Índice

1. Introducción

Esta guía presenta una visión general de cómo pueden explotarse los servicios de CPS.

Dada la amplitud de los mismos y por tanto, la imposibilidad de detallarlos todos de forma minuciosa, la presente guía se centra en los aspectos más comunes y principales que debe tener en cuenta el desarrollador, siendo por tanto el resto, servicios que podrán ser explotados fácilmente una vez se alcance a dominar los aspectos mostrados en este documento. En cualquier caso, el desarrollador siempre puede hacer uso del departamento de soporte de CPS para consultar cualquier duda que le surja durante el propio desarrollo o durante el ciclo de vida de su producto.

Los fragmentos de código que se muestran en el presente documento están escritos de forma meramente académica con el propósito de hacer entender al lector los conceptos de los diversos aspectos que en él se tratan. Si bien pueden servir de guía al desarrollador para escribir el código fuente de su sistema, aconsejamos estructurar el código de un modo más ordenado, de forma que todo código repetitivo (o similar y que pueda ser parametrizado) se agrupe en métodos o funciones comunes que se puedan ejecutar reiteradamente cuando sea necesario.

Los ejemplos mostrados en esta guía utilizan como lenguajes de programación tanto VisualBasic.NET como C#.NET aunque el sistema puede ser explotado desde cualquier lenguaje de programación.

2. Modelos de integración

Existen dos formas de explotación de los servicios de CPS:

- **API Rest:** Es la forma nativa de conexión a CPS. Por su propia naturaleza, permite la comunicación desde cualquier sistema operativo, entorno o dispositivo programable con conexión a internet.
- **Dll:** Librería que se sirve de la API anterior y que proporciona un *CPSConnector* con el que poder acceder a todos los servicios de CPS de forma sencilla. Esta librería puede ser usada por distintos lenguajes de programación y sistemas operativos (Windows y Linux entre ellos) siempre que sean compatibles con .NET Standard 2.0.

Los detalles de modelos y endpoints disponibles se especifican con más detalle en: https://www.cloudpaymentservice.com/swagger. En dicha documentación se puede observar la posibilidad de personalización que contienen muchas de las peticiones gracias a un amplio número de parámetros. Pero esto no debe confundirse con complejidad ya que la mayoría de ellos son opcionales, por lo que se cumple el principio de "*Simplicity and power*" (Simplicidad y potencia) como se demuestra más adelante en los ejemplos de código expuestos en este documento.

Se opte por uno u otro modelo, es importante destacar que, por razones de seguridad y para evitar posibles ataques por DDOS, los servidores de CPS aceptarán un número máximo de peticiones por segundo a cada uno de los endpoints expuestos en la documentación anterior desde un mismo cliente. Superado dicho número, no se obtendrá la respuesta esperada a la petición y en su lugar se recibirá un mensaje en texto plano advirtiendo de dicha situación.

NOTA: *Aunque la estructuración y diseño del código para explotar la* ***API Rest*** *se dejan a elección del cliente (siempre que cumpla con las indicaciones de esta guía), cuando este sea el modelo de integración elegido sugerimos que, por facilidad de uso, el cliente se cree su propio CPSConnector en su propio lenguaje de programación. Para ello, dejamos este enlace con el código fuente de nuestra librería CPSConnector escrito en C#, con la idea de que pueda ser replicado al lenguaje de programación deseado.* ***Recordamos que no es necesario implementar todas las funcionalidades que en él aparecen sino sólo aquellas que requieran ser utilizadas y que el terminal a gobernar soporte****.*

3. Apertura y cierre de sesión

Antes de poder operar sobre un terminal CPS, debe iniciarse una sesión sobre el mismo. Para ello, CPS proporcionará al cliente las credenciales de acceso necesarias compuestas por los siguientes parámetros:

- **CPSCommerceCode:** Código de comercio CPS al que pertenece el terminal sobre el que se desea operar. Identifica al cliente CPS.
- **CPSCommercePassword:** Contraseña de acceso asignada al comercio anterior.

- **CPSTerminalCode:** Código que identifica al Terminal CPS sobre el que se desea operar.

EJEMPLO DE INICIO DE SESIÓN

DLL

```
Dim LoginRequest As LoginRequest = New LoginRequest()
LoginRequest.CPSCommerceCode = "215602532622" ' Código de comercio CPS
LoginRequest.CPSCommercePassword = "Password" ' Password del comercio CPS
LoginRequest.CPSTerminalCode = "211408393009" ' Código de terminal CPS

Dim APIError As String = Nothing
Dim HttpStatusCode As HttpStatusCode
Dim CPSConnector As CPSConnector = New CPSConnector()
AddHandler CPSConnector.TokenNotRefreshed, AddressOf TokenNotRefreshedhandler
' VER EXPLICACIÓN DEL EVENTO EN EPIGRAFE 4.1 Ejecucion por Dll

HttpStatusCode = CPSConnector.Login(LoginRequest, APIError)

If HttpStatusCode = System.Net.HttpStatusCode.OK Then
   ' SE HA PRODUCIDO EL LOGEO CORRECTAMENTE
   ' EJECUTAR PETICIONES. MIRAR EJEMPLO EPIGRAFE 4.1 Ejecucion por Dll
   CPSConnector.Logout()
Else
   Console.WriteLine("Error HTTP " + HttpStatusCode.ToString()+": "+APIError)
End If
```

EJEMPLO DE INICIO DE SESIÓN

API REST

```
string ResponseContent;
string ResponseContentType;
string url = "https://www.cloudpaymentservice.com/api/login";
string RequestContent = "{"CPSCommerceCode":"215602532622",
                         "CPSCommercePassword":"Password",
                         "CPSTerminalCode":"211408393009"}";

HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
request.Method = "POST";
request.ContentType = "application/json";

Byte[] byteArray = new System.Text.UTF8Encoding().GetBytes(RequestContent);
request.ContentLength = byteArray.Length;
request.GetRequestStream().Write(byteArray, 0, byteArray.Length);

HttpWebResponse response = (HttpWebResponse)request.GetResponse();
Stream responseStream = response.GetResponseStream();
StreamReader reader=new StreamReader(responseStream,System.Text.Encoding.UTF8);

ResponseContentType = response.ContentType; // Tipo de respuesta (JSON si OK)
ResponseContent = reader.ReadToEnd(); // Respuesta (generalmente en JSON)
HttpStatusCode httpStatusCode = response.StatusCode;

if (httpStatusCode == System.Net.HttpStatusCode.OK)
{
   // SE HA PRODUCIDO EL LOGEO CORRECTAMENTE
   // EJECUTAR PETICIONES. MIRAR EJEMPLO EPIGRAFE 4.2 Ejecucion por API REST
}
```

```
else
{
   Console.WriteLine("Error HTTP " + httpStatusCode.ToString()+": "+APIError);
}
```

Este inicio de sesión sobre el terminal **sólo es necesario que se realice una vez**. Una vez logados, se pueden ejecutar infinitas operaciones de forma continuada sobre el mismo. No obstante, pudiera ser necesario si no se consigue renovar el Token de autorización al que se hace referencia en el siguiente epígrafe *"4. Ejecución de peticiones"*.

