



# Referencia API SOAP Webpay Transbank S.A.

*DOCUMENTO DE ESPECIFICACIONES TRANSACCIÓN NORMAL  
(v 1.6)*

## Contenido

1	Control de cambios.....	2
2	Prefacio .....	2
2.1	Acerca de esta guía .....	2
2.2	Audiencia .....	2
2.3	Feedback para esta documentación.....	3
3	Transacción de Autorización Normal.....	4
3.1	Descripción de la Transacción de Autorización Normal .....	4
3.2	Secuencia de pago en una transacción normal .....	5
3.3	Descripción de métodos del Servicio Web de Transacción de Autorización Normal.....	10
3.3.1	Operación initTransaction .....	10
3.3.2	Operación getTransactionResult.....	11
3.3.3	Operación acknowledgeTrasaction.....	14
4	Ejemplos de integración con API SOAP Webpay – Transacción Normal .....	16
4.1	Ejemplo Java .....	17
4.2	Ejemplos PHP .....	21

# 1 Control de cambios

Fecha	Version	Descripción del cambio
12-12-12	1.0	Liberación inicial de documento general de API de integración con WS Transacción normal.  Futuros Release: <ul style="list-style-type: none"><li>Ejemplos de integración Microsoft .Net</li></ul>
03-01-13	1.1	Se adicionan ejemplos en PHP
15-01-13	1.2	Modificación de procedimiento de integración PHP
28-01-13	1.3	Se agrega ejemplo .Net
27-05-14	1.4	Flujo alternativo, modifica ejemplo php, se elimina ejemplo .Net
12-06-14	1.5	Mejoras según observaciones
06-08-14	1.6	Actualización en especificación de paymentTypeCode

## 2 Prefacio

### 2.1 Acerca de esta guía

Esta guía describe los aspectos técnicos que deben ser considerados en la integración con Webpay utilizando API SOAP. Describe el servicio Web para Transacción de Autorización Normal, sus operaciones y cómo estas deben ser utilizadas en un flujo de pago.

### 2.2 Audiencia

Esta guía está dirigida a implementadores que realizan la integración de Webpay en comercios utilizando la API SOAP para soportar en estos el pago con tarjetas bancarias.

Se recomienda que quién realice la integración posea conocimiento técnico de al menos en los siguientes temas:

- Servicios Web
- WS-Security
- Firma digital, generación y validación.

## 2.3 Feedback para esta documentación

Ayúdanos a mejorar esta información enviándonos comentarios a [soporte@transbank.cl](mailto:soporte@transbank.cl)

## 3 Transacción de Autorización Normal


### 3.1 Descripción de la Transacción de Autorización Normal

Una transacción de autorización normal (o transacción normal), corresponde a una solicitud de autorización financiera de un pago con tarjetas de crédito o débito, en donde quién realiza el pago ingresa al sitio del comercio, selecciona productos o servicio, y el ingreso asociado a los datos de la tarjeta de crédito o débito lo realiza en forma segura en Webpay.

El flujo de páginas para la transacción es el siguiente:



Resumen de los métodos del servicio Web de Transacción Normal

Método	Descripción general
<b><i>initTransaction</i></b>	<p>Permite inicializar una transacción en Webpay. Como respuesta a la invocación se genera un token que representa en forma única una transacción.</p> <p><b>Es importante considerar que una vez invocado este método, el token que es entregado tiene un periodo reducido de vida de 5 minutos, posterior a esto el token es caducado y no podrá ser utilizado en un pago.</b></p>
<b><i>getTransactionResult</i></b>	<p>Permite obtener el resultado de la transacción una vez Webpay ha resuelto su autorización financiera.</p>
<b><i>acknowledgeTrasaction</i></b>	<p>Indica a Webpay que se ha recibido conforme el resultado de la transacción.</p> <p> <b>El método <code>acknowledgeTrasaction</code> debe ser invocado siempre, independientemente del resultado entregado por el método <code>getTransactionResult</code>. Si la invocación no se realiza en un período de 30 segundos, Webpay reversará la transacción, asumiendo que el comercio no pudo informarse de su resultado, evitando así el cobro al tarjetahabiente.</b></p>

## 3.2 Secuencia de pago en una transacción normal

El siguiente diagrama ilustra la secuencia de pago y cómo participan los distintos actores en una transacción normal.

