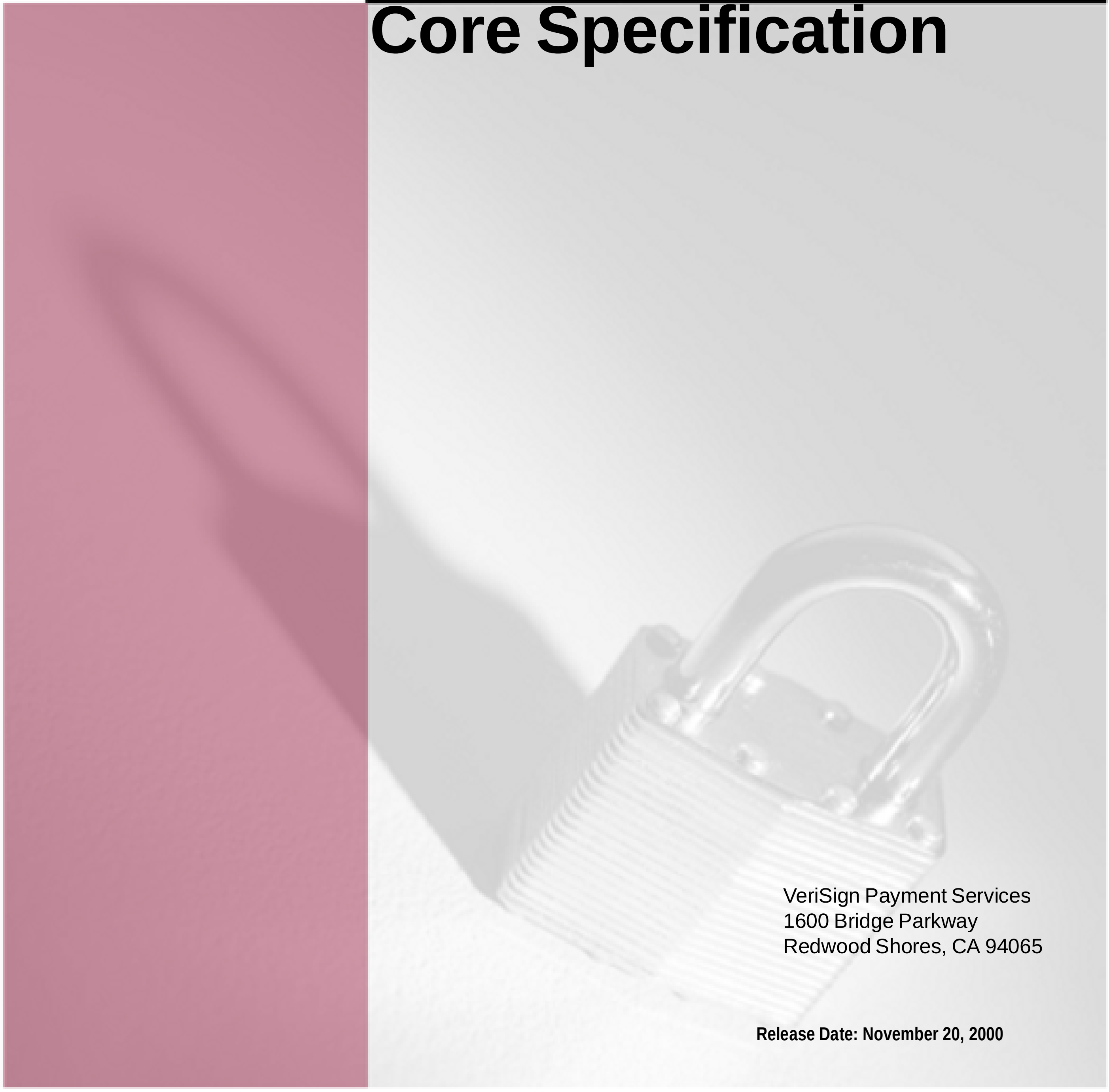
XMLPay 1.0 Core Specification



VeriSign Payment Services
1600 Bridge Parkway
Redwood Shores, CA 94065

Release Date: November 20, 2000

XML Pay 1.0: Core Specification

Trademark Notices

Note This document may describe features and/or functionality that are not present in your software or your service agreement. Contact your account representative to learn more about what is available with this VeriSign product.

