Central Payment Portal Integration Guide

Publication Date: September 10, 2010

CBOSS 7332 Southern Blvd., Suite 3, Boardman, Ohio 44512, Telephone: (330) 726-0429 Fax: (330) 726-0499

Table of Contents

List of Figures

List of Tables

1 - Introduction

The Central Payment Portal (CPP) is a robust electronic payment processing gateway for online credit card and electronic check payments.

Integrating with the CPP gateway incorporates the functionality of a custom built client web site with the secure, scalable, and robust electronic payment processing engine that is built inside the CPP gateway.

Hosted and non-hosted solutions are available for payment processing. For hosted solutions all payment intake is performed in a Payment Card Industry Data Security Standard (PCI DSS) certified environment. Non-hosted solutions provide the ability to perform payment intake on a client site but use the processing engine hosted in a PCI DSS certified environment.

The CPP gateway also provides extensive reconciliation capabilities as well as the ability to perform real time updates of client systems as soon as a payment authorization has been successfully completed.

2 - Purpose

The purpose of this document is to provide an overview of the CPP gateway payment processing capabilities.

The document will describe the various hosting and integration methods available for processing payments with the gateway and include detailed example code for each of the integration methods. Each of the available payment information parameters for the CPP gateway is described in detail.

It will also provide details on how a client system may receive real time updates of successful authorizations.

3 - Scope

The scope of this document includes the overview of the CPP gateway payment processing capabilities, the hosting and integration methods available, the description of the payment information parameters available, and how to receive real time updates of successful authorizations.

This document includes example code on how to integrate with the CPP gateway and it is assumed that the reader is familiar with the various web protocols.

4 - Hosted vs. Non-Hosted Solutions

The CPP gateway provides client web sites with the ability to complete customer payment transactions directly on the client web site or to redirect customers to a customizable payment intake form hosted at CBOSS.

4.1 - Hosted Solution

The benefit of a hosted solution is the reduced development/integration time as well as reduced client liability because no customer payment information is entered on the client web site.

The look and feel of the hosted payment intake form and receipt is customizable so that they may match the look of the client web site. A customizable payment intake form is shown in the following figure.

Figure 1 - Example Payment Intake

There are two methods to integrate with the hosted solution. To use the first method, the client web site performs an HTTP post to a specified hosted URL specifying the payment information via form parameters.

To use the second method, the client web site launches into the CPP gateway by redirecting the customer to a specified hosted URL with the query string specifying the payment

information via a tokenized value. The tokenized payment information query string value is generated using a platform independent Windows Communication Foundation (WCF) service.

4.1.1 - Hosted Modes

The CPP gateway hosted solution has two payment processing modes of operation available: **Stay on Site Mode** and **Redirect Mode**. The mode chosen depends on the client web site business processes in place. Most client web sites choose to use the **Stay on Site Mode** where all payment processing and receipt display is performed by the CPP gateway hosted solution.

A customizable payment receipt form is shown in the following figure.

Figure 2 - Example Payment Receipt

Client web sites which require their customers to continue with other business logic after payment processing can chose to use the **Redirect Mode** which immediately returns transaction results to the client web site. When using **Redirect Mode** no payment receipt is displayed after successful authorization in the CPP gateway, the client web site is responsible for any payment receipt display.

4.1.2 - Stay on Site Mode

In this mode all payment processing and receipt displays are performed by the CPP gateway. To direct customers back to the client web site after the transaction is completed through the CPP gateway or to cancel the payment process, include the following variables in the integration payment information: **HomeURL** and **CancelURL**.

If the **HomeURL** is present in the integration payment information the **Continue** button is displayed on the receipt page which when pressed will redirect the browser to the specified URL.

If the **CancelURL** is present in the integration payment information the **Cancel** button is displayed on the payment intake form which when pressed will redirect the browser to the specified URL.

4.1.3 - Redirect Mode

In this mode all the payment processing is performed by the CPP gateway but the results are returned to the client web site. The client web site would host a publicly accessible **Successful Authorization** web page which would display a custom receipt and perform any additional processing required.

The system also has the ability to redirect upon unsuccessful authorization. This is not a requirement for this mode. If not setup the customer is simply provided with the ability to enter different payment information. The client web site would host a publicly accessible **Unsuccessful Authorization** web page which would display an error message and perform any additional processing required.

The **Successful Authorization** and **Unsuccessful Authorization** URLs need to be setup (administered by CBOSS) prior to performing any transactions in order to instruct where to redirect the customer after a transaction is complete.

Appropriate payment information parameters will be passed with each redirected page. Each parameter will be formatted according to URL specifications and collectively passed as a URL query string back to the appropriate URL in the client application.

4.2 - Non-Hosted Solutions

The benefit of a non-hosted solution is that the complete process remains on the client web site and allows for complete customization of the payment process.

To integrate with the non-hosted solution, the client web site simply communicates with a hosted web service specifying the complete payment information.

5 - Integration Process

The integration process involves the setup or an originator, developing the application to pass payment information, and finally testing the complete business process.

5.1 - Originator Setup

An originator will be setup for the client web site by CBOSS. The originator is an application identification number that is associated with a merchant account and uniquely identifies the client web site and determines how the payments will be processed.

The client will be provided by CBOSS with information to begin the integration. Typical integration information is shown below:

Table 1 - Example Integration Information

Name	Value
Staged Originator ID	1
Staged Payment Page	http://domain.cpp.dev.cboss.com/Payment.aspx
Staged Administration Portal	http://domain.cpp.dev.cboss.com/Admin
Staged Tokenization Service	http://domain.cpp.dev.cboss.com/Service/PublicTokenization.svc
Production Payment Page	https://domain.cpp.cboss.com/Payment.aspx
Production Administration Portal	https:// domain.cpp.cboss.com/Admin
Production Tokenization Service	http://domain.cpp.cboss.com/Service/PublicTokenization.svc
Administration Portal Username	JDoe
Administration Portal Password	****
Test ACH Routing Number	041001039
Test ACH Account Number	74123
Test Credit Card American Express	378282246310005
Test Credit Card American Express	371449635398431
Test Credit Card Amex Corporate	378734493671000
Test Credit Card Diners Club	30569309025904
Test Credit Card Diners Club	38520000023237
Test Credit Card Discover	6011111111111117
Test Credit Card Discover	6011000990139424
Test Credit Card JCB	3530111333300000
Test Credit Card JCB	3566002020360505
Test Credit Card MasterCard	5555555555554444
Test Credit Card MasterCard	5105105105105100
Test Credit Card Visa	4111111111111111
Test Credit Card Visa	4012888888881881

5.2 - Integration Development

This section describes in detail the integration process for performing the post and redirect methods for the hosted environment as well as the integration method for non-hosted environment.