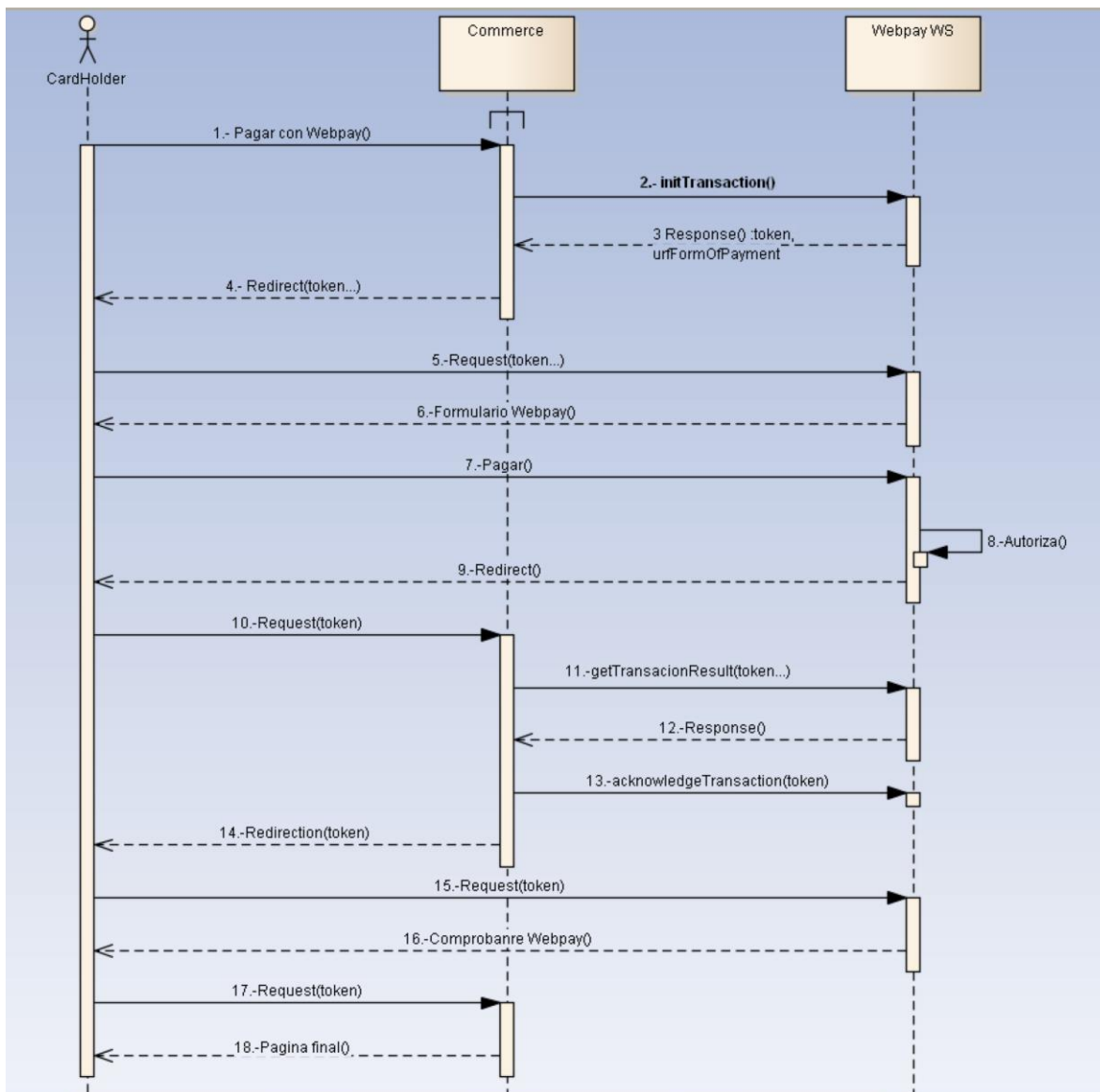


Diagrama de secuencia de Transacción Normal

### Descripción de la secuencia:

1. Una vez seleccionado los bienes o servicios, tarjetahabiente decide pagar a través de Webpay.
2. El comercio inicia una transacción en Webpay, invocando el método **initTransaction(...)**.
3. Webpay procesa el requerimiento y entrega como resultado de la operación el token de la transacción y URL de redireccionamiento a la cual se deberá redirigir al tarjetahabiente.
4. Comercio redirecciona al tarjetahabiente hacia Webpay, con el token de la transacción a la URL indicada en punto 3. La redirección se realiza enviando por método POST el token en variable **token\_ws**.
5. El navegador Web del tarjetahabiente realiza una petición HTTPS a Webpay, en base al redireccionamiento generado por el comercio en el punto 4.
6. Webpay responde al requerimiento desplegando el formulario de pago de Webpay. Desde este punto la comunicación es entre Webpay y el tarjetahabiente, sin interferir el comercio. El formulario de pago de Webpay despliega, entre otras cosas, el monto de la transacción, información del comercio como nombre y logotipo, las opciones de pago a través de crédito o débito.
7. Tarjetahabiente ingresa los datos de la tarjeta, hace clic en pagar en formulario Webpay.
8. Webpay procesa la solicitud de autorización (primero autenticación bancaria y luego la autorización de la transacción).
9. Una vez resuelta la autorización, **Webpay retorna el control al comercio**, realizando un redireccionamiento HTTP/HTTPS hacia la página de transición del comercio, en donde se envía por método POST el token de la transacción en la variable **token\_ws**.
10. El navegador Web del tarjetahabiente realiza una petición HTTP/HTTPS al sitio del comercio, en base a la redirección generada por Webpay en el punto 9.
11. El sitio del comercio recibe la variable **token\_ws** e invoca el segundo método Web, **getTransactionResult ()** (mientras se despliega la página de transición<sup>1</sup>), para obtener el resultado de la autorización. Se recomienda que el resultado de la autorización sea persistida en los sistemas del comercio, ya que este método se puede invocar una única vez por transacción.
12. Webpay responde el resultado de la invocación del método **getTransactionResult()**.

---

<sup>1</sup> El detalle de la página de transición se encuentra descrito en Anexo A, del documento de descripción general de la API SOAP.

13. Para informar a Webpay que el resultado de la transacción se ha recibido sin problemas, el sistema del comercio consume el tercer método **acknowledgeTransaction()**.