Contents

CHAPTER 1

Introduction

This document, the *XMLPay 1.0 Core Specification*, defines an XML syntax for payment transaction requests, responses, and receipts in a payment processing network.

The typical user of XMLPay is an Internet merchant or merchant aggregator who wants to dispatch credit card, corporate purchase card, Automated Clearing House (ACH), or other payment requests to a financial processing network.

Using the data type definitions specified by XMLPay, a user creates a client payment request and dispatches it—using a mechanism left unspecified by XMLPay—to an associated XMLPay-compliant server component. Responses are also formatted in XML and convey the results of the payment requests to the client.

XMLPay includes support for digitally-signed XML objects. Digital signatures are used both for the purpose of authenticating requests and responses and as a foundation for a higher-level digital receipt architecture based on an X.509 Public Key Infrastructure.

XMLPay uses the digital signature format being specified by the joint IETF/W3C XML Digital Signature Working Group.

Note For specific examples of how to submit XML documents using the Payflow Pro client API, see the Payflow Pro SDK download package itself.

About this Document

This document is organized as follows:

- Chapter 2, “XMLPay Overview” describes XML and XMLPay, presenting processing models, networking, messsaging and related specifications.
- Chapter 3, “XMLPay Syntax” presents the syntax for transaction requests, responses, and receipts using a simplified notation.
- Chapter 4, “XMLPay Transaction Profiles” lists the transactions supported for each tender type—ACH, Card, Check—along with the data elements used for each of those transactions.
- Chapter 5, “XMLPay Examples” gives several XMLPay document samples.
- Appendix A,“XMLPay Schemas” provides standard W3C schemas for XMLPay and XMLPay Types.
- Appendix B,“XMLPay DTD” presents the Document Type Definition XMLPay schema.
- Appendix C,“Transaction Results” lists VeriSign transaction result codes and response messages as well as Address Verification Service result codes.

CHAPTER 2

XMLPay Overview

About XML

XML stands for *e**X**tensible **M**arkup **L**anguage*. XML is derived from Standardized General Markup Language (SGML) and HyperText Markup Language (HTML). In a sense, XML is SGML "lite", but XML manages to maintain SGML's strength as well as HTML's simplicity. What's more, XML can be converted to HTML.

The main advantage of XML is that text can be meaningfully annotated. In XML, markers identify and tag the text. But the markers themselves have no defined meaning; it is the applications that define the markers.

XML allows complex transactions to be structured, and client integration is simplified through the exchange of XML documents. Since XML provides support for digital signatures, documents from unknown sources can be trusted. In addition, XML can easily produce large documents, such as transaction logs and reports.

Benefits of XML

The main benefits of XML are that it:

- Allows text annotation
- Presents text, data and content as a structured document to applications
- Facilitates integration of diverse applications

In addition to these two benefits, XML is easy to:

- Read (all text)
- Parse and validate
- Search for content
- Produce

Well-formed XML Document

A well-formed XML document conforms to XML syntax. A well-formed XML document must have:

- An XML processing instruction at the beginning (prolog)
- A single root element
- Matching (case sensitive) start and end tags for all elements
- All XML elements properly nested
- Attribute values in quotes

Example 1

```
<?xml  version="1.0"?>
<Card>
       <CardType> MasterCard </CardType>
       <CardNumber>495576040004</CardNumber>
       <ExpDate>200011</ExpDate>
       CVNum>828</CVNum>
</Card>
```

Example 2

```
<?xml  version="1.0"?>
<Items ID="IDTI123">
       <Item Number="1">
              <Description>Electric Toothbrush</Description>
              <Quantity>1</Quantity>
              <UnitPrice>100</UnitPrice>
              <TotalAmt>100</TotalAmt>
       </Item>
       <Item Number="2">
              <Description>ToothPaste</Description>
              <Quantity>20</Quantity>
              <UnitPrice>2</UnitPrice>
              <TotalAmt>40</TotalAmt>
       </Item>
       …
</Items>
```

About XMLPay

XMLPay defines an XML syntax for payment transaction requests, responses, and receipts in a payment processing network.

The typical user of XMLPay is an Internet merchant or merchant aggregator who wants to dispatch credit card, corporate purchase card, Automated Clearing House (ACH), or other payment requests to a financial processing network.

Using the data type definitions specified by XMLPay, a user creates a client payment request and dispatches it—using a mechanism left unspecified by XMLPay—to an associated XMLPay-compliant server component. Responses are also formatted in XML and convey the results of the payment requests to the client.