5.2.1 - Hosted Integration Post Method

To use this integration method, the client web site performs an HTTP post to a specified hosted URL specifying the payment information via form parameters.

At a minimum, the **OriginatorID** and **Amount** payment information parameters must be specified as form parameters. This is demonstrated in the following example.

```
<html>
<head>
    <title>Example Hosted Basic Post - Central Payment Portal | cboss, Inc.</title>
</head>
<body>
    <form method="post" action="http://experiment.cpp.cboss.com/Payment.aspx">

    <!-- Add the REQUIRED input parameters. -->
    <input type="hidden" name="OriginatorID" value="1" />
    <input type="text" name="Amount" value="44.76" />

    <input type="submit" name="Pay" value="Pay" />
    </form>
</body>
</html>
```

Additional optional payment information parameters such as the **Comment1**, **HomeUrl**, and **CancelUrl** may be specified as demonstrated in the following more detailed example.

```
<html>
<head>
    <title>Example Hosted Complex Post - Central Payment Portal | cboss, Inc.</title>
</head>
<body>
    <form method="post" action="http://experiment.cpp.cboss.com/Payment.aspx">

    <!-- Add the REQUIRED input parameters. -->
    <input type="hidden" name="OriginatorID" value="42" />
    <input type="hidden" name="Amount" value="44.76" />

    <!-- Add the OPTIONAL input parameters. -->
    <input type="hidden" name="AmountLine1Name" value="Subtotal" />
    <input type="hidden" name="AmountLine1Value" value="41.5" />
    <input type="hidden" name="AmountLine2Name" value="Convenience Fee" />
    <input type="hidden" name="AmountLine2Value" value="3.26" />
    <input type="hidden" name="CancelURL" value="http://Domain/Cancel.aspx" />
    <input type="hidden" name="HomeURL" value="http://Domain/Home.aspx?Data=ABC&ID=123" />
    <input type="hidden" name="Number" value="123458" />
    <input type="hidden" name="Comment1" value="ITEM:=XYZ,TUV" />
    <input type="hidden" name="Comment2" value="USER:=JSMITH" />

    <!-- Add the OPTIONAL basket input parameters. -->
    <input type="hidden" name="ItemCount" value="2" />
    <input type="hidden" name="ItemDescription1" value="XYZ" />
    <input type="hidden" name="ItemPrice1" value="25.64" />
    <input type="hidden" name="ItemQuantity1" value="1" />
    <input type="hidden" name="ItemDescription2" value="TUV" />
    <input type="hidden" name="ItemPrice2" value="7.93" />
    <input type="hidden" name="ItemQuantity2" value="2" />

    <!-- Add the OPTIONAL field populate input paramters. -->
    <input type="hidden" name="PaymentType" value="Credit" />
    <input type="hidden" name="BillingAddressLine1" value="11 Main Street" />
    <input type="hidden" name="BillingAddressLine2" value="#1" />
    <input type="hidden" name="BillingCity" value="Anytown" />
    <input type="hidden" name="BillingCountry" value="USA" />
    <input type="hidden" name="BillingEmail" value="john@doe.com" />
```

```
    <input type="hidden" name="BillingFirstName" value="John" />
    <input type="hidden" name="BillingLastName" value="Doe" />
    <input type="hidden" name="BillingMiddleName" value="James" />
    <input type="hidden" name="BillingPhoneAreaCode" value="111" />
    <input type="hidden" name="BillingPhoneNumber" value="2223333" />
    <input type="hidden" name="BillingState" value="OH" />
    <input type="hidden" name="BillingZip" value="12345" />

    <input type="submit" name="Pay" value="Pay" />
    </form>
</body>
</html>
```

5.2.2 - Hosted Integration Redirect Method

To use this integration method the client web site launches into the CPP gateway by redirecting the customer to a specified hosted URL with the query string specifying the payment information via a tokenized value.

The following are the three steps to this integration process:

1. Construct payment information string of name/value pairs.
2. Pass the payment information string to a WCF service which will return a token.
3. Construct a payment URL with the token specified in the query string of the URL.

Each payment item is made up of a name/value pair. The name/value pairs must be separated by an equals sign (=). The series of name/value pairs must be separated by an ampersand (&). An example of a payment information string would be the following:

OriginatorID=42&Amount=44.76

If any of the special characters delimiting the series of name/value pairs is required in the value portion of the name/value pair, then proper encoding must be performed so that the parsing of the string can be performed. The following are the special characters which must be substituted for any values:

Special Character	Encoded Value
&	%26
=	%3d
?	%3f

For example, the following payment information string is not valid:

OriginatorID=42&Amount=44.76&HomeURL=http://Domain/Home.aspx?Data=ABC&ID=123

But the following payment information string is valid:

OriginatorID=42&Amount=44.76&HomeURL=http://Domain/Home.aspx%3fData%3dABC%26ID%3d123

Many programming languages offer a URL encoding utility that may be used to perform the encoding. If available, it is highly recommended to take advantage of those utilities. Otherwise, a simple text substitution routine may be easily developed based on the information above.

After the payment information string is constructed, a call to the Tokenize method of the tokenization WCF service is performed passing the payment information string. The Tokenize method returns a token which is used in place of the payment information (e.g. ddbb47ed-3f9f-4161-833f-3b850e3fa8ea).