Tal y como puede observarse en el código anterior referente al modelo de integración por *Dll*, el cierre de sesión se ejecuta a través del método *Logout* del *CPSConnector* una vez se hayan ejecutado todas las peticiones deseadas y tras haberse logado correctamente. Sin embargo, también puede observar cómo dicho cierre no se realiza si el modelo de integración usado es el nativo por API REST. Es más, en la documentación de la API ni siquiera verá un endpoint *Logout*.

Es decir, el cierre de sesión debe realizarse sí y sólo sí el modelo de integración usado es por *Dll*. Esto se debe a que, al ser dicha *Dll* la encargada internamente de renovar los Tokens antes de que caduquen, la misma posee un *timer* interno dedicado a realizar dicha renovación, que debe ser detenido una vez que no se vaya a usar más el objeto *CPSConnector*. En caso de no cerrar la sesión mediante el método *Logout* y finalizar la aplicación donde se integre este *CPSConnector*, es posible que esta no acabe cerrándose por dicho motivo, incluso cuando su interfaz gráfica sí lo haga.

4. Ejecución de peticiones

Tras el inicio de sesión sobre el terminal, la ejecución de peticiones es muy sencilla. No obstante, es aquí donde reside la mayor diferencia entre usar el modo de integración por **API Rest** o el modo de integración por **Dll**, ya que para poder ejecutar cada petición de manera segura debe implementarse lo que se conoce como flujo de autenticación "OAuth".

4.1. Ejecución por Dll

En caso de utilizar el modelo de integración por **Dll**, el desarrollador puede despreocuparse de implementar dicho flujo ya que este ya se encuentra implementado dentro del *CPSConnector* que dicha librería proporciona.

EJEMPLO DE EJECUCIÓN DE PETICION

DLL

```
Dim PaymentRequest As PaymentRequest = New PaymentRequest()
Dim PaymentResponse As PaymentResponse = New PaymentResponse()

PaymentRequest.Amount = 8.5 ' Importe
HttpStatusCode=CPSConnector.Payment(PaymentRequest,PaymentResponse,APIError)

If HttpStatusCode = System.Net.HttpStatusCode.OK Then
   ' TRATAR RESPUESTA
   Console.WriteLine(PaymentResponse.ToString())
Else
   Console.WriteLine("Error HTTP " + HttpStatusCode.ToString()+": "+APIError)
```

```
End If
```

A pesar de ello, debe tenerse en cuenta que ante cualquier problema que imposibilite al *CPSConnector* conectar con los servidores de CPS para renovar el token (como por ejemplo, caída de internet), dicho *CPSConnector* no podrá renovar el Token y será por tanto necesario realizar un nuevo Login. Esta situación podrá gestionarse a través de su evento *TokenNotRefreshed*:

EJEMPLO DE GESTIÓN DEL EVENTO *TokenNotRefreshed*

DLL

```
Private Sub TokenNotRefreshedHandler(sender As Object,e As
TokenNotRefreshedEventArgs)

  Dim LoginRequest As LoginRequest = New LoginRequest()
  LoginRequest.CPSCommerceCode = "215602532622" ' Código de comercio CPS
  LoginRequest.CPSCommercePassword = "Password" ' Password del comercio CPS
  LoginRequest.CPSTerminalCode = "211408393009" ' Código de terminal CPS

  Dim APIError As String = Nothing

  While CPSConnector.Login(LoginRequest,APIError)<>System.Net.HttpStatusCode.OK
     Console.WriteLine("Terminal no operativo": " + APIError)
     Threading.Thread.Sleep(5000)
  End While

End Sub
```

4.2. Ejecución por API REST

En este caso, la petición HTTP debe realizarse de manera "autorizada", por lo que debe incluir en su cabecera un Token de Autenticación. Dicho Token fue recibido en la respuesta obtenida al Login previo.

EJEMPLO DE EJECUCIÓN DE PETICIÓN

API REST

```
string AccessToken = "?" // Recibido en ResponseContent del Login previo

url = "https://www.cloudpaymentservice.com/api/payment";
RequestContent = "{"Amount":8.5}";

HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
request.Method = "POST";
request.ContentType = "application/json";
request.Headers.Add(HttpRequestHeader.Authorization,"Bearer "+AccessToken);

Byte[] byteArray = new System.Text.UTF8Encoding().GetBytes(RequestContent);
request.ContentLength = byteArray.Length;
request.GetRequestStream().Write(byteArray, 0, byteArray.Length);

HttpWebResponse response = (HttpWebResponse)request.GetResponse();
Stream responseStream = response.GetResponseStream();
StreamReader reader=new StreamReader(responseStream,System.Text.Encoding.UTF8);
```

```
ResponseContentType = response.ContentType; // Tipo de respuesta (JSON si OK)
ResponseContent = reader.ReadToEnd(); // Respuesta (generalmente en JSON)
HttpStatusCode httpStatusCode = response.StatusCode;

if (httpStatusCode == System.Net.HttpStatusCode.OK)
{
   // TRATAR RESPUESTA
   Console.WriteLine(ResponseContent);
}
else
{
   Console.WriteLine("Error HTTP " + httpStatusCode.ToString()+": "+APIError);
}
```

Lo más destacable, no obstante, es que el AccessToken obtenido en la respuesta al Login caduca a los 5 minutos desde su obtención. Por tanto, toda petición que quiera ser ejecutada pasado ese tiempo no podrá ser autenticada con dicho AccessToken.