XMLPay Instruments

XMLPay supports payment processing using the following payment instruments:

- Retail credit and debit cards
- Corporate purchase cards: Levels 1, 2 and 3
- Internet checks
- ACH

XMLPay Operations

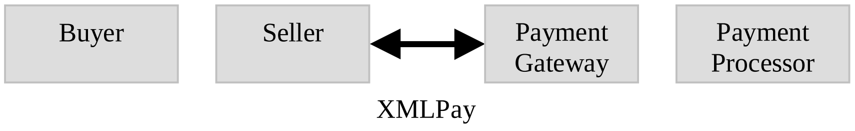
Typical XMLPay operations include:

- Funds authorization and capture
- Sales and repeat sales
- Voiding of transactions

XMLPay Processing Models

XMLPay is intended for use in both Business-to-Consumer (B2C) and Business-to-Business (B2B) payment processing applications.

Business-to-Consumer

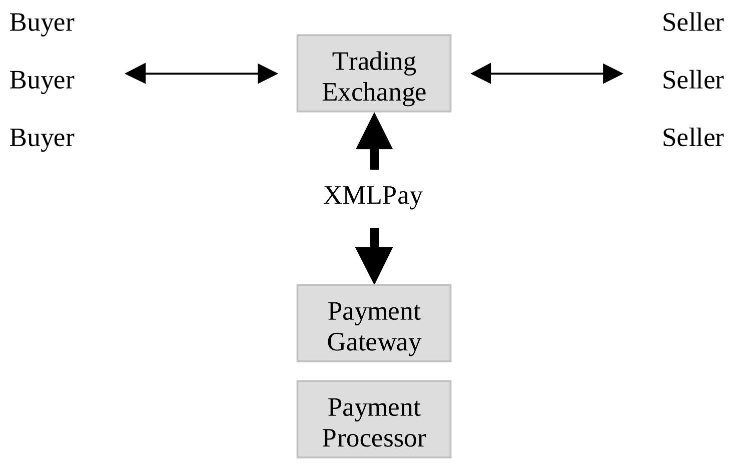


In a B2C transaction, the Buyer presents a payment instrument (e.g., credit card number) to a Seller in order move money from the Buyer to the Seller (or vice-versa in the case of a credit or refund).

Use of XMLPay comes into play when the Seller needs to forward the Buyer's payment information on to a Payment Processor. The Seller formats a XMLPayRequest and submits it either directly to an XMLPay-compliant payment processor or, as pictured, indirectly via a XMLPay-compliant Payment Gateway. Responses have type XMLPayResponse.

The Buyer-to-Seller and Payment Gateway-to-Payment Processor channels are typically left unaffected by use of XMLPay. For example, XMLPay is typically not used in direct communications between the buyer and the seller. Instead, conventional HTML form submission or other Internet communication methods are typically used. Similarly, because Payment Processors often differ considerably in the formats they specify for payment requests, it is often desired to localize XMLPay server logic at the Payment Gateway, leaving the legacy connections between gateways and processors unchanged.

Business-to-Business



When used in support of B2B transactions, the Seller does not typically initiate XMLPay requests. Instead, an aggregator or trading exchange uses XMLPay to communicate business-focused purchasing information (such as level 3 corporate purchase card data) to a payment gateway.

In this way, the trading exchange links payment execution to other XML-based communications between Buyers and Sellers such as Advance Shipping Notice delivery, Purchase Order communication, or other B2B communication functions.

XMLPay Networking

XMLPay has a flexible approach to the networking aspects of payment communication. Achieving network security and efficient communications are the primary goals.

Data Transport

The mechanics of how XMLPay requests are passed to server processing components is outside the scope of the core XMLPay specification. Instead, these transport mechanisms are documented in separate XMLPay transport specifications.

The typical application scenario involves passing payment requests synchronously to a server over a persistent TCP/IP connection that is not torn down until a response is received or a timeout occurs. Alternatively, asynchronous processing paradigms (such as store and forward) are also possible.

Security and Authentication

Payment transactions typically include highly sensitive data such as credit card numbers, their associated expiration dates, payment amounts, and line-item invoice details. Such information must be protected when transmitted over insecure networks. It is recommended that XMLPay messages be encrypted using the strongest encryption available when passing over insecure networks, for example, using either SSL or S/MIME as appropriate for the chosen processing model.