**NOTA:** *De no ser consumido o demorar más de 30 segundos en su consumo, Webpay realizará la reversa de la transacción, asumiendo que existieron problemas de comunicación. En este caso el método retorna una Excepción indicando la situación.*

14. Una vez recibido el resultado de la transacción e informado a Webpay su correcta recepción, el sitio del comercio debe redirigir al tarjetahabiente nuevamente a Webpay, con la finalidad de desplegar el comprobante de pago. Es importante realizar este punto para que el tarjetahabiente entienda que el proceso de pago fue exitoso, y que involucrará un cargo a su tarjeta bancaria. **El redireccionamiento a Webpay se hace utilizando como destino la URL informada por el método getTransactionResult()** enviando por método POST el token de la transacción en la variable token\_ws.
15. Webpay recibe un requerimiento con la variable token\_ws
16. Webpay identifica la transacción y despliega el comprobante de pago al tarjetahabiente.
17. Una vez visualizado el comprobante de pago por un periodo acotado de tiempo, el tarjetahabiente es redirigido de vuelta al sitio del comercio, por medio de redireccionamiento con el token en la variable token\_ws enviada por método POST hacia la pagina final informada por el comercio en el método initTransaction.
18. Sitio del comercio despliega página final de pago<sup>2</sup>.

---

<sup>2</sup> El detalle de la página de transición se encuentra descrito en Anexo A, 6.1.2 del documento de descripción general de la API SOAP.



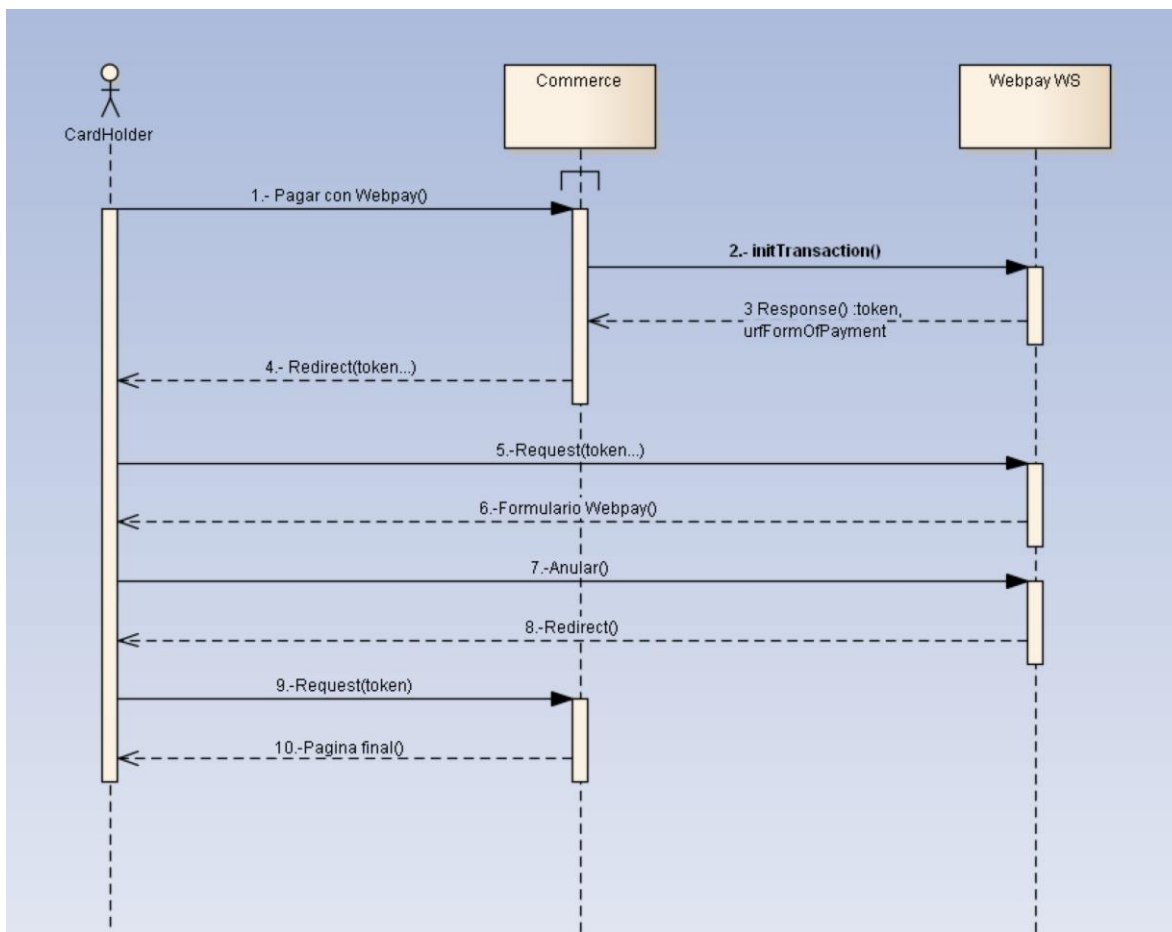


Diagrama de secuencia de Transacción Normal anulada

#### Descripción de secuencia alternativa, anular:

1. Pasos de 1 a 6 son identicos a la secuencia normal.
7. Tarjetahabiente hace clic en “anular”, en formulario Webpay.
8. Webpay retorna el control al comercio, realizando un redireccionamiento HTTP/HTTPS hacia la página de final del comercio, en donde se envía por método POST el token de la transacción en la variable **TBK\_TOKEN**.
9. El comercio con la variable **TBK\_TOKEN** debe, invocar el segundo método Web, **getTransactionResult ()** (mientras se despliega la página de transición<sup>3</sup>), para obtener el resultado de la autorización. En este caso debe obtener una excepcion, pues el pago fue abortado.

