos.
Last4Digits	string	4	Últimos cuatro dígitos de la tarjeta del cliente.
CommerceRut	string	10	Número único para el pago de impuestos - Ejemplo: 12345678-9.
CommerceName	string	25	Nombre del Comercio.
BranchName	string	25	Nombre de la Sucursal.
BranchAddress	string	40	Dirección de la Sucursal.
BranchDistrict	string	25	Comuna de la Sucursal.

**Response Object:**

```
{  
    "FunctionCode": 102,  
    "ResponseCode": 0,  
    "ResponseMessage": "Aprobado",  
    "CommerceCode": 597029414300,  
    "TerminalId": "ABCD1234",  
    "AuthorizationCode": "ABCD1234",  
    "OperationID": 123456,  
    "Success": true,  
    "TransToken": "8000001231060620240923213405",  
    "ExpiryDate": "0828",  
    "IssuerId": "01",  
    "Last4Digits": "1234",  
    "CommerceRut": "12345678-9",  
    "CommerceName": "Getnet SPA",  
    "BranchName": "Work Café Agustinas",  
    "BranchAddress": "Agustinas 920",  
    "BranchDistrict": "Santiago"  
}
```



## 6.4 CIERRE

Comando enviado por el PDV para solicitarle al POS el cierre de las ventas del día , este cierre es impreso en un ticket en el POS. Este método no requiere de parámetros.

### Request:

Nombre Parámetro	Tipo	Longitud	Descripción
Command*	int	-	Comando a enviar para solicitar una acción en el POS.
PrintOnPos	bool	-	Verdadero: solicita la impresión en papel en el POS. Falso: No solicita la impresión al POS. Ambas operaciones envían una respuesta al PDV. <b>[campo opcional]</b>
SecondsTimeOut	int	-	Cantidad de segundos de timeout para esperar por la respuesta del POS. <b>[campo opcional]</b> <b>[no es requerido en el request que va al POS]</b>
PaymentResultTimeout	int	-	Cantidad de segundos de espera para la pantalla de Resultado de la Venta, sea aprobada o rechazada. <b>[campo opcional]</b>
DateTime	string	-	Fecha actual en el momento de enviar el comando. Se usa para que la firma del mensaje sea siempre diferente.

(\*) La data en gris corresponde a parámetros que se envía de forma interna en la API, no se solicitan en el método expuesto al desarrollador del PDV.

### Request Object:

```
{  
    "Command": 103,  
    "PrintOnPos": false,  
    "PaymentResultTimeout": 2,  
    "DateTime": "2023-12-28 22:35:12"  
}
```

**Response:**

Nombre Nodo	Tipo	Longitud	Descripción
FunctionCode	int	-	Tipo de función del POS.
ResponseCode	int	-	Código de la respuesta.
ResponseMessage	string	30	Mensaje de texto que representa la respuesta de la operación puede ser Aprobado o Rechazado.
CommerceCode	long	-	Código único del comercio, aprobado por Getnet para generar transacciones.
TerminalId	string	20	Código del terminal POS.
Success	bool	-	Resultado de la operación.
SaleDetails	List	-	Una lista con los mismos parámetros de respuesta de una Venta agrupados en un arreglo. <b>Solo si el parámetro PrintOnPos viene en false desde el request.</b>

**Response Object 1: Solo para cuando el PrintOnPos venga en true desde el request.**

```
{
    "FunctionCode": 103,
    "ResponseCode": 0,
    "ResponseMessage": "Aprobado",
    "CommerceCode": 550062700310,
    "TerminalId": "ABC1234C",
    "Success": true,
    "SaleDetails": null
}
```

**Response Object 2: Solo para cuando el PrintOnPos venga en false desde el request.**

```
{
    "FunctionCode": 103,
    "ResponseCode": 0,
    "ResponseMessage": "Aprobado",
    "CommerceCode": 550062700310,
    "TerminalId": "ABC1234C",
    "Success": true,
    "SaleDetails": [
        {
            "FunctionCode": 100,
            "ResponseCode": 0,
```



```
"ResponseMessage": "Aprobado",
"CommerceCode": 550062700310,
"TerminalId": "ABC1234C",
"Ticket": "123456789012345678901234",
"AuthorizationCode": "XZ123456",
"Amount": 15000,
"SharesNumber": 3,
"SharesAmount": 5000,
"Last4Digits": 6677,
"OperationId": 60,
"CardType": "CR",
"AccountingDate": "2023-12-28 22:35:12",
"AccountNumber": "30000000000",
"CardBrand": "AX",
"RealDate": "2023-12-28 22:35:12",
"EmployeeId": 1,
"Tip": 1500,
"SaleType": 1,
"PosMode": 1,
"Cashback": 1000,
"TransToken": "8000001231060620240923213405",
"ExpiryDate": "0828",
"EntryMode": "23",
"AID": "315041592E5359532E4444463031",
"CommerceRut": "12345678-9",
"CommerceName": "Getnet SPA",
"BranchName": "Work Café Agustinas",
"BranchAddress": "Agustinas 920",
"BranchDistrict": "Santiago",
"Bin": "12345678",
"PaymentId": "2c341d28-491b-4cf8-aec7-eeb60136b7a5"
} ,
{
"FunctionCode": 100,
"ResponseCode": 0,
"ResponseMessage": "Aprobado",
"CommerceCode": 550062700310,
"TerminalId": "ABC1234C",
"Ticket": "123456789012345678901234",
"AuthorizationCode": "XZ123456",
"Amount": 15000,
"SharesNumber": 3,
```

```
        "SharesAmount": 5000,  
        "Last4Digits": 6677,  
        "OperationId": 60,  
        "CardType": "CR",  
        "AccountingDate": "2023-12-28 22:35:12",  
        "AccountNumber": "300000000000",  
        "CardBrand": "AX",  
        "RealDate": "2023-12-28 22:35:12",  
        "EmployeeId": 1,  
        "Tip": 1500,  
        "SaleType": 1,  
        "PosMode": 1,  
        "Cashback": 1000,  
        "TransToken": "8000001231060620240923213405",  
        "ExpiryDate": "0828",  
        "EntryMode": "23",  
        "AID": "315041592E5359532E4444463031",  
        "CommerceRut": "12345678-9",  
        "CommerceName": "Getnet SPA",  
        "BranchName": "Work Café Agustinas",  
        "BranchAddress": "Agustinas 920",  
        "BranchDistrict": "Santiago",  
        "Bin": "12345678",  
        "PaymentId": "2c341d28-491b-4cf8-aec7-eeb60136b7a5"  
    }  
]  
}
```



## 6.5 TOTALES DE VENTAS

Comando enviado por el PDV para solicitarle al POS un resumen con la cantidad de ventas realizadas y el total de ventas. Este método no requiere de parámetros.

Esto permite que el PDV haga una comparación de las transacciones que no fueron reportadas desde el POS.

### Request:

Nombre Parámetro	Tipo	Longitud	Descripción
Command*	int	-	Comando a enviar para solicitar una acción en el POS.
PrintOnPos	bool	-	Verdadero: solicita la impresión en papel en el POS. Falso: No solicita la impresión al POS. Ambas operaciones envían una respuesta al PDV. <b>[campo opcional]</b>
SecondsTimeOut	int	-	Cantidad de segundos de timeout para esperar por la respuesta del POS. <b>[campo opcional]</b> <b>[no es requerido en el request que va al POS]</b>
DateTime	string	-	Fecha actual en el momento de enviar el comando. Se usa para que la firma del mensaje sea siempre diferente.

(\*) La data en gris corresponde a parámetros que se envía de forma interna en la API, no se solicitan en el método expuesto al desarrollador del PDV.

### Request Object:

```
{  
    "Command": 104,  
    "PrintOnPos": false,  
    "DateTime": "2023-12-28 22:35:12"  
}
```

**Response:**

Nombre Nodo	Tipo	Longitud	Descripción
FunctionCode	int	-	Tipo de función del POS.
ResponseCode	int	-	Código de la respuesta.
ResponseMessage	string	30	Mensaje de texto que representa la respuesta de la operación puede ser Aprobado o Rechazado.
TxCount	int	-	Cantidad de transacciones.
TxTotal	long	-	Suma total de los montos de cada transacción.
CountCreditSales	int	-	Cantidad total de ventas por concepto de tarjetas crédito
TotalCreditSales	long	-	Suma total de los montos de ventas por concepto de tarjetas crédito
CountDebitSales	int	-	Cantidad total de ventas por concepto de tarjetas débito
TotalDebitSales	long	-	Suma total de los montos de ventas por concepto de tarjetas débito
CountPrepaidSales	Int	-	Cantidad total de ventas por concepto de prepago
TotalPrepaidSales	long	-	Suma total de los montos de ventas por concepto de prepago
CountQrCreditSales	Int	-	Cantidad total de ventas por concepto de Crédito QR
TotalQrCreditSales	long	-	Suma total de los montos de ventas por concepto de Credito QR
CountQrDebitSales	Int	-	Cantidad total de ventas por concepto de débito QR
TotalQrDebitSales	long	-	Suma total de los montos de ventas por concepto de debito QR
CountQrPrepaidSales	Int	-	Cantidad total de ventas por concepto de prepago QR
TotalQrPrepaidSales	long	-	Suma total de los montos de ventas por concepto de prepago QR
CountCashSales	Int	-	Cantidad total de ventas por concepto de Efectivo
TotalCashSales	long	-	Suma total de los montos de ventas por concepto de Efectivo
CountCashSalesOff	Int	-	Cantidad total de ventas por concepto de efectivo offline



TotalCashSalesOff	long	-	Suma total de los montos de ventas por concepto de efectivo offline
CountCreditAdvances	Int	-	Cantidad total de avances por concepto de tarjeta de crédito
TotalCreditAdvances	long	-	Suma total de los montos de avances por concepto de tarjeta de crédito
CountCancelCredit	Int	-	Cantidad total de anulaciones o devoluciones por concepto de tarjeta de crédito.
TotalCancelCredit	long	-	Suma total de los montos de anulaciones o devoluciones por concepto de tarjeta de crédito.
CountCancelDebit	Int	-	Cantidad total de anulaciones o devoluciones por concepto de tarjeta de débito.
TotalCancelDebit	long	-	Suma total de los montos de anulaciones o devoluciones por concepto de tarjeta de débito.
CountCancelPrepaid	Int	-	Cantidad total de anulaciones o devoluciones por concepto de prepago.
TotalCancelPrepaid	long	-	Suma total de los montos de anulaciones o devoluciones por concepto de prepago.
CountCancelQrCredit	Int	-	Cantidad total de anulaciones o devoluciones por concepto de tarjeta de crédito QR.
TotalCancelQrCredit	long	-	Suma total de los montos de anulaciones o devoluciones por concepto de tarjeta de crédito QR.
CountCancelQrDebit	Int	-	Cantidad total de anulaciones o devoluciones por concepto de tarjeta de débito QR.
TotalCancelQrDebit	long	-	Suma total de los montos de anulaciones o devoluciones por concepto de tarjeta de débito QR.
CountCancelQrPrepaid	Int	-	Cantidad total de anulaciones o devoluciones por concepto de prepago QR.
TotalCancelQrPrepaid	long	-	Suma total de los montos de anulaciones o devoluciones por concepto de prepago QR.
CountTips	Int	-	Cantidad total por concepto de propinas.
TotalTips	long	-	Suma total de los montos por concepto de propinas.
CountReturns	Int	-	Cantidad total por concepto de vueltos.
TotalReturns	long	-	Suma total de los montos por concepto de vueltos.

**Response Object:**

{

"FunctionCode": 104,



```
"ResponseCode": 0,  
"ResponseMessage": "Aprobado",  
"TxCount": 3,  
"TxTotal": 15000,  
"CountCreditSales ": 3,  
"TotalCreditSales ": 15000,  
"CountDebitSales ": 3,  
"TotalDebitSales ": 15000,  
"CountPrepaidSales ": 3,  
"TotalPrepaidSales ": 15000,  
"CountQrCreditSales ": 3,  
"TotalQrCreditSales ": 15000,  
"CountQrDebitSales ": 3,  
"TotalQrDebitSales ": 15000,  
"CountQrPrepaidSales ": 3,  
"TotalQrPrepaidSales ": 15000,  
"CountCashSales ": 3,  
"TotalCashSales ": 15000,  
"CountCashSalesOff ": 3,  
"TotalCashSalesOff ": 15000,  
"CountCreditAdvances ": 3,  
"TotalCreditAdvances ": 15000,  
"CountCancelCredit ": 3,  
"TotalCancelCredit ": 15000,  
"CountCancelDebit ": 3,  
"TotalCancelDebit ": 15000,  
"CountCancelPrepaid ": 3,  
"TotalCancelPrepaid ": 15000,  
"CountCancelQrCredit ": 3,  
"TotalCancelQrCredit ": 15000,  
"CountCancelQrDebit ": 3,  
"TotalCancelQrDebit ": 15000,  
"CountCancelQrPrepaid ": 3,  
"TotalCancelQrPrepaid ": 15000,  
"CountTips": 3,  
"TotalTips": 15000,  
"CountReturns": 3,  
"TotalReturns": 15000  
}
```



## 6.6 DETALLE DE VENTAS

Comando enviado por el PDV para solicitarle al POS todas las transacciones que se han realizado y que quedan en la memoria del POS. Se le envía un parámetro para solicitar la impresión de todas las transacciones.

### Request:

Nombre Parámetro	Tipo	Longitud	Descripción
Command*	int	-	Comando a enviar para solicitar una acción en el POS.
PrintOnPos	bool	-	Verdadero: solicita la impresión en papel en el POS. Falso: No solicita la impresión al POS. Ambas operaciones envían una respuesta al PDV. <b>[campo opcional]</b>
SecondsTimeOut	int	-	Cantidad de segundos de timeout para esperar por la respuesta del POS. <b>[campo opcional]</b> <b>[no es requerido en el request que va al POS]</b>
DateTime	string	-	Fecha actual en el momento de enviar el comando. Se usa para que la firma del mensaje sea siempre diferente.

(\*) La data en gris corresponde a parámetros que se envía de forma interna en la API, no se solicitan en el método expuesto al desarrollador del PDV.

### Request Object:

```
{
    "Command": 105,
    "PrintOnPos": false,
    "DateTime": "2023-12-28 22:35:12"
}
```

### Response:

Nombre Nodo	Tipo	Longitud	Descripción
FunctionCode	int	-	Tipo de función del POS.
ResponseCode	int	-	Código de la respuesta.
ResponseMessage	string	30	Mensaje de texto que representa la respuesta de la operación puede ser Aprobado o Rechazado.
SaleDetails	List	-	Una lista con los mismos parámetros de respuesta de una Venta agrupados en un arreglo.

**Response Object:**

```
{  
    "FunctionCode": 105,  
    "ResponseCode": 0,  
    "ResponseMessage": "Aprobado",  
    "SaleDetails": [  
        {  
            "FunctionCode": 100,  
            "ResponseCode": 0,  
            "ResponseMessage": "Aprobado",  
            "CommerceCode": 550062700310,  
            "TerminalId": "ABC1234C",  
            "Ticket": "123456789012345678901234",  
            "AuthorizationCode": "XZ123456",  
            "Amount": 15000,  
            "SharesNumber": 3,  
            "SharesAmount": 5000,  
            "Last4Digits": 6677,  
            "OperationId": 60,  
            "CardType": "CR",  
            "AccountingDate": "2023-12-28 22:35:12",  
            "AccountNumber": "300000000000",  
            "CardBrand": "AX",  
            "RealDate": "2023-12-28 22:35:12",  
            "EmployeeId": 1,  
            "Tip": 1500,  
            "SaleType": 1,  
            "PosMode": 1,  
            "Cashback": 1000,  
            "TransToken": "8000001231060620240923213405",  
            "ExpiryDate": "0828",  
            "EntryMode": "23",  
            "AID": "315041592E5359532E4444463031",  
            "CommerceRut": "12345678-9",  
            "CommerceName": "Getnet SPA",  
            "BranchName": "Work Café Agustinas",  
            "BranchAddress": "Agustinas 920",  
            "BranchDistrict": "Santiago",  
            "Bin": "12345678"  
        },  
        {  
            "FunctionCode": 100,  
            "ResponseCode": 0,  
            "ResponseMessage": "Aprobado",  
            "CommerceCode": 550062700310,
```

```
"TerminalId": "ABC1234C",
"Ticket": "123456789012345678901234",
"AuthorizationCode": "XZ123456",
"Amount": 15000,
"SharesNumber": 3,
"SharesAmount": 5000,
"Last4Digits": 6677,
"OperationId": 60,
"CardType": "CR",
"AccountingDate": "2023-12-28 22:35:12",
"AccountNumber": "30000000000",
"CardBrand": "AX",
"RealDate": "2023-12-28 22:35:12",
"EmployeeId": 1,
"Tip": 1500,
"SaleType": 1,
"PosMode": 1,
"Cashback": 1000,
"TransToken": "8000001231060620240923213405",
"ExpiryDate": "0828",
"EntryMode": "23",
"AID": "315041592E5359532E4444463031",
"CommerceRut": "12345678-9",
"CommerceName": "Getnet SPA",
"BranchName": "Work Café Agustinas",
"BranchAddress": "Agustinas 920",
"BranchDistrict": "Santiago",
"Bin": "12345678"
}
]
}
```



## 6.7 POLLING

Comando enviado por el PDV para consultarle al POS si está conectado.

### Request:

Nombre Parámetro	Tipo	Longitud	Descripción
Command*	int	-	Comando a enviar para solicitar una acción en el POS.
DateTime	string	-	Fecha actual en el momento de enviar el comando. Se usa para que la firma del mensaje sea siempre diferente.

(\*) La data en gris corresponde a parámetros que se envía de forma interna en la API, no se solicitan en el método expuesto al desarrollador del PDV.

### Request Object:

```
{
    "Command": 106,
    "DateTime": "2023-12-28 22:35:12"
}
```

### Response:

Nombre Nodo	Tipo	Longitud	Descripción
FunctionCode	int	-	Tipo de función del POS.
ResponseCode	int	-	Código de la respuesta.
ResponseMessage	string	30	Mensaje de texto que representa la respuesta de la operación puede ser Aprobado o Rechazado.
Connected	bool	-	Verdadero: POS conectado. Falso: POS desconectado.

### Response Object:

```
{
    "FunctionCode": 106,
    "ResponseCode": 0,
    "ResponseMessage": "Aprobado",
    "Connected": true
}
```



## 6.8 CAMBIO A POS NORMAL

Comando enviado por el PDV para solicitarle al POS que cambie la configuración al modo normal. Esto significa que el POS bloquearía la comunicación entrante con el PDV.

**Nota:** Si se cambia el POS integrado a modo normal, se deberá volver a configurar en el POS para volver al modo integrado, ya que no es posible realizar esta configuración con un comando desde el PDV.

### Request:

Nombre Parámetro	Tipo	Longitud	Descripción
Command*	int	-	Comando a enviar para solicitar una acción en el POS.
DateTime	string	-	Fecha actual en el momento de enviar el comando. Se usa para que la firma del mensaje sea siempre diferente.

(\*) La data en gris corresponde a parámetros que se envía de forma interna en la API, no se solicitan en el método expuesto al desarrollador del PDV.

### Request Object:

```
{  
    "Command": 107,  
    "DateTime": "2023-12-28 22:35:12"  
}
```

### Response:

Nombre Nodo	Tipo	Longitud	Descripción
FunctionCode	int	-	Tipo de función del POS.
ResponseCode	int	-	Código de la respuesta.
ResponseMessage	string	30	Mensaje de texto que representa la respuesta de la operación puede ser Aprobado o Rechazado.
ChangeToNormalMode	bool	-	Devuelve verdadero y luego se bloquea la comunicación con el PDV.

**Response Object:**

```
{  
    "FunctionCode": 107,  
    "ResponseCode": 0,  
    "ResponseMessage": "Aprobado",  
    "ChangeToNormalMode": true  
}
```



## 6.9 TRANSACCIÓN DE DEVOLUCIÓN

Comando enviado por el PDV para solicitarle al POS la anulación de una venta realizada.

Esta operación es muy parecida a una cancelación, sin embargo, se produce posterior al cierre de las ventas, ya que la cancelación anula una venta realizada durante el mismo día, previo al cierre de la caja, puesto que la información aún está presente en el POS. Sin embargo, luego del cierre de la caja, todas las ventas son borradas el dispositivo y hay que ir a consultar al autorizador.

### Request:

Nombre Parámetro	Tipo	Longitud	Descripción
Command*	int	-	Comando a enviar para solicitar una acción en el POS.
AuthorizationCode	string	20	Código de autorización de la transacción.
Amount	long	-	Valor en pesos para realizar la transacción.
PrintOnPos	bool	-	Verdadero: solicita la impresión en papel en el POS. Falso: No solicita la impresión al POS. Ambas operaciones envían una respuesta al PDV. <b>[campo opcional]</b>
SecondsTimeOut	int	-	Cantidad de segundos de timeout para esperar por la respuesta del POS. <b>[campo opcional]</b> <b>[no es requerido en el request que va al POS]</b>
PlaceCardToPayTimeout	int	-	Cantidad de segundos de espera para la pantalla de Opere Tarjeta. <b>[campo opcional]</b>
PaymentResultTimeout	int	-	Cantidad de segundos de espera para la pantalla de Resultado de la Venta, sea aprobada o rechazada. <b>[campo opcional]</b>
DateTime	string	-	Fecha actual en el momento de enviar el comando. Se usa para que la firma del mensaje sea siempre diferente.

(\*) La data en gris corresponde a parámetros que se envía de forma interna en la API, no se solicitan en el método expuesto al desarrollador del PDV.

**Request Object:**

```
{
    "Command": 108,
    "AuthorizationCode": "XZ123456",
    "Amount": 1500,
    "PrintOnPos": false,
    "PlaceCardToPayTimeout": 5,
    "PaymentResultTimeout": 2,
    "DateTime": "2023-12-28 22:35:12"
}
```

**Response:**

Nombre Nodo	Tipo	Longitud	Descripción
FunctionCode	int	-	Tipo de función del POS.
ResponseCode	int	-	Código de la respuesta.
ResponseMessage	string	30	Mensaje de texto que representa la respuesta de la operación puede ser Aprobado o Rechazado.
CommerceCode	long	-	Código único del comercio, aprobado por Getnet para generar transacciones.
TerminalId	string	20	Código del terminal POS.
AuthorizationCode	string	20	Código de autorización de la transacción.
OperationId	int	-	Número de la operación.
Success	bool	-	Resultado de la operación.
DateTime	string	-	Fecha y hora de la transacción.
TransToken	string	30	Dato de largo máximo que corresponde al TID + ID Sucursal + OperationID + yyyyMMddhhmmss
ExpiryDate	string	4	Corresponde al formato numérico de la fecha de expiración de la tarjeta YYMM.
IssuerId	string	2	Corresponde a uno de los valores de la tabla 10.5 de los anexos.
Last4Digits	string	4	Últimos cuatro dígitos de la tarjeta del cliente.
CommerceRut	string	10	Número único para el pago de impuestos - Ejemplo: 12345678-9.
CommerceName	string	25	Nombre del Comercio.

BranchName	string	25	Nombre de la Sucursal.
BranchAddress	string	40	Dirección de la Sucursal.
BranchDistrict	string	25	Comuna de la Sucursal.

**Response Object:**

```
{  
    "FunctionCode": 108,  
    "ResponseCode": 0,  
    "ResponseMessage": "Aprobado",  
    "CommerceCode": 597029414300,  
    "TerminalId": "ABCD1234",  
    "AuthorizationCode": "ABCD1234",  
    "OperationID": 123456,  
    "Success": true,  
    "DateTime": "2023-12-28 22:35:12",  
    "TransToken": "8000001231060620240923213405",  
    "ExpiryDate": "0828",  
    "IssuerId": "01",  
    "Last4Digits": "1234",  
    "CommerceRut": "12345678-9",  
    "CommerceName": "Getnet SPA",  
    "BranchName": "Work Café Agustinas",  
    "BranchAddress": "Agustinas 920",  
    "BranchDistrict": "Santiago"  
}
```



## 6.10 DUPLICADO OTROS

Comando enviado por el PDV para solicitarle al POS la impresión de la última transacción o bien una en específico.

### Request:

Nombre Parámetro	Tipo	Longitud	Descripción
Command*	int	-	Comando a enviar para solicitar una acción en el POS.
OperationId	int	-	Número de la operación en caso de imprimir un ticket específico o por defecto 0 en caso de ser el último ticket.
PrintOnPos	bool	-	Verdadero: solicita la impresión en papel en el POS. Falso: No solicita la impresión al POS. Ambas operaciones envían una respuesta al PDV. <b>[campo opcional]</b>
SecondsTimeOut	int	-	Cantidad de segundos de timeout para esperar por la respuesta del POS. <b>[campo opcional]</b> <b>[no es requerido en el request que va al POS]</b>
DateTime	string	-	Fecha actual en el momento de enviar el comando. Se usa para que la firma del mensaje sea siempre diferente.