Será por tanto necesario obtener previamente un nuevo AccessToken (ya sea antes o después de que expire el actual) atacando al endpoint "*RefreshToken*" de manera "anónima" (sin añadir cabecera de autenticación).

(Véase https://www.cloudpaymentserver.com/swagger/ui/index#!/Session/API_RefreshToken)

La petición a este endpoint "*RefreshToken*" deberá contener tanto el AccessToken actual como el RefreshToken (Recibido también junto al primero en la respuesta al Login), necesario para poder realizar el refresco del primero.

Si ambos son validados como correctos por los servidores de CPS, se devolverá una respuesta similar a la obtenida en el Login con los nuevos AccessToken y RefreshToken a usar a partir de ese momento y que de nuevo deberán gestionarse igualmente que los anteriores.

Por último, debe tenerse presente que el RefreshToken también expira, en este caso a los 60 minutos, por lo que si pasado dicho tiempo este no ha sido usado para obtener un nuevo AccessToken, ambos Tokens quedarán revocados y será por tanto necesario realizar un nuevo Login.

Aunque la implementación de este modelo de autenticación se deja en manos del desarrollador, aconsejamos la creación de un *Timer* que, a intervalos algo anteriores a que expire el AccessToken (por ejemplo, cada 4'5 minutos), se encargue de refrescar el mismo y de esta forma tenerlo siempre vigente y listo para cuando se requiera ejecutar una petición.

5. Tratamiento de respuestas

Aunque ya se ha adelantado algo en los ejemplos anteriores, es interesante comprender correctamente el tratamiento que debe darse a las respuestas obtenidas a cada petición.

En primer lugar, el HttpStatusCode que se observa en los anteriores ejemplos nos indica el código HTTP de respuesta obtenido desde los servidores CPS y se refiere a si estos aceptan la petición como procesable o no.

Aunque cada uno de los endpoints en https://www.cloudpaymentservice.com/swagger expone cuales son los posibles códigos HTTP de respuesta que puede devolver, los códigos más comunes son:

HTTP STATUS CODE	REASON
200	Ok
400	BadRequest: Invalid request
401	Unauthorized: Access token expired
429	Too Many Request
500	Internal Server Error: Exception connecting CPS Server

Nota: *el código 429, aunque no aparece en la documentación indicada, será devuelto por cualquier endpoint en caso de superar el número máximo de peticiones permitidas por segundo.*

Por tanto, un **HttpStatusCode igual a 200 indica que la petición ha sido aceptada como procesable por los servidores de CPS**. En caso de que así sea, deberá investigarse a continuación el objeto *ResponseResultInfo* obtenido en la respuesta, ya que nos facilitará la siguiente información acerca del resultado de su procesamiento:

PROPIEDAD	SIGNIFICADO
Success	Indica si la operación ha podido ejecutarse correctamente
ResultCode	En caso de que la operación se haya ejecutado satisfactoriamente (Success=True), indica el resultado funcional de la operación. El valor 0 indica que la operación se ha realizado satisfactoriamente.
ResultInfo	Información en texto plano relativa al resultado final de la operación. Normalmente muestra un mensaje descriptivo cuando ResultCode no es 0.

Para ejemplificar todo lo anterior, se exponen los siguientes casos:

CASUÍSTICA	RESULTADOS	EXPLICACIÓN
Se intenta ejecutar una petición de cobro sobre un terminal pero no se indica el importe a cobrar	HTTPStatusCode: 400	La petición no es procesable. Está mal compuesta.
Se intenta ejecutar una petición de cobro con un AccessToken caducado o inexistente	HTTPStatusCode: 401	La petición no es procesable. No está autorizada
Se intenta ejecutar una petición sobre un terminal que ya está procesando otra petición solicitada por otro cliente HTTP	HTTPStatusCode: 200 ResponseResultInfo.Success: False	La petición es procesable, pero no puede ser procesada
Se intenta ejecutar una petición de cobro sobre un terminal y la tarjeta bancaria empleada está caducada	HTTPStatusCode: 200 ResponseResultInfo.Success: True ResponseResultInfo.ResultCode: >0	La petición ha sido procesada, pero su resultado no ha sido el esperado

Se intenta ejecutar una petición de cobro sobre un terminal y la tarjeta bancaria empleada no tiene saldo suficiente para poder realizar el cargo solicitado	HTTPStatusCode: 200 ResponseResultInfo.Success: True ResponseResultInfo.ResultCode: >0	La petición ha sido procesada, pero su resultado no ha sido el esperado
Se intenta ejecutar una petición de cobro sobre un terminal y se acaba realizando el cargo deseado en la tarjeta	HTTPStatusCode: 200 ResponseResultInfo.Success: True ResponseResultInfo.ResultCode: 0	La petición ha sido procesada y su resultado ha sido el esperado

6. Timeout de ejecución

Como en otros muchos desarrollos, es importante:

- Controlar el tiempo máximo que el sistema está dispuesto a esperar para recibir respuesta a una solicitud enviada, en este caso, a los servidores de CPS.
- Determinar cuál será el comportamiento del sistema si se supera dicho tiempo sin haber recibido respuesta.

Si bien el segundo punto se deja completamente en manos del desarrollador, sobre el primero mostramos a continuación un par de ejemplos de cómo hacerlo en cada uno de los dos modelos de integración:

EJEMPLO DE TIMEOUT

DLL

```
Dim LoginRequest As LoginRequest = New LoginRequest()
LoginRequest.CPSCommerceCode = "215602532622" ' Código de comercio CPS
LoginRequest.CPSCommercePassword = "Password" ' Password del comercio CPS
LoginRequest.CPSTerminalCode = "211408393009" ' Código de terminal CPS
LoginRequest.ResponseTimeOut = 500 ' Segundos máximos de espera en cualquier petición que se realice en la sesión que se va a iniciar con este Login a menos que en cada una de ellas se especifique un valor concreto

Dim APIError As String = Nothing
Dim HttpStatusCode As HttpStatusCode
Dim CPSConnector As CPSConnector = New CPSConnector()

HttpStatusCode = CPSConnector.Login(LoginRequest, APIError)

If HttpStatusCode = System.Net.HttpStatusCode.OK Then

  Dim PaymentRequest As PaymentRequest = New PaymentRequest()
  Dim PaymentResponse As PaymentResponse = New PaymentResponse()

  PaymentRequest.Amount = 8.5 ' Importe
  PaymentRequest.ResponseTimeOut = 600 ' Segundos máximos de espera para esta solicitud concreta
  HttpStatusCode=CPSConnector.Payment(PaymentRequest,PaymentResponse,APIError)

  If HttpStatusCode = System.Net.HttpStatusCode.OK Then
     ' TRATAR RESPUESTA
    Console.WriteLine(PaymentResponse.ToString())
```

```
  Else
    Console.WriteLine("Error HTTP " + HttpStatusCode.ToString()+": "+APIError)
  End If
  CPSConnector.Logout()
Else
  Console.WriteLine("Error HTTP " + HttpStatusCode.ToString()+": "+APIError)
End If
```