<sup>3</sup> El detalle de la página de transición se encuentra descrito en Anexo A, 6.1.1 del documento de descripción general de la API SOAP.

10. El comercio debe informar al tarjeta habiente que su pago no se completo, según anexo glosa transacción no autorizada

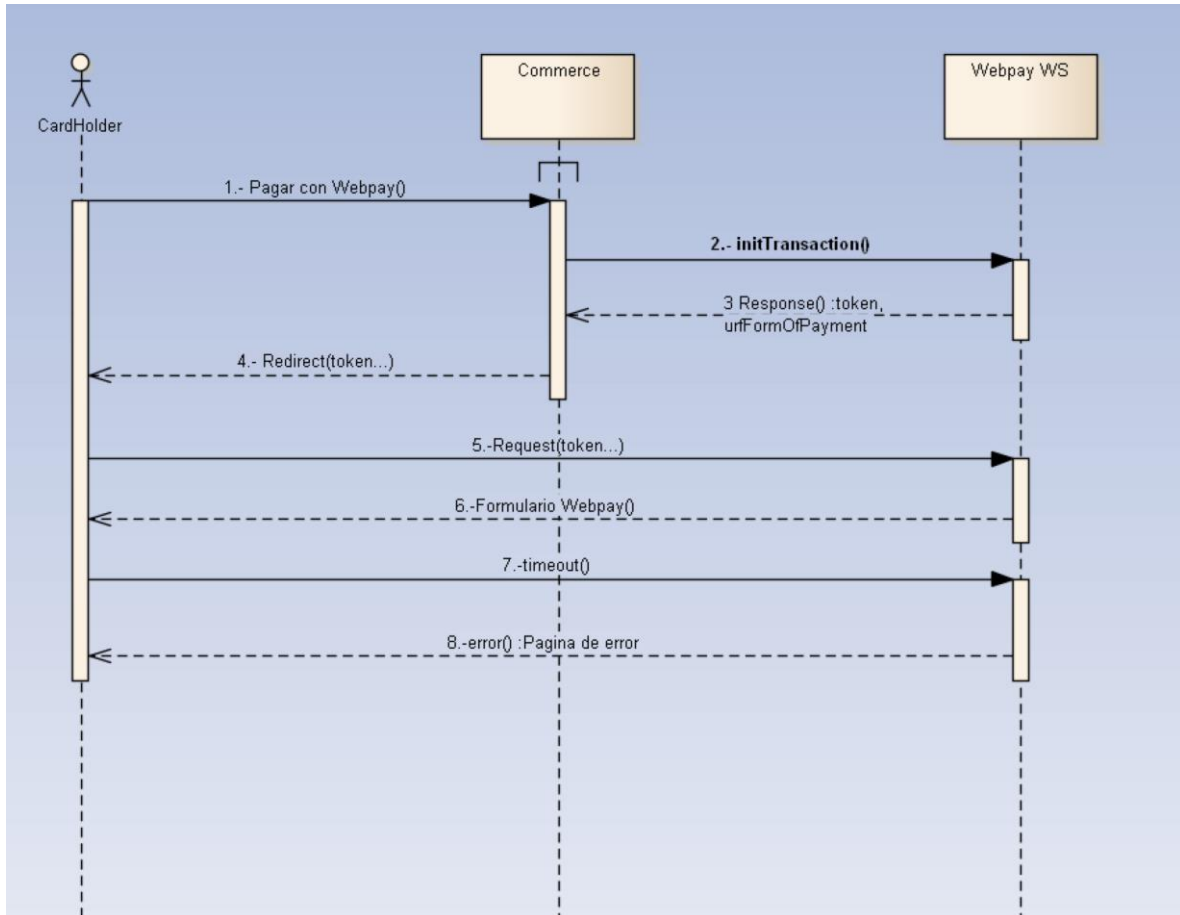


Diagrama de secuencia de Transacción Normal

**Descripción de secuencia alternativa, timeout:**

1. Pasos de 1 a 6 son identicos a la secuencia normal.
7. Tarjetahabiente esta en formulario Webpay, pero no presiona pagar durante 10 minutos. Esto causa un timeout en dicho formulario.
8. Webpay genera un error de timeout, se presenta una pantalla indicando que ocurrio un error. No se regresa automaticamente al comercio.

### 3.3 Descripción de métodos del Servicio Web de Transacción de Autorización Normal

A continuación se describen cada uno de las operaciones que deben ser utilizadas en una Transacción Normal.

#### 3.3.1 Operación `initTransaction`

Método que permite iniciar una transacción de pago Webpay.

**Parámetro de entrada:** `type initTransaction`

Nombre	Descripción
<b>WSTransactionType</b>	<code>xs:wsTransactionType</code>  Indica el tipo de transacción, su valor debe ser siempre <code>TR_NORMAL_WS</code>
<b>sessionId</b>	<code>xs:string</code>  (Opcional) Identificador de sesión, uso interno de comercio, este valor es devuelto al final de la transacción.  Largo máximo: 61
<b>returnURL</b>	<code>xs:anyURI</code>  (Obligatorio) URL del comercio, a la cual Webpay redireccionará posterior al proceso de autorización. Largo máximo: 256
<b>finalURL</b>	<code>xs:anyURI</code>  (Obligatorio) URL del comercio a la cual Webpay redireccionará posterior al voucher de éxito de Webpay. Largo máximo: 256
<b>transactionDetails</b>	<code>tns:wsTransactionDetail</code>  (Obligatorio) Lista de objetos del tipo <code>wsTransactionDetail</code> , el cual contiene datos de la transacción. Máxima cantidad de repeticiones es de 1 para este tipo de transacción. <code>wsTransactionDetail</code> está descrito a continuación.