(\*) La data en gris corresponde a parámetros que se envía de forma interna en la API, no se solicitan en el método expuesto al desarrollador del PDV.

### Request Object:

```
{  
    "Command": 109,  
    "OperationId": 0,  
    "PrintOnPos": false,  
    "DateTime": "2023-12-28 22:35:12"  
}
```

**Response:**

Nombre Nodo	Tipo	Longitud	Descripción
FunctionCode	int	-	Tipo de función del POS.
ResponseCode	int	-	Código de la respuesta.
ResponseMessage	string	30	Mensaje de texto que representa la respuesta de la operación puede ser Aprobado o Rechazado.
CommerceCode	long	-	Código único del comercio, aprobado por Getnet para generar transacciones.
TerminalId	string	20	Código del terminal POS.
Ticket	string	24	Este dato lo genera el PDV en el comprobante que se genera para la venta (número de boleta). [campo opcional]
AuthorizationCode	string	20	Código de autorización de la transacción.
Amount	long	-	Valor en pesos de la venta a transaccionar.
SharesNumber	Int	-	Multiplicador.
SharesAmount	Int	-	Número mínimo de en el que se calcula la venta.
Last4Digits	string	-	Últimos cuatro dígitos de la tarjeta del cliente.
OperationId	int	-	Número de la operación.
CardType	string	2	Tipo de Tarjeta.
AccountingDate	string	-	Fecha en la que se contabilizará la venta En formato ISO YYYY-MM-DD HH-mm-ss
AccountNumber	string	30	Número de cuenta.
CardBrand	string	2	Marca de la tarjeta utilizada en la transacción.
RealDate	string		Fecha actual.
EmployeeId	int	-	Número que representa al vendedor.
Tip	long	-	Propina de la venta.
SaleType	int	-	Dato que identifica el tipo de venta afecta/exenta
PosMode	int	1	Permite conocer si la venta se realizó en el Modo Pos Integrado o Pos Normal.0 = no integrado   1 = integrado
Cashback	long	-	Valor del vuelto solicitado.
TransToken	string	30	Dato de largo máximo que corresponde al TID + ID Sucursal + OperationID + yyyyMMddhhmmss



ExpiryDate	string	4	Corresponde al formato numérico de la fecha de expiración de la tarjeta YYMM.
EntryMode	string	2	Corresponde a uno de los valores de la tabla 10.4 de los anexos.
AID	string	-	Es el Application Identifier para EMV.
CommerceRut	string	10	Número único para el pago de impuestos - Ejemplo: 12345678-9.
CommerceName	string	25	Nombre del Comercio.
BranchName	string	25	Nombre de la Sucursal.
BranchAddress	string	40	Dirección de la Sucursal.
BranchDistrict	string	25	Comuna de la Sucursal.
Bin	string	8	Número BIN de la tarjeta que realizó la compra.

### Response Object:

```
{
    "FunctionCode": 109,
    "ResponseCode": 0,
    "ResponseMessage": "Aprobado",
    "CommerceCode": 550062700310,
    "TerminalId": "ABC1234C",
    "Ticket": "123456789012345678901234",
    "AuthorizationCode": "XZ123456",
    "Amount": 15000,
    "SharesNumber": 3,
    "SharesAmount": 5000,
    "Last4Digits": 6677,
    "OperationId": 60,
    "CardType": "CR",
    "AccountingDate": "2023-12-28 22:35:12",
    "AccountNumber": "300000000000",
    "CardBrand": "AX",
    "RealDate": "2023-12-28 22:35:12",
    "EmployeeId": 1,
    "Tip": 1500,
    "SaleType": 1,
    "PosMode": 1,
    "Cashback": 1000,
```

```
"TransToken": "8000001231060620240923213405",
"ExpiryDate": "0828",
"EntryMode": "23",
"AID": "315041592E5359532E4444463031",
"CommerceRut": "12345678-9",
"CommerceName": "Getnet SPA",
"BranchName": "Work Café Agustinas",
"BranchAddress": "Agustinas 920",
"BranchDistrict": "Santiago",
"Bin": "12345678"
}
```



## 6.11 COMPRAS POR VENDEDOR

Comando enviado por el PDV para solicitarle al POS la impresión todas las ventas asignadas a un vendedor en el POS Getnet. En caso de querer imprimir las ventas realizadas por un vendedor en particular se debe enviar el identificador que es un dato numérico de largo 4 (en caso de estar habilitado).

### Request:

Nombre Parámetro	Tipo	Longitud	Descripción
Command*	int	-	Comando a enviar para solicitar una acción en el POS.
EmployeeId	int	-	Número que representa al vendedor. En caso de ser todos se podría enviar un 0 y en caso de ser distinto de 0 indicaría que es un vendedor en particular.
PrintOnPos	bool	-	Verdadero: solicita la impresión en papel en el POS. Falso: No solicita la impresión al POS. Ambas operaciones envían una respuesta al PDV. <b>[campo opcional]</b>
SecondsTimeOut	int	-	Cantidad de segundos de timeout para esperar por la respuesta del POS. <b>[campo opcional]</b> <b>[no es requerido en el request que va al POS]</b>
DateTime	string	-	Fecha actual en el momento de enviar el comando. Se usa para que la firma del mensaje sea siempre diferente.

(\*) La data en gris corresponde a parámetros que se envía de forma interna en la API, no se solicitan en el método expuesto al desarrollador del PDV.

### Request Object:

```
{  
    "Command": 110,  
    "EmployeeId": 1234,  
    "PrintOnPos": false,  
    "DateTime": "2023-12-28 22:35:12"  
}
```

**Response:**

Nombre Nodo	Tipo	Longitud	Descripción
FunctionCode	int	-	Tipo de función del POS.
ResponseCode	int	-	Código de la respuesta.
ResponseMessage	string	40	Mensaje de texto que representa la respuesta de la operación puede ser Aprobado o Rechazado.
SaleDetails	List	-	Una lista con los parámetros de detalle por cada venta.
Date	string	2	Día del mes.
EmployeeId	int	-	Número que representa al vendedor. <b>[campo opcional]</b>
Ticket	string	10	Este dato lo genera el PDV en el comprobante que se genera para la venta (número de boleta). <b>[campo opcional]</b>
Amount	long	-	Valor en pesos de la venta a transaccionar.
Nc	string	2	¿?
Tc	string	2	¿?
Tx	string	2	¿?

**Response Object:**

```
{
    "FunctionCode": 110,
    "ResponseCode": 0,
    "ResponseMessage": "Aprobado",
    "SaleDetails": [
        {
            "Date": "DD",
            "EmployeeId": 1234,
            "Ticket": "BBBBBBBB",
            "Amount": 999999999,
            "Nc": "CC",
            "Tc": "TT",
            "Tx": "XX"
        },
        {
            "Date": "DD",
            "Seller": "VVVV",
        }
    ]
}
```



```
"Ticket": "BBBBBBBBB",  
"Amount": 999999999,  
"Nc": "CC",  
"Tc": "TT",  
"Tx": "XX"  
}  
]  
}
```



## 6.12 REPORTE DE PROPINAS

Comando enviado por el PDV para solicitarle al POS la impresión todas las propinas de los vendedores en el POS Getnet. En caso de querer imprimir las propinas de un vendedor en particular se debe enviar el identificador que es un dato numérico de largo 4 (en caso de estar habilitado).

### Request:

Nombre Parámetro	Tipo	Longitud	Descripción
Command*	int	-	Comando a enviar para solicitar una acción en el POS.
EmployeeId	int	-	Número que representa al vendedor. En caso de ser todos se podría enviar un 0 y en caso de ser distinto de 0 indicaría que es un vendedor en particular.
PrintOnPos	bool	-	Verdadero: solicita la impresión en papel en el POS. Falso: No solicita la impresión al POS. Ambas operaciones envían una respuesta al PDV. <b>[campo opcional]</b>
SecondsTimeOut	int	-	Cantidad de segundos de timeout para esperar por la respuesta del POS. <b>[campo opcional]</b> <b>[no es requerido en el request que va al POS]</b>
DateTime	string	-	Fecha actual en el momento de enviar el comando. Se usa para que la firma del mensaje sea siempre diferente.

(\*) La data en gris corresponde a parámetros que se envía de forma interna en la API, no se solicitan en el método expuesto al desarrollador del PDV.

### Request Object:

```
{
    "Command": 111,
    "EmployeeId": 1234,
    "PrintOnPos": false,
    "DateTime": "2023-12-28 22:35:12"
}
```

**Response:**

Nombre Nodo	Tipo	Longitud	Descripción
FunctionCode	int	-	Tipo de función del POS.
ResponseCode	int	-	Código de la respuesta.
ResponseMessage	string	40	Mensaje de texto que representa la respuesta de la operación puede ser Aprobado o Rechazado.
TipReport	List	-	Una lista con los parámetros de detalle por propina.
EmployeeId	int	-	Número que representa al vendedor. <b>[campo opcional]</b>
Tx	string	3	?
Sales	long	-	Monto de las ventas totales.
Tips	long	-	Monto de las propinas.

**Response Object:**

```
{  
    "FunctionCode": 111,  
    "ResponseCode": 0,  
    "ResponseMessage": "Aprobado",  
    "TipReport": [  
        {  
            "EmployeeId": 1234,  
            "Tx": "NNN",  
            "Sales": 999999999,  
            "Tips": 99999999  
        },  
        {  
            "EmployeeId": 1234,  
            "Tx": "NNN",  
            "Sales": 999999999,  
            "Tips": 99999999  
        }  
    ]  
}
```



## 6.13 COMPRA PREDETERMINADA

Comando enviado por el PDV para solicitarle al POS el cambio de tipo de compra predeterminada por defecto.

- Cobro afecto
- Cobro exento
- Factura afecta
- Factura exenta
- Recaudación afecta
- Recaudación exenta

### Request:

Nombre Parámetro	Tipo	Longitud	Descripción
Command*	int	-	Comando a enviar para solicitar una acción en el POS.
SaleType	int	-	Define el tipo de operación que pueden ser: CA, CE, FA, FE, RA, RE, según la definición arriba.
DateTime	string	-	Fecha actual en el momento de enviar el comando. Se usa para que la firma del mensaje sea siempre diferente.

(\*) La data en gris corresponde a parámetros que se envía de forma interna en la API, no se solicitan en el método expuesto al desarrollador del PDV.

Comando SaleType	Tipo
1	Compra Afecta
2	Factura Afecta
3	Compra Exenta
4	Factura Exenta
5	Recaudación Afecta
6	Recaudación Exenta

**Request Object:**

```
{  
    "Command": 113,  
    "SaleType": 1,  
    "DateTime": "2023-12-28 22:35:12"  
}
```

**Response:**

Nombre Nodo	Tipo	Longitud	Descripción
FunctionCode	int	-	Tipo de función del POS.
ResponseCode	int	-	Código de la respuesta.
ResponseMessage	string	30	Mensaje de texto que representa la respuesta de la operación puede ser Aprobado o Rechazado.
SaleType	int	-	Dato que identifica el tipo de venta afecta/exenta

**Response Object:**

```
{  
    "FunctionCode": 113,  
    "ResponseCode": 0,  
    "ResponseMessage": "Aprobado",  
    "SaleType": 1  
}
```



## 6.14 REPORTE DE PARÁMETROS

Comando enviado por el PDV para solicitarle al POS el reporte de todos los parámetros configurados en el POS Getnet.

### Request:

Nombre Parámetro	Tipo	Longitud	Descripción
Command*	int	-	Comando a enviar para solicitar una acción en el POS.
PrintOnPos	bool	-	Verdadero: solicita la impresión en papel en el POS. Falso: No solicita la impresión al POS. Ambas operaciones envían una respuesta al PDV. <b>[campo opcional]</b>
SecondsTimeOut	int	-	Cantidad de segundos de timeout para esperar por la respuesta del POS. <b>[campo opcional]</b> <b>[no es requerido en el request que va al POS]</b>
DateTime	string	-	Fecha actual en el momento de enviar el comando. Se usa para que la firma del mensaje sea siempre diferente.

(\*) La data en gris corresponde a parámetros que se envía de forma interna en la API, no se solicitan en el método expuesto al desarrollador del PDV.

### Request Object:

```
{  
    "Command": 114,  
    "PrintOnPos": false,  
    "DateTime": "2023-12-28 22:35:12"  
}
```

**Response:**

Nombre Nodo	Tipo	Longitud	Descripción
FunctionCode	int	-	Tipo de función del POS.
ResponseCode	int	-	Código de la respuesta.
ResponseMessage	string	30	Mensaje de texto que representa la respuesta de la operación puede ser Aprobado o Rechazado.
ApplicationName	string	-	Nombre de la aplicación.
Version	string	-	Versión del sistema.
So	string	-	Sistema operativo.
EmvModule	string	-	Módulo EMV (Europay Mastercard and Visa)
Sn	string	-	Número de serie.
Model	string	-	Modelo.
CommunicationType	string	-	Tipo de comunicación.
PrimaryIp	string	-	IP Primaria.
SecondIp	string	-	IP Secundaria.
Company	string	-	Compañía.
Apn	string	-	APN
SimId	string	-	Número de chip SIM.
SucursalId	string	-	Número de Sucursal.
TerminalId	string	-	Número de Terminal.
Tip	bool	-	Configuración de propinas.
Ticket	bool	-	Configuración de boletas.
Employee	bool	-	Configuración de empleados.
VoucherAsTicket	bool	-	Configuración para el comprobante de boletas.
Return	bool	-	Configuración para vuelto.
IssuerQuotas	bool	-	Configuración para recibir cuotas del emisor.
MinimumIssuerQuotas	int	-	Mínimo de cuotas del emisor.
MaximumIssuerQuotas	int	-	Máximo de cuotas del emisor.
TradeQuotas	bool	-	Configuración de las cuotas comercio.

MinimumTradeQuotas	int	-	Mínimo de cuotas comercio.
MaximumTradeQuotas	int	-	Máximo de cuotas comercio.
MinimumAmount	int	-	Monto mínimo de venta.

### Response Object \*:

```
{
  "FunctionCode": 115,
  "ResponseCode": 0,
  "ResponseMessage": "Aprobado",
  "ApplicationName": "GPA",
  "Version": "G27R12.0.2-10",
  "So": "release-31343200-A300",
  "EmvModule": "7.15.1",
  "Sn": "390-370-304",
  "Model": "V240m 3GPlus",
  "CommunicationType": "3G",
  "PrimaryIp": "190.208.26.170",
  "SecondIp": "190.196.71.248",
  "Company": "Movistar",
  "Apn": "M2M.MOVISTAR.CL",
  "SimId": "8934075400010355505",
  "SucursalId": "00000000000088",
  "TerminalId": "20000235",
  "Tip": true,
  "Boleta": true,
  "Employee": true,
  "VoucherAsTicket": true,
  "Return": true,
  "IssuerQuotas": true,
  "MinimumTradeQuotas": 2,
  "MaximumTradeQuotas": 60,
  "TradeQuotas": true,
  "MinimumTradeQuotas": 2,
  "MaximumTradeQuotas": 6,
  "MinimumAmount": 100,
}
```



## 6.15 REPORTE SIM

Comando enviado por el PDV para consultarle al POS si tiene señal de conexión y los parámetros del chip SIM.

### Request:

Nombre Parámetro	Tipo	Longitud	Descripción
Command*	int	-	Comando a enviar para solicitar una acción en el POS.
PrintOnPos	bool	-	Verdadero: solicita la impresión en papel en el POS. Falso: No solicita la impresión al POS. Ambas operaciones envían una respuesta al PDV. <b>[campo opcional]</b>
SecondsTimeOut	int	-	Cantidad de segundos de timeout para esperar por la respuesta del POS. <b>[campo opcional]</b> <b>[no es requerido en el request que va al POS]</b>
DateTime	string	-	Fecha actual en el momento de enviar el comando. Se usa para que la firma del mensaje sea siempre diferente.

(\*) La data en gris corresponde a parámetros que se envía de forma interna en la API, no se solicitan en el método expuesto al desarrollador del PDV.

### Request Object:

```
{  
    "Command": 115,  
    "PrintOnPos": false,  
    "DateTime": "2023-12-28 22:35:12"  
}
```

**Response:**

Nombre Nodo	Tipo	Longitud	Descripción
FunctionCode	int	-	Tipo de función del POS.
ResponseCode	int	-	Código de la respuesta.
ResponseMessage	string	30	Mensaje de texto que representa la respuesta de la operación puede ser Aprobado o Rechazado.
Connected	bool	-	Verdadero: POS conectado. Falso: POS desconectado.
Company	String	20	Compañía del SIM <b>[campo opcional]</b>
ApnSim	String	20	APN del Sim <b>[campo opcional]</b>
ApnConfig	String	20	APN configurada en el dispositivo <b>[campo opcional]</b>
SimId	String	20	Id del SIM <b>[campo opcional]</b>
SimProvider	String	20	SIM proveedor <b>[campo opcional]</b>
SimUsed	String	20	SIM Utilizada <b>[campo opcional]</b>
BatteryPercentage	int	-	Porcentaje de batería del dispositivo <b>[campo opcional]</b>
IntensitySignal	String	20	Intensidad de señal del dispositivo <b>[campo opcional]</b>
Rssi	String	20	RSSI <b>[campo opcional]</b>
SimStatus	String	20	Status del Sim <b>[campo opcional]</b>
RadioLevel	String	20	Nivel de radio del Sim <b>[campo opcional]</b>
Imei	String	20	IMEI del dispositivo <b>[campo opcional]</b>
NetworkConnection	String	20	Conexión de red del dispositivo <b>[campo opcional]</b>
NetworksEnabled	String	20	Redes habilitadas en el dispositivo <b>[campo opcional]</b>
Lac	String	20	LAC. <b>[campo opcional]</b>
CallId	String	20	Cell Id. <b>[campo opcional]</b>
NetworkType	String	20	Tipo de red. <b>[campo opcional]</b>
NetworkPlmn	String	20	PLMN de Red. <b>[campo opcional]</b>
SimPlmn	String	20	PLMN de SIM. <b>[campo opcional]</b>
NetworkLog	String	20	Registros en la red. <b>[campo opcional]</b>

**Response Object:**

```
{  
    "FunctionCode": 115,  
    "ResponseCode": 0,  
    "ResponseMessage": "Aprobado",  
    "Connected": true,  
    "Company": "Aprobado",  
    "ApnSim": "Aprobado",  
    "ApnConfig": "Aprobado",  
    "SimId": "Aprobado",  
    "SimProvider": "Aprobado",  
    "SimUsed": "Aprobado",  
    "BatteryPercentage": 90,  
    "IntensitySignal": "Aprobado",  
    "Rssi": "Aprobado",  
    "SimStatus": "Aprobado",  
    "RadioLevel": "Aprobado",  
    "Imei": "Aprobado",  
    "NetworkConnection": "Aprobado",  
    "NetworksEnabled": "Aprobado",  
    "Lac": "Aprobado",  
    "CallId": "Aprobado",  
    "NetworkType": "Aprobado",  
    "NetworkPlmn": "Aprobado",  
    "SimPlmn": "Aprobado",  
    "NetworkLog": "Aprobado"  
}
```



## 6.16 CANCELAR VENTA

Comando enviado por el PDV para solicitarle al POS la cancelación o anulación de la transacción o proceso de venta enviado desde la caja al POS Getnet ésta puede ser usada en cualquier momento.

### Request:

Nombre Parámetro	Tipo	Longitud	Descripción
Command*	int	-	Comando a enviar para solicitar una acción en el POS.
DateTime	string	-	Fecha actual en el momento de enviar el comando. Se usa para que la firma del mensaje sea siempre diferente.

(\*) La data en gris corresponde a parámetros que se envía de forma interna en la API, no se solicitan en el método expuesto al desarrollador del PDV.

### Request Object:

```
{
    "Command": 116,
    "DateTime": "2023-12-28 22:35:12"
}
```

### Response:

Nombre Nodo	Tipo	Longitud	Descripción
FunctionCode	int	-	Tipo de función del POS.
ResponseCode	int	-	Código de la respuesta.
ResponseMessage	string	40	Mensaje de texto que representa la respuesta de la operación puede ser Aprobado o Rechazado.

### Response Object:

```
{
    "FunctionCode": 116,
    "ResponseCode": 0,
    "ResponseMessage": "Aprobado"
}
```

**6.17 VENTA MULTICOMERCIO**

Comando enviado por el PDV para solicitarle al POS una venta multicomiccio.

**Request:**

Nombre Parámetro	Tipo	Longitud	Descripción
Command*	int	-	Comando a enviar para solicitar una acción en el POS.
Amount	long	-	Valor en pesos para realizar la transacción.
PrintOnPos	bool	-	Verdadero: solicita la impresión en papel en el POS. Falso: No solicita la impresión al POS. Ambas operaciones envían una respuesta al PDV. <b>[campo opcional]</b>
SaleType	Int	-	Define el tipo de venta en caso de ser distinta a la predeterminada. <b>[campo opcional]</b>
SendMessage	bool	-	Permite enviar mensajes intermedios a la Caja.
SecondsTimeOut	int	-	Cantidad de segundos de timeout para esperar por la respuesta del POS. <b>[campo opcional]</b> <b>[no es requerido en el request que va al POS]</b>
RestrictedCards	Bines	-	Los bines indicados NO pueden realizar compras. Los otros se deben aceptar.
AllowedCards	Bines	-	Los bines indicados SI pueden realizar compras. Los otros se deben rechazar.
RutCommerceSon	string	10	Número de RUT del comercio hijo en el formato 12345678-9.
CommerceData	CommerceData	-	Objeto que contiene los datos del comercio.
CommerceParams	CommerceParams	-	Objeto que contiene los parámetros del comercio.
PlaceCardToPayTimeout	int	-	Cantidad de segundos de espera para la pantalla de Opere Tarjeta. <b>[campo opcional]</b>
PaymentResultTimeout	int	-	Cantidad de segundos de espera para la pantalla de Resultado de la Venta, sea aprobada o rechazada. <b>[campo opcional]</b>
DateTime	string	-	Fecha actual en el momento de enviar el comando. Se usa para que la firma del mensaje sea siempre diferente.

(\*) La data en gris corresponde a parámetros que se envía de forma interna en la API, no se solicitan en el método expuesto al desarrollador del PDV.

**Request Object:**

```
{  
    "Command": 120,  
    "Amount": 15000,  
    "PrintOnPos": false,  
    "SaleType": 1,  
    "RestrictedCards": ["12345678", "87654321", "23456789"],  
    "AllowedCards": ["12345678", "87654321", "23456789"],  
    "RutCommerceSon": "12345678-9",  
    "CommerceData": [  
        "LegalName": "La Mega Empresa Ltda.",  
        "CommerceNumber": "123456789012345",  
        "CommerceRut": "12.345.678-9",  
        "BranchNumber": "123456789012345",  
        "BranchName": "Casa Matriz",  
        "LittleBranchName": "La Mega",  
        "BranchAddress": "Tobalaba 1234",  
        "BranchDistrict": "Las Condes",  
        "TerminalId": "12345678",  
        "SerialNumber": "12ABC6789012"  
    ],  
    "CommerceParams": [  
        "IndicadorBiomoneda": 0,  
        "IndicadorBoleta": 1,  
        "IndicadorNoVendedor": 1,  
        "IndicadorComprobanteComoBoleta": 1,  
        "IndicadorPropina": 1,  
        "IndicadorVuelto": 1,  
        "IndicadorCuotasEmisor": 1,  
        "MinimoCuotasEmisor": "1",  
        "MaximoCuotasEmisor": "12",  
        "IndicadorCuotasComercio": 1,  
        "MinimoCuotasComercio": "1",  
        "MaximoCuotasComercio": "12",  
        "IndicadorCuotasTasaCero": 1,  
        "MinimoCuotasTasaCero": "1",  
        "MaximoCuotasTasaCero": "12",  
        "IndicadorCuotasTasaInteresConocida": 1,  
        "MinimoCuotasTasaInteresConocida": "1",  
        "MaximoCuotasTasaInteresConocida": "12",  
        "TipoProductoCreditoVisa": 1,  
    ]  
}
```



```
"TipoProductoDebitoVisa": 1,  
"TipoProductoDebitoVisaElectron": 1,  
"TipoProductoPrepagoVisa": 1,  
"TipoProductoCreditoMastercard": 1,  
"TipoProductoDebitoMastercard": 1,  
"TipoProductoDebitoMaestro": 1,  
"TipoProductoPrepagoMastercard": 1,  
"TipoProductoCreditoAmex": 1,  
"TipoProductoDebitoAmex": 1,  
"TipoProductoPrepagoAmex": 1,  
"TipoProductoMagna": 1,  
"NumeroDeFolio": 1,  
"PosAvance": 1  
  
],  
"PlaceCardToPayTimeout": 5,  
"PaymentResultTimeout": 2,  
"DateTime": "2023-12-28 22:35:12"  
}
```

Comando SaleType	Tipo
1	Compra Afecta
2	Factura Afecta
3	Compra Exenta
4	Factura Exenta
5	Recaudación Afecta
6	Recaudación Exenta



### 6.17.1 BINES

Modelo de datos para los restrictedCards y allowedCards de SaleMC.