Nota 1: *Cuando se usa el modelo de integración por **Dll**, dicho TimeOut puede establecerse en el Login de forma que a partir de ese momento, dicho TimeOut será el utilizado por todas las peticiones que se realicen dentro de la sesión iniciada por el mismo a menos que en cada petición se indique específicamente un valor distinto. Si en el Login no se especifica ningún valor, el valor por defecto es **INFINITO** (Así ocurre en el ejemplo del epígrafe "2. Inicio de sesión" de este documento).*

Nota 2: *Cuando se usa el modelo de integración por **Dll**, si en una petición no se fija un valor concreto para ResponseTimeOut, esta tomará como valor de TimeOut el fijado en el inicio de sesión realizado con el comentado Login.*

EJEMPLO DE TIMEOUT

API REST

```
string ResponseContent;
string ResponseContentType;
string url = "https://www.cloudpaymentservice.com/api/login";
string RequestContent = "{"CPSCommerceCode":"215602532622",
                          "CPSCommercePassword":"Password",
                          "CPSTerminalCode":"211408393009"}";

HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
request.Method = "POST";
request.ContentType = "application/json";
request.Timeout = 500 * 1000; // Tiempo en milisegundos

Byte[] byteArray = new System.Text.UTF8Encoding().GetBytes(RequestContent);
request.ContentLength = byteArray.Length;
request.GetRequestStream().Write(byteArray, 0, byteArray.Length);

HttpWebResponse response = (HttpWebResponse)request.GetResponse();
Stream responseStream = response.GetResponseStream();
StreamReader reader=new StreamReader(responseStream,System.Text.Encoding.UTF8);

ResponseContentType = response.ContentType; // Tipo de respuesta (JSON si OK)
ResponseContent = reader.ReadToEnd(); // Respuesta (generalmente en JSON)
HttpStatusCode httpStatusCode = response.StatusCode;

if (httpStatusCode == System.Net.HttpStatusCode.OK)
{
   // EJECUTAR PETICIÓN. MIRAR EJEMPLO EPIGRAFE 3.2 Ejecucion por API REST
}
else
{
   Console.WriteLine("Error HTTP " + httpStatusCode.ToString()+": "+APIError);
}
```

Nota 1: *Cuando se usa el modelo de integración por **API REST**, el TimeOut sólo puede ser fijado en cada petición. El valor por defecto de dicho TimeOut (si no se establece ninguno) o el valor a*

fijar si se desea que sea infinito dependerá del lenguaje de programación usado por el desarrollador.

Al fijar un valor a este TimeOut, recomendamos se tenga presente, según la operación a ejecutar, el tiempo real que puede llegar a ser necesario para ejecutar la misma. Por ejemplo:

- **Una petición de pago en un terminal desatendido para pagar con tarjeta bancaria:** Debe tener presente que es posible que el usuario tenga que buscar su tarjeta en la cartera antes de introducirla, que introduzca la tarjeta al revés en el terminal y tenga que extraerla y volverla a meter de nuevo, que se equivoque un par de veces antes de digitar el pin correcto, que la central de pagos apruebe la operación y responda… Todo ello supone bastante tiempo, por lo que si a dicha petición establecemos un TimeOut de por ejemplo 10 segundos, muy probablemente no vamos a dar tiempo al punto de venta para que sepa si la operación ha acabado realizándose de forma satisfactoria o no.
- **Una petición de cambio de idioma en el terminal:** Es algo que debe ser rápido por lo que un TimeOut de 15 segundos (lo que ya de por sí da bastante margen) debería ser suficiente por ejemplo.

7. Entornos de trabajo

CPS pone a disposición de sus clientes tanto un entorno de PRODUCCIÓN real como un entorno de PREPRODUCCIÓN donde realizar pruebas:

- Producción: https://www.cloudpaymentservice.com/api
- Preproducción: https://www.cloudpaymentservice.com:444/api

La forma de especificar contra cuál de los dos entornos se desea trabajar se ejemplifica a continuación:

EJEMPLO DE SELECCIÓN DE ENTORNO DE TRABAJO

DLL

```
Dim LoginRequest As LoginRequest = New LoginRequest()
LoginRequest.CPSCommerceCode = "215602532622" ' Código de comercio CPS
LoginRequest.CPSCommercePassword = "Password" ' Password del comercio CPS
LoginRequest.CPSTerminalCode = "211408393009" ' Código de terminal CPS


Dim APIError As String = Nothing
Dim HttpStatusCode As HttpStatusCode
Dim CPSApiUrl As String = "https://www.cloudpaymentservice.com:444/api"
Dim CPSConnector As CPSConnector = New CPSConnector(CPSApiUrl)

HttpStatusCode = CPSConnector.Login(LoginRequest, APIError)

If HttpStatusCode = System.Net.HttpStatusCode.OK Then
   ‘ EJECUTAR PETICIONES. MIRAR EJEMPLO EPIGRAFE 3.1 Ejecucion por Dll
   CPSConnector.Logout()
Else
   Console.WriteLine("Error HTTP " + HttpStatusCode.ToString()+": "+APIError)
End If
```

Nota: *Si al constructor de CPSConnector no se le pasa el parámetro CPSApiUrl, conectará contra el entorno de* ***PRODUCCIÓN****.*