## *wsTRANSACTIONDETAIL*

*Descripción: Tipo de dato contiene detalles de la transacción*

Campo	Descripción
<b>amount</b>	xs:decimal  Monto de la transacción. Máximo 2 decimales para USD.  Largo máximo: 10
<b>buyOrder</b>	xs:string  Orden de compra de la tienda. <sup>4</sup>  Largo máximo: 26
<b>commerceCode</b>	xs:string  Código comercio de la tienda entregado por Transbank.  Largo: 12
<b>sharesAmount</b>	  Parametro no usado.
<b>sharesNumber</b>	  Parametro no usado

**Parámetros de salida: type wsInitTransactionOutput**

Campo	Descripción
<b>token</b>	xs:string  Token de la transacción.  Largo: 64
<b>url</b>	xs:string  URL de formulario de pago Webpay Largo máximo: 256

### 3.3.2 Operación getTransactionResult

Método que permite obtener el resultado de la transacción y los datos de la misma.

---

<sup>4</sup> Debe cumplir con el formato de caracteres permitidos

**Parámetros de entrada: *getTransactionResult***

Campo	Descripción
<b>tokenInput</b>	<b>xs:string</b>  Token de la transacción.  Largo: 64

**Parámetros de salida: *transactionResultOutput***

Campo	Descripción
<b>buyOrder</b>	<b>xs:string</b>  Orden de compra de la tienda.  Largo máximo: 26
<b>sessionId</b>	<b>xs:string</b>  Identificador de sesión, uso interno de comercio, este valor es devuelto al final de la transacción. Largo máximo: 61
<b>cardDetails</b>	<b>Tns:carddetails</b>  Objeto que representa los datos de la tarjeta de crédito del tarjetahabiente. Descrito mas adelante.
<b>accountingDate</b>	<b>xs:string</b>  Fecha de la autorización.  Largo: 4, formato MMDD
<b>transactionDate</b>	<b>xs:string</b>  Fecha y hora de la autorización.  Largo: 6 formato: MMDDHHmm
<b>VCI</b>	<b>xs:string</b>  Resultado de la autenticación para comercios Webpay Plus y/o 3D Secure, los valores posibles son los siguientes: <ul style="list-style-type: none"><li>• TSY: Autenticación exitosa</li><li>• TSN: autenticación fallida.</li><li>• TO: Tiempo máximo excedido para autenticación.</li><li>• ABO: Autenticación abortada por tarjetahabiente.</li><li>• U3: Error interno en la autenticación.</li><li>• Puede ser vacío si la transacción no se autentico.</li></ul> Largo máximo: 3
<b>urlRedirection</b>	<b>xs:string</b>  URL de redirección para visualización de voucher.

Campo	Descripción
	Largo máximo: 256
<b>detailsOutput</b>	tns:transactionDetails  Objeto que contiene el detalle de la transacción financiera. Descrito más adelante

### **CARDDETAIL (DETALLES TARJETA DE CRÉDITO)**

Descripción: Tipo de dato contiene detalles de la tarjeta de crédito.

Campo	Descripción
<b>cardNumber</b>	xs:string  4 ultimos números de la tarjeta de crédito del tarjeta habiente. Solo para comercios autorizados por Transbank se envía el numero completo.  Largo máximo: 16
<b>cardExpirationDate</b>	xs:string  (Opcional) Fecha de expiración de la tarjeta de crédito del tarjetahabiente. Formato YYMM Solo para comercios autorizados por Transbank.  Largo máximo: 4

### **DETAILSOUTPUT**

Campo	Descripción
<b>authorizationCode</b>	xs:string  Código de autorización de la transacción  Largo máximo: 6
<b>paymentTypeCode</b>	xs:string  Tipo de pago de la transacción. VD = Venta Debito VN = Venta Normal VC = Venta en cuotas SI = 3 cuotas sin interés S2=2 cuotas sin interés NC = N Cuotas sin interés
<b>responseCode</b>	xs:string Código de respuesta de la autorización. Valores posibles:

Campo	Descripción
	<b>0</b> Transacción aprobada. <b>-1</b> Rechazo de transacción. <b>-2</b> Transacción debe reintentarse. <b>-3</b> Error en transacción. <b>-4</b> Rechazo de transacción. <b>-5</b> Rechazo por error de tasa. <b>-6</b> Excede cupo máximo mensual. <b>-7</b> Excede límite diario por transacción. <b>-8</b> Rubro no autorizado.
<b>Amount</b>	<code>xs:decimal</code> Monto de la transacción. Largo máximo: 10
<b>sharesAmount</b>	<code>xs:decimal</code> Valor de la cuota. Largo máximo: 9
<b>sharesNumber</b>	<code>xs:int</code> Cantidad de cuotas Largo máximo: 2
<b>commerceCode</b>	<code>xs:string</code> Código comercio de la tienda Largo: 12
<b>buyOrder</b>	<code>xs:string</code> Orden de compra de la tienda. Largo máximo: 26

### 3.3.3 Operación **acknowledgeTrasaction**

Método que permite informar a Webpay la correcta recepción del resultado de la transacción.