Nombre Parámetro	Tipo	Longitud	Descripción
RestrictedCards	String[]	8	Arreglo de bines numéricos.
AllowedCards	String[]	8	Arreglo de bines numéricos.

### 6.17.2 COMMERCE DATA

Modelo de datos para los datos del comercio en el campo CommerceData de SaleMC.

Nombre Parámetro	Tipo	Longitud	Descripción
LegalName	string	60	Razón social del comercio
CommerceNumber	string	15	Id comercio
CommerceRut	string	13	Rut del comercio
BranchNumber	string	15	Número de la sucursal
BranchName	string	60	Nombre de la sucursal
LittleBranchName	string	22	Nombre corto de la sucursal
BranchAddress	string	60	Dirección de la sucursal
BranchDistrict	string	60	Comuna de la sucursal
TerminalId	string	8	Código del terminal
SerialNumber	string	12	Numero de Serie del POS



### 6.17.3 COMMERCE PARAMS

Modelo de datos para los parámetros en el campo CommerceParams de SaleMC.

Nombre Parámetro	Tipo	Longitud	Descripción
IndicadorBiomoneda	int	1	Indica el tipo de moneda con el cual va a funcionar el terminal 1= Bimoneda 0= Peso Chileno Nota: este valor se obtiene en CPS del campo Moneda de la Terminal Física, de acuerdo a lo siguiente: Si el campo tiene definido Pesos Chilenos, Bimoneda = 0. Si el campo tiene definido Dólares, Bimoneda = 1
IndicadorBoleta	int	1	Permite habilitar/ deshabilitar el ingreso de boleta 1= Permite digitar boleta 0= No permite digitar boleta
IndicadorNoVendedor	int	1	Permite habilitar/deshabilitar ingreso No. de Vendedor 1 = Habilita Empleado 0 = Deshabilita Empleado
IndicadorComprobanteComoBoleta	int	1	Permite habilitar/deshabilitar funcionalidad comprobante como boleta de acuerdo y Boleta Electrónica 0 = Deshabilita Voucher Boleta 1 = Habilita Voucher como Boleta 2 = Habilita Boleta Electrónica
IndicadorPropina	int	1	Permite habilitar/deshabilitar ingreso de propina 1 = Habilita Propina 0 = Deshabilita Propina
IndicadorVuelto	int	1	Permite habilitar/deshabilitar ofrecer vuelto en transacciones Débito 1= Habilita Vuelto (Cahsback) 0 = Deshabilita
IndicadorCuotasEmisor	int	1	Habilitar/deshabilitar Cuotas Emisores 1 = Activo Cuotas Emisor 0 = Deshabilita
MínimoCuotasEmisor	string	2	Mínimo de cuotas para Cuotas Emisor Si el campo Indicador cuota Emisor es = 0; esta irá 00



MaximoCuotasEmisor	string	2	Máximo de cuotas para Cuotas Emisor Si el campo Indicador cuota Emisor es = 0; esta irá 00
IndicadorCuotasComercio	int	1	Habilitar/deshabilitar Cuotas Comercio 1 = Activo Cuotas comercio 0 = Deshabilita
MinimoCuotasComercio	string	2	Mínimo de cuotas para Cuotas Comercio Si el campo Indicador cuota comercio es = 0; esta irá 00
MaximoCuotasComercio	string	2	Máximo de cuotas para Cuotas Comercio Si el campo Indicador cuota comercio es = 0; esta irá 00
IndicadorCuotasTasaCero	int	1	Habilitar/deshabilitar Cuotas Tasa Cero 1 = Activo Cuotas Tasa Cero. 0 = Deshabilita
MinimoCuotasTasaCero	string	2	Mínimo Cuotas para Cuotas Tasa Cero Si el campo Indicador cuota tasa cero es = 0; esta irá 00
MaximoCuotasTasaCero	string	2	Máximo Cuotas para Cuotas Tasa Cero Si el campo Indicador cuota tasa cero es = 0; esta irá 00
IndicadorCuotasTasalInteresConocida	int	1	Habilitar/deshabilitar Cuotas Tasa de Interés conocida. 1 = Activo Tasa de Interés Conocida. 0 = Deshabilita
MinimoCuotasTasalInteresConocida	string	2	Mínimo Cuotas para Cuotas Tasa de Interés conocida. Si el campo Indicador cuota tasa de Interés conocida es = 0; esta irá 00
MaximoCuotasTasalInteresConocida	string	2	Máximo Cuotas para Cuotas Tasa de Interés conocida. Si el campo Indicador cuota tasa de Interés conocida es = 0; esta irá 00
TipoProductoCreditoVisa	int	1	Permite habilitar o deshabilitar el producto Crédito Visa 1= Habilita producto Crédito Visa 0= Deshabilita producto Crédito Visa
TipoProductoDebitoVisa	int	1	Permite habilitar o deshabilitar el producto Débito Visa 1= Habilita producto Débito Visa 0= Deshabilita producto Débito Visa
TipoProductoDebitoVisaElectron	int	1	Permite habilitar o deshabilitar el producto Débito Visa Electrón 1= Habilita producto Débito Visa Electrón 0= Deshabilita producto Débito Visa Electrón



TipoProductoPrepagoVisa	int	1	Permite habilitar o deshabilitar el producto Prepago Visa 1= Habilita producto Prepago Visa 0= Deshabilita producto Prepago Visa
TipoProductoCreditoMastercard	int	1	Permite habilitar o deshabilitar el producto Crédito MasterCard 1= Habilita producto Crédito MasterCard 0= Deshabilita producto Crédito MasterCard
TipoProductoDebitoMastercard	int	1	Permite habilitar o deshabilitar el producto Débito MasterCard 1= Habilita producto Débito MasterCard 0= Deshabilita producto Débito MasterCard
TipoProductoDebitoMaestro	int	1	Permite habilitar o deshabilitar el producto Débito Maestro 1= Habilita producto Débito Maestro 0= Deshabilita producto Débito Maestro
TipoProductoPrepagoMastercard	int	1	Permite habilitar o deshabilitar el producto Prepago MasterCard 1= Habilita producto Prepago MasterCard 0= Deshabilita producto Prepago MasterCard
TipoProductoCreditoAmex	int	1	Permite habilitar o deshabilitar el producto Crédito Amex 1= Habilita producto Crédito Amex 0= Deshabilita producto Crédito Amex
TipoProductoDebitoAmex	int	1	Permite habilitar o deshabilitar el producto Débito Amex 1= Habilita producto Débito Amex 0= Deshabilita producto Débito Amex
TipoProductoPrepagoAmex	int	1	Permite habilitar o deshabilitar el producto Prepago Amex 1= Habilita producto Prepago Amex 0= Deshabilita producto Prepago Amex
TipoProductoMagna	int	1	Permite habilitar o deshabilitar el producto Magna (este campo será para uso futuro una vez se tenga la definición del BIN y configuración para este producto). 1= Habilita producto Magna 0= Deshabilita producto Magna Enviar siempre en 1



NumeroDeFolio	int	1	Indica si la sucursal tiene habilitado el valor 'Número de Folio' entre las características de la terminal configuradas por sucursal 1= Habilita Número de Folio 0= Deshabilita Número de Folio
PosAvance	int	1	Indica si la terminal permite Avances en Efectivo 1= Habilita Avance en Efectivo 0= Deshabilita Avance en Efectivo

**Response:**

Nombre Nodo	Tipo	Longitud	Descripción
FunctionCode	int	-	Tipo de función del POS.
ResponseCode	int	-	Código de la respuesta. ( <i>tabla de códigos de respuesta en los Anexos</i> )
ResponseMessage	string	30	Mensaje de texto que representa la respuesta de la operación puede ser Aprobado o Rechazado.
CommerceCode	long	-	Código único de sucursal, aprobado por Getnet para generar transacciones.
TerminalId	string	20	Código del terminal POS.
Ticket	string	24	Este dato lo genera el PDV en el comprobante que se genera para la venta (número de boleta). [campo opcional]
AuthorizationCode	string	20	Código de autorización de la transacción.
Amount	long	-	Valor en pesos de la venta a transaccionar.
SharesNumber	Int	-	Numero de cuotas.
SharesAmount	Int	-	Valor de cada cuota <b>[campo opcional]</b>
Last4Digits	int	-	Últimos cuatro dígitos de la tarjeta del cliente.
OperationId	int	-	Corresponde al número de comprobante que genera el POS.
CardType	string	2	Tipo de Tarjeta.
AccountingDate	string	-	Fecha y hora de la transacción.
AccountNumber	string	30	Número de cuenta. <b>[campo opcional]</b>
CardBrand	string	2	Marca de la tarjeta utilizada en la transacción.



RealDate	string	-	Fecha actual.
EmployeeId	int	-	Número que representa al vendedor. <b>[campo opcional]</b>
Tip	long	-	Propina de la venta. <b>[campo opcional]</b>
SaleType	int	-	Dato que identifica el tipo de venta afecta/exenta
PosMode	int	1	Permite conocer si la venta se realizó en el Modo Pos Integrado o Pos Normal. 0 = no integrado   1 = integrado
Cashback	long	-	Valor del vuelto solicitado.
TransToken	string	30	Dato de largo máximo que corresponde al TID + ID Sucursal + OperationID + yyyyMMddhhmmss
ExpiryDate	string	4	Corresponde al formato numérico de la fecha de expiración de la tarjeta YYMM.
EntryMode	string	2	Corresponde a uno de los valores de la tabla 10.4 de los anexos.
AID	string	-	Es el Application Identifier para EMV.
CommerceRut	string	10	Número único para el pago de impuestos - Ejemplo: 12345678-9.
CommerceName	string	25	Nombre del Comercio.
BranchName	string	25	Nombre de la Sucursal.
BranchAddress	string	40	Dirección de la Sucursal.
BranchDistrict	string	25	Comuna de la Sucursal.
Bin	string	8	Número BIN de la tarjeta que realizó la compra.
PaymentId	string	-	Identificación única de la transacción SEP.
HostRRN	string	-	Código de autorización de la transacción. Campo 37 del mensaje ISO.
Mti	string	4	Valor fijo 0200
DE11	string	6	System Trace Audit Number, valor numérico de 6 posiciones.
DE12	string	6	Time Local Transaction, valor numérico de 6 posiciones.
DE13	string	4	Date Local Transaction, valor numérico de 4 posiciones.

**Response Object:**

```
{  
    "FunctionCode": 120,  
    "ResponseCode": 0,  
    "ResponseMessage": "Aprobado",  
    "CommerceCode": 550062700310,  
    "TerminalId": "ABC1234C",  
    "Ticket": "ABC123",  
    "AuthorizationCode": "XZ123456",  
    "Amount": 15000,  
    "SharesNumber": 3,  
    "SharesAmount": 5000,  
    "Last4Digits": "6677",  
    "OperationId": 60,  
    "CardType": "CR",  
    "AccountingDate": "2023-12-28 22:35:12",  
    "AccountNumber": "30000000000",  
    "CardBrand": "AX",  
    "RealDate": "2023-12-28 22:35:12",  
    "EmployeeId": 1,  
    "Tip": 1500,  
    "SaleType": 1,  
    "PosMode": 1,  
    "Cashback": 1000,  
    "TransToken": "8000001231060620240923213405",  
    "ExpiryDate": "0828",  
    "EntryMode": "23",  
    "AID": "315041592E5359532E4444463031",  
    "CommerceRut": "12345678-9",  
    "CommerceName": "Getnet SPA",  
    "BranchName": "Work Café Agustinas",  
    "BranchAddress": "Agustinas 920",  
    "BranchDistrict": "Santiago",  
    "Bin": "12345678",  
    "PaymentId": "2c341d28-491b-4cf8-aec7-eeb60136b7a5",  
    "hostRRN": "433413000062",  
    "mti": "0200",  
    "de11": "621234",  
    "de12": "131340",  
    "de13": "1129"  
}
```

**6.18 ANULACIÓN MULTICOMERCIO**

Comando enviado por el PDV para solicitarle al POS una anulación multicomiccio.

**Request:**

Nombre Parámetro	Tipo	Longitud	Descripción
Command*	int	-	Comando a enviar para solicitar una acción en el POS.
Amount	long	-	Valor en pesos para realizar la transacción.
OperationId	int	-	Número del comprobante
PrintOnPos	bool	-	True: Solicitud la impresión en papel en el POS. False: No solicita la impresión al POS.
RutCommerceSon	string	10	Número de RUT del comercio hijo en el formato 12345678-9.
HostRRN	string	-	Código de autorización de la transacción. Campo 37 del mensaje ISO.
SaleType	int	-	Define el tipo de venta en caso de ser distinta a la predeterminada. Se definen los diferentes valores más abajo.
CommerceData	CommerceData	-	Objeto que contiene los datos del comercio.
OriginalData	DE90	-	Objeto de tipo DE90.
PlaceCardToPayTimeout	int	-	Cantidad de segundos de espera para la pantalla de Opere Tarjeta. <b>[campo opcional]</b>
PaymentResultTimeout	int	-	Cantidad de segundos de espera para la pantalla de Resultado de la Venta, sea aprobada o rechazada. <b>[campo opcional]</b>
DateTime	string	-	Fecha actual en el momento de enviar el comando. Se usa para que la firma del mensaje sea siempre diferente.

(\*) La data en gris corresponde a parámetros que se envía de forma interna en la API, no se solicitan en el método expuesto al desarrollador del PDV.

**Request Object:**

```
{  
    "Command": 122,  
    "Amount": 15000,  
    "OperationId": 12,  
    "PrintOnPos": false,  
    "RutCommerceSon": "12345678-9",  
    "HostRRN": "12345",  
    "SaleType": 1,  
    "CommerceData": [  
        "LegalName": "La Mega Empresa Ltda.",  
        "CommerceNumber": "123456789012345",  
        "CommerceRut": "12.345.678-9",  
        "BranchNumber": "123456789012345",  
        "BranchName": "La Mega - Las Condes",  
        "LittleBranchName": "La Mega",  
        "BranchDistrict": "Las Condes",  
        "BranchAddress": "Tobalaba 1234",  
        "TerminalId": "12345678",  
        "PosSerialNumber": "12ABC6789012"  
    ],  
    "OriginalData": [  
        "Mti": "0200",  
        "DE11": "123456",  
        "DE12": "123456",  
        "DE13": "1234"  
    ],  
    "PlaceCardToPayTimeout": 5,  
    "PaymentResultTimeout": 2,  
    "DateTime": "2023-12-28 22:35:12"  
}
```



### 6.18.1 COMMERCE DATA

Modelo de datos para los datos del comercio en el campo CommerceData de SaleMC.

Nombre Parámetro	Tipo	Longitud	Descripción
LegalName	string	60	Razón social del comercio
CommerceNumber	string	15	Id comercio
CommerceRut	string	13	Rut del comercio
BranchNumber	string	15	Número de la sucursal
BranchName	string	60	Nombre de la sucursal
LittleBranchName	string	22	Nombre corto de la sucursal
BranchAddress	string	60	Dirección de la sucursal
BranchDistrict	string	60	Comuna de la sucursal
TerminalId	string	8	Código del terminal
SerialNumber	string	12	Numero de Serie del POS

### 6.18.2 DE90

Modelo de datos para un Objeto Data Element específico para un CancellationMc.

Nombre Parámetro	Tipo	Longitud	Descripción
Mti	string	4	Valor fijo 0200
DE11	string	6	System Trace Audit Number, valor numérico de 6 posiciones.
DE12	string	6	Time Local Transaction, valor numérico de 6 posiciones.
DE13	string	4	Date Local Transaction, valor numérico de 4 posiciones.

**Response:**

Nombre Nodo	Tipo	Longitud	Descripción
FunctionCode	int	-	Tipo de función del POS.
ResponseCode	int	-	Código de la respuesta.
ResponseMessage	string	30	Mensaje de texto que representa la respuesta de la operación puede ser Aprobado o Rechazado.
CommerceCode	long	-	Código único del comercio, aprobado por Getnet para generar transacciones.
TerminalId	string	20	Código del terminal POS.
AuthorizationCode	string	20	Código de autorización de la transacción.
OperationId	int	-	Número de la operación.
Success	bool	-	Resultado de la operación.
TransToken	string	30	Dato de largo máximo que corresponde al TID + ID Sucursal + OperationID + yyyyMMddhhmmss
ExpiryDate	string	4	Corresponde al formato numérico de la fecha de expiración de la tarjeta YYMM.
IssuerId	string	2	Corresponde a uno de los valores de la tabla 10.5 de los anexos.
Last4Digits	string	4	Últimos cuatro dígitos de la tarjeta del cliente.
CommerceRut	string	10	Número único para el pago de impuestos - Ejemplo: 12345678-9.
CommerceName	string	25	Nombre del Comercio.
BranchName	string	25	Nombre de la Sucursal.
BranchAddress	string	40	Dirección de la Sucursal.
BranchDistrict	string	25	Comuna de la Sucursal.

**Response Object:**

```
{  
    "FunctionCode": 122,  
    "ResponseCode": 0,  
    "ResponseMessage": "Aprobado",  
    "CommerceCode": 597029414300,  
    "TerminalId": "ABCD1234",  
    "AuthorizationCode": "ABCD1234",  
    "OperationID": 123456,  
    "Success": true,  
    "TransToken": "8000001231060620240923213405",  
    "ExpiryDate": "0828",  
    "IssuerId": "01",  
    "Last4Digits": "1234",  
    "CommerceRut": "12345678-9",  
    "CommerceName": "Getnet SPA",  
    "BranchName": "Work Café Agustinas",  
    "BranchAddress": "Agustinas 920",  
    "BranchDistrict": "Santiago"  
}
```



## 6.19 DEVOLUCIÓN MULTICOMERCIO

Comando enviado por el PDV para solicitarle al POS la anulación de una venta multicomiccio realizada.

Esta operación es muy parecida a una cancelación, sin embargo, se produce posterior al cierre de las ventas, ya que la cancelación anula una venta realizada durante el mismo día, previo al cierre de la caja, puesto que la información aún está presente en el POS. Sin embargo, luego del cierre de la caja, todas las ventas son borradas el dispositivo y hay que ir a consultar al autorizador.

### Request:

Nombre Parámetro	Tipo	Longitud	Descripción
Command*	int	-	Comando a enviar para solicitar una acción en el POS.
AuthorizationCode	string	20	Código de autorización de la transacción
Amount	long	-	Valor en pesos para realizar la transacción
PrintOnPos	bool	-	True: Solicitud la impresión en papel en el POS. False: No solicita la impresión al POS. Ambas operaciones envían una respuesta al PDV.
RutCommerceSon	string	10	Número de RUT del comercio hijo en el formato 12345678-9.
CommerceData	CommerceData	-	Objeto que contiene los datos del comercio.
PlaceCardToPayTimeout	int	-	Cantidad de segundos de espera para la pantalla de Operar Tarjeta. <b>[campo opcional]</b>
PaymentResultTimeout	int	-	Cantidad de segundos de espera para la pantalla de Resultado de la Venta, sea aprobada o rechazada. <b>[campo opcional]</b>
DateTime	string	-	Fecha actual en el momento de enviar el comando. Se usa para que la firma del mensaje sea siempre diferente.

### Request Object:

```
{
    "Command": 123,
    "AuthorizationCode": "XZ123456",
    "Amount": 1500,
    "PrintOnPos": false,
    "RutCommerceSon": "12345678-9",
    "CommerceData": [
        "LegalName": "La Mega Empresa Ltda.",
        "CommerceNumber": "123456789012345",
        "CommerceRut": "12.345.678-9",
    ]
}
```



```
"BranchNumber": "123456789012345",
"BranchName": "Casa Matriz",
"LittleBranchName": "La Mega",
"BranchAddress": "Tobalaba 1234",
"BranchDistrict": "Las Condes",
"TerminalId": "12345678",
"SerialNumber": "12ABC6789012"
],
"PlaceCardToPayTimeout": 5,
"PaymentResultTimeout": 2,
"DateTime": "2023-12-28 22:35:12"
}
```

### 6.19.1 COMMERCE DATA

Modelo de datos para los datos del comercio en el campo CommerceData de SaleMC.

Nombre Parámetro	Tipo	Longitud	Descripción
LegalName	string	60	Razón social del comercio
CommerceNumber	string	15	Id comercio
CommerceRut	string	13	Rut del comercio
BranchNumber	string	15	Número de la sucursal
BranchName	string	60	Nombre de la sucursal
LittleBranchName	string	22	Nombre corto de la sucursal
BranchAddress	string	60	Dirección de la sucursal
BranchDistrict	string	60	Comuna de la sucursal
TerminalId	string	8	Código del terminal
SerialNumber	string	12	Numero de Serie del POS

**Response:**

Nombre Nodo	Tipo	Longitud	Descripción
FunctionCode	int	-	Tipo de función del POS.
ResponseCode	int	-	Código de la respuesta.
ResponseMessage	string	30	Mensaje de texto que representa la respuesta de la operación puede ser Aprobado o Rechazado.
CommerceCode	long	-	Código único del comercio, aprobado por Getnet para generar transacciones.
TerminalId	string	20	Código del terminal POS.
AuthorizationCode	string	20	Código de autorización de la transacción.
OperationId	int	-	Número de la operación.
Success	bool	-	Resultado de la operación.
DateTime	string	-	Fecha y hora de la transacción.
TransToken	string	30	Dato de largo máximo que corresponde al TID + ID Sucursal + OperationID + yyyyMMddhhmmss
ExpiryDate	string	4	Corresponde al formato numérico de la fecha de expiración de la tarjeta YYMM.
IssuerId	string	2	Corresponde a uno de los valores de la tabla 10.5 de los anexos.
Last4Digits	string	-	Últimos cuatro dígitos de la tarjeta del cliente.

**Response Object:**

```
{  
    "FunctionCode": 123,  
    "ResponseCode": 0,  
    "ResponseMessage": "Aprobado",  
    "CommerceCode": 597029414300,  
    "TerminalId": "ABCD1234",  
    "AuthorizationCode": "ABCD1234",  
    "OperationID": 123456,  
    "Success": true,  
    "DateTime": "2023-12-28 22:35:12",  
    "TransToken": "8000001231060620240923213405",  
    "ExpiryDate": "0828",  
    "IssuerId": "01",  
    "Last4Digits": "1234"  
}
```



## 6.20 MAIN POS DATA

Comando enviado por el PDV para solicitarle al POS los datos del Comercio.

### Request:

Nombre Parámetro	Tipo	Longitud	Descripción
Command*	int	-	Comando a enviar para solicitar una acción en el POS.
DateTime	string	-	Fecha actual en el momento de enviar el comando. Se usa para que la firma del mensaje sea siempre diferente.

(\*) La data en gris corresponde a parámetros que se envía de forma interna en la API, no se solicitan en el método expuesto al desarrollador del PDV.

### Request Object:

```
{  
    "Command": 124,  
    "DateTime": "2023-12-28 22:35:12"  
}
```

### Response:

Nombre Nodo	Tipo	Longitud	Descripción
FunctionCode	int	-	Tipo de función del POS.
ResponseCode	int	-	Código de la respuesta.
ResponseMessage	string	30	Mensaje de texto que representa la respuesta de la operación puede ser Aprobado o Rechazado.
CommerceData	CommerceData	-	Objeto que contiene los datos del comercio.