EJEMPLO DE SELECCIÓN DE ENTORNO DE TRABAJO

API REST

```
string ResponseContent;
string ResponseContentType;
string url = "https://www.cloudpaymentservice.com:444/api/login";
string RequestContent = "{"CPSCommerceCode":"215602532622",
                          "CPSCommercePassword":"Password",
                          "CPSTerminalCode":"211408393009"}";

HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
request.Method = "POST";
request.ContentType = "application/json";

Byte[] byteArray = new System.Text.UTF8Encoding().GetBytes(RequestContent);
request.ContentLength = byteArray.Length;
request.GetRequestStream().Write(byteArray, 0, byteArray.Length);

HttpWebResponse response = (HttpWebResponse)request.GetResponse();
Stream responseStream = response.GetResponseStream();
StreamReader reader=new StreamReader(responseStream,System.Text.Encoding.UTF8);

ResponseContentType = response.ContentType; // Tipo de respuesta (JSON si OK)
ResponseContent = reader.ReadToEnd(); // Respuesta (generalmente en JSON)
HttpStatusCode httpStatusCode = response.StatusCode;

if (httpStatusCode == System.Net.HttpStatusCode.OK)
{
   // EJECUTAR PETICIONES. MIRAR EJEMPLO EPIGRAFE 3.2 Ejecucion por API REST
}
else
{
   Console.WriteLine("Error HTTP " + httpStatusCode.ToString()+": "+APIError);
}
```

En el entorno de preproducción se podrán realizar todas las pruebas necesarias durante el desarrollo en un entorno 100% similar al entorno real. De todos modos, deben tenerse presente las siguientes consideraciones:

- ***Entorno de producción:*** En este entorno todas las operaciones de cobro realizadas a través de CPS serán reales y serán facturadas al cliente a final de mes.

- ***Entorno de preproducción:*** Es un entorno idéntico al anterior sobre el que los clientes de CPS podrán testear la comunicación entre sus sistemas y CPS para gobernar los medios de pagos deseados y simular la gestión de su comercio a través del área de clientes como si de un sistema real se tratase. En este entorno, y en función del medio de pago seleccionado, es posible que ciertas operaciones de cobro puedan ser reales y otras no, por lo que el cliente deberá tenerlo en cuenta a la hora de realizar sus pruebas, aunque en cualquier caso, siempre será informado previamente desde CPS de dicha

situación. Lo que sí es seguro es que ninguna de estas operaciones serán facturadas al cliente a final de mes por parte de CPS.

Como se ha comentado previamente, al igual que el entorno de producción, el entorno de preproducción también dispone de un área de clientes totalmente funcional, por lo que podrá acceder a ella durante las labores de desarrollo si le es necesario:

- Producción: https://www.cloudpaymentservice.com/Account/Login
- Preproducción: https://www.cloudpaymentservice.com:444/Account/Login

Por último, también es importante aclarar que las credenciales de acceso asignadas por CPS al cliente *(CPSCommerceCode, CPSCommercePassword y CPSTerminalCode)* están ligadas a un entorno de trabajo, no siendo por tanto válidas para el otro entorno. Es decir, se usarán unas credenciales durante el desarrollo distintas a las que deban ser usadas cuando el sistema se encuentre en producción real.

8. El modo compartido

Aunque inicialmente se configura como *"Desactivado" (excepto en los terminales de e-commerce que se hace como "Automático" por motivos internos)*, todos los terminales CPS tienen la posibilidad de activar el modo compartido.

El modo compartido permite a un terminal CPS ser utilizado simultáneamente desde múltiples puntos de venta, ya sean atendidos o desatendidos, con el objetivo de ahorrar costes en terminales físicos. Véanse los siguientes ejemplos:

- ***Ejemplo 1:*** Imagine tres TPVs atendidos en una tienda que quieran usar el mismo terminal de pago atendido (datáfono).
- ***Ejemplo 2:*** Imagine tres máquinas vending distintas que quieran hacer uso del mismo terminal de pago desatendido.

Las arquitecturas representadas en ambos ejemplos suponen claramente un ahorro de costes en terminales físicos respecto a disponer de un terminal de cobro distinto para cada punto de venta, pero generarían errores si mientras que el terminal de cobro está realizando una operación solicitada por uno de los puntos de venta (TPV o maquina vending), recibe otra petición desde algún otro.

Para evitar este tipo de problemas, el modo compartido se encuentra inicialmente desactivado, lo que supondrá en esos casos obtener *ResponseResultInfo.Success = False* y *ResponseResultInfo.ResultInfo = 'Terminal ocupado'* en la respuesta a la petición enviada por el segundo punto de venta.

Sin embargo, con el modo compartido podemos cambiar este comportamiento de dos formas distintas:

- **Modo compartido Automático:** Cuando el modo compartido se fija en "Automático", el desarrollador pude despreocuparse completamente de si el terminal de pago está siendo usado o no por otro punto de venta en el momento en el que su sistema desea enviarle una petición. En este caso, sencillamente debe enviar la petición y esta se

ejecutará en el terminal inmediatamente después de que la actual haya finalizado, en el caso de que ya haya una actualmente en curso. No obstante, es importante destacar que en este modo, cuando coexistan a la vez varias peticiones pendientes de ejecución provenientes desde distintos puntos de venta, si bien todas se ejecutarán automáticamente una detrás de otra cuando finalice la anterior, no se puede garantizar que se ejecuten en el mismo orden en que se han solicitado, por lo que el tiempo de espera de respuesta de alguna de ellas es posible que sea elevado si fue una de las primeras en enviarse y finalmente es una de las últimas en ejecutarse.

- **Modo compartido Manual:** Aunque en la mayoría de las ocasiones, fijar el modo compartido en "Automático" es suficiente, es posible que el desarrollador requiera un mayor control que le permita saber si un terminal está o no siendo utilizado por otro punto venta cuando su sistema requiera usar dicho terminal. En dicho caso, puede fijarse el modo compartido en "Manual", lo que obligará a cada punto de venta a solicitar el control del terminal antes de usarlo y a liberarlo una vez haya terminado.

  Esta obtención y liberación del control del terminal se realiza con los endpoints *"StartTransaction"* y *"EndTransaction"*, de modo que cada punto de venta que se haya logado contra el terminal, antes de enviarle cualquier petición deberá enviarle un *"StartTransaction"*, ejecutar entonces todas las peticiones que desee sobre él si le ha sido concedido el acceso y ejecutar un *"EndTransaction"* cuando ya no requiera más del mismo, para así liberarlo y que pueda ser usado por otro punto de venta.

  Obviamente la ejecución de *"StartTransaction"* devolverá *ResponseResultInfo.Success = False* si el terminal ya está siendo controlado por otro punto de venta.