***Parámetros de entrada: getTransactionResult***

Campo	Descripción
<b>tokenInput</b>	<code>xs:string</code>  Token de la transacción.  Largo: 64



## 4 Ejemplos de integración con API SOAP Webpay – Transacción Normal

La presente sección, entrega ejemplos de uso de la API SOAP para transacción normal en los lenguajes Java, PHP y .net. Tienen por objetivo exponer una forma factible de integración con API SOAP Webpay para resolver los siguientes puntos asociados a la integración:

- Generación de cliente o herramienta para consumir los servicios Web, lo cual permite abstraerse de la complejidad de mensajería SOAP asociada a los Webservice y hacer uso de las operaciones del servicio.
- Firma del mensaje y validación de firma en la respuesta, existen frameworks y herramientas asociadas a cada lenguaje de programación que implementan el estándar WS Security, lo que se requiere es utilizar una de estas, configurarla y que realice el proceso de firma digital del mensaje.

Estos ejemplos son sólo una guía / ayuda de integración, y no abarcan el proceso completo de llamada a los WS. En ningún caso son una obligatoriedad su implementación, y el comercio es libre de realizar la integración como más le acomode.

## 4.1 Ejemplo Java

Este ejemplo hará uso de los siguientes frameworks para consumir los servicios Web de Webpay utilizando WS Security:

- **Apache CXF**, es un framework open source que ayuda a construir y consumir servicios Web en Java. En este ejemplo se utilizará para:
  - Generar el cliente del Webservice o STUBS.
  - Consumir los servicios Web
- **Apache WSS4J**, proporciona la implementación del estándar WS Security, nos permitirá:
  - Firmar los mensajes SOAP antes de enviarlo a Webpay.
  - Validar la firma de la respuesta del servicio Web de Webpay.
- **Spring framework 3.0**, permite que CXF y WSS4J trabajen en conjunto, también se utiliza para configurar WS Security en la firma del mensaje SOAP.

Pasos a seguir:

### 1. Generación de cliente del Webservice.

Para generar el código Java que implementará el cliente SOAP se utilizará `wsdl2java` de CXF, el cual toma el WSDL del servicio y genera todas las clases necesarias para invocar el servicio Web. Más información en <http://cxf.apache.org/docs/wsdl-to-java.html>

```
wsdl2java -autoNameResolution <URL del wsdl>
```

## 2. Configuración de WS Security

Para configurar WS Security en CXF se deben habilitar y configurar los interceptores que realicen el trabajo de firmado del mensaje. La configuración de los interceptores se puede realizar a través de la API de servicios Web o a través del XML de configuración de Spring, en este caso se realizará a través de Spring en el archivo applicationContext.xml de la aplicación.

Se deben habilitar y configurar 2 interceptores, uno para realizar la firma de los mensajes enviados al invocar una operación del servicio Web de Webpay y otro para validar la firma de la respuesta del servicio Web.

### Interceptor de salida

```
<bean class="org.apache.cxf.ws.security.wss4j.WSS4JOutInterceptor"
      id="signOutRequestInterceptor">
  <constructor-arg>
    <map>
      <entry key="signaturePropFile" value="signatureOut.properties"/>
      <entry key="user" value="${alias.client}"/>
      <entry key="action" value="Signature"/>
      <entry key="passwordCallbackClass"
            value="com.transbank.webpay.wsse.ClientCallBack"/>
      <entry key="signatureParts"
            value="{Element}{http://schemas.xmlsoap.org/soap/envelope/}Body"/>
    </map>
  </constructor-arg>
</bean>
```

### Interceptor de entrada

```
<bean class="org.apache.cxf.ws.security.wss4j.WSS4JInInterceptor"
      id="signInRequestInterceptor">
  <constructor-arg>
    <map>
      <entry key="action" value="Signature" />
      <entry key="signaturePropFile" value="signatureIn.properties"/>
      <entry key="passwordCallbackClass"
            value="com.transbank.webpay.wsse.ServerCallBack"/>
      <entry key="signatureParts"
            value="{Element}{http://schemas.xmlsoap.org/soap/envelope/}Body"/>
    </map>
  </constructor-arg>
</bean>
```

Las clases ClientCallBack y ServerCallBack implementan la interfaz javax.security.auth.callback.CallbackHandler, su implementación permite al framework de seguridad recuperar la contraseña para acceder al almacén de llaves de la aplicación (Java Key Store) que almacena los certificados digitales.

### 3. Llamada a operaciones del Webservice

### 4. Llamada a operaciones del Webservice

#### Operación initTransaction

```
private WsWebpayService service;
private List<WsTransactionDetail> transactionDetails= new ArrayList<
WsTransactionDetail>
private WsTransactionDetail transactionDetail = new WsTransactionDetail();
private WsInitTransactionInput transactionInput = new WsInitTransactionInput();
private WsInitTransactionOutput transactionOutput = new WsInitTransactionOutput();

transactionInput.setSessionId("12312423");
transactionInput.setReturnURL("http://www.midominio.com/recibetoken.do");
transactionInput.setFinalURL("http://www.midominio.com/resultado.do");

transactionDetail.setAmount(new BigDecimal(1000));
transactionDetail.setCommerceCode("59702512345");
transactionDetail.setBuyOrder("123456789");
transactionDetails.add(transactionDetail);

transactionInput.setWsTransactionType(WsTransactionType.TR_NORMAL_WS);
transactionInput.getTransactionDetails().add(transactionDetails);

transactionOutput = service.initTransaction(transactionInput);

/*Token y URL de redirección*/
transactionOutput.getUrl();
transactionOutput.getToken();
```