**Response Object:**

```
{  
    "FunctionCode": 124,  
    "ResponseCode": 0,  
    "ResponseMessage": "Aprobado",  
    "CommerceData": {  
        "LegalName": "La Mega Empresa Ltda.",  
        "CommerceNumber": "123456789012345",  
        "CommerceRut": "12.345.678-9",  
        "BranchNumber": "123456789012345",  
        "BranchName": "Casa Matriz",  
        "LittleBranchName": "La Mega",  
        "BranchAddress": "Tobalaba 1234",  
        "BranchDistrict": "Las Condes",  
        "TerminalId": "12345678",  
        "SerialNumber": "12ABC6789012"  
    }  
}
```



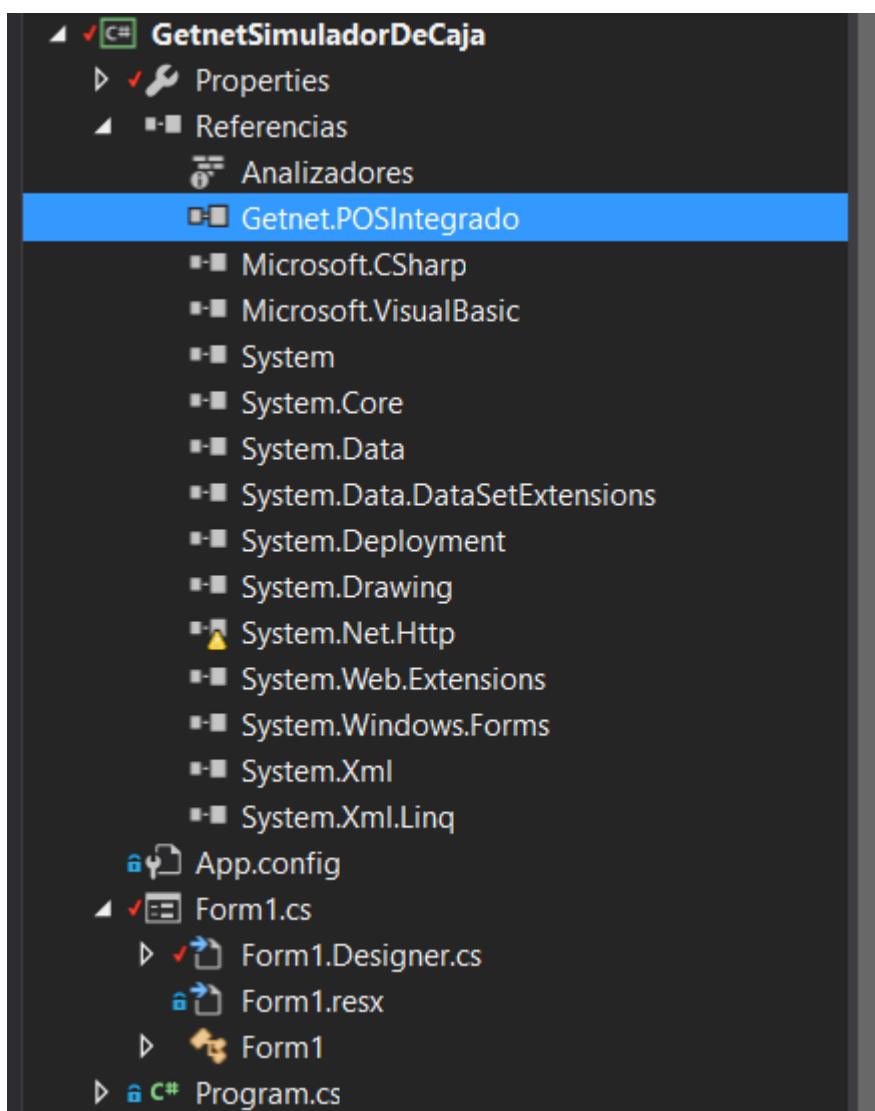
## 7 PROCESO DE INTEGRACIÓN PARA PROYECTOS WINFORM C# .NET

Para el proceso de integración se entrega el proyecto de Simulador de Caja tipo Winform en C# .Net y un Simulador de Caja para Web (desarrollado con Javascript para el frontend y con C# .Net para el backend) de manera que es posible revisarlo en su totalidad para tener un ejemplo claro de cómo se puede modificar el software de Caja para incorporar la integración con Getnet POS Integrado.

En este caso hablaremos de la integración para un software de caja desarrollado en la modalidad Winform. A continuación, entregamos algunos tips importantes:

### 7.1 LIBRERIA

Lo primero es agregar la referencia de la DLL dentro del proyecto.





## 7.2 AGREGAR LA LIBRERÍA EN LA SECCIÓN DE USING

Será necesario incorporar esta línea al inicio de la clase para tener acceso a los métodos de la librería.

```
using Getnet.POSIntegrado;
using Getnet.POSIntegrado.Responses;
```

## 7.3 CONTROLAR LA COMUNICACIÓN SERIAL Y TIMEOUTS

Para esto se requiere inicializar algunas variables globales dentro de la clase, como las siguientes:

```
//Declaración de variables globales
private static readonly SerialPort serialPort =
POSIntegrado.Instance.GetnetSerialPort();

private static readonly System.Timers.Timer timeoutForConnection =
POSIntegrado.Instance.GetTimeoutForConnetion();

private static readonly System.Timers.Timer timeoutForPosResponse =
POSIntegrado.Instance.GetTimeoutForPosResponse();
```

Explicación:

17. **Variable serialPort:** Permitirá controlar la comunicación por el cable USB con el POS.
18. **Variable timeoutForConnection:** Permitirá controlar los tiempos de espera para recibir los TRUE que validan que el POS recibió las solicitudes.
19. **Variable timeoutForResponse:** Permitirá controlar los tiempos de espera para las respuestas de cada comando enviado al POS Getnet.

## 7.4 DEFINIR MANEJADORES DE LA COMUNICACIÓN HACIA LA CAJA

En este paso se inicializan orejas de escucha en el puerto serial y en los timers para esperar las respuestas y controlar los timeouts. Esto se encuentra dentro del método Form1()

```
//Manejador de las respuestas por SerialPort
serialPort.DataReceived += new
SerialDataReceivedEventHandler(serialPort_DataReceived);

//Control de tiempo para esperar por la conexión del POS
timeoutForConnection.Elapsed += timeoutForConnectionElapsed;

//Control de tiempo para esperar respuestas del POS
timeoutForPosResponse.Elapsed += timeoutForPosResponseElapsed;
```



## 7.5 CONTROLAR EL CIERRE DE PUERTOS

En un método del ciclo de vida del Winform se puede llamar al cierre de puertos cuando se cierra la aplicación o cuando se requiera cerrar la comunicación con el POS Getnet.

```
//Cierra la conexión y finalización de subprocessos de recepción antes de cerrar la
//aplicación
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    POSIntegrado.Instance.UsbClosePort();
    POSIntegrado.Instance.UsbDisposePort();
}
```

## 7.6 CONTROLAR LOS TIMERS

Estos métodos se inicializaron (en el punto 6.4) más arriba y se comportan tal como un CATCH tras finalizar el tiempo de espera. De manera que pueden permitir mostrar un mensaje de alerta al usuario de la Caja o bien controlarlo de manera inteligente dentro del software de Caja para indicar que se agotó el tiempo de espera y que el POS no está conectado o bien superó el tiempo de espera para responder el comando solicitado.

```
//Control del timeout para la conexión con el POS
private void timeoutForConnectionElapsed(object sender, ElapsedEventArgs e)
{
    POSIntegrado.Instance.StopTimerForConnetion();
    POSIntegrado.Instance.StopTimerForPosResponse();
    MessageBox.Show("There is no connection to the POS.", "Error");
}

//Control del timeout para las respuestas del POS
private void timeoutForPosResponseElapsed(object sender, ElapsedEventArgs e)
{
    POSIntegrado.Instance.StopTimerForPosResponse();
    MessageBox.Show("The POS has timed out waiting for the response.", "Error");
}
```

## 7.7 CONOCER LOS PUERTOS DISPONIBLES

Si fuera necesario solicitar los puertos disponibles en Windows se puede ocupar este método.

```
string[] ports = POSIntegrado.Instance.ListPorts();
```



## 7.8 CONECTAR AL PUERTO SERIAL

Para conectarse al puerto serial COM disponible se pueden realizar de la siguiente manera:

**EN DESARROLLO:** Se puede definir en duro el puerto de comunicaciones y la velocidad de transmisión de datos (BaudRate) para hacer las pruebas.

```
try
{
    //Conexión por USB
    POSIntegrado.Instance.UsbConnect("COM1", 115200);
}
catch (Exception ex)
{
    MessageBox.Show("Error connecting to COM port: " + ex.Message);
}
```

**EN PRODUCCIÓN:** Se requerirá la configuración desde el Agente POS. En este caso no se define ni el nombre del puerto ni la velocidad de transmisión de datos (BaudRate) dado que la componente la va a buscar a la ruta de la configuración del Agente POS (Ver punto 4.14.)

```
try
{
    //Conexión por USB
    POSIntegrado.Instance.UsbConnect();
}
catch (Exception ex)
{
    MessageBox.Show("Error connecting to COM port: " + ex.Message);
}
```

**NOTA:** Estas son solo recomendaciones. Si usted no quiere usar el Agente POS puede usar la misma configuración del ambiente de desarrollo y validar que el nombre del puerto en la Caja será el mismo que usted definió en su código.



## 7.9 ENVIAR SOLICITUDES AL POS

El envío de comandos al POS Getnet es tan simple como llamar a un método y la librería se encarga de todo el procesamiento de los datos para construir el JSON, firmar los datos y enviar el mensaje así:

### POLLING:

```
try
{
    POSIntegrado.Instance.Poll();
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

### SALE:

```
try
{
    POSIntegrado.Instance.Sale(
        25000, "A1234", false, (POSCommands.SaleType)0, false, 1, 60);
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

### LAST VOUCHER:

```
try
{
    POSIntegrado.Instance.LastVoucher();
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

### CANCELLATION:

```
try
{
    POSIntegrado.Instance.Refund(1234);
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

**CLOSE:**

```
try
{
    POSIntegrado.Instance.Close();
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

**TOTALS:**

```
try
{
    POSIntegrado.Instance.Totals();
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

**DETAILS:**

```
try
{
    POSIntegrado.Instance.Details();
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

**NORMAL MODE:**

```
try
{
    POSIntegrado.Instance.SetNormalMode();
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

**REFUND:**

```
try
{
    POSIntegrado.Instance.Return(1234, 25000);
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

**DUPLICATE:**

```
try
{
    POSIntegrado.Instance.DuplicateOthers(1234);
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

**SELLER SALES:**

```
try
{
    POSIntegrado.Instance.SalesBySeller(1);
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

**TIP REPORT:**

```
try
{
    POSIntegrado.Instance.TipReport(1);
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

**DEFAULT SALE:**

```
try
{
    POSIntegrado.Instance.DefaultSaleType((POSCommands.SaleType) 0);
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

**PARAMS:**

```
try
{
    POSIntegrado.Instance.ParameterReport();
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

**SIM REPORT:**

```
try
{
    POSIntegrado.Instance.SimReport();
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

**CANCEL SALE:**

```
try
{
    POSIntegrado.Instance.CancelSale();
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

**NOTA:** Todos los métodos expuestos aquí tienen una sobrecarga, lo que significa que se pueden enviar en su forma más simple (como se exponen aquí) o bien agregarle parámetros como el PrintOnPos o Seconds (para controlar el timeout).



## 7.10 LEER LAS RESPUESTAS DEL POS

Para leer las respuestas desde el POS Getnet solo será necesario implementar un método ya definido más arriba en el punto 6.4. de la siguiente manera:

```
//Detector de mensajes entrantes por USB
private void serialPort_DataReceived(object sender, SerialDataReceivedEventArgs e)
{
    try
    {
        string message = POSIntegrado.Instance.UsbReadData();
        POSIntegrado.Instance.StopTimerForConnection();
        dynamic dataProcessed = POSIntegrado.Instance.ProcessData(message);
    }
    catch (Exception ex)
    {
        ExceptionMessage(ex.Message);
    }
}
```

Es importante mencionar que al recibir el mensaje en la variable “**message**” se puede ver tal como se recibe en el Simulador de Caja, pero no es necesario descomponerlo o intentar deserializarlo porque en el siguiente paso la variable “**dataProcessed**” nos entrega un objeto con los valores de cada respuesta ya procesados y con lo cual podemos usarlos directamente para leer los datos y procesarlos en el Software de Caja.

## 7.11 CASTEO DE OBJETOS RESPONSES

La variable “**dataProcessed**” se define de tipo **dynamic** porque soporta cualquier tipo de objeto de las respuestas definidas por la librería. Si quisieramos transformarla internamente dentro del desarrollo de la Caja se puede hacer de la siguiente manera:

### POLLING:

```
PollResponse response = dataProcessed;
```

### SALE, LAST VOUCHER, DUPLICATE:

```
SaleResponse sale = dataProcessed;
```

### CANCELATION:

```
RefundResponse cancelation = dataProcessed;
```

### CLOSE:

```
CloseResponse close = dataProcessed;
```

### TOTALS:

```
TotalsResponse totals = dataProcessed;
```

**DETAILS:**

```
DetailsResponse details = dataProcessed;
```

**NORMAL MODE:**

```
SetNormalModeResponse normalMode = dataProcessed;
```

**REFUND:**

```
ReturnResponse refund = dataProcessed;
```

**SELLER SALES:**

```
SalesBySellerResponse sellerSales = dataProcessed;
```

**TIP REPORT:**

```
TipReportResponse tipsReport = dataProcessed;
```

**DEFAULT SALE:**

```
DefaultSaleTypeResponse defaultSale = dataProcessed;
```

**PARAMS:**

```
ParameterReportResponse parameters = dataProcessed;
```

**SIM REPORT:**

```
SimReportResponse simReport = dataProcessed;
```

**CANCEL SALE:**

```
CancelSaleResponse cancelSale = dataProcessed;
```

**EJEMPLO:**

Aquí proponemos una manera de hacerlo. Se pueden crear tantas variables como tipos de respuestas hay y luego simplemente validar el comando y castearlo para obtener acceso a las propiedades del modelo y poder usar los datos directamente en el Software de Caja.

**NOTA:** No es necesario leer el resultado del puerto serial y deserializarlo para procesarlo de forma manual, la librería ya hace esto.

```
//Variables para castear la data procesada
PollResponse poll;
SaleResponse sale;
RefundResponse cancelation;
CloseResponse close;
TotalsResponse totals;
DetailsResponse details;
SetNormalModeResponse normalMode;
```



```
ReturnResponse refund;
SalesBySellerResponse sellerSales;
TipReportResponse tipReport;
DefaultSaleTypeResponse defaultSale;
ParameterReportResponse parameters;
SimReportResponse simReport;
CancelSaleResponse cancelSale;
SaleIntermediateMessageResponse saleIntermediateMessage;

if (dataProcessed != null)
{
    //Procesar un POLL
    if (dataProcessed.FunctionCode == (int)POSCommands.Function.Poll)
    {
        poll = dataProcessed;
    }

    //Procesar un SALE, LAST VOUCHER y DUPLICATE
    if (dataProcessed.FunctionCode == (int)POSCommands.Function.Sale
        || dataProcessed.FunctionCode == (int)POSCommands.Function.LastVoucher
        || dataProcessed.FunctionCode == (int)POSCommands.Function.DuplicateOthers)
    {
        sale = dataProcessed;
    }

    //Procesar un CANCELLATION
    if (dataProcessed.FunctionCode == (int)POSCommands.Function.Refund)
    {
        cancelation = dataProcessed;
    }

    //Procesar un CLOSE
    if (dataProcessed.FunctionCode == (int)POSCommands.Function.Close)
    {
        close = dataProcessed;
    }

    //Procesar un TOTALS
    if (dataProcessed.FunctionCode == (int)POSCommands.Function.Totals)
    {
        totals = dataProcessed;
    }

    //Procesar un DETAILS
    if (dataProcessed.FunctionCode == (int)POSCommands.Function.Details)
    {
        details = dataProcessed;
    }

    //Procesar un NORMAL MODE
    if (dataProcessed.FunctionCode == (int)POSCommands.Function.SetNormalMode)
    {
        normalMode = dataProcessed;
    }
}
```



```
//Procesar un REFUND
if (dataProcessed.FunctionCode == (int)POSCommands.Function.Return)
{
    refund = dataProcessed;
}

//Procesar un SELLER SALES
if (dataProcessed.FunctionCode == (int)POSCommands.Function.SalesBySeller)
{
    sellerSales = dataProcessed;
}

//Procesar un TIP REPORT
if (dataProcessed.FunctionCode == (int)POSCommands.Function.TipReport)
{
    tipReport = dataProcessed;
}

//Procesar un DEFAULT SALE
if (dataProcessed.FunctionCode == (int)POSCommands.Function.DefaultSaleType)
{
    defaultSale = dataProcessed;
}

//Procesar un PARAMS
if (dataProcessed.FunctionCode == (int)POSCommands.Function.ParameterReport)
{
    parameters = dataProcessed;
}

//Procesar un SIM REPORT
if (dataProcessed.FunctionCode == (int)POSCommands.Function.SimReport)
{
    simReport = dataProcessed;
}

//Procesar un CANCEL SALE
if (dataProcessed.FunctionCode == (int)POSCommands.Function.CancelSale)
{
    cancelSale = dataProcessed;
}

//Procesar un INTERMEDIATE MESSAGES
if (dataProcessed.FunctionCode ==
(int)POSCommands.Function.IntermediateMessages)
{
    saleIntermediateMessage = dataProcessed;
}
```

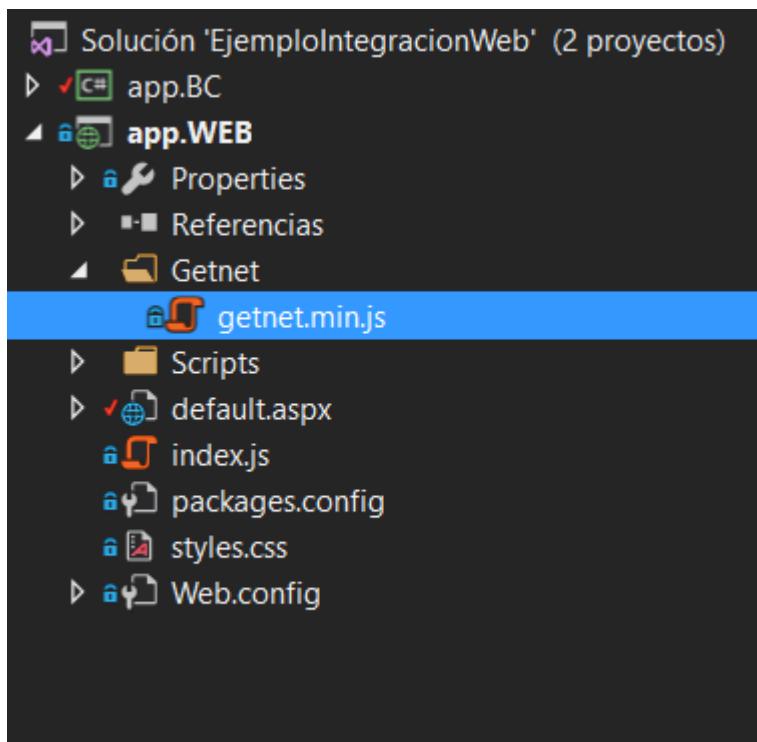


## 8 PROCESO DE INTEGRACIÓN PARA PROYECTOS WEB C# .NET Y JAVASCRIPT

En este caso hablaremos de la integración para un software de Caja desarrollado en la modalidad Web. A continuación, entregamos algunos tips importantes:

### 8.1 LIBRERIA

Lo primero es agregar la referencia de la componente **Javascript** dentro del proyecto. En este caso se agregó la librería minificada.





## 8.2 ENTENDIENDO EL PROYECTO DE EJEMPLO

Dentro del paquete de archivos viene un ejemplo de un proyecto WEB para .Net. Hemos creado un archivo llamado “**default.aspx**” que básicamente contiene lo siguiente:

1. Botones para cada uno de los comandos a enviar al POS Getnet.
2. Un control **textarea** para mostrar el resultado de la comunicación.
3. Un botón de refresh para limpiar el **textarea**.

A continuación, mostraremos ejemplos para la implementación de los botones para cada comando. Solo es necesario definir un botón del lado del frontend en HTML algo así como en este ejemplo:

```
<button id="poll_btn" type="button">Poll</button>
<button id="sale_btn" type="button">Sale</button>
<button id="last_voucher_btn" type="button">Last Voucher</button>
<button id="cancelation_btn" type="button">Cancelation</button>
<button id="close_btn" type="button">Close</button>
<button id="totals_btn" type="button">Totals</button>
<button id="details_btn" type="button">Details</button>
<button id="normal_mode_btn" type="button">Normal Mode</button>
<button id="refund_btn" type="button">Refund</button>
<button id="duplicate_btn" type="button">Duplicate</button>
<button id="seller_sales_btn" type="button">Seller Sales</button>
<button id="tips_report_btn" type="button">Tips Report</button>
<button id="default_sale_btn" type="button">Default Sale</button>
<button id="params_btn" type="button">Params</button>
<button id="sim_report_btn" type="button">Sim Report</button>
<button id="cancel_sale_btn" type="button">Cancel Sale</button>
```



### 8.3 AGREGAR LA LIBRERÍA EN UN ARCHIVO JS

Se ha creado un segundo archivo llamado “**index.js**”. En este archivo hemos agregado toda la integración requerida del lado del frontend. Para demostrar un ejemplo de cómo sería necesario realizar la integración usando Javascript. Será necesario incorporar esta línea al inicio del archivo para tener acceso a los métodos de la librería.

```
import Getnet from '/Getnet/getnet.min.js';
```

### 8.4 CREAR VARIABLES PARA ASOCIAZLOS CON LOS BOTÓNES DEL FRONTEND

Aquí creamos una relación entre el botón en HTML y el Javascript para luego agregarles la funcionalidad que queremos.

```
// Declaración de variables
var poll_btn
    , sale_btn
    , last_voucher_btn
    , cancelation_btn
    , close_btn
    , totals_btn
    , details_btn
    , normal_mode_btn
    , refund_btn
    , duplicate_btn
    , seller_sales_btn
    , tips_report_btn
    , default_sale_btn
    , params_btn
    , sim_report_btn
    , cancel_sale_btn
    , refresh_btn;
```

### 8.5 CONTROLAR LOS LISTENERS

Luego se deben crear acciones que conectarán la funcionalidad a los botones:

```
// Declaración de listeners
document.addEventListener("DOMContentLoaded", () => {
    setControls();
    setListeners();
});

// Relación del botón con su variable correspondiente
function setControls() {
    poll_btn = document.querySelector("#poll_btn");
    sale_btn = document.querySelector("#sale_btn");
    last_voucher_btn = document.querySelector("#last_voucher_btn");
    cancelation_btn = document.querySelector("#cancelation_btn");
    close_btn = document.querySelector("#close_btn");
    totals_btn = document.querySelector("#totals_btn");
    details_btn = document.querySelector("#details_btn");
```



```
normal_mode_btn = document.querySelector("#normal_mode_btn");
refund_btn = document.querySelector("#refund_btn");
duplicate_btn = document.querySelector("#duplicate_btn");
seller_sales_btn = document.querySelector("#seller_sales_btn");
tips_report_btn = document.querySelector("#tips_report_btn");
default_sale_btn = document.querySelector("#default_sale_btn");
params_btn = document.querySelector("#params_btn");
sim_report_btn = document.querySelector("#sim_report_btn");
cancel_sale_btn = document.querySelector("#cancel_sale_btn");
refresh_btn = document.querySelector("#refresh_btn");
}

// Implementación de los manejadores
function setListeners() {
    poll_btn.addEventListener("click", handlePoll);
    sale_btn.addEventListener("click", handleSale);
    last_voucher_btn.addEventListener("click", handleLastVoucher);
    cancelation_btn.addEventListener("click", handleRefund);
    close_btn.addEventListener("click", handleClose);
    totals_btn.addEventListener("click", handleTotals);
    details_btn.addEventListener("click", handleDetails);
    normal_mode_btn.addEventListener("click", handleSetNormalMode);
    refund_btn.addEventListener("click", handleReturn);
    duplicate_btn.addEventListener("click", handleDuplicateOthers);
    seller_sales_btn.addEventListener("click", handleSalesBySeller);
    tips_report_btn.addEventListener("click", handleTipReport);
    default_sale_btn.addEventListener("click", handleDefaultSaleType);
    params_btn.addEventListener("click", handleParameterReport);
    sim_report_btn.addEventListener("click", handleSimReport);
    cancel_sale_btn.addEventListener("click", handleCancelSale);
    refresh_btn.addEventListener("click", limpiarTextArea);
}
```

## 8.6 ENVIAR SOLICITUDES AL POS

El envío de comandos al POS Getnet es tan simple como llamar a un método y la librería se encarga de todo el procesamiento de los datos para construir el JSON, firmar los datos y enviar el mensaje.