  También es importante destacar que, una vez concedido el control a un punto de venta con *"StartTransaction"*, se entiende que dicho punto de venta ha abierto una sesión contra el mismo. Dicha sesión expirará cuando el punto de venta ejecute un *"EndTransaction"* o cuando durante la misma se produzca un tiempo continuado de inactividad (no envío de ningún comando) de 120 segundos (sin tener en cuenta los tiempos de respuesta a cada comando).

EJEMPLO DE MODO COMPARTIDO MANUAL

DLL

```
' SUPONIENDO QUE YA SE HA REALIZADO EL LOGIN CORRECTAMENTE

Dim HttpStatusCode As HttpStatusCode
Dim StartRes As New StartTransactionResponse()
Dim APIError As String = Nothing

HttpStatusCode = CPSConnector.StartTransaction(StartRes, APIError)
If HttpStatusCode = HttpStatusCode.OK Then
   If StartRes.ResponseResultInfo.Success AndAlso StartRes.ResultCode = 0 Then

      ' SE HA OBTENIDO EL CONTROL DEL TERMINAL.
```

```
      ' EJECUTAMOS TODAS LAS PETICIONES QUE SE DESEEN SOBRE EL TERMINAL…

      ' FINALIZAMOS LA TRANSACCIÓN PARA LIBERAR EL TERMINAL.
      Dim EndRes As New EndTransactionResponse()

      HttpStatusCode = CPSConnector.EndTransaction(EndRes, APIError)
      If HttpStatusCode = HttpStatusCode.OK Then
         If EndRes.ResponseResultInfo.Success AndAlso EndRes.ResultCode=0 Then
            ' SE HA FINALIZADO LA TRANSACCIÓN Y LIBERADO EL TERMINAL
         Else
            Console.WriteLine(StartRes.ResponseResultInfo.ResultInfo)
         End If
      Else
         Console.WriteLine("Error HTTP"+HttpStatusCode.ToString()+":"+APIError)
      End If

   Else
      Console.WriteLine("No ha sido posible obtener el control del terminal: "+
StartRes.ResponseResultInfo.ResultInfo) ' ResultInfo contendrá algo similar a
“Terminal ocupado”
   End If
Else
   Console.WriteLine("Error HTTP"+HttpStatusCode.ToString()+":"+APIError)
End If
```

EJEMPLO DE MODO COMPARTIDO MANUAL

API REST

```
string AccessToken = "?" // Recibido en ResponseContent del Login previo
string url = "https://www.cloudpaymentservice.com/api/starttransaction";

HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
request.Method = "GET";
request.Headers.Add(HttpRequestHeader.Authorization,"Bearer "+AccessToken);

HttpWebResponse response = (HttpWebResponse)request.GetResponse();
Stream responseStream = response.GetResponseStream();
StreamReader reader=new StreamReader(responseStream,System.Text.Encoding.UTF8);

ResponseContentType = response.ContentType; // Tipo de respuesta (JSON si OK)
ResponseContent = reader.ReadToEnd(); // Respuesta (generalmente en JSON)
HttpStatusCode httpStatusCode = response.StatusCode;

if (httpStatusCode == System.Net.HttpStatusCode.OK)
{
   // TRATAR ResponseContent COMO UNA CADENA JSON PARA DETERMINAR LOS VALORES
DE RESPONSERESULTINFO.SUCCESS, RESPONSERESULTINFO.RESULTCODE Y
RESPONSERESULTINFO.RESULTINFO

   if (ResponseResultInfo.Success && StartRes.ResultCode == 0)
   {

      // SE HA OBTENIDO EL CONTROL DEL TERMINAL.

      // EJECUTAMOS TODAS LAS PETICIONES QUE SE DESEEN SOBRE EL TERMINAL…
```

```
      // FINALIZAMOS LA TRANSACCIÓN PARA LIBERAR EL TERMINAL.

      string url = "https://www.cloudpaymentservice.com/api/endtransaction";

      request = (HttpWebRequest)WebRequest.Create(url);
      request.Method = "GET";
      request.Headers.Add(HttpRequestHeader.Authorization,"Bearer
"+AccessToken);

      response = (HttpWebResponse)request.GetResponse();
      responseStream = response.GetResponseStream();
      reader = new StreamReader(responseStream,System.Text.Encoding.UTF8);

      ResponseContentType = response.ContentType; // Tipo de respuesta (JSON si
OK)
      ResponseContent = reader.ReadToEnd(); // Respuesta (generalmente en JSON)
      httpStatusCode = response.StatusCode;

      if (httpStatusCode == System.Net.HttpStatusCode.OK)
      {
         // TRATAR ResponseContent COMO UNA CADENA JSON PARA DETERMINAR LOS
VALORES DE RESPONSERESULTINFO.SUCCESS, RESPONSERESULTINFO.RESULTCODE Y
RESPONSERESULTINFO.RESULTINFO

         if (ResponseResultInfo.Success && StartRes.ResultCode == 0)
         {
             // SE HA FINALIZADO LA TRANSACCIÓN Y LIBERADO EL TERMINAL.
         }
         else
         {
            Console.WriteLine(StartRes.ResponseResultInfo.ResultInfo);
         }
      }
      else
      {
         Console.WriteLine("Error HTTP " + httpStatusCode.ToString()+":
"+APIError);
      }
   }
   else
   {
      Console.WriteLine("No ha sido posible obtener el control del terminal: "+
StartRes.ResponseResultInfo.ResultInfo); ' ResultInfo contendrá algo similar a
“Terminal ocupado”
   }
}
else
{
   Console.WriteLine("Error HTTP " + httpStatusCode.ToString()+": "+APIError);
}
```

Por último, debemos comentar que tanto **la activación/desactivación del modo compartido en un terminal** como **la configuración del tiempo continuado de inactividad para dar una transacción por finalizada** en el mismo, **es algo que debe ser solicitado al servicio técnico de CPS**, no siendo posible configurar dichos valores por tanto por el propio desarrollador.

Nótese además que si bien el cambio de modo compartido entre “Desactivado” y “Automático” y viceversa puede realizarse sin problemas, el modo “Manual” sí requerirá al desarrollador que todos los puntos de venta que compartan dicho terminal realicen los “StartTransactión” y “EndTransaction” correspondientes y viceversa, es decir, pasar de modo “Manual” a cualquiera de los otros dos impide al desarrollador realizar dichas operaciones.