#### Operación getTransactionResult

```
/*Se asume que se obtuvo el token, el cual fue enviado por Webpay a La URL
notificada en el parámetro returnURL al invocar al método initTransaction, el
parámetro enviado por post se llama token_ws*/

private WsWebpayService service;
private TransactionResultOutput transactionResultOutput;
transactionResultOutput = service.getTransactionResult(token);

/* transactionResultOutput contendrá los parámetros de resultado de la
transacción*/
```

### **Operación acknowledgeTransaction()**

*/\*Se asume que se obtuvo el token, este método tiene por objetivo indicarle a Webpay que se obtuvo el resultado de la transacción correctamente. Este método es void\*/*

```
private WWebpayService service;  
service.acknowledgeTransaction(token);
```

URL:

<http://cxf.apache.org/docs/ws-security.html>

<http://ws.apache.org/wss4j/>

<http://cxf.apache.org/docs/wsdl-to-java.html>

## 4.2 Ejemplos PHP

El siguiente ejemplo está basado en PHP versión 5, sobre el cual se utilizaron las siguientes bibliotecas de software para realizar la invocación de los servicios web de Webpay bajo el estándar WSS:

- **Biblioteca de seguridad:** archivo compuesto de tres clases que integran librerías nativas PHP de validación y verificación. Estas clases nos permitirán generar la seguridad suficiente a través de métodos de encriptación y desencriptación.
- **WSSE-PHP:** integra las librerías de seguridad XML y genera un documento XML-SOAP seguro. Depende de las librerías de seguridad XML.
- **SOAP-VALIDATION:** clase encargada de la validación de mensajes SOAP seguros de respuesta. Verifica la autenticidad e integridad del mensaje. Depende de WSSE-PHP.

Los fuentes pueden ser descargados desde <https://github.com/OrangePeople/php-wss-validation>

Pasos a seguir:

### 1. Generación de cliente del Servicio Web:

Antes de la integración se debe tener instalado y funcionando un servidor HTTP Apache y configurar el directorio para generar una salida a través del navegador web. Si se quiere optar por una alternativa más sencilla, Apache provee un directorio por omisión, que varía según el sistema operativo. Generalmente en plataformas Linux es “/var/www” y en Windows “C:\<directorio hacia Apache>\htdocs”.

A continuación, para generar las clases necesarias que conectan a los servicios Web, se puede utilizar la herramienta Easy WSDL2PHP. La documentación necesaria e información de descarga se encuentra en <http://sourceforge.net/projects/easywsdl2php/>.

Una vez descargados los fuentes, se deben copiar en el directorio de apache que posee la salida por navegador.

Se hace la llamada por navegador del archivo **wsdl2php.php** y se obtiene la siguiente pantalla:

## Easy WSDL2PHP Generator

ej. <http://soap.amazon.com/schemas2/AmazonWebServices.wsdl>

Url:

Class Name:

Se escribe la URL del **archivo wsdl** al se quiere conectar, un **nombre de clase** y luego se presiona el botón **Generate Code**. Luego de esto se muestra una pantalla como la siguiente:

### Easy WSDL2PHP Generator

ej. <http://soap.amazon.com/schemas2/AmazonWebServices.wsdl>

Url:

Class Name:

```
<?php
class getTransactionResult{
var $tokenInput;//string
}
class getTransactionResultResponse{
var $return;//transactionResultOutput
}
class transactionResultOutput{
var $accountingDate;//string
var $buyOrder;//string
var $cardDetail;//cardDetail
var $detailOutput;//wsTransactionDetailOutput
var $sessionId;//string
var $transactionDate;//string
var $urlRedirection;//string
var $VCI;//string
}
class cardDetail{
var $cardNumber;//string
var $cardExpirationDate;//string
}
class wsTransactionDetailOutput{
var $authorizationCode;//string
var $paymentTypeCode;//string
var $responseCode;//int
}
class wsTransactionDetail{
var $sharesAmount;//decimal
var $sharesNumber;//int
var $amount;//decimal
var $commerceCode;//string
var $buyOrder;//string
}
class acknowledgeTransaction{
var $tokenInput;//string
}
```

Una vez que se obtiene el resultado mostrado en la imagen se copia el código PHP generado y se guarda en un archivo, el cual representará el stub del servicio Web. Una vez realizado este proceso ya se tienen las clases necesarias para poder integrarse con los servicios web de Webpay.