At that point a URL may be constructed that the user may be redirected to as the following demonstrates:

http://domain.cpp.dev.cboss.com/Payment.aspx?id=ddbb47ed-3f9f-4161-833f-3b850e3fa8ea

The following example is written in C# and demonstrates the payment information construction, URL encoding, tokenization service, and payment URL construction.

```
// ---------------------------------------------------------------------------
// <copyright file="HostedRedirect.aspx.cs" company="cboss">
// Copyright (c) 2010. All rights reserved.
// </copyright>
// <author>Douglas J. Knickerbocker</author>
// ---------------------------------------------------------------------------
namespace Cboss.CentralPayment.UI.Web.Example
{
    using System;
    using System.Web;

    using Cboss.CentralPayment.UI.Web.Example.DataTransferService;

    /// <summary>
    /// An example page which constructs a URL with tokenized payment data.
    /// </summary>
    public partial class HostedRedirect : System.Web.UI.Page
    {
        #region Methods

        protected void Page_Load(object sender, EventArgs e)
        {
            string paymentURL, paymentInformation, tokenizedPaymentInformation;
            TokenizationClient tokenizationClient;

            // Specify the payment URL.
            paymentURL = "http://experiment.cpp.cboss.com/Payment.aspx";

            // Add the REQUIRED input parameters.
            paymentInformation = string.Format("{0}={1}", "OriginatorID", "42");
            paymentInformation += string.Format("&{0}={1}", "Amount", "44.76");

            // Add the OPTIONAL input parameters.
            paymentInformation += string.Format("&{0}={1}", "AmountLine1Name", "Subtotal");
            paymentInformation += string.Format("&{0}={1}", "AmountLine1Value", "41.5");
            paymentInformation += string.Format("&{0}={1}", "AmountLine2Name", "Convenience Fee");
            paymentInformation += string.Format("&{0}={1}", "AmountLine2Value", "3.26");
            paymentInformation += string.Format("&{0}={1}", "CancelURL",
HttpUtility.UrlEncode("http://Domain/Cancel.aspx"));
            paymentInformation += string.Format("&{0}={1}", "HomeURL",
HttpUtility.UrlEncode("http://Domain/Home.aspx?Data=ABC&ID=123"));
            paymentInformation += string.Format("&{0}={1}", "Number", "123458");
            paymentInformation += string.Format("&{0}={1}", "Comment1", "ITEM:=XYZ,TUV");
            paymentInformation += string.Format("&{0}={1}", "Comment2", "USER:=JSMITH");

            // Add the OPTIONAL basket input parameters.
            paymentInformation += string.Format("&{0}={1}", "ItemCount", "2");
            paymentInformation += string.Format("&{0}={1}", "ItemDescription1", "XYZ");
            paymentInformation += string.Format("&{0}={1}", "ItemPrice1", "25.64");
            paymentInformation += string.Format("&{0}={1}", "ItemQuantity1", "1");
            paymentInformation += string.Format("&{0}={1}", "ItemDescription2", "TUV");
            paymentInformation += string.Format("&{0}={1}", "ItemPrice2", "7.93");
            paymentInformation += string.Format("&{0}={1}", "ItemQuantity2", "2");

            // Add the OPTIONAL field populate input paramters.
            paymentInformation += string.Format("&{0}={1}", "PaymentType", "Credit");
            paymentInformation += string.Format("&{0}={1}", "BillingAddressLine1", "11 Main Street");
            paymentInformation += string.Format("&{0}={1}", "BillingAddressLine2", "#1");
```

```
            paymentInformation += string.Format("&{0}={1}", "BillingCity", "Anytown");
            paymentInformation += string.Format("&{0}={1}", "BillingCountry", "USA");
            paymentInformation += string.Format("&{0}={1}", "BillingEmail", "john@doe.com");
            paymentInformation += string.Format("&{0}={1}", "BillingFirstName", "John");
            paymentInformation += string.Format("&{0}={1}", "BillingLastName", "Doe");
            paymentInformation += string.Format("&{0}={1}", "BillingMiddleName", "James");
            paymentInformation += string.Format("&{0}={1}", "BillingPhoneAreaCode", "111");
            paymentInformation += string.Format("&{0}={1}", "BillingPhoneNumber", "2223333");
            paymentInformation += string.Format("&{0}={1}", "BillingState", "OH");
            paymentInformation += string.Format("&{0}={1}", "BillingZip", "12345");

            // Use the data transfer service to tokenize the payment information.
            tokenizationClient = new TokenizationClient();
            tokenizedPaymentInformation = tokenizationClient.Tokenize(paymentInformation).ToString();

            // Construct the payment URL with the tokenized payment information.
            this.HyperLinkIntegration.NavigateUrl = string.Format("{0}?id={1}", paymentURL,
tokenizedPaymentInformation);
            this.HyperLinkIntegration.Text = this.HyperLinkIntegration.NavigateUrl;
        }

        #endregion Methods
    }
}
```

5.2.3 - Non-Hosted Integration Method

To use this method, a client web site would build a web page with the various intake fields (e.g. credit card number, billing address). Based on the data entered on that web page construct the payment information and calls the payment web service to authorize the payment.

The following are the two steps to this integration process:

1. Construct payment information string of name/value pairs.
1. Call the payment web service with that payment information which will return the authorization response.

The process of construction the name/value pairs is exactly the same as described in the previous section. The only difference is that additional payment parameters may need to be specified such as **PaymentType** and **CreditCardNumber**.