Because payment transactions also typically modify or reveal highly sensitive data such as account balances, it is essential that the identity of the submitting party be authenticated. XMLPay provides two built-in authentication mechanisms:

1) Userids & passwords, and

2) Digitally signed transactions.

However, use of neither authentication method is mandated. Authentication is not required at the XMLPay level because it is often more conveniently implemented at the transport level instead. For example, HTTP userid & passwords, SSL client authentication, or S/MIME signed messages can provide suitable authentication mechanisms for XMLPay data.

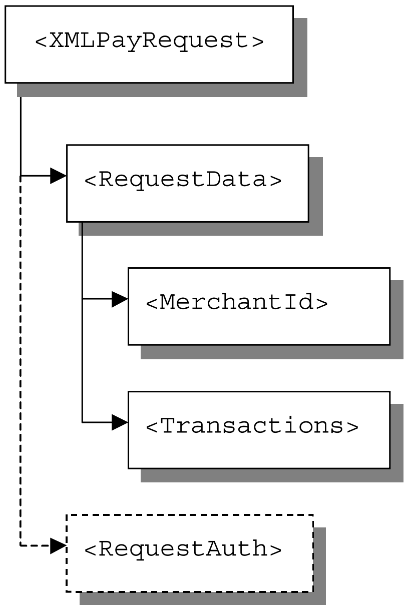
Finally, in some environments there are requirements to authenticate the buyers in e-commerce transactions. XMLPay provides a mechanism to include buyer authentication information, in a variety of formats, in payment transactions. For more information on how this mechanism can be employed, see VeriSign's Authenticated Payment^{TM} white paper.

XMLPay Messaging

The highest-level XMLPay structures represent payment transaction requests, responses, and receipts.

XMLPayRequest

Payment transactions are submitted, one or more at a time, as XMLPayRequest documents. The high-level structure of a request looks like this:



Vendor identifies the merchant of record for the transaction within the target payment processing network. Note that the merchant of record may be different than the submitting party in a delegated processing model.

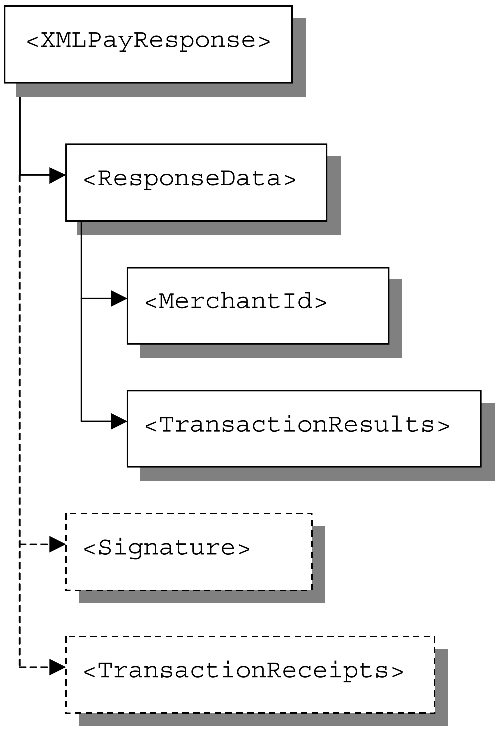
Transactions is the list of payment transactions to be processed.

RequestAuth is an optional structure used to authenticate the submitting party, in the absence of transport level authentication.

See Chapter 3, “XMLPay Syntax” for a detailed description of request documents.

XMLPayResponse

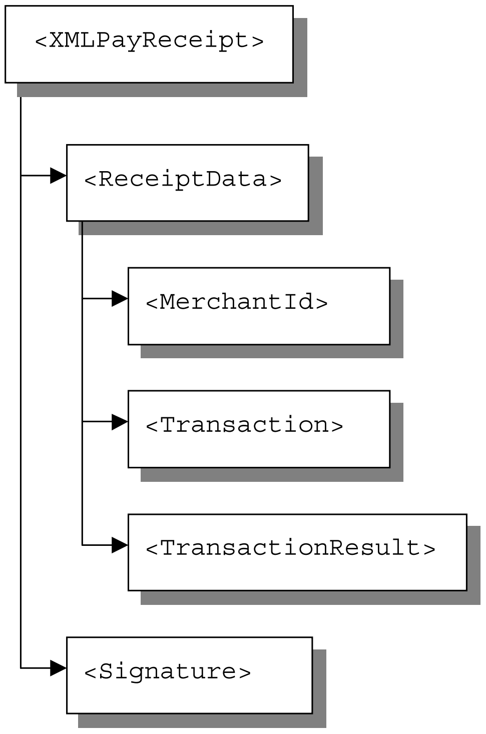
Each XMLPayRequest submission produces a corresponding XMLPayResponse document containing results for each submitted transaction request. The high-level structure of a response looks like this:



See Chapter 3, “XMLPay Syntax” for a detailed description of response documents.

XMLPayReceipt

Receipts provide verifiable records of transactions. Using XMLPayReceipt documents, a payment processing network may optionally provide receipts for each XMLPay transaction processed. The high-level structure of a receipt looks like this:



See Chapter 3, “XMLPay Syntax” for a detailed description of receipt documents.

Related Specifications

Companion specifications include XMLPay: HTTP Transport, which defines the basic HTTP transport mechanism, XMLPay: Registration, which defines an XML schema for the registration of merchant account information with a payment processing network, and XMLPay: Reporting, which defines an XML schema for reports of a merchant's payment processing activity.

XMLPay: HTTP Transport

This companion specification defines a simple HTTP mechanism for submitting XMLPay requests to a payment processing network and receiving back the corresponding XMLPay responses.

See the document *XMLPay: HTTP Transport* for more information.

XMLPay: Registration

In the core XMLPay specification, it is assumed that the entities submitting requests are already registered clients of a payment-processing network. More specifically, it is assumed that they can submit valid payment transactions in that network. It is further assumed that merchants or other requesting parties may have registered agents who are able to execute transactions on their behalf.

Turning on a business for online payment process is a moderately complex process that involves the manual intervention of several parties, including not only the merchant or trading exchange itself but also the merchant's acquiring bank and a payment processor to move money between buyer and seller banks. Both resellers and emerging B2B marketplaces are increasingly demanding more automated access to the process of enabling online merchant payment accounts. In response, XMLPay includes a specification defining data formats and processing infrastructure supporting the automation of merchant registrations. See the document *XMLPay: Registration* for more information.

XMLPay: Reporting

Payment processing applications demand robust reporting functionality not only for the purposes of reconciliation but also fraud protection and other auditing functionality. See the document *XMLPay: Reporting* for more information.

CHAPTER 3

XMLPay Syntax

We now present the syntax for transaction requests, responses, and receipts using a simplified notation (described below). The complete syntax, expressed in W3C XML-schema notation, is included in Appendix A,“XMLPay Schemas”. In Appendix B,“XMLPay DTD” a Document Type Definition representation of the schema is included

Here is an example of the notation used to present the XMLPay syntax:

```
<Example>
        (element)
        (optionalElement)?
        (alternateElement1|alternateElement2)
        (element)+
        (element)*
</Example>
```

The “(element)” notation indicates the occurrence of a (possibly complex) XML element (e.g. <element>...</element>) defined elsewhere. The “?” notation indicates an optional element. The “|” notation separates alternative elements, any one of which is allowed. The “+” notation indicates that one or more occurrences of an element are allowed. The “*” notation similarly indicates that zero or more occurrences of an element are allowed.

Note For specific examples of how to submit XML documents using the Payflow Pro client API, see the Payflow Pro SDK download package itself.

The XMLPayRequest Document

```
<XMLPayRequest Timeout=?>
      <RequestData>
            (Vendor)
            (Partner)
            <Transactions>
                  (Transaction)+
            </Transactions>
      </RequestData>
      (RequestAuth)?
</XMLPayRequest>
```

`Vendor` identifies the merchant of record for the transaction within the target payment processing network. Note that the merchant of record may be different than the submitting party in a delegated processing model.

`Partner` identifies the submitting party.

`Transaction` and **`RequestAuth`** are defined below.

`Timeout` is an optional attribute that, if provided, puts an upper bound on the time that the server will spend processing the request. Multiple transactions submitted within a single request execute independently of one another. Results will be returned for any transactions in a request that complete before a timeout occurs.