**2. Crear una clase que extienda de SoapClient** (SoapClient es la clase nativa que provee PHP para utilización de servicios Web)(En el ejemplo se denominará MySoap)

```
<?php
//Notar que se incluyen dos archivos que se proveen en La Librería de encriptación
require_once('xmlseclibs.php');
require_once('soap-wsse.php');
class MySoap extends SoapClient {
private $useSSL=false;
    function __construct($wsdl,$options){
        $locationparts = parse_url($wsdl);
        $this->useSSL = $locationparts['scheme']=="https" ? true:false;
        return parent::__construct($wsdl,$options);
    }

    function __doRequest($request, $location, $saction, $version) {
        if ($this->useSSL){
            $locationparts = parse_url($location);
            $location = 'https://';
            if(isset($locationparts['host'])) $location .=
            $locationparts['host'];
            if(isset($locationparts['port'])) $location .=
            ':'.$locationparts['port'];
            if(isset($locationparts['path'])) $location .=
            $locationparts['path'];
            if(isset($locationparts['query'])) $location .=
            '?'.$locationparts['query'];
        }

        $doc = new DOMDocument('1.0');
        $doc->LoadXML($request);
        $objWSSE = new WSSESoap($doc);
        $objKey = new XMLSecurityKey(XMLSecurityKey::RSA_SHA1,array('type'
=> 'private'));
        $objKey->LoadKey('privada_amb_orange.key', TRUE);
        $options = array("insertBefore" => TRUE);
        $objWSSE->signSoapDoc($objKey, $options);
        $objWSSE->addIssuerSerial('certificado_amb_orange.crt');
        $objKey = new XMLSecurityKey(XMLSecurityKey::AES256_CBC);
        $objKey->generateSessionKey();
        $retVal = parent::__doRequest($objWSSE->saveXML(), $location,
        $saction, $version);
        $doc = new DOMDocument();
        $doc->LoadXML($retVal);
        return $doc->saveXML();
    }
}
?>
```

Las constantes **PRIVATE\_KEY** Y **CERT\_FILE** son las rutas de la llave privada y certificado del comercio, respectivamente.



### 3. Incluir la clase generada en el paso anterior en el archivo principal de los servicios.

Se debe incluir con la sentencia `require_once` la clase generada en el paso anterior.

Ejemplo:

```
require_once("mysoap.php");
```

### 4. Editar el archivo stub creado en el paso 1

Se debe editar el método `__construct` del stub tal como se muestra en el ejemplo:

**Donde dice:**

```
$this->soapClient = new SoapClient($url, array("classmap" => self::$classmap,
"trace" => true, "exceptions" => true));
```

**Debe quedar: (utilizando el nombre de clase del paso 2)**

```
$this->soapClient = new MySoap($url, array("classmap" => self::$classmap, "trace"
=> true, "exceptions" => true));
```

### 5. Invocación de operaciones del servicio web de Webpay.

Para todos los ejemplos se debe hacer referencia a los archivos de la librería descargada

```
require_once('soap-wsse.php');
require_once('soap-validation.php');
require_once('<archivo que contiene la clase stub creado en el paso 1>');
```

### Operación InitTransaction:

```
$wsInitTransactionInput = new wsInitTransactionInput();
$wsTransactionDetail = new wsTransactionDetail();

/*Variables de tipo string*/
$wsInitTransactionInput->wSTransactionType = $transactionType;
$wsInitTransactionInput->commerceId = $commerceId;
$wsInitTransactionInput->buyOrder = $buyOrder;
$wsInitTransactionInput->sessionId = $sessionId;
$wsInitTransactionInput->returnURL = $returnUrl;
$wsInitTransactionInput->finalURL = $finalUrl;

$wsTransactionDetail->commerceCode = $commerceCode;
$wsTransactionDetail->buyOrder = $buyOrder;
$wsTransactionDetail->amount = $amount;
$wsTransactionDetail->sharesNumber = $shareNumber;
$wsTransactionDetail->sharesAmount = $shareAmount;

$wsInitTransactionInput->transactionDetails = $wsTransactionDetail;

$webpayService = new WebpayService($url_wsdl);
$initTransactionResponse = $webpayService->initTransaction(
    array("wsInitTransactionInput" => $wsInitTransactionInput)
);

$xmlResponse = $webpayService->soapClient->__getLastResponse();

$soapValidation = new SoapValidation($xmlResponse, SERVER_CERT);
$validationResult = $soapValidation->getValidationResult();

/*Invocar sólo si $validationResult es TRUE*/
$wsInitTransactionOutput = $initTransactionResponse->return;
```

### Observaciones:

- La clase **WebpayService** contiene los servicios principales y es el nombre que se generó el stub en el paso 1.
- La constante **SERVER\_CERT** es la ruta del archivo del certificado cliente entregado por Transbank.
- **initTransaction** es un método de la clase WebpayService y representa la llamada al servicio initTransaction.
- La variable **\$xmlResponse** es un string del xml-soap que responde el servidor.
- La variable **\$validationResult** es el resultado de tipo boolean de la validación del mensaje de respuesta. Para un caso correcto el valor es **TRUE**, de lo contrario es **FALSE**.
- La variable **\$wsInitTransactionOutput** contiene los datos que entrega el servidor. Esta variable sólo debe invocarse después de un resultado correcto en el mensaje SOAP de respuesta.

#### Operación getTransactionResult:

```
$webpayService = new WebpayService($url_wsdl);  
$getTransactionResult = new getTransactionResult();  
$getTransactionResult->tokenInput = $_POST['token_ws'];  
$getTransactionResultResponse = $webpayService->getTransactionResult(  
    $getTransactionResult);  
$transactionResultOutput = $getTransactionResultResponse->return;
```

#### Operación acknowledgeTransaction:

```
$webpayService = new WebpayService($url_wsdl);  
$acknowledgeTransaction = new acknowledgeTransaction();  
$acknowledgeTransaction->tokenInput = $_POST['token_ws'];  
$acknowledgeTransactionResponse = $webpayService->acknowledgeTransaction(  
    $acknowledgeTransaction);  
$xmlResponse = $webpayService->soapClient->__getLastResponse();  
$soapValidation = new SoapValidation($xmlResponse, SERVER_CERT);  
$validationResult = $soapValidation->getValidationResult();
```

#### Observaciones:

- El valor de **\$\_POST['token\_ws']** contiene un string del token de la transacción que entrega Webpay.