The following example is written in C# and demonstrates the payment information construction and payment service.

```
// ----------------------------------------------------------------------------
// <copyright file="NonHosted.aspx.cs" company="cboss">
// Copyright (c) 2010. All rights reserved.
// </copyright>
// <author>Douglas J. Knickerbocker</author>
// ----------------------------------------------------------------------------
namespace Cboss.CentralPayment.UI.Web.Example
{
    using System;

    using Cboss.CentralPayment.UI.Web.Example.PaymentService;

    /// <summary>
    /// An example page which authorized payment information with the payment
    /// service.
    /// </summary>
    public partial class NonHosted : System.Web.UI.Page
    {
        #region Methods

        protected void ButtonPay_Click(object sender, EventArgs e)
        {
            string paymentInformation;
            Payment payment;
```

```
            // Add the REQUIRED input parameters.
            paymentInformation = string.Format("{0}={1}", "OriginatorID", "1");
            paymentInformation += string.Format("&{0}={1}", "Amount", "44.76");

            // Add the OPTIONAL input parameters.
            paymentInformation += string.Format("&{0}={1}", "AmountLine1Name", "Subtotal");
            paymentInformation += string.Format("&{0}={1}", "AmountLine1Value", "41.5");
            paymentInformation += string.Format("&{0}={1}", "AmountLine2Name", "Convenience Fee");
            paymentInformation += string.Format("&{0}={1}", "AmountLine2Value", "3.26");
            paymentInformation += string.Format("&{0}={1}", "Number", "12345");
            paymentInformation += string.Format("&{0}={1}", "Comment1", "ITEM:=XYZ,TUV");
            paymentInformation += string.Format("&{0}={1}", "Comment2", "USER:=JSMITH");

            // Add the OPTIONAL basket input parameters.
            paymentInformation += string.Format("&{0}={1}", "ItemCount", "2");
            paymentInformation += string.Format("&{0}={1}", "ItemDescription1", "XYZ");
            paymentInformation += string.Format("&{0}={1}", "ItemPrice1", "25.64");
            paymentInformation += string.Format("&{0}={1}", "ItemQuantity1", "1");
            paymentInformation += string.Format("&{0}={1}", "ItemDescription2", "TUV");
            paymentInformation += string.Format("&{0}={1}", "ItemPrice2", "7.93");
            paymentInformation += string.Format("&{0}={1}", "ItemQuantity2", "2");

            // Add the OPTIONAL field populate input paramters.
            paymentInformation += string.Format("&{0}={1}", "PaymentType", "Credit"); // "Cash", "Check"
            paymentInformation += string.Format("&{0}={1}", "CreditCardCVVCode", "123");
            paymentInformation += string.Format("&{0}={1}", "CreditCardExpirationMonth", "05");
            paymentInformation += string.Format("&{0}={1}", "CreditCardExpirationYear", "2012");
            paymentInformation += string.Format("&{0}={1}", "CreditCardNumber", "4111111111111111");
            paymentInformation += string.Format("&{0}={1}", "CreditCardType", "Visa");
            paymentInformation += string.Format("&{0}={1}", "BankAccountNumber", "74123");
            paymentInformation += string.Format("&{0}={1}", "BankRoutingNumber", "041001039");
            paymentInformation += string.Format("&{0}={1}", "CashReceiptNumber", "32124");
            paymentInformation += string.Format("&{0}={1}", "BillingAddressLine1", "11 Main Street");
            paymentInformation += string.Format("&{0}={1}", "BillingAddressLine2", "#1");
            paymentInformation += string.Format("&{0}={1}", "BillingCity", "Anytown");
            paymentInformation += string.Format("&{0}={1}", "BillingCountry", "US");
            paymentInformation += string.Format("&{0}={1}", "BillingEmail", "john@doe.com");
            paymentInformation += string.Format("&{0}={1}", "BillingFirstName", "John");
            paymentInformation += string.Format("&{0}={1}", "BillingLastName", "Doe");
            paymentInformation += string.Format("&{0}={1}", "BillingMiddleName", "James");
            paymentInformation += string.Format("&{0}={1}", "BillingPhoneAreaCode", "111");
            paymentInformation += string.Format("&{0}={1}", "BillingPhoneNumber", "2223333");
            paymentInformation += string.Format("&{0}={1}", "BillingState", "OH");
            paymentInformation += string.Format("&{0}={1}", "BillingZip", "12345");

            // Use the payment service to authorized the payment information.
            payment = new Payment();
            this.LabelPaymentResponse.Text = payment.Submit(paymentInformation);
        }

        #endregion Methods
    }
}
```

5.3 - Testing

Testing and verification includes the following steps:

1. Run test transactions during development, staging, and/or production environments by using the test accounts and URLs provided. In addition, test for all payment options that are set up in originator setup (all credit card types, electronic check or cash).
2. Verify that the transactions are working properly by generating a transaction report in the CPP Administrative area and verify that the customer order receipt was received.
3. Test that reconciliation files are being sent in proper format, at set times.

6 - Real Time Client System Updates

The originator may be setup to provide backend notification to the client system of the successful authorization of a payment. This is performed by setting the originator to one of the following updates modes:

- No Update
- XML Update
- XML Update Lite

The XML Update Lite mode is performed by the CPP gateway calling a well defined web service function on the client system that provides the Number, Comment 1, and Comment 2 information for a payment. The XML Update mode is the same as the XML Update Lite mode with the exception that it includes more information for a payment.

The CPP gateway immediately attempts to notify the client system after successful authorization. In the event the notification is unsuccessful in communicating with the client system, a retry will occur at regular intervals to attempt to notify the client system.

A stub Microsoft.Net Web service project is provided that can be adapted to the client's needs by having the client develop the SOAPXMLLite or SOAPXML method as required. Once the stub Web service project has been properly developed and deployed, the client will need to provide CBOSS with the URL of the deployment so that the originator may be configured to perform the XML Update Lite or XML Update to the appropriate URL.

The SOAPXMLLite and SOAPXML methods must return one of three statuses:

- Success
- Failure
- Do Not Retry

Additionally, in the event of Failure or Do Not Retry, it is helpful if the client provides a reason why the failure occurred or why they do not want to retry. This display is performed by setting the error message property in the SOAP header.

7 - Payment Information Parameters

The various parameters available for hosted solutions and non-hosted solutions are described in this section.

Please note that in a hosted solution using the **Redirect Mode** processing, each of the parameters described below are present in the redirected URLs tokenized string and may be used in the building of a receipt page.

7.1 - Required Parameters

The parameters shown below are required for both hosted and non-hosted solutions.

Table 2 – Required Parameters

Name	Format	Description
OriginatorID	integer	OriginatorID is the application identification number. This number is assigned to the merchant/originating application. **Example:** 1
Amount	money	The total amount to be charged in US dollars. The transaction amount must be in the format ddd[…].cc without a dollar sign. **Example:** 44.76

7.2 - Optional Parameters

The parameters shown below are optional for both hosted and non-hosted solutions.