### POLLING:

```
function handlePoll() {
    Getnet.SetCallback(pollResponse);
    Getnet.Poll();
}
```

### SALE:

```
function handleSale() {
    Getnet.SetCallback(saleResponse);
    Getnet.Sale(15000, 1234, false, Getnet.POSCommands.Function.Sale, true, 1);
}
```

**LAST VOUCHER:**

```
function handleLastVoucher() {  
    Getnet.SetCallback(lastVoucherResponse);  
    Getnet.LastVoucher();  
}
```

**CANCELLATION:**

```
function handleRefund() {  
    Getnet.SetCallback(refundResponse);  
    Getnet.Refund(1234);  
}
```

**CLOSE:**

```
function handleClose() {  
    Getnet.SetCallback(closeResponse);  
    Getnet.Close();  
}
```

**TOTALS:**

```
function handleTotals() {  
    Getnet.SetCallback(totalsResponse);  
    Getnet.Totals();  
}
```

**DETAILS:**

```
function handleDetails() {  
    Getnet.SetCallback(detailsResponse);  
    Getnet.Details();  
}
```

**NORMAL MODE:**

```
function handleSetNormalMode() {  
    Getnet.SetCallback(setNormalModeResponse);  
    Getnet.SetNormalMode();  
}
```

**REFUND:**

```
function handleReturn() {  
    Getnet.SetCallback(returnResponse);  
    Getnet.Return(1234, 25000);  
}
```

**DUPLICATE:**

```
function handleDuplicateOthers() {  
    Getnet.SetCallback(duplicateOthersResponse);  
    Getnet.DuplicateOthers(1234);  
}
```

**SELLER SALES:**

```
function handleSalesBySeller() {  
    Getnet.SetCallback(salesBySellerResponse);  
    Getnet.SalesBySeller(1);  
}
```

**TIP REPORT:**

```
function handleTipReport() {  
    Getnet.SetCallback(tipReportResponse);  
    Getnet.TipReport(1);  
}
```

**DEFAULT SALE:**

```
function handleDefaultSaleType() {  
    Getnet.SetCallback(defaultSaleTypeResponse);  
    Getnet.DefaultSaleType(Getnet.POSCommands.Function.DefaultSaleType);  
}
```

**PARAMS:**

```
function handleParameterReport() {  
    Getnet.SetCallback(parameterReportResponse);  
    Getnet.ParameterReport();  
}
```

**SIM REPORT:**

```
function handleSimReport() {  
    Getnet.SetCallback(simReportResponse);  
    Getnet.SimReport();  
}
```

**CANCEL SALE:**

```
function handleCancelSale() {  
    Getnet.SetCallback(cancelSaleRequestResponse);  
    Getnet.CancelSale();  
}
```



## 8.7 LEER LAS RESPUESTAS DEL POS

Para leer las respuestas desde el POS Getnet solo será necesario implementar un método que quede en espera del **RESPONSE** y permita controlar esto por cada comando asignando al **callback** de la comunicación en métodos ya definidos en la sección de envío de solicitudes de la siguiente manera:

```
//Control de las respuestas
function postMethod(csMethod, data) {
    if (!data.Received)
        ApiCall(csMethod, data);
}

function ApiCall(csMethod, body) {
    fetch('default.aspx/' + csMethod, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({ data: body })
    })
        .then(response => {
            if (!response.ok) {
                throw new Error('Error de red o respuesta no exitosa');
            }
            return response.json();
        })
        .then(res => {
            console.log(res);
        })
        .catch(error => {
            console.error('Hubo un problema con la petición fetch:', error.message);
        });
}
```

Finalmente, por cada comando se debe enviar la respuesta al backend del archivo “**default.aspx**” de manera que los datos puedan ser procesados en el servidor.

### POLLING:

```
function pollResponse(response) {
    postMethod('PollResponse', response);
}
```

### SALE:

```
function saleResponse(response) {
    postMethod('SaleResponse', response);
}
```

**LAST VOUCHER:**

```
function lastVoucherResponse(response) {  
    postMethod('LastVoucherResponse', response);  
}
```

**CANCELLATION:**

```
function refundResponse(response) {  
    postMethod('RefundResponse', response);  
}
```

**CLOSE:**

```
function closeResponse(response) {  
    postMethod('CloseResponse', response);  
}
```

**TOTALS:**

```
function totalsResponse(response) {  
    postMethod('TotalsResponse', response);  
}
```

**DETAILS:**

```
function detailsResponse(response) {  
    postMethod('DetailsResponse', response);  
}
```

**NORMAL MODE:**

```
function setNormalModeResponse(response) {  
    postMethod('SetNormalModeResponse', response);  
}
```

**REFUND:**

```
function returnResponse(response) {  
    postMethod('ReturnResponse', response);  
}
```

**DUPLICATE:**

```
function duplicateOthersResponse(response) {  
    postMethod('DuplicateOthersResponse', response);  
}
```

**SELLER SALES:**

```
function salesBySellerResponse(response) {  
    postMethod('SalesBySellerResponse', response);  
}
```

**TIP REPORT:**

```
function tipReportResponse(response) {  
    postMethod('TipReportResponse', response);  
}
```

**DEFAULT SALE:**

```
function defaultSaleTypeResponse(response) {  
    postMethod('DefaultSaleTypeResponse', response);  
}
```

**PARAMS:**

```
function parameterReportResponse(response) {  
    postMethod('ParameterReportResponse', response);  
}
```

**SIM REPORT:**

```
function simReportResponse(response) {  
    postMethod('SimReportResponse', response);  
}
```

**CANCEL SALE:**

```
function cancelSaleResponse(response) {  
    postMethod('CancelSaleResponse', response);  
}
```

**NOTA:** No es necesario leer el resultado del puerto serial y deserializarlo para procesarlo de forma manual, la librería ya hace esto.

## 8.8 PROCESAMIENTO DE LA RESPUESTA EN EL BACKEND

En el archivo “**default.aspx.cs**” hemos dejado un método por cada comando que permita procesar los datos de respuesta llamando a la componente **.DLL** dentro de un proyecto BC así:

```
[WebMethod]  
public static void PollResponse(object data)  
{  
    BC.IntegracionGetnet.CommandHandler.Instance.HandlePollAnswer(data);  
}
```

**NOTA:** Esta es la forma de procesar la respuesta del comando **POLL**. Para el resto de los comandos por favor revisar el proyecto WEB de ejemplo.



## 8.9 DESERIALIZACIÓN DE LOS DATOS

Luego dentro del proyecto **app.BC** hay un namespace llamado **IntegracionGetnet** y dentro hay una clase llamada **CommandHandler**. Para cada comando hemos dejado un procesamiento de datos separado, que sirve de ejemplo:

```
public void HandlePollAnswer(object jsonResponse)
{
    try
    {
        ResponseGetnet resonseGetnet = DeserializarResponse(jsonResponse);
        PollResponse res =
JsonConvert.DeserializeObject<PollResponse>(resonseGetnet.JsonSerialized);
    }
    catch (Exception ex)
    {
        throw new Exception(ex.Message);
    }
}

//-----
/// <summary>
/// Deserialización de la respuesta.
/// </summary>
/// <param name="jsonResponse">Json de la respuesta.</param>
/// <returns>Modelo de Datos.</returns>
private ResponseGetnet DeserializarResponse(object jsonResponse)
{
    return
JsonConvert.DeserializeObject<ResponseGetnet>(JsonConvert.SerializeObject(jsonResponse
));
}

/// <summary>
/// Modelo de Datos.
/// </summary>
class ResponseGetnet
{
    public string JsonSerialized { get; set; }
    public string Sign { get; set; }
}
```

De esta manera se obtiene un objeto por cada tipo de respuesta de manera que el Software de Caja pueda utilizar los datos necesarios para procesar las operaciones de venta y finalizar con la integración.



## 8.10 PROCESO ESPECIAL PARA EL NAVEGADOR FIREFOX

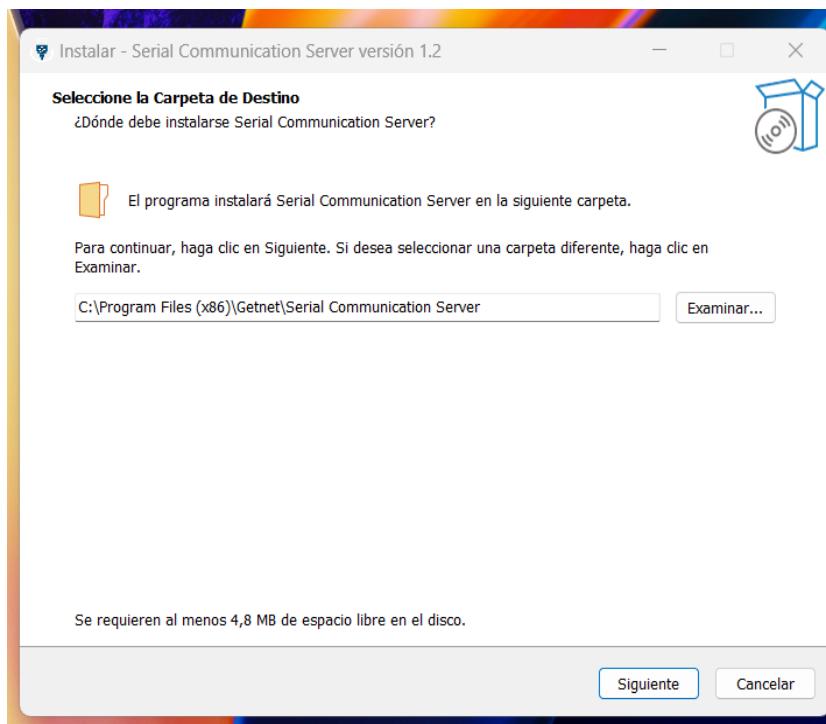
Para los usuarios que utilicen el navegador Firefox hay una excepción respecto al uso de la comunicación Serial COM, dado que por motivos de seguridad este navegador particular prohíbe el uso de esta tecnología y en este sentido hemos desarrollado un Software especial de comunicación serial intermedio de la librería Getnet Javascript para Firefox.

## 8.11 PROCEDIMIENTO DE INSTALACIÓN

1. Ejecutar instalador “Serial Communication Setup”

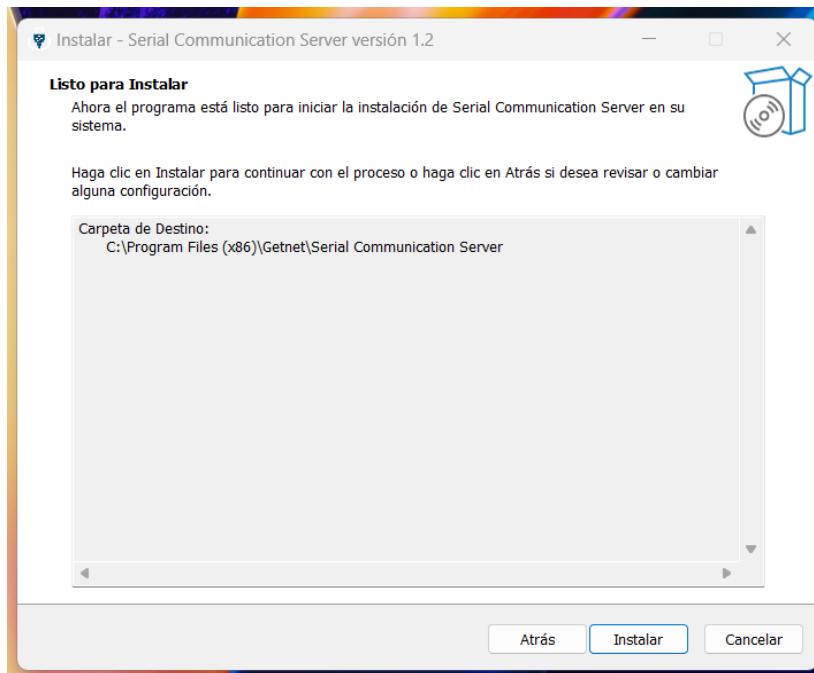


2. En este paso, por defecto el programa se instalará en la carpeta **C:\Program Files (x86)\Getnet\Serial Communication Server** (Recomendado)  
Presionar Siguiente.

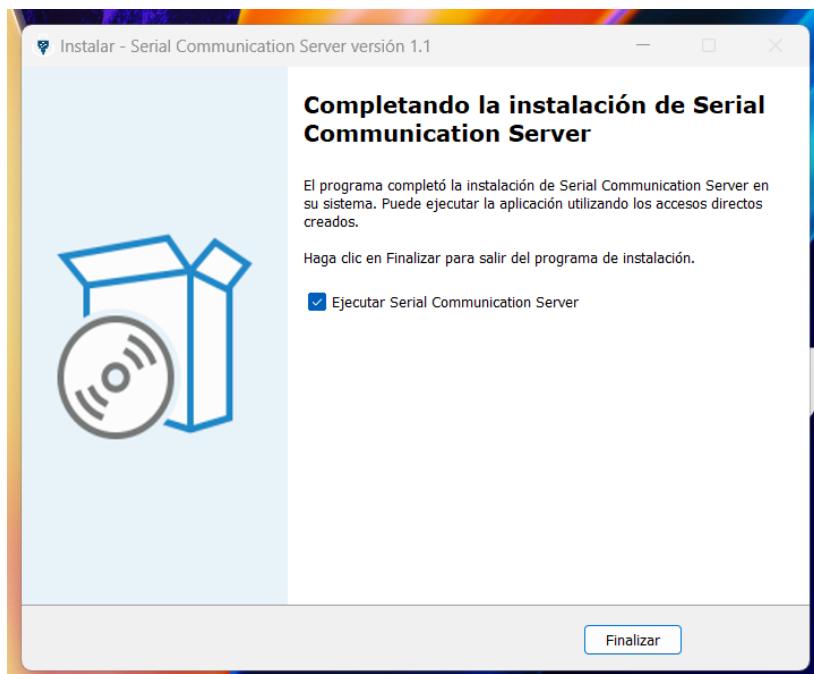




3. En este paso muestra un resumen y un mensaje de “listo para instalar”, solo se debe presionar “Instalar”

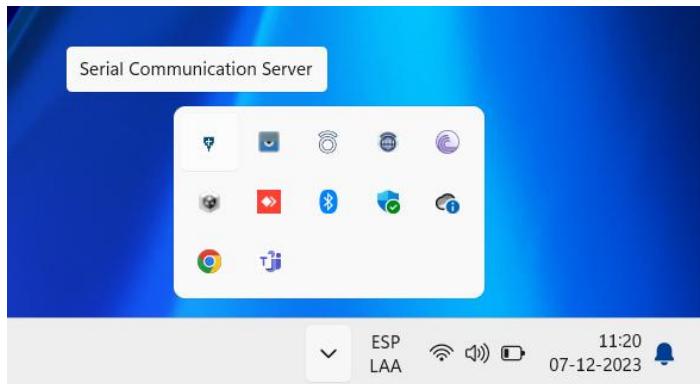


4. Finalmente, el mensaje de instalación exitosa. Recomendado dejar la casilla “Ejecutar Serial Communication Server” seleccionada para iniciar automáticamente el programa.





- Luego revisar si el programa fue iniciado correctamente, en la sección oculta de las aplicaciones en la barra de tareas de Windows, es el ícono de software primero a la izquierda en la imagen, al posicionarse con el cursor encima del ícono debería desplegarse el nombre "Serial Communication Server".



- Finalmente debemos asegurarnos de que el programa esté en los programas de inicio de Windows. Para ello debemos ir a configuración de Windows, luego a aplicaciones, presionar inicio (aplicaciones que se inician automáticamente sin iniciar sesión) y ver que esté registrado el programa ahí.

Aplicaciones > Inicio

En la mayoría de los casos, se iniciarán minimizadas o solo se podrán iniciar como una tarea en segundo plano. Ordenar por: Nombre

Nombre	Estado	Opciones
AnyDesk	Activado	<input checked="" type="checkbox"/>
BitTorrent Web	Activado	<input checked="" type="checkbox"/>
CommunicationServer	Activado	<input checked="" type="checkbox"/>
Google Chrome	Activado	<input checked="" type="checkbox"/>
Intel® Graphics Command Center Startup Task	Desactivado	<input type="checkbox"/>
Microsoft Defender	Desactivado	<input type="checkbox"/>
Microsoft Edge	Desactivado	<input type="checkbox"/>
Microsoft OneDrive	Activado	<input checked="" type="checkbox"/>
Microsoft Teams	Desactivado	<input type="checkbox"/>

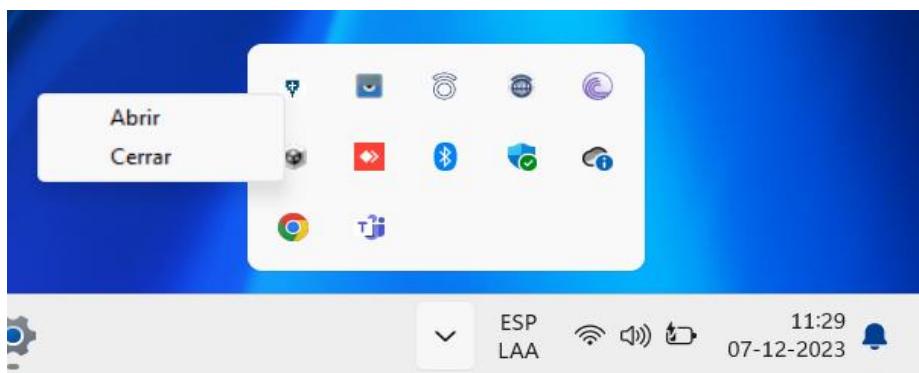
En caso de que no esté debemos activarlo, así el programa se iniciará automáticamente al iniciar Windows.



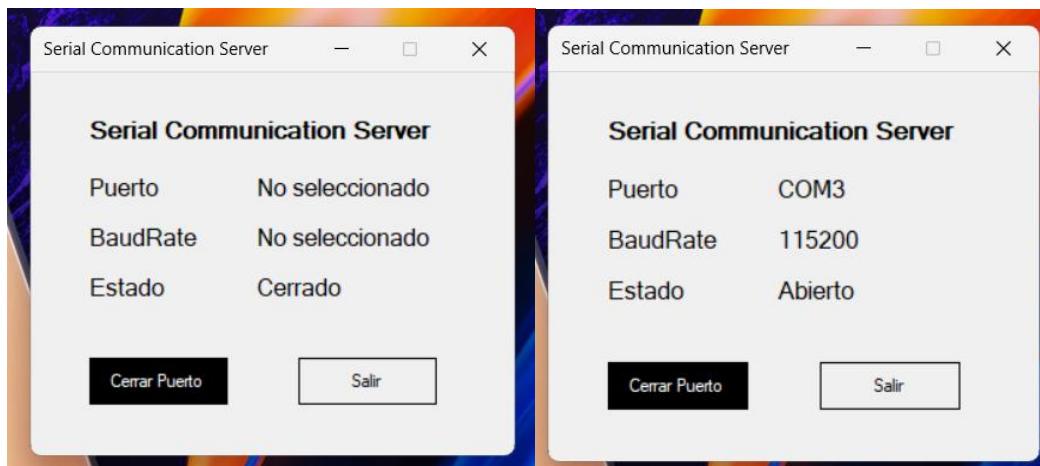
## 8.12 FUNCIONAMIENTO DEL SERIAL COMMUNICATION SERVER

El software es un programa que se conecta a la librería de JavaScript vía websocket y sirve de intermediario de comunicación serial entre la librería y el POS Getnet.

1. Al hacer click derecho se desplegará una pequeña venta donde hay dos opciones “Abrir” y “Cerrar”.



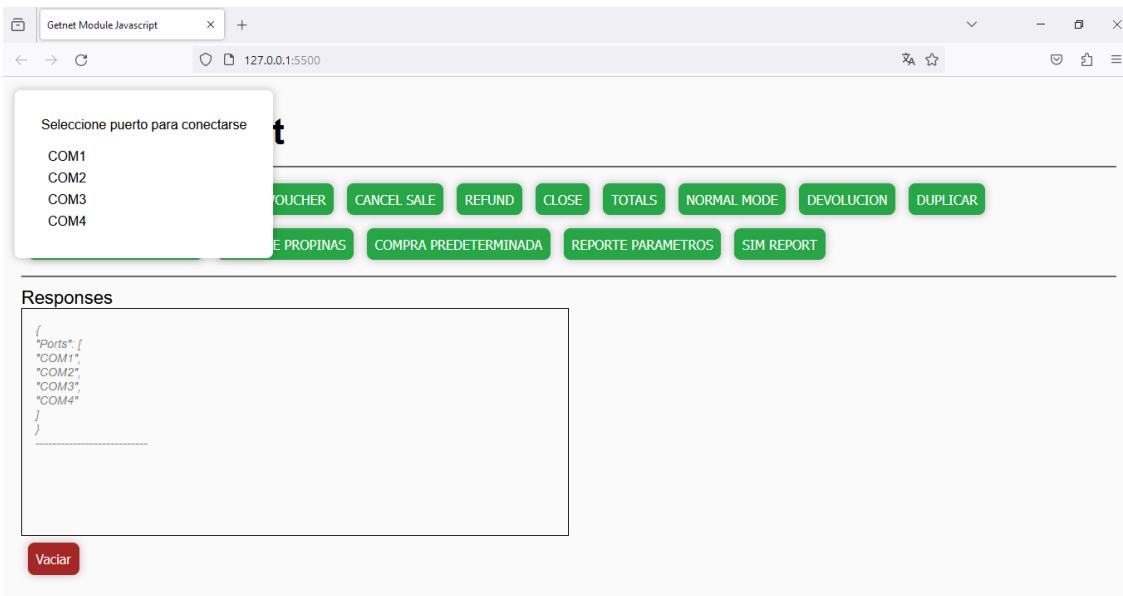
2. Al presionar “Abrir”, se abrirá la ventana.
3. La ventana contiene los siguientes datos:
  - a. Puerto: Puerto COM seleccionado en el front end a través de la librería Getnet Javascript.
  - b. BaudRate: Velocidad de transmisión de los datos.
  - c. Estado: Estado del puerto COM, puede estar “Abierto” o “Cerrado”.



4. El botón “Cerrar Puerto” cierra el puerto y se detiene la comunicación (recomendado no cerrar el puerto, mantener siempre el puerto abierto).
5. El botón “Salir” oculta la aplicación en los iconos ocultos.
6. Al minimizar la ventana queda en la barra de tareas (no recomendado).
7. Al cerrar la ventana se cierra el programa (no recomendado).
8. En caso de cerrar la aplicación debe volver a inicializarla en la carpeta donde fue instalada: “C:\Program Files (x86)\Serial Communication Server\CommunicationServer.exe”

## 8.13 FUNCIONAMIENTO EN FIREFOX

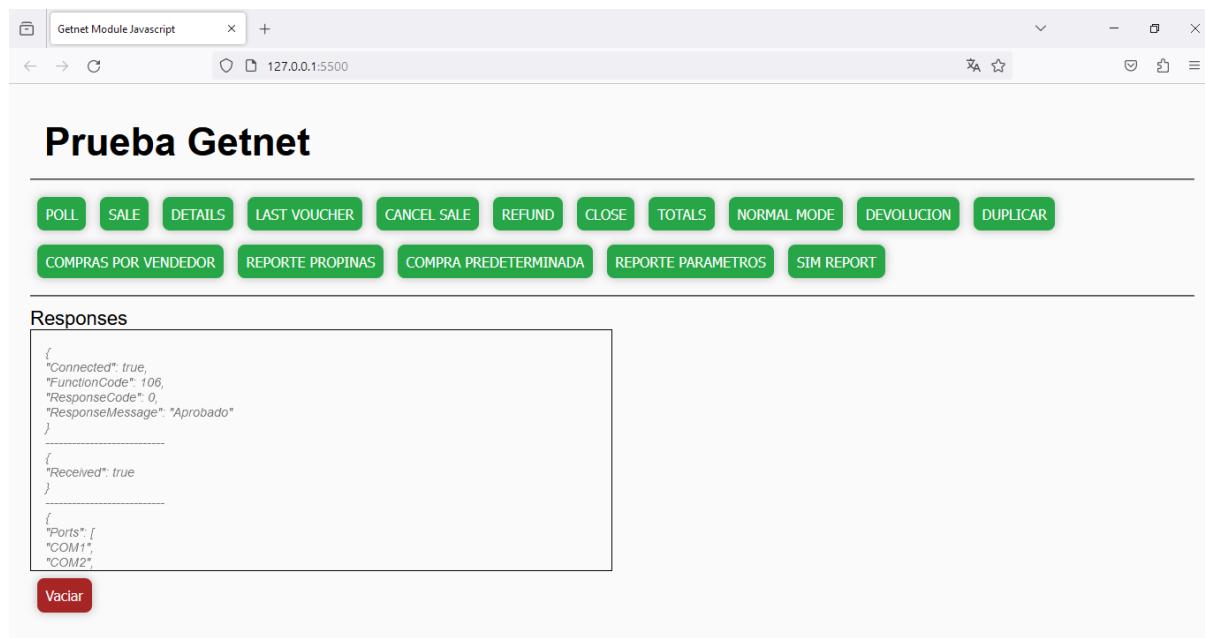
Al presionar cualquier botón primero la componente pedirá seleccionar un puerto, donde se debe presionar el puerto deseado.



The screenshot shows a Firefox browser window with the title "Getnet Module Javascript". The address bar displays "127.0.0.1:5500". A modal dialog box is open, prompting the user to "Seleccione puerto para conectarse" (Select port to connect). It lists four options: COM1, COM2, COM3, and COM4. Below the list are several green buttons: VOUCHER, CANCEL SALE, REFUND, CLOSE, TOTALS, NORMAL MODE, DEVOLUCION, DUPLICAR, REPROPIAS, COMPA PREDETERMINADA, REPORTE PARAMETROS, and SIM REPORT. At the bottom of the dialog is a red "Vaciar" button. The main content area is titled "Responses" and contains a JSON object:

```
{
  "Ports": [
    "COM1",
    "COM2",
    "COM3",
    "COM4"
  ]
}
```

Entonces la conexión debe estar funcionando como para poder presionar cualquiera de los comandos en pantalla.