9. El comando TagReading

El endpoint “TagReading” permite solicitar al terminal que lea el PAN almacenado en un Tag (coloquialmente, que lea el número de una tarjeta).

En función del terminal físico que estemos controlando, esta petición puede provocar por ejemplo que el terminal solicite al usuario la introducción de la tarjeta o simplemente que se quede a la espera de que la misma sea acercada o introducida.

Pero igualmente, en función del terminal físico y, sobre todo, si la lectura no se ha realizado por proximidad (contactless) sino por introducción del Tag, es posible que, una vez haya sido leído y hayamos obtenido el PAN como respuesta, para poder proceder a su extracción sea necesario enviar una petición “EndCurrentOperation”.

Por tanto, aconsejamos que siempre que obtenga respuesta a un “TagReading”, se envíe un “EndCurrentOperation” para asegurarse de que el terminal solicite al usuario la extracción del Tag. En el peor de los casos, es posible que dicha orden simplemente no sea necesaria en dicho terminal, lo cual no supondría un problema.

No obstante, es importante reseñar que entre la respuesta al “TagReading” y el envío del “EndCurrentOperation” es posible realizar otras operaciones si se desean, como por ejemplo “Payment”. En este caso por ejemplo, primero solicitaríamos la lectura del Tag y, una vez conocido su PAN y estando el Tag aún introducido en el terminal, solicitaríamos un cobro. Una vez se obtenga la respuesta a dicho cobro ya podríamos solicitar la extracción del Tag con “EndCurrentOperation”.

En referencia al párrafo anterior, téngase presente que para ejecutar “Payment” no es necesario realizar antes un “TagReading”, ya que si la tarjeta no está introducida en ese momento, el propio terminal se encargará de solicitar su introducción automáticamente si así es necesario por su naturaleza. La diferencia entre el caso expuesto en el párrafo anterior y el expuesto en este es que el primero permite conocer el PAN de la tarjeta (y otras propiedades más, como su tipología por ejemplo) antes de lanzar el cobro, lo que permitiría no acabar lanzándolo si así se decidiese en función de dicha información. Un caso muy típico de esta casuística es cuando se desea dar al usuario la posibilidad de introducir previamente al pago un Tag privado que sirva para identificarlo antes de realizar el cobro. De esta forma, solicitaríamos la lectura de Tag y una vez sabido su PAN y sobre todo su tipología (que nos indicará si se trata de una tarjeta bancaria o una de identificación), decidiríamos si lanzar el cobro contra dicho Tag o si por el contrario solicitar su extracción para dar por identificado al usuario y lanzar una orden de pago para, ahora ya sí, realizar el cobro.

10. Los terminales de e-commerce

Los terminales CPS para e-commerce requieren de una mención especial dada su naturaleza.

En primer lugar debemos aclarar que existen dos tipos de terminales e-commerce:

- **Los destinados a ser integrados en una tienda virtual on-line**: Caso típico de un carrito de compra en cualquier web.
- **Los destinados a ser accedidos mediante un enlace enviado al cliente por SMS y/o mail**.

En función del tipo de terminal (que será acordado por el cliente con CPS en el momento de su alta), en las respuestas a la petición de pago sobre el mismo obtendremos la siguiente información:

- **Los destinados a ser integrados en una tienda virtual on-line**: En Response.eCommerceinfoResponse.PaymentHtml se obtendrá el código HTML del botón a incluir en la Web para poder lanzar el pago. Para temas estéticos y de CSS, dicho botón está asignado a la clase "CPSPaymentButton", que puede definir en su CSS. También puede reemplazar en el código HTML dicha cadena por la clase CSS que desee antes de incluir el código HTML en su web.
- **Los destinados a ser accedidos mediante un enlace enviado al cliente por SMS y/o mail**: En este caso, en Response.eCommerceinfoResponse.PaymentUrl obtendrá la URL a la que el cliente deberá acceder para procesar el pago. La forma de envío de dicha URL al cliente dependerá de lo acordado por el comercio con CPS en el momento del alta del terminal. El envío puede ser realizado por el comercio según el medio que desee, o puede solicitar a CPS que se envíe al cliente por SMS y/o por Mail en el mismo momento de generación de la operación. En cualquier caso, el comercio siempre recibirá esta URL en la respuesta.

Sea cual sea el tipo de terminal empleado, una vez el cliente acceda a la web de pago, este simplemente deberá introducir los datos de la tarjeta y aceptar el pago, lanzándose automáticamente a continuación el proceso de autenticación del titular mediante el protocolo 3DSecure determinado por el emisor de la tarjetas (ya sea la versión del mismo la 3DSv1 o la EMV 3DSv2) si es que así lo tiene establecido dicho emisor.

Una vez la operación haya finalizado ocurrirá lo siguiente:

- **Si la operación se ha realizado satisfactoriamente:**
    1. Se volverá a enviar la respuesta a dicha petición de pago (pero ahora con toda la información del pago ya completa) a la Url fijada en la petición de pago como *eCommerceURLs.UrlNotification* si es que se fijó alguna. Esto permitirá al comercio disponer de toda la información relativa al cobro por si quiere procesarla de alguna forma.
    2. Se devolverá la sesión del navegador del cliente al comercio para que continúe navegando en su sitio web. En concreto lo hará a la Url fijada en la petición de pago como *eCommerceURLs.UrlOK.*
- **Si la operación no se ha realizado satisfactoriamente:**

1. Se devolverá la sesión del navegador del cliente al comercio para que continúe navegando en su sitio web. En concreto lo hará a la Url fijada en la petición de pago como *eCommerceURLs.UrlKO.*

Dado que en la propia web de pago se informa al finalizar la operación de si esta ha sido o no satisfactoria, muchas comercios optan por que las Url's *"UrlOk"* y *"UrlKO"* sean la misma, pero obviamente es algo que se deja en manos del desarrollador en cada caso.

10.1. Tokenización de tarjetas

Ciertos comercios que integran una tienda virtual on-line, requieren que su web sea capaz de almacenar y gestionar, para cada usuario, cuáles son sus tarjetas bancarias de forma que este no tenga que facilitar todos los datos de la/s misma/s cada vez que va a realizar una compra, sobre todo cuando realiza compras recurrentes.

Pero en la mayoría de los casos, dichos comercios no cumplen la normativa PCI DSS, necesaria para poder procesar y tratar datos de tarjetas bancarias en "crudo", por lo que les es imposible llevar a cabo dicha implementación.