Table 3 – Optional Parameters

Name	Format	Description
AmountLine1Value	money	The transaction amount less the charge such as a fee to be charged in US dollars. It must be in the format ddd[…].cc without a dollar sign. **Note:** If AmountLine1Value contains a value, AmountLine2Value must also contain a value, and both line values should add up to the payment amount. **Note:** The CPP gateway does not verify that the amounts in Line 1 and Line 2 add up to the payment amount. It just provides additional details for the payment process. **Example:** 41.5
AmountLine2Value	money	The charge amount such as a fee to be charged in US dollars. It must be in the format ddd[…].cc without a dollar sign. **Example:** 3.26
AmountLine1Name	char[50]	Name that will appear on the payment summary for the

		value of AmountLine1. **Example:** Subtotal
AmountLine2Name	char[50]	Name that will appear on the payment summary for the value of AmountLine2. **Example:** Convenience Fee
CancelURL	text	Include the cancel field to direct customers to a custom page if the transaction is cancelled. This field value will be the target of a "Cancel" button to return the user to another system. If this field is not included, the "Cancel" button is not displayed. **Note:** The field value must be a fully-qualified URL. **Example:** http://DomainName/Payment/Cancel.aspx
HomeURL	text	Include the home field to direct customers to a custom page after a successful payment is displayed. This field value will be the target of a "Home" button to return the user to another system. If this field is not included, the "Home" button is not displayed. **Note:** The field value must be a fully-qualified URL. **Example:** http://DomainName/Payment/Success.aspx
Number	char[50]	The actual payment number to use. Maximum length for the payment number is 50 characters. If a number is not supplied, a number will be generated by the CPP gateway. The format of the generated number is not guaranteed, but it will generally be sequential. **Note:** Care must be taken to ensure that the number is unique to the CPP gateway either by segmenting numbers by originator, or using the number generation facilities in the CPP gateway. **Example:** 12345
Comment1	char[128]	Custom information stored with the payment. **Example:** ITEM:=XYZ,TUV
Comment2	char[128]	Custom information stored with the payment. **Example:** USER:=JSMITH

7.3 - Basket Parameters

If basket information is passed to the CPP gateway, then all basket information fields must be present. If basket information is not included, then all basket information fields should be omitted. The first item is identified by **ItemDescription1**, **ItemQuantity1**, and **ItemPrice1**. Additional items are identified by increasing numbers, with no limit imposed

by the system. In practice, it is advisable to limit the number of basket items to less than 20.

The **Amount** field is still required, but it does not have to be the sum of (**ItemPrice** * **ItemQuantity**)[1..n]. This allows for shipping, tax, etc., to be included in the total charge amount.

The parameters shown below are optional for both hosted and non-hosted solutions.

Table 4 – Basket Parameters

Name	Format	Description
ItemCount	Integer	Value of the total items to be stored **Example:** 2
ItemDescription1, ItemDescription2, … ItemDescriptionN	char[200]	Descriptions corresponding to the line item **Example:** XYZ, TUV
ItemQuantity1, ItemQuantity2, … ItemQuantityN	Integer	Quantities corresponding to the item description **Example:** 1, 2
ItemPrice1, ItemPrice2, … ItemPriceN	Money	Prices corresponding to the item descriptions **Example:** 25.64, 7.93

7.4 - Payment Parameters

The parameters shown below are optional for hosted solutions and required for non-hosted solutions.

In a hosted solution, using these parameters will populate the payment processing intake form upon the initial page load. In a non-hosted solution the only required information is that which corresponds to the payment type specified. The only exception to this rule is the **CreditCardCVVCode** and **CreditCardType** which are not required for **Credit** payments.

Table 5 – Payment Parameters

Name	Format	Description
PaymentType	text	Specifies a 'Credit', 'Check', or 'Cash' type of payment. **Example:** Credit
CreditCardCVVCode	integer	The CVV2 code on the credit card. This is only used when the payment type is set to 'Credit'. **Note:** In *Redirect Mode* this value becomes '***'. **Example:** 123
CreditCardExpirationMonth	integer	The expiration month of the credit card. This is only used when the payment type is set to 'Credit'.

		Note: In *Redirect Mode* this value becomes '****'. **Example:** 05
CreditCardExpirationYear	integer	The expiration year of the credit card. This is only used when the payment type is set to 'Credit'. **Note:** In *Redirect Mode* this value becomes '****'. **Example:** 2012
CreditCardNumber	integer	The credit card number. This is only used when the payment type is set to 'Credit'. **Note:** In *Redirect Mode* this value becomes '*********XXXX'. **Example:** 4111111111111111
CreditCardType	text	The 'American Express', 'Discover', 'MasterCard', or 'Visa' credit card type. This is only used when the payment type is set to 'Credit'. **Example:** Visa
BankAccountNumber	integer	The account number of the electronic check. This is only used when the payment type is set to 'Check'. **Note:** In *Redirect Mode* this value becomes '*********XXXXX'. **Example:** 74123
BankRoutingNumber	integer	The routing number for the electronic check. This is only used when the payment type is set to 'Check'. **Note:** In *Redirect Mode* this value becomes '*********'. **Example:** 041001039
CashReceiptNumber	char[100]	The receipt number of the cash. This is only used when the payment type is set to 'Cash'. **Example:** 32124

7.5 - Billing Parameters

The parameters shown below are optional for both hosted and non-hosted solutions.

In a hosted solution, using these parameters will populate the payment processing intake form upon the initial page load.