The screenshot shows a Firefox browser window with the title "Prueba Getnet". The address bar displays "127.0.0.1:5500". The interface features a header with various green buttons: POLL, SALE, DETAILS, LAST VOUCHER, CANCEL SALE, REFUND, CLOSE, TOTALS, NORMAL MODE, DEVOLUCION, DUPLICAR, COMPRAS POR VENDEDOR, REPORTE PROPIAS, COMPA PREDETERMINADA, REPORTE PARAMETROS, and SIM REPORT. Below the header is a section titled "Responses" containing a JSON object:

```
{
  "Connected": true,
  "FunctionCode": 106,
  "ResponseCode": 0,
  "ResponseMessage": "Aprobado"
}

{
  "Received": true
}

{
  "Ports": [
    "COM1",
    "COM2"
  ]
}
```

At the bottom of this section is a red "Vaciar" button.



## 9 PROCESO DE INTEGRACIÓN PARA PROYECTOS JAVA

Lo primero es realizar las importaciones necesarias de los archivos entregados en el SDK, aquí exponemos todos los necesarios, pero depende, si requerirás integrar todos los comandos quizás solo importes los necesarios y no todo.

```
import com.fazecast.jSerialComm.SerialPort;
import com.fazecast.jSerialComm.SerialPortDataListener;
import com.fazecast.jSerialComm.SerialPortEvent;
import java.util.logging.Level;
import java.util.logging.Logger;
import posintegradogetnet.POSCommands;
import posintegradogetnet.POSCommands.Function;
import posintegradogetnet.POSIntegrado;
import posintegradogetnet.Utils;
import posintegradogetnet.exceptions.CloseException;
import posintegradogetnet.exceptions.CustomException;
import posintegradogetnet.exceptions.RefundException;
import posintegradogetnet.exceptions.ReturnException;
import posintegradogetnet.exceptions.SaleException;
import posintegradogetnet.exceptions.TotalsException;
import posintegradogetnet.responses.CancelSaleResponse;
import posintegradogetnet.responses.CloseResponse;
import posintegradogetnet.responses.DefaultSaleTypeResponse;
import posintegradogetnet.responses.DetailsResponse;
import posintegradogetnet.responses.DuplicateOthersResponse;
import posintegradogetnet.responses.LastVoucherResponse;
import posintegradogetnet.responses.ParameterReportResponse;
import posintegradogetnet.responses.PollResponse;
import posintegradogetnet.responses.RefundResponse;
import posintegradogetnet.responses.ReturnResponse;
import posintegradogetnet.responses.SaleIntermediateMessageResponse;
import posintegradogetnet.responses.SaleResponse;
import posintegradogetnet.responses.SalesBySellerResponse;
import posintegradogetnet.responses.SetNormalModeResponse;
import posintegradogetnet.responses.SimReportResponse;
import posintegradogetnet.responses.TipReportResponse;
import posintegradogetnet.responses.TotalsResponse;
```

### 9.1 DECLARACIÓN DE VARIABLES SEGÚN EL COMANDO

Lo segundo es crear las variables que serán necesarias según sea el caso. Aquí exponemos todos los comandos.

```
//Control de la conexión Serial.
private static SerialPort serialPort;

//Variables para recibir respuestas desde el POS Getnet.
private static SaleResponse saleCommand;
private static LastVoucherResponse lastVoucherCommand;
private static RefundResponse refundCommand;
private static CloseResponse closeCommand;
private static TotalsResponse totalsCommand;
```



```
private static DetailsResponse detailsCommand;
private static PollResponse pollCommand;
private static SetNormalModeResponse setNormalModeCommand;
private static ReturnResponse returnCommand;
private static DuplicateOthersResponse duplicateOthersCommand;
private static SalesBySellerResponse salesBySellerCommand;
private static TipReportResponse tipReportCommand;
private static DefaultSaleTypeResponse defaultSaleTypeCommand;
private static ParameterReportResponse parameterReportCommand;
private static SimReportResponse simReportCommand;
private static CancelSaleResponse cancelSaleCommand;
private static SaleIntermediateMessageResponse
saleIntermediateMessageCommand;
```

## 9.2 CONECTAR AL PUERTO SERIAL

Para conectarse al puerto serial COM disponible se pueden realizar de la siguiente manera:

**EN DESARROLLO:** Se puede definir en duro el puerto de comunicaciones y la velocidad de transmisión de datos (BaudRate) para hacer las pruebas.

```
try
{
    //Conexión por USB
    POSIntegrado.getInstance().usbConnect("COM4", 115200);

} catch (CustomException ex) {
    System.err.println("Error: " + ex.getMessage());
}
```

**EN PRODUCCIÓN:** Se requerirá la configuración desde el Agente POS. En este caso no se define ni el nombre del puerto ni la velocidad de transmisión de datos (BaudRate) dado que la componente la va a buscar a la ruta de la configuración del Agente POS (Ver punto 4.14.)

```
try
{
    //Conexión por USB
    POSIntegrado.getInstance().usbConnect();

} catch (CustomException ex) {
    System.err.println("Error: " + ex.getMessage());
}
```

**NOTA:** Estas son solo recomendaciones. Si usted no quiere usar el Agente POS puede usar la misma configuración del ambiente de desarrollo y validar que el nombre del puerto en la Caja será el mismo que usted definió en su código.



### 9.3 ESCUCHAR LA COMUNICACIÓN POR EL PUERTO SERIAL

Se debe definir un listener para obtener todas las respuestas del POS y de esta manera recibir los objetos ya procesados para utilizarlos en la integración:

```
serialPort = POSIntegrado.getInstance().getnetSerialPort();
serialPort.addDataListener(new SerialPortDataListener() {
    @Override
    public void serialEvent(SerialPortEvent spe) {
        if (spe.getEventType() == SerialPort.LISTENING_EVENT_DATA_AVAILABLE) {
            try {
                //Procesamiento de las respuestas desde el POS Getnet.
                String dataReceived = POSIntegrado.getInstance().dataReceived();

                String dataToCheck = "{\"Received\":true}";
                if (dataReceived.contains(dataToCheck)) {
                    Utils.getInstance().stopTimerForConnection();
                }
                Object dataProcessed = POSIntegrado.getInstance().processData(dataReceived);
                if (dataProcessed != null)
                {
                    //Casteo de la respuesta segun el comando enviado.
                    Function function = POSCommands.GetResponseClass(
                        dataProcessed.getClass().getName());
                    switch(function) {
                        case SALE:
                            saleCommand = (SaleResponse)dataProcessed;
                            break;
                        case LAST_VOUCHER:
                            lastVoucherCommand = (LastVoucherResponse)dataProcessed;
                            break;
                        case REFUND:
                            refundCommand = (RefundResponse)dataProcessed;
                            break;
                        case CLOSE:
                            closeCommand = (CloseResponse)dataProcessed;
                            break;
                        case TOTALS:
                            totalsCommand = (TotalsResponse)dataProcessed;
                            break;
                        case DETAILS:
                            detailsCommand = (DetailsResponse)dataProcessed;
                            break;
                        case POLL:
                            pollCommand = (PollResponse)dataProcessed;
                            break;
                        case SET_NORMAL_MODE:
                            setNormalModeCommand = (SetNormalModeResponse)dataProcessed;
                            break;
                        case RETURN:
                            returnCommand = (ReturnResponse)dataProcessed;
                            break;
                        case DUPLICATE_OTHERS:
                            duplicateOthersCommand = (DuplicateOthersResponse)dataProcessed;
                            break;
                        case SALES_BY_SELLER:
                            salesBySellerCommand = (SalesBySellerResponse)dataProcessed;
                            break;
                        case TIP_REPORT:
                            tipReportCommand = (TipReportResponse)dataProcessed;
                            break;
                        case DEFAULT_SALE_TYPE:
                            defaultSaleTypeCommand = (DefaultSaleTypeResponse)dataProcessed;
                            break;
                    }
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
});
```



```
        case PARAMETER_REPORT:
            parameterReportCommand = (ParameterReportResponse)dataProcessed;
            break;
        case SIM_REPORT:
            simReportCommand = (SimReportResponse)dataProcessed;
            break;
        case CANCEL_SALE:
            cancelSaleCommand = (CancelSaleResponse)dataProcessed;
            break;
        case SALE_INTERMEDIATE_MESSAGE:
            saleIntermediateMessageResponseCommand =
(SaleIntermediateMessageResponse)dataProcessed;
            break;
        }
        dispose();
    }
} catch (CustomException ex) {
    System.err.println("Error: " + ex.getMessage());
}
}
}

@Override
public int getListeningEvents() {
    return SerialPort.LISTENING_EVENT_DATA_AVAILABLE |
    SerialPort.LISTENING_EVENT_FRAMING_ERROR;
}
});
```

#### 9.4 ENVIAR SOLICITUDES AL POS

El envío de comandos al POS Getnet es tan simple como llamar a un método y la librería se encarga de todo el procesamiento de los datos para construir el JSON, firmar los datos y enviar el mensaje.

##### POLLING:

```
POSIntegrado.getInstance().poll();
```

##### SALE:

```
POSIntegrado.getInstance().sale(
    12000, "1234", false, POSCommands.SaleType.SALE, false, 1, 60);
```

##### LAST VOUCHER:

```
POSIntegrado.getInstance().lastVoucher();
```

##### CANCELLATION:

```
POSIntegrado.getInstance().refund(60);
```

##### CLOSE:

```
POSIntegrado.getInstance().close(false, 60);
```

##### TOTALS:

```
POSIntegrado.getInstance().totals();
```

**DETAILS:**

```
POSIntegrado.getInstance().details();
```

**NORMAL MODE:**

```
POSIntegrado.getInstance().setNormalMode();
```

**REFUND:**

```
POSIntegrado.getInstance().returns("XZ123456", 12000);
```

**DUPLICATE:**

```
POSIntegrado.getInstance().duplicateOthers(60);
```

**SELLER SALES:**

```
POSIntegrado.getInstance().salesBySeller(1);
```

**TIP REPORT:**

```
POSIntegrado.getInstance().tipReport(1);
```

**DEFAULT SALE:**

```
POSIntegrado.getInstance().defaultSaleType(POSCommands.SaleType.SALE);
```

**PARAMS:**

```
POSIntegrado.getInstance().parameterReport();
```

**SIM REPORT:**

```
POSIntegrado.getInstance().simReport();
```

**CANCEL SALE:**

```
POSIntegrado.getInstance().cancelSale();
```



## 9.5 CONTROL DE TIMEOUTS

Estos métodos permiten mostrar un mensaje de alerta al usuario de la Caja o bien controlarlo de manera inteligente dentro del software de Caja para indicar que se agotó el tiempo de espera y que el POS no está conectado o bien superó el tiempo de espera para responder el comando solicitado.

```
Utils.getInstance().setTimerForConnection(4);
Utils.getInstance().setListenerForConnection(() -> {
    if (Utils.timerForConnection != null) {
        Utils.getInstance().stopTimerForPosResponse();
        System.err.println("Error: There is no connection to the POS.");
    }
});

Utils.getInstance().setListenerForPosResponse(() -> {
    if (Utils.timerForPosResponse != null)
        System.err.println("Error: The POS has timed out waiting for the
response.");
});
```

## 9.6 CONTROL DE LA COMUNICACIÓN

Una recomendación sabia es que al finalizar la comunicación puedas cerrar los objetos que controla el puerto serial y de esta manera usar de forma asertiva los recursos del PC de la Caja. Este método se llama justo después de la recepción de las respuestas y lo que hace es finalizar la comunicación tras la llamada de cada método o comando enviado al POS.

```
private static void dispose() throws CustomException {
    try {
        POSIntegrado.getInstance().disposePort();
        serialPort.flushIOBuffers();
        serialPort.closePort();
        serialPort = null;
    } catch (CustomException ex) {
        Logger.getLogger(SimuladorDeCajaGetnet.class.getName())
            .log(Level.SEVERE, null, ex);
    }
}
```

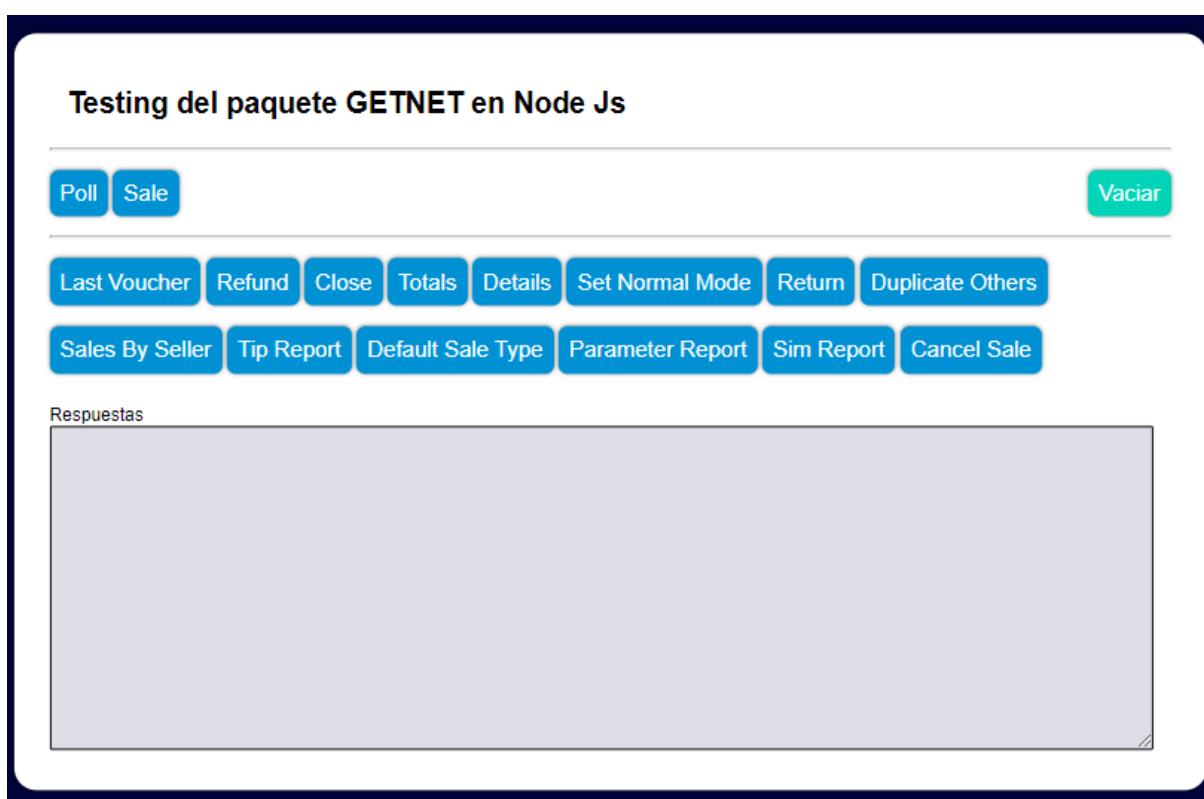


## 10 PROCESO DE INTEGRACIÓN PARA PROYECTOS NODE.JS

Antes de realizar la prueba del Simulador de Caja primero se deben realizar los pasos de habilitación de puertos COM, validación de configuración de puertos COM y realizar los pasos del agente POS, para proceder a realizar las pruebas y entender el flujo.

### 10.1 INICIAR SIMULADOR DE POS Y CAJA

Para realizar las pruebas iniciales debemos abrir el Simulador de Caja en Node.JS (en modo depuración con el código de Ejemplo utilizando “**node index.js**” o “**nodemon index.js**” o “**node --watch index.js**”) y conectarlo en el otro puerto disponible para establecer la conexión. Luego abrir en el navegador la url **http://localhost:3000**. Y luego el Simulador de POS o bien el POS físico para que pueda responder a las pruebas.

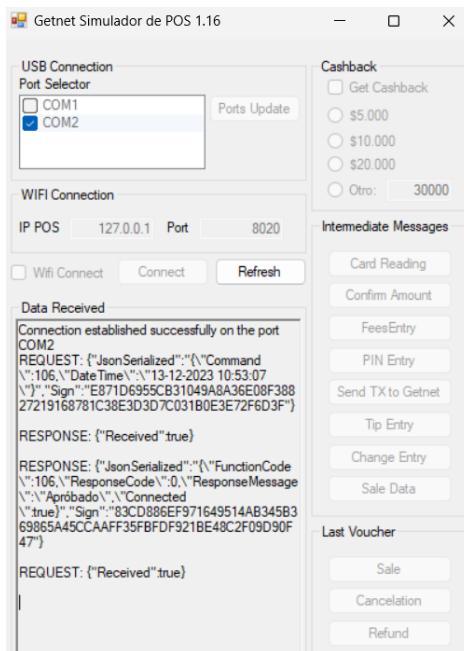


En este caso solo bastaría pulsar los botones para validar las funcionalidades respectivas.



## 10.2 PROBAR FUNCIONAMIENTO DE COMANDOS

Presionar el botón **Poll** para enviar comando de conexión. Al recibir el “**RESPONSE**” la conexión se ha realizado con éxito, y puede probar todos los botones.



### Testing del paquete GETNET en Node Js

**Poll**
**Sale**
**Vaciar**

Last Voucher
Refund
Close
Totals
Details
Set Normal Mode
Return
Duplicate Others

Sales By Seller
Tip Report
Default Sale Type
Parameter Report
Sim Report
Cancel Sale

**Respuestas**

```

2. Mensaje Recibido: 28/12/2023 10:03:59
{
  "FunctionCode": 106,
  "ResponseCode": 0,
  "ResponseMessage": "Aprobado",
  "Connected": true
}

1. Mensaje Recibido: 28/12/2023 10:03:59
{
  "Received": true
}
  
```

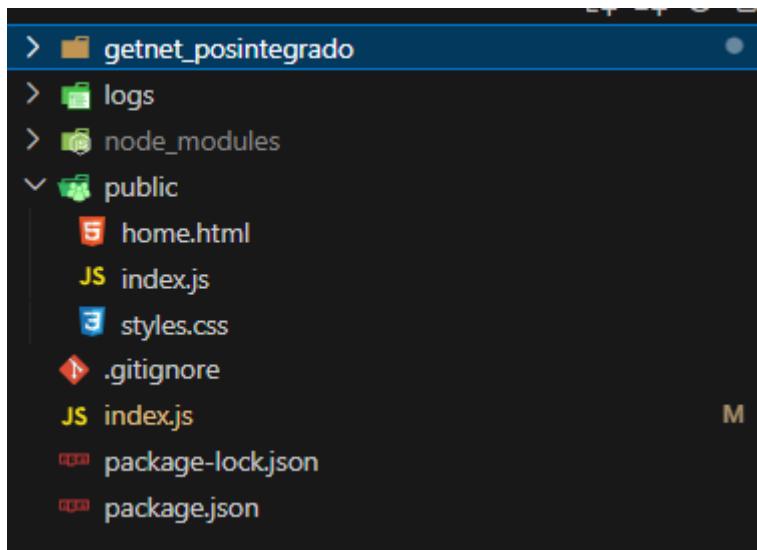


### 10.3 PROCESO DE INTEGRACIÓN

Para el proceso de integración se entrega el proyecto de Simulador de Caja de manera que se puede revisar en su totalidad y así tener un ejemplo claro de cómo se puede modificar la aplicación de caja para incorporar la integración con Getnet.

### 10.4 LIBRERIA

El primer paso es agregar la carpeta de librería en la raíz (puede ser agregada en cualquier carpeta excepto dentro de la carpeta node\_modules porque no es un módulo de Node.JS).



### 10.5 AGREGAR LA LIBRERÍA EN LA SECCIÓN REQUIRE

```
const PosIntegrado = require('./getnet_posintegrado');  
const POSCommands = require('./getnet_posintegrado/lib/PosCommands');
```



## 11 PROCESO DE INTEGRACIÓN PARA PROYECTOS PHP

Es un lenguaje de Scripting, y se ejecuta en el servidor. Dependemos de otro lenguaje que se ejecute desde el lado del navegador del usuario para poder ejecutar el código de PHP.

### 11.1 DESDE JAVASCRIPT

Necesitamos utilizar el comando fetch con la ruta donde se encuentra nuestro código PHP que utilizará la interfaz del PostIntegrado. Además, se deben enviar ciertos parámetros para que puedan ser leídos por nuestro Backend, estos son:

- **Action:** Corresponde al comando ejecutado desde el navegador del usuario.
- **Data:** información adicional que corresponde a los datos que se utilizan en cada comando.

Ejemplo:

```
fetch('http://localhost/getnet/php/index.php', {
  mode: 'no-cors',
  method: 'POST',
  headers: {
  },
  body: JSON.stringify({
    action: csMethod,
    data: body.JsonSerialized
  })
})
.then(response => {
  if (!response.ok) {
    throw new Error('Error de red o respuesta no exitosa');
  }
  return response.json();
})
.then(res => {
  console.log(res);
})
.catch(error => {
  console.error('Hubo un problema con la petición fetch:', error.message);
});
```

### 11.2 DESDE PHP

Se debe copiar la estructura de archivos definida para ello. La cual contiene los archivos necesarios para procesar la información entregada por el POS

src/Exceptions

src/Responses

POSIntegrado.php



Una vez copiada la estructura debemos integrarla con el proyecto de PHP que utilizaremos para consumir los mensajes desde el POS

- Incluir en la página PHP el archivo de POSIntegrado.php

```
require_once "POSIntegrado.php";
```

- Luego, para evitar mapear cada una de las variables que vienen desde JS, leeremos todas las variables con la función file\_get\_content()

```
$input = file_get_contents( filename: "php://input");
```

- Una vez obtenida la entrada de datos desde JS, procederemos a decodificarlas y transformarlas a un formato JSON.

```
// Decodificar el JSON
$data = json_decode($input, associative: true);
```

Con estos pasos tendremos lo básico para decodificar los mensajes desde el POS y transformarlos a un objeto definido para cada comando recibido. Antes de comenzar con la decodificación de cada comando, es necesario validar ciertos parámetros que contengan la información necesaria.