Para solventar este problema, CPS pone a disposición de sus clientes lo que se conoce como Tokenización de tarjetas. De esta forma, al introducir los datos de la tarjeta en la primera compra, es posible solicitar a CPS que en la respuesta sea devuelto un Token que identifique unívocamente a dicha tarjeta exclusivamente para dicho comercio. Además de dicho Token, en la respuesta también se obtendrá otra información parcialmente identificativa de la misma, como su PAN enmascarado o asteriscado, donde ciertos dígitos del mismo son sustituidos por asteriscos.

Esto permitirá al comercio, ahora sí, almacenar y gestionar dicha información de la tarjeta junto con el mencionado Token, lo que posibilitará en futuras compras ofertar al usuario con cuál de sus tarjetas desea realizar el pago sin necesidad de que este vuelva a digitar toda la información de la misma, ya que bastará con indicar a CPS el Token de la tarjeta contra el que se desea realizar el cobro en el momento del mismo.

Para llevar todo esto a cabo, bastará con utilizar la propiedad *TagInfoRequest.TokenizedTag* en la petición de cobro de la siguiente forma:

- **No se le asigna ningún valor:** El usuario deberá introducir todos los datos de la tarjeta que desea utilizar para realizar el pago en la URL de pago a la que se ha redirigido la navegación y no se obtendrá ningún Token que identifique a la misma en la respuesta obtenida. Esta es la casuística normal en la que no se usa Tokenización.
- **Se le asigna el valor *"REQUIRED"*:** Ocurrirá lo mismo que en el caso anterior, solo que en esta ocasión si será devuelto en la respuesta un Token que identifica unívocamente a dicha tarjeta exclusivamente dentro de dicho comercio y que podrá ser utilizado en futuras compras. En concreto, se obtendrá en la propiedad *TagInfoResponse.TokenizedTag* de dicha respuesta.
- **Se le asigna como valor un Token obtenido anteriormente:** El cargo se realizará automáticamente sobre la tarjeta identificada por dicho Token, sin solicitar de nuevo al usuario que introduzca todos los datos de la misma.

11. Ejemplo de trazas para API REST

Se presentan a continuación a modo de ejemplo, las trazas **completas** de petición y respuesta para una petición de pago de 10 € contra un terminal desatendido. Recuerde que en la petición realmente no es obligatorio indicar todos los parámetros como aquí aparecen, sino solo aquellos no sean opcionales *(Véase https://www.cloudpaymentservice.com/swagger)* o que aun siéndolos, se requieran por algún motivo:

11.1. Petición

{"PaymentType":"Normal","AuthenticationMode":"Auto","OperationsGroup":null,"VehicleInfo":{"LicensePlate":null,"Kilometers":0},"Amount":10.0,"SalesLines":[],"PaymentMethod":"GenericTag","PaymentCause":null,"PaymentText":null,"PayerInfoRequest":{"PayerName":null,"PayerAddress":null,"PayerMobile":null,"PayerEMail":null},"eCommerceInfoRequest":{"UrlOK":"","UrlKO":"","UrlNotification":"","PaymentUrlExpiryUTC":null},"ReceiptsType":"Text","SalesDocumentIdentification":null,"TagReadingConfiguration":{"TagType":"Card","ReadingSensor":"CardReader","PANEncoding":"Text","PANByteOrder":"LSB"},"TagInfoRequest":{"PAN":null,"PIN":0,"UTCTagExpiry":null,"TokenizedTag":null},"Currency":"Euro","FlashMode":"AlwaysOff","Language":"Spanish","OperatorId":"CPSConnectorTester","ResponseTimeOut":null,"CPSAPIVersion":"CPS_API_Version_1"}

11.2. Respuesta

{"SalesLines":[],"AppliedTotalDiscount":0.0,"OperationsGroup":"","TransactionId":"12310*#*075002180301124542313207","AmountInfo":{"Currency":"Euro","CoinAmount":0.0,"BillAmount":0.0,"TagAmount":10.0,"EuroRate":1.0},"PaymentKind":"Unknow","ProcessedPaymentMethod":"Unknow","InvoiceableByPaymentProcessingEntity":false,"PayerInfoResponse":{"PayerName":null,"PayerEmail":null,"PayerMobilePhone":null,"PayerHomePhone":null,"PayerWorkPhone":null,"PayerTown":null,"PayerCountry":null,"PayerAddressLine1":null,"PayerAddressLine2":null,"PayerAddressLine3":null,"PayerPostCode":null,"PayerProvince":null},"eCommerceInfoResponse":{"IntegrationMode":"RedirectionToUrl","PaymentUrl":null,"PaymentHtml":null,"BrowserIP":null,"BrowserJavaEnabled":null,"BrowserJavaScriptEnabled":null,"BrowserIETF_BCP47Language":null,"BrowserDescription":null,"BrowserTimeZone":null},"CPSOperationNumber":"202303783559","AuthenticationInfo":{"AuthenticationMethod":"Unknow","AuthenticationMethodProtocolVersion":null,"NeedsSignature":false,"SignatureData":null,"SignatureType":null},"ReceiptsInfo":{"ReceiptsType":"Text","ClientReceipt":" EJEMPLAR PARA EL CLIENTE\n\n SIMULADOR !\n\n OPERACION CONTACTLESS\n\nCOMERCIO : nnnnnnnn\nTERMINAL : nn\n\n**********NNNN\nTitular\nCad: YYMM\n\n VENTA\n\nAUT.457368 CRED Pedido:12310\nFECHA yyyy-mm-dd hh:mi:ss\n Etiqueta APP\nAPLICACION: A00000000XX010\nN.TRANS: 004772\nRESP: 00\nTVR: 0080008000\n\n 10.00 EUR\n\n","MerchantReceipt":""},"TagInfoResponse":{"PANEncoding":"Text","TagKind":"Unknown","HashedPAN":null,"ReadingMethod":"Contactless","IsContactless":true,"IsEMV":null,"IsMobile":null,"IsExpired":null,"PIN":0,"CardTracks":null,"NIF":null,"TagCountry":null,"UTCTagExpiry":null,"PAN":"**********NNNN","TokenizedTag":null},"ResponseResultInfo":{"ResultInfo":"OPERACION ACEPTADA","Success":true,"ResultCode":0}}