Table 6 – Billing Parameters

Name	Format	Description

BillingFirstName	char[50]	The first name of the account holder. **Example:** John
BillingMiddleName	char[50]	The middle name of the account holder. **Example:** James
BillingLastName	char[50]	The last name of the account holder. **Example:** Doe
BillingAddressLine1	char[100]	The address line 1 of the account holder. **Example:** 11 Main Street
BillingAddressLine2	char[100]	The address line 2 of the account holder. **Example:** #1
BillingCity	char[50]	The city of the account holder. **Example:** Anytown
BillingState	char[2]	The state abbreviation of the account holder. **Example:** OH
BillingZip	char[10]	The zip of the account holder. **Example:** 12345
BillingCountry	char[50]	The country of the account holder. **Example:** US
BillingPhoneAreaCode	integer	The area code of the account holder. **Example:** 111
BillingPhoneNumber	integer	The phone number of the account holder. **Example:** 2223333
BillingEmail	char[50]	The email address of the account holder. **Example:** john@doe.com

7.6 - Output Parameters

The parameters shown below are output only for both hosted and non-hosted solutions.

Table 7 – Output Parameters

Name	Format	Description
DateAuthorized	date	The date and time the payment was authorized.

		Example: 11/29/2006 1:50:54 PM
Errror	boolean	A flag to determine if an error has occurred during payment processing. **Example:** False
ErrorMessage	text	The error message if an error or system error has occurred.
SystemError	boolean	A flag to determine if a system error has occurred during payment processing. **Example:** False
Status	text	The 'Submitted', 'Authorized', or 'Rejected' status of the payment. **Example:** Authorized
Token	text	A unique token that represents the payment which may be used for further payment resubmissions if the payment is 'Submitted' or 'Rejected'.

8 - Conclusion

The CPP gateway is a robust and secure PCI DSS certified electronic payment processing gateway for online credit card and electronic check payments that can easily be integrated with your business applications.

This document provided an overview of what the CPP gateway had to offer for payment processing capabilities. The various integration methods were described and detailed example code for each of the integration methods was made available to make the integration process simple.

Appendix A - Glossary of Terms

A list of terms used in this document is included here for reference.

Query String
The HTTP query string is specified by the values following the question mark (?). Several different processes can generate a query string. Query strings are widely used on the Web for both dynamic and static links to databases. They are appended to the URL using the special characters as (?), (&), and (=).

SOAP
Although not used as an acronym anymore, the protocol provides a standard, extensible, framework for packaging and exchanging XML messages. It is defined by the different ways the technology can be interpreted.

The Service Oriented Architecture Protocol message represents the information needed to invoke a service or reflect the results of a service invocation, and contains the information specified in the service interface definition.

The Simple Object Access Protocol message represents a method invocation on a remote object, and the serialization of in the argument list of that method that must be moved from the local environment to the remote environment.

Web Services
A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface descried in a machine processable format (specifically Web Services Description Language, WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

Appendix B - Tokenization WCF Service WSDL

The following is the WSDL for the tokenization WCF service.

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
xmlns:wsap="http://schemas.xmlsoap.org/ws/2004/08/addressing/policy"
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:msc="http://schemas.microsoft.com/ws/2005/12/wsdl/contract"
xmlns:tns="http://www.cboss.com/DataTransferTokenizer/1.0"
xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl" xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:wsa10="http://www.w3.org/2005/08/addressing" xmlns:wsx="http://schemas.xmlsoap.org/ws/2004/09/mex"
xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata" name="PublicTokenization"
targetNamespace="http://www.cboss.com/DataTransferTokenizer/1.0" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsp:Policy wsu:Id="WSHttpBinding_Tokenization_policy">
    <wsp:ExactlyOne>
      <wsp:All>
        <wsaw:UsingAddressing />
      </wsp:All>
    </wsp:ExactlyOne>
  </wsp:Policy>
  <wsdl:types>
    <xsd:schema targetNamespace="http://www.cboss.com/DataTransferTokenizer/1.0/Imports">
      <xsd:import
schemaLocation="http://experiment.cpp.cboss.com/Services/DataTransfer/1.0/PublicTokenization.svc?xsd=xsd0"
namespace="http://www.cboss.com/DataTransferTokenizer/1.0" />
      <xsd:import
schemaLocation="http://experiment.cpp.cboss.com/Services/DataTransfer/1.0/PublicTokenization.svc?xsd=xsd2"
namespace="http://schemas.datacontract.org/2004/07/Cboss.DataTransferTokenizer.Application.Presentation" />
      <xsd:import
schemaLocation="http://experiment.cpp.cboss.com/Services/DataTransfer/1.0/PublicTokenization.svc?xsd=xsd1"
namespace="http://schemas.microsoft.com/2003/10/Serialization/" />
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="Tokenization Detokenize InputMessage">
    <wsdl:part name="parameters" element="tns:Detokenize" />
  </wsdl:message>
  <wsdl:message name="Tokenization_Detokenize_OutputMessage">
    <wsdl:part name="parameters" element="tns:DetokenizeResponse" />
  </wsdl:message>
  <wsdl:message name="Tokenization_Detokenize_NotFoundFaultFault_FaultMessage">
    <wsdl:part
xmlns:q1="http://schemas.datacontract.org/2004/07/Cboss.DataTransferTokenizer.Application.Presentation"
name="detail" element="q1:NotFoundFault" />
  </wsdl:message>
  <wsdl:message name="Tokenization Tokenize InputMessage">
    <wsdl:part name="parameters" element="tns:Tokenize" />
  </wsdl:message>
  <wsdl:message name="Tokenization_Tokenize_OutputMessage">
    <wsdl:part name="parameters" element="tns:TokenizeResponse" />
  </wsdl:message>
  <wsdl:portType name="Tokenization">
    <wsdl:operation name="Detokenize">
      <wsdl:input wsaw:Action="http://www.cboss.com/DataTransferTokenizer/1.0/Tokenization/Detokenize"
message="tns:Tokenization_Detokenize_InputMessage" />
      <wsdl:output wsaw:Action="http://www.cboss.com/DataTransferTokenizer/1.0/Tokenization/DetokenizeResponse"
message="tns:Tokenization_Detokenize_OutputMessage" />
      <wsdl:fault
wsaw:Action="http://www.cboss.com/DataTransferTokenizer/1.0/Tokenization/DetokenizeNotFoundFaultFault"
name="NotFoundFaultFault" message="tns:Tokenization Detokenize NotFoundFaultFault FaultMessage" />
    </wsdl:operation>
    <wsdl:operation name="Tokenize">
      <wsdl:input wsaw:Action="http://www.cboss.com/DataTransferTokenizer/1.0/Tokenization/Tokenize"
message="tns:Tokenization_Tokenize_InputMessage" />
      <wsdl:output wsaw:Action="http://www.cboss.com/DataTransferTokenizer/1.0/Tokenization/TokenizeResponse"
message="tns:Tokenization Tokenize OutputMessage" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="WSHttpBinding_Tokenization" type="tns:Tokenization">
    <wsp:PolicyReference URI="#WSHttpBinding_Tokenization_policy" />
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="Detokenize">
      <soap12:operation soapAction="http://www.cboss.com/DataTransferTokenizer/1.0/Tokenization/Detokenize"
style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
```