- Validar que después de la transformación a formato JSON, sea valido.

```
if ($data !== null) {
    /**
}
```

- También debemos validar que el método utilizado desde JS para enviar la información hacia PHP haya sido un método POST y que exista información dentro del parámetro action.

```
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($data['action'])) {
```



### 11.3 COMANDOS

- **POLL:** desde JS se debe enviar en el parámetro action el valor PollRequest. Como parámetro se le debe enviar el parámetro data del objeto que transformamos a JSON. Debemos utilizar el comando estático desde el archivo POSIntegrado.php el método poll. Una vez procesado la información, este nos devolverá un objeto de tipo PollResponse. En caso de producirse algún error, este arrojara un objeto del tipo PollException.

```
$response = POSIntegrado::poll($data['data']);
```

- **SALE:** desde JS se debe enviar en el parámetro action el valor SaleRequest. Como parámetro se le debe enviar el parámetro data del objeto que transformamos a JSON. Debemos utilizar el comando estático desde el archivo POSIntegrado.php el método sale. Una vez procesado la información, este nos devolverá un objeto de tipo SaleResponse. En caso de producirse algún error, este arrojara un objeto del tipo SaleException.

```
$response = POSIntegrado::sale($data['data']);
```

- **LAST VOUCHER:** desde JS se debe enviar en el parámetro action el valor LastVoucherRequest. Como parámetro se le debe enviar el parámetro data del objeto que transformamos a JSON. Debemos utilizar el comando estático desde el archivo POSIntegrado.php el método lastVoucher. Una vez procesado la información, este nos devolverá un objeto de tipo SaleResponse. En caso de producirse algún error, este arrojara un objeto del tipo LastVoucherException.

```
$response = POSIntegrado::lastVoucher($data['data']);
```

- **REFUND:** desde JS se debe enviar en el parámetro action el valor RefundRequest. Como parámetro se le debe enviar el parámetro data del objeto que transformamos a JSON. Debemos utilizar el comando estático desde el archivo POSIntegrado.php el método refund. Una vez procesado la información, este nos devolverá un objeto de tipo RefundResponse. En caso de producirse algún error, este arrojara un objeto del tipo RefundException.

```
$response = POSIntegrado::refund($data['data']);
```

- **CLOSE:** desde JS se debe enviar en el parámetro action el valor CloseRequest. Como parámetro se le debe enviar el parámetro data del objeto que transformamos a JSON. Debemos utilizar el comando estático desde el archivo POSIntegrado.php el método close. Una vez procesado la información, este nos devolverá un objeto de tipo CloseResponse. En caso de producirse algún error, este arrojara un objeto del tipo CloseException.

```
$response = POSIntegrado::close($data['data']);
```



- **TOTALS:** desde JS se debe enviar en el parámetro action el valor TotalsRequest. Como parámetro se le debe enviar el parámetro data del objeto que transformamos a JSON. Debemos utilizar el comando estático desde el archivo POSIntegrado.php el método totals. Una vez procesado la información, este nos devolverá un objeto de tipo TotalsResponse. En caso de producirse algún error, este arrojara un objeto del tipo TotalsException.

```
$response = POSIntegrado::totals($data['data']);
```

- **DETAILS:** desde JS se debe enviar en el parámetro action el valor DetailsRequest. Como parámetro se le debe enviar el parámetro data del objeto que transformamos a JSON. Debemos utilizar el comando estático desde el archivo POSIntegrado.php el método details. Una vez procesado la información, este nos devolverá un objeto de tipo DetailsResponse. En caso de producirse algún error, este arrojara un objeto del tipo DetailsException.

```
$response = POSIntegrado::details($data['data']);
```

- **SET NORMAL MODE:** desde JS se debe enviar en el parámetro action el valor SetNormalModeRequest. Como parámetro se le debe enviar el parámetro data del objeto que transformamos a JSON. Debemos utilizar el comando estático desde el archivo POSIntegrado.php el método normalMode. Una vez procesado la información, este nos devolverá un objeto de tipo SetNormalModeResponse. En caso de producirse algún error, este arrojara un objeto del tipo SetNormalModeException.

```
$response = POSIntegrado::normalMode($data['data']);
```

- **RETURN:** desde JS se debe enviar en el parámetro action el valor ReturnRequest. Como parámetro se le debe enviar el parámetro data del objeto que transformamos a JSON. Debemos utilizar el comando estático desde el archivo POSIntegrado.php el método return. Una vez procesado la información, este nos devolverá un objeto de tipo ReturnResponse. En caso de producirse algún error, este arrojara un objeto del tipo ReturnException.

```
$response = POSIntegrado::return($data['data']);
```

- **DUPLICATE OTHERS:** desde JS se debe enviar en el parámetro action el valor Duplicate OthersRequest. Como parámetro se le debe enviar el parámetro data del objeto que transformamos a JSON. Debemos utilizar el comando estático desde el archivo POSIntegrado.php el método duplicate. Una vez procesado la información, este nos devolverá un objeto de tipo SaleResponse. En caso de producirse algún error, este arrojara un objeto del tipo DuplicateOthersException.

```
$response = POSIntegrado::duplicate($data['data']);
```



- **SALES BY SELLER:** desde JS se debe enviar en el parámetro action el valor SalesBySellerRequest. Como parámetro se le debe enviar el parámetro data del objeto que transformamos a JSON. Debemos utilizar el comando estático desde el archivo POSIntegrado.php el método salesBySeller. Una vez procesado la información, este nos devolverá un objeto de tipo SalesBySellerResponse. En caso de producirse algún error, este arrojara un objeto del tipo SalesBySellerException.

```
$response = POSIntegrado::salesBySeller($data['data']);
```

- **TIP REPORT:** desde JS se debe enviar en el parámetro action el valor TipReportRequest. Como parámetro se le debe enviar el parámetro data del objeto que transformamos a JSON. Debemos utilizar el comando estático desde el archivo POSIntegrado.php el método TipReport. Una vez procesado la información, este nos devolverá un objeto de tipo TipReportResponse. En caso de producirse algún error, este arrojara un objeto del tipo TipReportException.

```
$response = POSIntegrado::tipReport($data['data']);
```

- **DEFAULT SALE TYPE:** desde JS se debe enviar en el parámetro action el valor DefaultSaleTypeRequest. Como parámetro se le debe enviar el parámetro data del objeto que transformamos a JSON. Debemos utilizar el comando estático desde el archivo POSIntegrado.php el método defaultSale. Una vez procesado la información, este nos devolverá un objeto de tipo DefaultSaleTypeResponse. En caso de producirse algún error, este arrojara un objeto del tipo DefaultSaleTypeException.

```
$response = POSIntegrado::defaultSale($data['data']);
```

- **PARAMETER REPORT:** desde JS se debe enviar en el parámetro action el valor parameterReportRequest. Como parámetro se le debe enviar el parámetro data del objeto que transformamos a JSON. Debemos utilizar el comando estático desde el archivo POSIntegrado.php el método parametersReport. Una vez procesado la información, este nos devolverá un objeto de tipo ParameterReportResponse. En caso de producirse algún error, este arrojara un objeto del tipo ParameterReportException.

```
$response = POSIntegrado::parametersReport($data['data']);
```

- **SIM REPORT:** desde JS se debe enviar en el parámetro action el valor SimReportRequest. Como parámetro se le debe enviar el parámetro data del objeto que transformamos a JSON. Debemos utilizar el comando estático desde el archivo POSIntegrado.php el método simReport. Una vez procesado la información, este nos devolverá un objeto de tipo SimReportResponse. En caso de producirse algún error, este arrojara un objeto del tipo SimReportException.

```
$response = POSIntegrado::simReport($data['data']);
```



- **CANCEL SALE:** desde JS se debe enviar en el parámetro action el valor CancelSaleRequest. Como parámetro se le debe enviar el parámetro data del objeto que transformamos a JSON. Debemos utilizar el comando estático desde el archivo POSIntegrado.php el método cancelSale. Una vez procesado la información, este nos devolverá un objeto de tipo CancelSaleResponse.

```
$response = POSIntegrado::cancelSale($data['data']);
```



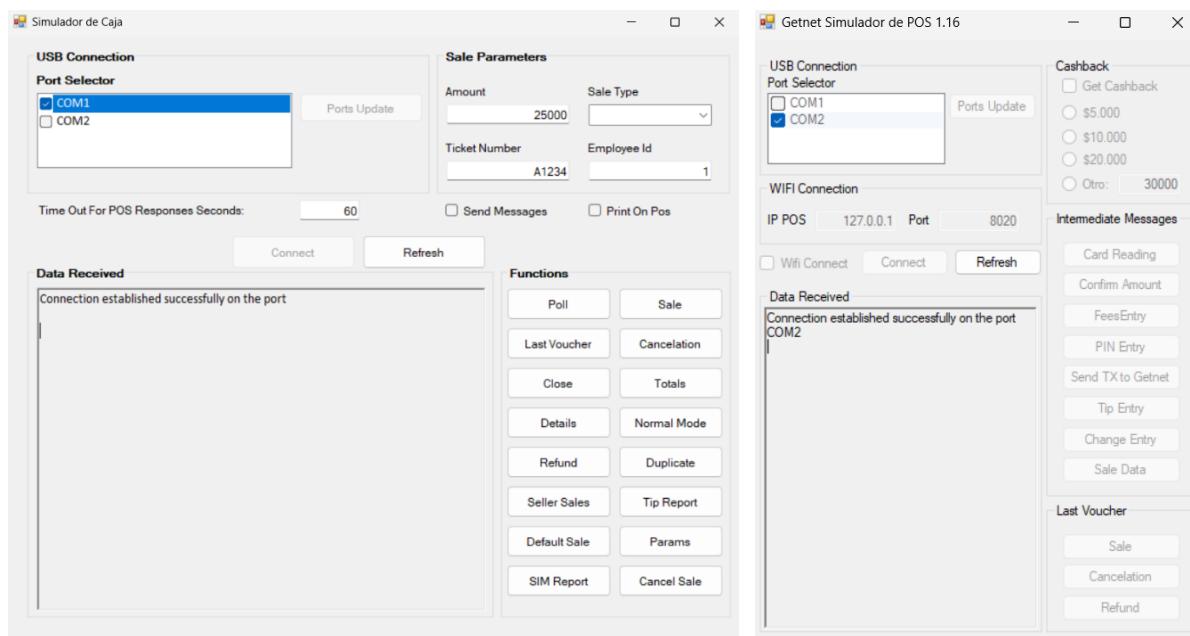
## 12 PROCESO DE INTEGRACIÓN PARA PROYECTOS EN C++

El siguiente documento contempla una guía paso a paso para realizar la integración POS Integrado Getnet con cualquier software de caja desarrollado en C++/Cli .Net Framework (desde la versión 4.5 en adelante).

Antes de realizar la prueba del simulador de Caja primero se debe realizar los pasos de habilitación de puertos COM, validacion de configuración de puertos COM y realizar los pasos del agente POS. Una vez realizados los pasos anteriores se procede a realizar las pruebas y entender el flujo.

### 12.1 INICIAR SIMULADORES DE POS Y CAJA

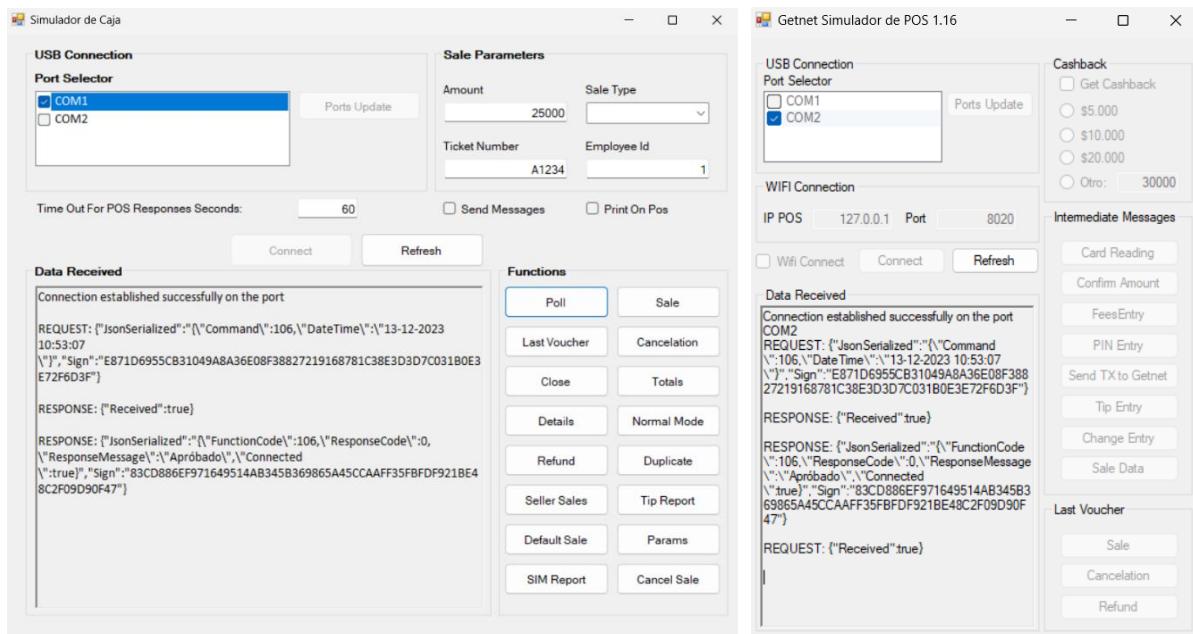
Abrir el simulador de POS y conectarlo en el puerto correspondiente, luego abrir o depurar el simulador de caja C++ y conectarlo en el otro puerto disponible para establecer la conexión.





## 12.2 PROBAR FUNCIONAMIENTOS DE COMANDOS

Presionar el botón Poll para enviar comando de conexión. Al recibir el “RESPONSE” la conexión se ha realizado con éxito, y puede probar todos los botones.

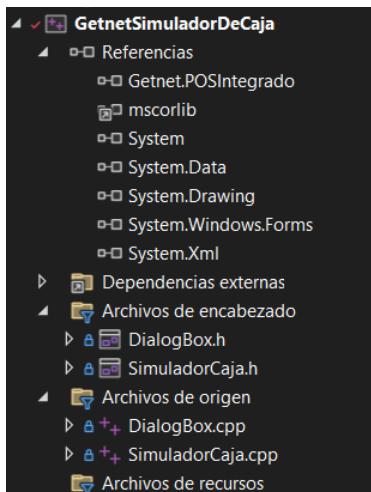


## 12.3 PROCESO DE INTEGRACION PARA PROYECTOS WINFORM C++/.NET FRAMEWORK

Para el proceso de integración se entrega el proyecto de Simulador de Caja tipo Winform en C++ .Net Framework de manera que se puede revisar en su totalidad y así tener un ejemplo claro de como se puede modificar el software de caja para incorporar la integración con GetnetPOSIntegrado.

## 12.4 LIBRERÍA

El primer paso es agregar la referencia de la DLL dentro del proyecto:





## 12.5 AGREGAR LA LIBRERÍA EN LA SECCIÓN DE USING

```
using namespace GetnetPOSIntegrado;
```

## 12.6 CONTROLAR LA COMUNICACIÓN SERIAL Y TIMEOUTS

### Archivo de encabezado SimuladorCaja.h

Se requiere inicializar algunas variables globales dentro de la clase como las siguientes:

```
SerialPort^ serialPort;  
static System::Timers::Timer^ timeoutForPosResponse =  
    POSIntegrado::Instance()->GetTimeoutForPosResponse();  
static System::Timers::Timer^ timeoutForConnection =  
    POSIntegrado::Instance()->GetTimeoutForConnection();
```

Luego se debe crear un método de inicialización de variables:

```
void InicializacionVariables();
```

### Archivo de origen SimuladorCaja.cpp

En este archivo se realizará la inicialización de las variables dentro del metodo InicializacionVariables

```
void SimuladorCaja::InicializacionVariables() {  
    serialPort = POSIntegrado::Instance()->GetnetSerialPort();  
    // Más código...  
}
```

### Explicación:

- Variable **serialPort**: Permitirá controlar la comunicación por el cable USB con el POS
- Variable **timeoutForConnection**: Permitirá controlar los tiempos de espera para escribir los TRUE que validan que el POS recibió las solicitudes.
- Variable **timeoutForPosResponse**: Permitirá controlar los tiempos de espera para las respuestas de cada comando enviado al POS Getnet.



## 12.7 DEFINIR MANEJADORES DE LA COMUNICACIÓN HACIA LA CAJA

En este paso se inicializan orejas de escucha en el puerto serial y en los timers para esperar las respuestas y controlar los timeouts. Esto se encuentra dentro de InicializarVariables();

```
//Manejador de las respuestas por SerialPort
serialPort->DataReceived += gcnew SerialDataReceivedEventHandler(
    this, &SimuladorCaja::serialPort_DataReceived
);

//Control de tiempo para esperar por la conexión del POS
timeoutForConnection->Elapsed += gcnew System::Timers::ElapsedEventHandler(
    this, &SimuladorCaja::timeoutForConnectionElapsed
);

//Control de tiempo para esperar respuestas del POS
timeoutForPosResponse->Elapsed += gcnew System::Timers::ElapsedEventHandler(
    this, &SimuladorCaja::timeoutForPosResponseElapsed
);
```

## 12.8 CONTROLAR EL CIERRE DE PUERTOS

En un método de ciclo de vida del WinForm se puede llamar al cierre de puertos cuando se cierra la aplicación o cuando se requiera cerrar la comunicación con el POS Getnet.

```
System::Void SimuladorCaja::SimuladorCaja_FormClosing(
    System::Object^ sender, System::Windows::Forms::FormClosingEventArgs^ e) {
    POSIntegrado::Instance()->UsbClosePort();
}
```



## 12.9 CONTROLAR LOS TIMERS

Estos métodos se inicializaron más arriba y se comportan tal como un CATCH tras finalizar el tiempo de espera. De manera que pueden permitir mostrar un mensaje de alerta al usuario de la caja o bien controlarlo de manera inteligente dentro del software de caja para indicar que se agotó el tiempo de espera y que el POS no está conectado o bien superó el tiempo de espera para responder el comando solicitado.

### Archivo de encabezado SimuladorCaja.h

```
//Control del timeout para la conexión con el POS
void timeoutForConnectionElapsed(
    System::Object^ sender, System::Timers::ElapsedEventArgs^ e
);

//Control del timeout para las respuestas del POS
void timeoutForPosResponseElapsed(
    System::Object^ sender, System::Timers::ElapsedEventArgs^ e
);
```

### Archivo de origen SimuladorCaja.cpp

```
void SimuladorCaja::timeoutForConnectionElapsed(
    System::Object^ sender, System::Timers::ElapsedEventArgs^ e) {
    POSIntegrado::Instance()->StopTimerForConnection();
    POSIntegrado::Instance()->StopTimerForPosResponse();
    MessageBox::Show("There is no connection to the POS.", "Error");
}

void SimuladorCaja::timeoutForPosResponseElapsed(
    System::Object^ sender, System::Timers::ElapsedEventArgs^ e) {
    POSIntegrado::Instance()->StopTimerForPosResponse();
    MessageBox::Show("The POS has timed out waiting for the response.", "Error");
}
```

## 12.10 CONOCER LOS PUERTOS DISPONIBLES

Si fuera necesario solicitar los puertos disponibles en Windows se puede ocupar este método

```
array<String^>^ ports = POSIntegrado::Instance()->ListPorts();
```



## 12.11 CONECTAR AL PUERTO SERIAL

Para conectarse al Puerto serial COM disponible se puede realizar de la siguiente manera:

**EN DESARROLLO:** Se puede definir “en duro” el puerto de comunicaciones y la velocidad de transmisión de datos (BaudRate) para hacer las pruebas

```
try
{
    // Conexion por USB
    POSIntegrado::Instance()->UsbConnect("COM1", 115200);
}
catch (Exception^ ex)
{
    ShowMessage("Error connecting to COM port: " + ex->Message);
}
```

**EN PRODUCCIÓN:** Se requerirá la configuración desde el Agente POS. En este caso no se define ni el nombre del puerto ni la velocidad de transmisión de datos (BaudRate) dado que la componente la va a buscar a la ruta de la configuración del Agente Pos.

```
try
{
    POSIntegrado::Instance()->UsbConnect();
}
catch (Exception^ ex)
{
    ShowMessage("Error connecting to COM port: " + ex->Message);
}
```

**NOTA:** Estas son solo recomendaciones. Si usted no quiere usar el Agente POS puede usar la misma configuración del ambiente de desarrollo y validar que el nombre del puerto en la Caja será el mismo que usted definió en su código.



## 12.12 ENVIAR SOLICITUDES AL POS

El envío de comandos al POS Getnet es tan simple como llamar a un método y la librería se encarga de todo el procesamiento de los datos para construir el JSON, firmar los datos y enviar el mensaje así:

### POLLING

```
try {
    POSIntegrado::Instance()->Poll();
}
catch (Exception^ ex) {
    MessageBox::Show(ex->Message);
}
```

### SALE

```
try {
    POSIntegrado::Instance()->Sale(
        1000, "A1234", false, (int)POSCommands::SaleType::Sale, false, 1, 60
    );
}
catch (Exception^ ex) {
    MessageBox::Show(ex->Message);
}
```

### LAST VOUCHER

```
try {
    POSIntegrado::Instance()->LastVoucher();
}
catch (Exception^ ex) {
    MessageBox::Show(ex->Message);
}
```

### CANCELLATION

```
try {
    POSIntegrado::Instance()->Refund(1234);
}
catch (Exception^ ex) {
    MessageBox::Show(ex->Message);
}
```

### CLOSE

```
try {
    POSIntegrado::Instance()->Close();
}
catch (Exception^ ex) {
    MessageBox::Show(ex->Message);
}
```

**TOTALS**

```
try {
    POSIntegrado::Instance()->Totals();
}
catch (Exception^ ex) {
    MessageBox::Show(ex->Message);
}
```

**DETAILS**

```
try {
    POSIntegrado::Instance()->Details();
}
catch (Exception^ ex) {
    MessageBox::Show(ex->Message);
}
```

**NORMAL MODE**

```
try {
    POSIntegrado::Instance()->SetNormalMode();
}
catch (Exception^ ex) {
    MessageBox::Show(ex->Message);
}
```

**REFUND**

```
try {
    POSIntegrado::Instance()->Return("A1234", 25000);
}
catch (Exception^ ex) {
    MessageBox::Show(ex->Message);
}
```

**DUPLICATE**

```
try {
    POSIntegrado::Instance()->DuplicateOthers(1234);
}
catch (Exception^ ex) {
    MessageBox::Show(ex->Message);
}
```

**SELLER SALES**

```
try {
    POSIntegrado::Instance()->SalesBySeller(1);
}
catch (Exception^ ex) {
    MessageBox::Show(ex->Message);
}
```

**TIPS REPORT**

```
try {
    POSIntegrado::Instance()->TipReport(1);
}
catch (Exception^ ex) {
    MessageBox::Show(ex->Message);
}
```

**DEFAULT SALE**

```
try {
    POSIntegrado::Instance()->DefaultSaleType(POSCommands::SaleType::Sale);
}
catch (Exception^ ex) {
    MessageBox::Show(ex->Message);
}
```

**PARAMS**

```
try {
    POSIntegrado::Instance()->ParameterReport();
}
catch (Exception^ ex) {
    MessageBox::Show(ex->Message);
}
```

**SIM REPORT**

```
try {
    POSIntegrado::Instance()->SimReport();
}
catch (Exception^ ex) {
    MessageBox::Show(ex->Message);
}
```



## CANCEL SALE

```
try {
    POSIntegrado::Instance()->CancelSale();
}
catch (Exception^ ex) {
    MessageBox::Show(ex->Message);
}
```

**NOTA:** Todos los métodos expuestos aquí tienen una sobrecarga, lo que significa que se pueden enviar en su forma más simple (como se exponen aquí) o bien agregarle parámetros como el PrintOnPos o Seconds (para controlar el timeout).

## 12.13 LEER LAS RESPUESTAS DEL POS

Para leer las respuestas del POS será necesario implementar un método ya definido más arriba de la siguiente

```
void SimuladorCaja::serialPort_DataReceived(
System::Object^ sender, System::IO::Ports::SerialDataReceivedEventArgs^ e) {
    try {
        receivedMessage = POSIntegrado::Instance()->UsbReadData();
        this->Invoke(gcnew MethodInvoker(this, &SimuladorCaja::DataReceived));
    }
    catch (Exception^ ex) {
        ExceptionMessage(ex->Message);
    }
}
```

Debido que este método se ejecuta en otro hilo del procesamiento, al ejecutar la línea this->invoke estamos devolviéndonos al hilo principal y por ende podemos hacer uso de los controles del formulario así procesamos la data dentro del hilo, en caso de no querer hacerlo procesar los datos dentro del mismo método evento void SimuladorCaja::serialPort\_DataReceived. Entonces en el método DataReceived se procesarían los datos, donde la variable dataProcessed ya tendría procesada la respuesta por ende solo quedaría hacer uso de ella en la manera que se deseé.

```
void SimuladorCaja::DataReceived() {
    try
    {
        POSIntegrado::Instance()->StopTimerForConnection();
        rtbMessages->AppendText("RESPONSE: " + receivedMessage + "\n\n");
        Object^ dataProcessed = POSIntegrado::Instance()->ProcessData(receivedMessage);
        // más código...
    }
    catch (Exception^ ex)
    {
        ExceptionMessage(ex->Message);
    }
}
```



## 12.14 CASTEO DE OBJETOS RESPONSE

La variable dataProcessed se define de tipo Object porque almacena cualquier tipo de objeto de las respuestas definidas por la librería. Si quisieramos transformarla internamente dentro del desarrollo de la caja se puede hacer de la siguiente manera:

Primeramente se obtiene el tipo de objeto, con ello luego podemos hacer la consulta de qué tipo de respuesta que se obtuvo y así poder hacer el casteo.

```
Type^ tipo = dataProcessed->GetType();
```

### POLLING

```
if (tipo == PollResponse::typeid){  
    PollResponse^ poll = static_cast<PollResponse^>(dataProcessed);  
}
```

### SALE, LAST VOUCHER, DUPLICATE

```
if (tipo == SaleResponse::typeid) {  
    SaleResponse^ sale = static_cast<SaleResponse^>(dataProcessed);  
}
```

### CANCELLATION

```
if (tipo == RefundResponse::typeid) {  
    RefundResponse^ cancelation = static_cast<RefundResponse^>(dataProcessed);  
}
```

### CLOSE

```
if (tipo == CloseResponse::typeid) {  
    CloseResponse^ close = static_cast<CloseResponse^>(dataProcessed);  
}
```

### TOTALS

```
if (tipo == TotalsResponse::typeid) {  
    TotalsResponse^ totals = static_cast<TotalsResponse^>(dataProcessed);  
}
```

### DETAILS

```
if (tipo == DetailsResponse::typeid) {  
    DetailsResponse^ details = static_cast<DetailsResponse^>(dataProcessed);  
}
```

### NORMAL MODE

```
if (tipo == SetNormalModeResponse::typeid) {  
    SetNormalModeResponse^ normalMode =  
        static_cast<SetNormalModeResponse^>(dataProcessed);  
}
```

**REFUND**

```
if (tipo == ReturnResponse::typeid) {  
    ReturnResponse^ refund = static_cast<ReturnResponse^>(dataProcessed);  
}
```

**SELLER SALES**

```
if (tipo == SalesBySellerResponse::typeid) {  
    SalesBySellerResponse^ sellerSales =  
        static_cast<SalesBySellerResponse^>(dataProcessed);  
}
```

**TIP REPORT**

```
if (tipo == TipReportResponse::typeid) {  
    TipReportResponse^ tipReport = static_cast<TipReportResponse^>(dataProcessed);  
}
```

**DEFAULT SALE**

```
if (tipo == DefaultSaleTypeResponse::typeid) {  
    DefaultSaleTypeResponse^ defaultSale =  
        static_cast<DefaultSaleTypeResponse^>(dataProcessed);  
}
```

**PARAMS**

```
if (tipo == ParameterReportResponse::typeid) {  
    ParameterReportResponse^ parameters =  
        static_cast<ParameterReportResponse^>(dataProcessed);  
}
```