```
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
      <wsdl:fault name="NotFoundFaultFault">
        <soap12:fault use="literal" name="NotFoundFaultFault" namespace="" />
      </wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="Tokenize">
      <soap12:operation soapAction="http://www.cboss.com/DataTransferTokenizer/1.0/Tokenization/Tokenize"
style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="PublicTokenization">
    <wsdl:port name="WSHttpBinding_Tokenization" binding="tns:WSHttpBinding_Tokenization">
      <soap12:address
location="http://experiment.cpp.cboss.com/Services/DataTransfer/1.0/PublicTokenization.svc" />
      <wsa10:EndpointReference>

<wsa10:Address>http://experiment.cpp.cboss.com/Services/DataTransfer/1.0/PublicTokenization.svc</wsa10:Address>
        <Identity xmlns="http://schemas.xmlsoap.org/ws/2006/02/addressingidentity">
          <Dns>localhost</Dns>
        </Identity>
      </wsa10:EndpointReference>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Appendix C - Integration Web Service WSDL

The following is the WSDL for the integration Web service.

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://www.cboss.com/WebService/CentralPayment/Integration"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="http://www.cboss.com/WebService/CentralPayment/Integration"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified"
targetNamespace="http://www.cboss.com/WebService/CentralPayment/Integration">
      <s:element name="SOAPXMLUpdateLite">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="OrderNumber" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="OrderComment1" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="OrderComment2" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="SOAPXMLUpdateLiteResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="SOAPXMLUpdateLiteResult"
type="tns:IntegrationSoapResponse" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="IntegrationSoapResponse">
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="ResponseID" type="s:int" />
          <s:element minOccurs="0" maxOccurs="1" name="ResponseDescription" type="s:string" />
        </s:sequence>
      </s:complexType>
      <s:element name="IntegrationSoapHeader" type="tns:IntegrationSoapHeader" />
      <s:complexType name="IntegrationSoapHeader">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="ErrorMessage" type="s:string" />
        </s:sequence>
      </s:complexType>
      <s:element name="SOAPXMLUpdate">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="RealTime" type="s:boolean" />
            <s:element minOccurs="0" maxOccurs="1" name="OriginatorID" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="OriginatorName" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="OrderID" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="OrderStatus" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="OrderNumber" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="OrderComment1" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="OrderComment2" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="BatchNumber" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="PaymentType" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="TransactionAmount" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="AmountLine1" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="AmountLine2" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="TransactionDate" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="AuthorizationDate" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="SettleDate" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="OwnerFirstName" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="OwnerMiddleName" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="OwnerLastName" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="OwnerAddress1" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="OwnerAddress2" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="OwnerCity" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="OwnerStateID" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="OwnerState" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="OwnerZip" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="OwnerCountry" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="OwnerEmail" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="OwnerPhone" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="BillingFirstName" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="BillingMiddleName" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="BillingLastName" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="BillingAddress1" type="s:string" />
```

```
            <s:element minOccurs="0" maxOccurs="1" name="BillingAddress2" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="BillingCity" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="BillingStateID" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="BillingState" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="BillingZip" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="BillingCountry" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="BillingEmail" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="BillingPhone" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="ReceiptNumber" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="ProcessorOrderNumber" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="TransactionStatus" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="Items" type="tns:ArrayOfItem" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="ArrayOfItem">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="unbounded" name="Item" nillable="true" type="tns:Item" />
        </s:sequence>
      </s:complexType>
      <s:complexType name="Item">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="ItemID" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="Quantity" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="Description" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="Price" type="s:string" />
        </s:sequence>
      </s:complexType>
      <s:element name="SOAPXMLUpdateResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="SOAPXMLUpdateResult"
type="tns:IntegrationSoapResponse" />
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:schema>
  </wsdl:types>
  <wsdl:message name="SOAPXMLUpdateLiteSoapIn">
    <wsdl:part name="parameters" element="tns:SOAPXMLUpdateLite" />
  </wsdl:message>
  <wsdl:message name="SOAPXMLUpdateLiteSoapOut">
    <wsdl:part name="parameters" element="tns:SOAPXMLUpdateLiteResponse" />
  </wsdl:message>
  <wsdl:message name="SOAPXMLUpdateLiteIntegrationSoapHeader">
    <wsdl:part name="IntegrationSoapHeader" element="tns:IntegrationSoapHeader" />
  </wsdl:message>
  <wsdl:message name="SOAPXMLUpdateSoapIn">
    <wsdl:part name="parameters" element="tns:SOAPXMLUpdate" />
  </wsdl:message>
  <wsdl:message name="SOAPXMLUpdateSoapOut">
    <wsdl:part name="parameters" element="tns:SOAPXMLUpdateResponse" />
  </wsdl:message>
  <wsdl:message name="SOAPXMLUpdateIntegrationSoapHeader">
    <wsdl:part name="IntegrationSoapHeader" element="tns:IntegrationSoapHeader" />
  </wsdl:message>
  <wsdl:portType name="IntegrationSoap">
    <wsdl:operation name="SOAPXMLUpdateLite">
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Called from the central payment engine to perform
customized updates.</documentation>
      <wsdl:input message="tns:SOAPXMLUpdateLiteSoapIn" />
      <wsdl:output message="tns:SOAPXMLUpdateLiteSoapOut" />
    </wsdl:operation>
    <wsdl:operation name="SOAPXMLUpdate">
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Called from the central payment engine to perform
customized updates.</documentation>
      <wsdl:input message="tns:SOAPXMLUpdateSoapIn" />
      <wsdl:output message="tns:SOAPXMLUpdateSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="IntegrationSoap" type="tns:IntegrationSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
    <wsdl:operation name="SOAPXMLUpdateLite">
      <soap:operation soapAction="http://www.cboss.com/WebService/CentralPayment/Integration/SOAPXMLUpdateLite"
style="document" />
      <wsdl:input>
        <soap:body use="literal" />
        <soap:header message="tns:SOAPXMLUpdateLiteIntegrationSoapHeader" part="IntegrationSoapHeader"
use="literal" />
      </wsdl:input>
      <wsdl:output>
```

```
        <soap:body use="literal" />
        <soap:header message="tns:SOAPXMLUpdateLiteIntegrationSoapHeader" part="IntegrationSoapHeader"
use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="SOAPXMLUpdate">
      <soap:operation soapAction="http://www.cboss.com/WebService/CentralPayment/Integration/SOAPXMLUpdate"
style="document" />
      <wsdl:input>
        <soap:body use="literal" />
        <soap:header message="tns:SOAPXMLUpdateIntegrationSoapHeader" part="IntegrationSoapHeader"
use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
        <soap:header message="tns:SOAPXMLUpdateIntegrationSoapHeader" part="IntegrationSoapHeader"
use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="Integration">
    <documentation xmlns="http://schemas.xmlsoap.org/wsdl/" />
    <wsdl:port name="IntegrationSoap" binding="tns:IntegrationSoap">
      <soap:address
location="http://localhost/Projects.Net/cboss/WebService/Client/CentralPayment/Integration.asmx" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Appendix D - Payment Web Service WSDL

The following is the WSDL for the payment web service.

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tns="http://www.cboss.com/cboss/WebService/CentralPayment/Payment"
xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
targetNamespace="http://www.cboss.com/cboss/WebService/CentralPayment/Payment"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified"
targetNamespace="http://www.cboss.com/cboss/WebService/CentralPayment/Payment">
      <s:element name="PreSubmit">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="Value" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="PreSubmitResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="PreSubmitResult" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="PaymentSoapHeader" type="tns:PaymentSoapHeader" />
      <s:complexType name="PaymentSoapHeader">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="UserLanguages" type="tns:ArrayOfString" />
        </s:sequence>
        <s:anyAttribute />
      </s:complexType>
      <s:complexType name="ArrayOfString">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="unbounded" name="string" nillable="true" type="s:string" />
        </s:sequence>
      </s:complexType>
      <s:element name="Submit">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="Value" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="SubmitResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="SubmitResult" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:schema>
  </wsdl:types>
  <wsdl:message name="PreSubmitSoapIn">
    <wsdl:part name="parameters" element="tns:PreSubmit" />
  </wsdl:message>
  <wsdl:message name="PreSubmitSoapOut">
    <wsdl:part name="parameters" element="tns:PreSubmitResponse" />
  </wsdl:message>
  <wsdl:message name="PreSubmitPaymentSoapHeader">
    <wsdl:part name="PaymentSoapHeader" element="tns:PaymentSoapHeader" />
  </wsdl:message>
  <wsdl:message name="SubmitSoapIn">
    <wsdl:part name="parameters" element="tns:Submit" />
  </wsdl:message>
  <wsdl:message name="SubmitSoapOut">
    <wsdl:part name="parameters" element="tns:SubmitResponse" />
  </wsdl:message>
  <wsdl:message name="SubmitPaymentSoapHeader">
    <wsdl:part name="PaymentSoapHeader" element="tns:PaymentSoapHeader" />
  </wsdl:message>
  <wsdl:portType name="PaymentSoap">
    <wsdl:operation name="PreSubmit">
```

```
      <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">PreSubmits a payment using the base-64
encoded value that contains the originator hash value as well as the TripleDES cipher data for the payment.
Authorization does not occur, however a payment token is created.</wsdl:documentation>
      <wsdl:input message="tns:PreSubmitSoapIn" />
      <wsdl:output message="tns:PreSubmitSoapOut" />
    </wsdl:operation>
    <wsdl:operation name="Submit">
      <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Submits a payment using the base-64
encoded value that contains the originator hash value as well as the TripleDES cipher data for the
payment.</wsdl:documentation>
      <wsdl:input message="tns:SubmitSoapIn" />
      <wsdl:output message="tns:SubmitSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="PaymentSoap" type="tns:PaymentSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="PreSubmit">
      <soap:operation soapAction="http://www.cboss.com/cboss/WebService/CentralPayment/Payment/PreSubmit"
style="document" />
      <wsdl:input>
        <soap:body use="literal" />
        <soap:header message="tns:PreSubmitPaymentSoapHeader" part="PaymentSoapHeader" use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="Submit">
      <soap:operation soapAction="http://www.cboss.com/cboss/WebService/CentralPayment/Payment/Submit"
style="document" />
      <wsdl:input>
        <soap:body use="literal" />
        <soap:header message="tns:SubmitPaymentSoapHeader" part="PaymentSoapHeader" use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="PaymentSoap12" type="tns:PaymentSoap">
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="PreSubmit">
      <soap12:operation soapAction="http://www.cboss.com/cboss/WebService/CentralPayment/Payment/PreSubmit"
style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
        <soap12:header message="tns:PreSubmitPaymentSoapHeader" part="PaymentSoapHeader" use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="Submit">
      <soap12:operation soapAction="http://www.cboss.com/cboss/WebService/CentralPayment/Payment/Submit"
style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
        <soap12:header message="tns:SubmitPaymentSoapHeader" part="PaymentSoapHeader" use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="Payment">
    <wsdl:port name="PaymentSoap" binding="tns:PaymentSoap">
      <soap:address location="http://experiment.cpp.cboss.com/Services/Payment.asmx" />
    </wsdl:port>
    <wsdl:port name="PaymentSoap12" binding="tns:PaymentSoap12">
      <soap12:address location="http://experiment.cpp.cboss.com/Services/Payment.asmx" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```