**SIM REPORT**

```
if (tipo == SimReportResponse::typeid) {  
    SimReportResponse^ simReport = static_cast<SimReportResponse^>(dataProcessed);  
}
```

**CANCEL SALE**

```
if (tipo == CancelSaleResponse::typeid) {  
    CancelSaleResponse^ cancelSale =  
        static_cast<CancelSaleResponse^>(dataProcessed);  
}
```



## 12.15 EJEMPLO

Aquí proponemos una manera de hacerlo. Se pueden crear tantas variables como tipos de respuestas hay y luego simplemente validar el comando y castearlo para obtener acceso a las propiedades del modelo y poder usar los datos directamente en el software de Caja

**NOTA: No es necesario leer el resultado del puerto serial y deserializarlo para procesarlo de forma manual, la librería ya hace esto.**

```
POSIntegrado::Instance()->StopTimerForConnection();
rtbMessages->AppendText("RESPONSE: " + receivedMessage + "\n\n");
Object^ dataProcessed = POSIntegrado::Instance()->ProcessData(receivedMessage);

//Variables para castear la data procesada
PollResponse^ poll;
SaleResponse^ sale;
RefundResponse^ cancelation;
CloseResponse^ close;
TotalsResponse^ totals;
DetailsResponse^ details;
SetNormalModeResponse^ normalMode;
ReturnResponse^ refund;
SalesBySellerResponse^ sellerSales;
TipReportResponse^ tipReport;
DefaultSaleTypeResponse^ defaultSale;
ParameterReportResponse^ parameters;
SimReportResponse^ simReport;
CancelSaleResponse^ cancelSale;

if (dataProcessed == nullptr) {
    return;
}
Type^ tipo = dataProcessed->GetType();

if (tipo == PollResponse::typeid){
    poll = static_cast<PollResponse^>(dataProcessed);
}
if (tipo == SaleResponse::typeid) {
    sale = static_cast<SaleResponse^>(dataProcessed);
}
if (tipo == RefundResponse::typeid) {
    cancelation = static_cast<RefundResponse^>(dataProcessed);
}
if (tipo == CloseResponse::typeid) {
    close = static_cast<CloseResponse^>(dataProcessed);
}
if (tipo == TotalsResponse::typeid) {
    totals = static_cast<TotalsResponse^>(dataProcessed);
}
if (tipo == DetailsResponse::typeid) {
    details = static_cast<DetailsResponse^>(dataProcessed);
}
```



```
if (tipo == SetNormalModeResponse::typeid) {  
    normalMode = static_cast<SetNormalModeResponse^>(dataProcessed);  
}  
if (tipo == ReturnResponse::typeid) {  
    refund = static_cast<ReturnResponse^>(dataProcessed);  
}  
if (tipo == SalesBySellerResponse::typeid) {  
    sellerSales = static_cast<SalesBySellerResponse^>(dataProcessed);  
}  
if (tipo == TipReportResponse::typeid) {  
    TipReportResponse^ tipReport = static_cast<TipReportResponse^>(dataProcessed);  
}  
if (tipo == DefaultSaleTypeResponse::typeid) {  
    defaultSale = static_cast<DefaultSaleTypeResponse^>(dataProcessed);  
}  
if (tipo == ParameterReportResponse::typeid) {  
    parameters = static_cast<ParameterReportResponse^>(dataProcessed);  
}  
if (tipo == SimReportResponse::typeid) {  
    simReport = static_cast<SimReportResponse^>(dataProcessed);  
}  
if (tipo == CancelSaleResponse::typeid) {  
    cancelSale = static_cast<CancelSaleResponse^>(dataProcessed);  
}
```



## 13 PROCESO DE INTEGRACIÓN PARA PROYECTOS EN LENGUAJE C

Para el proceso de integración se entrega el proyecto de Simulador de Caja tipo consola en C de manera que es posible revisarlo en su totalidad para tener un ejemplo claro de cómo se puede modificar el software de Caja para incorporar la integración con Getnet POS Integrado.

En este caso hablaremos de la integración para un software de caja desarrollado en C. A continuación, entregamos algunos tips importantes:

### LIBRERIA

Lo primero es agregar la referencia de la DLL dentro del proyecto.

Para agregar la librería se provee de varios archivos que se nombran a continuación

- Getnet.POSIntegrado.dll
- Getnet.POSIntegrado.lib
- Request.h
- Responses.h
- POSCommands.h
- Utils.h
- Getnet.POSIntegrado.h

### AGREGAR ARCHIVO DLL

El archivo Getnet.POSIntegrado.dll se debe agregar en la mismo directorio donde reside el archivo ejecutable del sistema de caja.

### AGREGAR ARCHIVO LIB

El archivo Getnet.POSIntegrado.lib se debe agregar al proyecto del sistema de caja, habitualmente hay un directorio Lib para agregar este tipo de archivos.

### AGREGAR LOS HEADERS

Será necesario incorporar estas líneas al inicio del módulo de código para tener acceso a las funciones de la librería.

```
#include <Request.h>
#include <Responses.h>
#include <POSCommands.h>
#include <Utils.h>
#include <Getnet.POSIntegrado.h>
```



### 13.1 CONECTAR AL PUERTO SERIAL

Para conectarse al puerto serial COM disponible se pueden realizar de la siguiente manera:

**EN DESARROLLO:** Se puede definir en duro el puerto de comunicaciones y la velocidad de transmisión de datos (BaudRate) para hacer las pruebas.

```
UsbConnect("COM1", 115200);
```

**EN PRODUCCIÓN:** Se requerirá la configuración desde el Agente POS. En este caso no se define ni el nombre del puerto ni la velocidad de transmisión de datos (BaudRate) dado que la componente la va a buscar a la ruta de la configuración del Agente POS.

```
UsbConnectWithPosConfig();
```

**NOTA:** Estas son solo recomendaciones. Si usted no quiere usar el Agente POS puede usar la misma configuración del ambiente de desarrollo y validar que el nombre del puerto en la Caja será el mismo que usted definió en su código.

### 13.2 ENVIAR SOLICITUDES AL POS

El envío de comandos al POS Getnet es tan simple como llamar a una función y la librería se encarga de todo el procesamiento de los datos para construir el JSON, firmar los datos y enviar el mensaje así:

**POLLING:**

```
Poll();
```

**SALE:**

```
Sale(25000, "A1234", false, 0, false, 1);
```

**LAST VOUCHER:**

```
LastVoucher(false);
```

**CANCELLATION:**

```
Refund(1234, false);
```

**CLOSE:**

```
Close(false);
```

**TOTALS:**

```
Totals(false);
```

**DETAILS:**

```
Details(false);
```

**NORMAL MODE:**

```
SetNormalMode();
```

**REFUND:**

```
Return(1234, 25000, false);
```

**DUPLICATE:**

```
DuplicateOthers(1234, false);
```

**SELLER SALES:**

```
SalesBySeller(1, false);
```

**TIP REPORT:**

```
TipReport(1, false);
```

**DEFAULT SALE:**

```
DefaultSaleType(0);
```

**PARAMS:**

```
ParameterReport(false);
```

**SIM REPORT:**

```
SimReport(false);
```

**CANCEL SALE:**

```
CancelSale();
```

### 13.3 LEER LAS RESPUESTAS DEL POS

Para leer las respuestas desde el POS Getnet solo será necesario llamar a las siguientes funciones:

```
char *message = UsbReadData();
void *dataProcessed = ProcessData(message, &functionCode);
```

La respuesta de la comunicación serial puede tomar un tiempo, por lo que la función UsbReadData se puede llamar dentro de un ciclo while hasta que se obtenga una respuesta o se cumpla un tiempo.

**Ejemplo de implementación UsbReadData**

```
time_t t = time(NULL);
while (t + timeOut >= time(NULL))
{
    char* message = UsbReadData();
    int received = strlen(message);
    if (received > 0)
    {
        dataReceived(message);
        if (strstr(message, "FunctionCode"))
        {
            break;
        }
    }
}
```



```
if (_kbhit())
{
    char c = _getch();
    if (c == 27)
    {
        break;
    }
}
```

Es importante mencionar que al recibir el mensaje en la variable “**message**” se puede ver tal como se recibe en el Simulador de Caja, pero no es necesario descomponerlo o intentar deserializarlo porque en el siguiente paso la variable “**dataProcessed**” nos entrega un puntero que se debe castear según el valor de la variable **functionCode**, con esto se puede acceder a las estructuras correspondientes con los valores de cada respuesta ya procesados y con lo cual podemos usarlos directamente para leer los datos y procesarlos en el Software de Caja.

#### 13.4 CASTEO DE RESPONSES

La variable “**dataProcessed**” se define como un puntero void porque soporta cualquier tipo de puntero según las respuestas definidas por la librería. Si quisieramos transformarla internamente dentro del desarrollo de la Caja y obtener la estructura correspondiente, se puede hacer de la siguiente manera:

**POLLING:**

```
PollResponse pollResponse = *(PollResponse*)dataProcessed;
```

**SALE, LAST VOUCHER, DUPLICATE:**

```
SaleResponse saleResponse = *(SaleResponse*)dataProcessed;
```

**CANCELLATION:**

```
RefundResponse refundResponse = *(RefundResponse*)dataProcessed;
```

**CLOSE:**

```
CloseResponse closeResponse = *(CloseResponse*)dataProcessed;
```

**TOTALS:**

```
TotalsResponse totalsResponse = *(TotalsResponse*)dataProcessed;
```

**DETAILS:**

```
DetailsResponse detailsResponse = *(DetailsResponse*)dataProcessed;
```

**NORMAL MODE:**

```
SetNormalModeResponse setNormalModeResponse =
    *(SetNormalModeResponse*)dataProcessed;
```

**REFUND:**

```
ReturnResponse returnResponse = *(ReturnResponse*)dataProcessed;
```

**SELLER SALES:**

```
SalesBySellerResponse salesBySellerResponse =  
    *(SalesBySellerResponse*)dataProcessed;
```

**TIP REPORT:**

```
TipReportResponse tipReportResponse = *(TipReportResponse*)dataProcessed;
```

**DEFAULT SALE:**

```
DefaultSaleTypeResponse defaultSaleTypeResponse =  
    *(DefaultSaleTypeResponse*)dataProcessed;
```

**PARAMS:**

```
ParameterReportResponse parameterReportResponse =  
    *(ParameterReportResponse*)dataProcessed;
```

**SIM REPORT:**

```
SimReportResponse simReportResponse = *(SimReportResponse*)dataProcessed;
```

**CANCEL SALE:**

```
CancelSaleResponse cancelSaleResponse = *(CancelSaleResponse*)dataProcessed;
```

### 13.5 EJEMPLO

Aquí proponemos una manera de hacerlo. Se pueden crear tantas variables como tipos de respuestas hay y luego simplemente validar el comando y castearlo para obtener acceso a las propiedades del modelo y poder usar los datos directamente en el Software de Caja.

**NOTA:** No es necesario leer el resultado del puerto serial y deserializarlo para procesarlo de forma manual, la librería ya hace esto.

```
if (dataProcessed)  
{  
    switch (functionCode)  
    {  
        case Function_Sale:  
            SaleResponse saleResponse = *(SaleResponse*)dataProcessed;  
            break;  
        case Function_LastVoucher:  
            SaleResponse saleResponse2 = *(SaleResponse*)dataProcessed;  
            break;  
        case Function_Refund:  
            RefundResponse refundResponse = *(RefundResponse*)dataProcessed;  
            break;  
        case Function_Close:  
            CloseResponse closeResponse = *(CloseResponse*)dataProcessed;  
            break;  
        case Function_Totals:  
            TotalsResponse totalsResponse = *(TotalsResponse*)dataProcessed;  
            break;  
        case Function_Details:  
            DetailsResponse detailsResponse = *(DetailsResponse*)dataProcessed;  
            break;  
    }  
}
```



```
        case Function_Poll:
            PollResponse pollResponse = *(PollResponse*)dataProcessed;
            break;
        case Function_SetNormalMode:
            SetNormalModeResponse setNormalModeResponse =
                *(SetNormalModeResponse*)dataProcessed;
            break;
        case Function_Return :
            ReturnResponse returnResponse = *(ReturnResponse*)dataProcessed;
            break;
        case Function_DuplicateOthers:
            SaleResponse saleResponse3 = *(SaleResponse*)dataProcessed;
            break;
        case Function_SalesBySeller:
            SalesBySellerResponse salesBySellerResponse =
                *(SalesBySellerResponse*)dataProcessed;
            break;
        case Function_TipReport:
            TipReportResponse tipReportResponse =
                *(TipReportResponse*)dataProcessed;
            break;
        case Function_DefaultSaleType:
            DefaultSaleTypeResponse defaultSaleTypeResponse =
                *(DefaultSaleTypeResponse*)dataProcessed;
            break;
        case Function_ParameterReport:
            ParameterReportResponse parameterReportResponse =
                *(ParameterReportResponse*)dataProcessed;
            break;
        case Function_SimReport:
            SimReportResponse simReportResponse =
                *(SimReportResponse*)dataProcessed;
            break;
        case Function_CancelSale:
            CancelSaleResponse cancelSaleResponse =
                *(CancelSaleResponse*)dataProcessed;
            break;
    }
}
```

### 13.6 CONTROLAR EL CIERRE DE PUERTOS

Se debe cerrar de puertos cuando se cierra la aplicación o cuando se requiera cerrar la comunicación con el POS Getnet. Para esto se provee de la siguiente función.

```
UsbClosePort();
```



## 14 EXCEPCIONES

Para todos los métodos, si ocurre un error al ejecutar la acción desde la Caja, se lanzará una excepción que llevará el mensaje de error que devuelve el framework.



## 15 ANEXOS

### 15.1 COMANDOS A ENVIAR AL POS

Comando	Códigos Propios	Códigos TBK
Venta	100	210
Ultima Venta	101	260
Anular Venta	102	1210
Cierre	103	510
Totales de Ventas	104	710
Detalles de Ventas	105	261
Polling	106	
Cambio a POS Normal	107	
Transacción de Devolución	108	
Duplicado Otros	109	
Compras por Vendedor	110	
Reporte de Propinas	111	
Mensajes Intermedios	112	
Compra Predeterminada	113	
Reporte de Parámetros	114	
Reporte SIM	115	
Cancelar Venta	116	



## 15.2 CODIGOS DE RESPUESTAS

Estos son todos los códigos que responde el POS Getnet en el campo ResponseCode y ResponseMessage respectivamente.

**NOTA:** Cabe recalcar que estos códigos no son los mismos que utiliza Transbank por lo tanto esto ya no podría considerarse un cambio simple de una DLL como se tenía pensado, dado que el desarrollador de la Caja debe considerar todas estas excepciones en sus funcionalidades para el software de la caja. Por defecto todos los códigos de respuesta 00 significan aprobaciones, cualquier dígito distinto es rechazo.

Código de respuesta interno	Glosa en POS	Nueva Glosa a definir
00	Aprobado	Aprobado
00	Aprobado	Aprobado
00	Identificación Aprobada	Aprobado
00	Aprobado VIP	Aprobado
00	Aprobado, Actualizar Track 3	Aprobado
00	Aprobado, tipo de cuenta especificada por el emisor	Aprobado
00	Aprobado, actualizar ICC	Aprobado
00	Reversa aceptada	Aprobado
00	Reversa rechazada, original no encontrada	Aprobado
00	Reversa rechazada, original no aprobada	Aprobado
00	Advice reconocido, sin responsabilidad financiera	Aprobado
00	Advice reconocido, con responsabilidad financiera	Aprobado
00	Aprobado	Aprobado
00	Aprobado	Aprobado
01	Negada, contactar emisor	Rechazado
02	Negada, contactar emisor por condiciones especiales	Rechazado
03	Comercio Invalido	Rechazado
03	Negada, comercio invalido	Rechazado



04	Capturar Tarjeta	Rechazado, Probar con otra Tarjeta
04	Negada, retener tarjeta no honrar	Rechazado, Probar con otra Tarjeta
04	Negada, retener tarjeta expirada	Rechazado, Probar con otra Tarjeta
04	Negada, retener tarjeta por sospecha de fraude	Rechazado, Probar con otra Tarjeta
04	Negada, retener tarjeta contactar adquiriente	Rechazado, Probar con otra Tarjeta
04	Negada, tarjeta restricta, retener	Rechazado, Probar con otra Tarjeta
04	Negada, retener tarjeta	Rechazado, Probar con otra Tarjeta
04	Negada, retener tarjeta	Rechazado, Probar con otra Tarjeta
04	Negada, retener tarjeta	Rechazado, Probar con otra Tarjeta
04	Negada, tarjeta perdida, retener	Rechazado, Probar con otra Tarjeta
04	Negada, tarjeta robada, retener	Rechazado, Probar con otra Tarjeta
04	Negada, retener tarjeta por sospecha de falsificación	Rechazado, Probar con otra Tarjeta
04	Negada, retener tarjeta ARQC invalido	Rechazado, Probar con otra Tarjeta
04	Negada, retener tarjeta CVR invalido	Rechazado, Probar con otra Tarjeta
04	Negada, retener tarjeta TVR invalido	Rechazado, Probar con otra Tarjeta
04	Negada, retener tarjeta ATC invalido	Rechazado, Probar con otra Tarjeta
04	Negada, retener tarjeta fallback invalido	Rechazado, Probar con otra Tarjeta



05	No Honrar	Rechazado
05	Negada, no honrar	Rechazado
05	Negada, sospecha de fraude	Rechazado, Probar con otra Tarjeta
05	Negada, contactar adquiriente	Rechazado
05	Negada, contactar departamento de seguridad adquiriente	Rechazado
05	Negada, sospecha de falsificación	Rechazado
05	No Honrar	Rechazado
10	Aprobación parcial	Rechazado
10	Aprobación parcial, tipo de cuenta especificada por el emisor	Rechazado
12	Transacción Invalida	Rechazado, transacción no permitida
12	Negada, transacción no permitida en el terminal	Rechazado, transacción no permitida
12	Fallo en verificación de información	Rechazado, transacción no permitida
12	Negada, falla en validación de ARQC	Rechazado, transacción no permitida
12	Negada, falla en validación de ARQC	Rechazado, transacción no permitida
12	Negada, falla en validación de CVR	Rechazado, transacción no permitida
12	Negada, falla en validación de CVR	Rechazado, transacción no permitida
12	Negada, falla en validación de TVR	Rechazado, transacción no permitida
12	Negada, falla en validación de TVR	Rechazado, transacción no permitida
12	Negada, falla en validación de ATC	Rechazado, transacción no permitida



12	Negada, falla en validación de ATC	Rechazado, transacción no permitida
12	Negada, chequear fallback	Rechazado, transacción no permitida
12	Negada, chequear fallback	Rechazado, transacción no permitida
12	Negada, banco emisor invalido	Rechazado, transacción no permitida
12	Negada, banco emisor invalido	Rechazado, transacción no permitida
12	Negada, vendedor invalido	Rechazado, transacción no permitida
12	Negada, vendedor invalido	Rechazado, transacción no permitida
12	Negada, vendedor invalido	Rechazado, transacción no permitida
12	Negada, vendedor no encontrado	Rechazado, transacción no permitida
12	Negada, vendedor no encontrado	Rechazado, transacción no permitida
12	Negada, vendedor información invalida	Rechazado, transacción no permitida
12	Negada, información de pago invalida	Rechazado, transacción no permitida
12	Negada, verificar manual	Rechazado, transacción no permitida
12	Negada, verificar manual	Rechazado, transacción no permitida
12	Negada, verificar manual	Rechazado, transacción no permitida
12	Negada, verificar manual	Rechazado, transacción no permitida
12	Negada, verificar manual	Rechazado, transacción no permitida



12	Transacción invalida	Rechazado, transacción no permitida
13	Negada, monto invalido	Rechazado, monto invalido
13	Monto Invalido	Rechazado, monto invalido
13	Negada, monto no informado	Rechazado, monto invalido
14	Negada, tarjeta invalida	Rechazado, Probar con otra Tarjeta
14	Negada, tarjeta sin registro	Rechazado, Probar con otra Tarjeta
14	Negada, tarjeta no efectiva	Rechazado, Probar con otra Tarjeta
14	Tarjeta Invalida	Rechazado, Probar con otra Tarjeta
19	Reenviar transacción	Rechazado, reintentar transacción
21	Transacción no encontrada	Rechazado
30	Mensaje mal formado	Rechazado
30	Error de formato	Rechazado
41	Tarjeta Perdida	Rechazado, Probar con otra Tarjeta
43	Tarjeta Extraviada	Rechazado, Probar con otra Tarjeta
46	Cuenta cerrada	Rechazado, Probar con otra Tarjeta
51	Negada, fondos insuficientes	Rechazado, excede monto
51	Saldo Insuficiente	Rechazado, excede monto



54	Negada, tarjeta expirada	Rechazado, Probar con otra Tarjeta
54	Tarjeta Expirada	Rechazado, Probar con otra Tarjeta
55	Negada, pin invalido	Rechazado, problema con pin
55	Negada, pinblock invalido	Rechazado, problema con pin
55	Negada, largo de pin invalido	Rechazado, problema con pin
55	Negada, cambio de pin requerido	Rechazado, problema con pin
55	Negada, nuevo pin invalido	Rechazado, problema con pin
55	Pin Invalido	Rechazado, problema con pin
57	Negada, pin requerido	Rechazado, transacción no permitida
57	Negada, tarifa no aceptada	Rechazado, transacción no permitida
57	Negada, tipo de cuenta	Rechazado, transacción no permitida
57	Negada, función no soportada	Rechazado, transacción no permitida
57	Negada, transacción no permitida en el emisor	Rechazado, transacción no permitida
57	Negada, violación de la ley	Rechazado, transacción no permitida
57	Transacción no permitida	Rechazado, transacción no permitida
57	Transacción duplicada	Rechazado, transacción no permitida
57	No se puede rastrear transacción original	Rechazado, transacción no permitida



57	MAC incorrecta	Rechazado, transacción no permitida
57	Tipo de captura invalida	Rechazado, transacción no permitida
58	Transacción Invalida	Rechazado, transacción no permitida
61	Negada, límite de retiro excedido	Rechazado, excede monto
61	Límite Excedido	Rechazado, excede monto
62	Negada, tarjeta restricta	Rechazado, Probar con otra Tarjeta
62	Negada, dominio restringido	Rechazado, Probar con otra Tarjeta
62	Tarjeta Restringida	Rechazado, Probar con otra Tarjeta
63	Negada, violación de seguridad	Rechazado
63	Control de seguridad rechazado	Rechazado
65	Negada, límite de frecuencia de retiro excedido	Rechazado, excede transacciones
65	Cantidad trxs. Excedida	Rechazado, excede transacciones
71	Pin no cambiado	Rechazado, problema con pin
75	Negada, tentativa de pin excedida	Rechazado, problema con pin
75	Tentativa de Pin Excedida	Rechazado, problema con pin
89	Error en validación de CVV	Rechazado
91	Emisor no operativo	Rechazado, Intento más tarde
91	Timeout con el emisor	Rechazado, Intento más tarde



91	Timeout con emisor	Rechazado, Intente más tarde
91	Emisor Indisponible (no responde)	Rechazado, Intente más tarde
92	No es posible rutear a destino	Rechazado, Intente más tarde
92	Emisor indisponible	Rechazado, Intente más tarde
92	Emisor sin conexión	Rechazado, Intente más tarde
94	Transacción ya cancelada	Rechazado, transacción no permitida
96	Negada, sincronización de llave de pin con error	Error de Sistema
96	Adquiriente no soportado	Error de Sistema
96	Traslado en proceso	Error de Sistema
96	Error de sistema	Error de Sistema
96	EPS Security Software/Hardware Error	Error de Sistema
96	Data base error	Error de Sistema
96	Código de moneda no soportado	Error de Sistema
96	Error en formato del monto	Error de Sistema
96	Error en formato de fecha	Error de Sistema
96	Error en formato de cuenta	Error de Sistema
96	Error en formato de información recurrente	Error de Sistema
96	Error de Sistema	Error de Sistema
1001	Reintente insertando tarjeta	Rechazado, reintente insertando tarjeta
1002	Denegada, validación de CVV2 incorrecta	Rechazado
1003	Error al imprimir por tapa de impresora abierta	Rechazo, impresora Tapa Abierta
1004	Error al imprimir por falta de papel	Rechazo, falta de papel



1005	Error al imprimir por atasco de papel	Rechazo, Atasco de papel
1006	Cancelado por POS	Cancelado por POS

**15.3 ABREVIATURAS DE IDENTIFICADOR DE TARJETAS EN LA MENSAJERIA**

Tarjeta	Abreviatura
VISA	VI
MASTERCARD	MC
CABAL	CA
CREDENCIAL	CR
AMEX	AX
CERRADA	CE
DINNERS	DC
PRESTO	TP
MAGNA	MG
MAS (CENCOSUD)	TM
RIPLEY	RP
EXTRA	EX
CMR	TC
REDCOMPRA	DB