Transaction

```
<Transaction Id=? CustRef=?>
      (Authorization|Capture|Sale|Credit|Void|ForceCapture|
       RepeatSale|GetStatus)
</Transaction>
```

`Id` is an optional attribute of a Transaction that can be used to track the transaction through the payment-processing network. The submitting merchant generates this transaction identifier, which should be unique among all transactions submitted by that merchant. Note that Id need not be *globally* unique across merchants, since the payment-processing network will interpret it within the context of the merchant associated with the transaction. If an Id attribute is provided in a transaction, it will be included in the matching TransactionResult in the resultant XMLPayResult. Similarly, CustRef is a merchant generated id identifying a specific customer of this merchant and associating it to this transaction.

Authorization

```
<Authorization>
      <PayData>
            (Invoice)
            (Tender)
      </PayData>
      (PayDataAuth)?
      (ExtData)*
</Authorization>
```

An authorization is used to verify the availability of funds, and reserve them for later capture.

`PayData` specifies the details of the purchase, within **`Invoice`**, as well as the payment **`Tender`** to use.

`PayDataAuth` is an optional element that coveys the payer's authorization, for use in Authenticated Payment.

`ExtData` is an optional element that may carry extended data (outside the syntax of the XMLPay schema).

Capture

```
<Capture>
      (PNRef)
      (Invoice)?
      (ExtData)*
</Capture>
```

A capture is used to transfer the funds secured by a previous authorization transaction, identified by **`PNRef`**, into the merchant's account.

An updated **`Invoice`** may optionally be provided, specifying any changes in the purchase details from the original invoice in the reference authorization.

`ExtData` is an optional element that may carry extended data (outside the syntax of the XMLPay schema).

Sale

```
<Sale>
      <PayData>
            (Invoice)
            (Tender)
      </PayData>
      (PayDataAuth)?
      (ExtData)*
</Sale>
```

A sale is used to verify the availability of and capture funds in one step.

The details of **`PayData, PayDataAuth,`** and **`ExtData`** are the same as for an authorization.

Credit

```
<Credit>
      (PNRef|Tender)
      (Invoice)?
      (ExtData)*
</Credit>
```

A credit is used to reverse a previous sale or capture.

The transaction to reverse is identified by **`PNRef`**. A credit may be run without a PNRef by providing the **`Tender`** for the account to be credited and **`Invoice`** for the amount.

In the case of a partial credit, **`Invoice`** must be provided, specifying detail on the items being returned.

`ExtData` is an optional element that may carry extended data (outside the syntax of the XMLPay schema).

Void

```
<Void>
      (PNRef)
      (ExtData)*
</Void>
```

A void is used to cancel a pending sale, capture, or credit.

The transaction to cancel is identified by **`PNRef`**. If the referenced transaction has already been processed, the void will fail.

ForceCapture

```
<ForceCapture>
      <PayData>
            (Invoice)
            (Tender)
      </PayData>
      (PayDataAuth)?
      (AuthCode)
      (ExtData)*
</ForceCapture>
```

ForceCapture captures funds reserved through an out-of-band authorization (e.g. a voice authorization received over the phone).

`AuthCode` is the authorization code received out-of-band.

The details of **`PayData, PayDataAuth,`** and **`ExtData`** are the same as for an authorization.

RepeatSale

```
<RepeatSale>
      (PNRef)
      (Invoice)?
      (ExtData)*
</RepeatSale>
```

RepeatSale repeats a previous sale transaction.

The previous sale transaction is identified by **`PNRef`**. Any information provided in **`Invoice`** will override the corresponding information in the referenced transaction. In particular, this transaction is allowed to increase or decrease the amount of the original sale.

`ExtData` is an optional element that may carry extended data (outside the syntax of the XMLPay schema).

GetStatus

```
<GetStatus>
       (PNRef)
       (ExtData)*
</GetStatus>
```

GetStatus queries the status of a previous transaction.

The transaction to query is identified by **`PNRef`**.

`ExtData` is an optional element that may carry extended data (outside the syntax of the XMLPay schema).

RequestAuth

```
<RequestAuth>
       (UserPass|Signature)
</RequestAuth>
```

The **`RequestAuth`** element provides authentication of the requestor through either a username and password, using UserPass, or a digital signature, using Signature.

In the case of a digital signature, the W3C XML Signature syntax is used and the signature is executed over the **`RequestData`**.

UserPass

```
<UserPass>
       (User)
       (UserDomain)?
       (Password)
</UserPass>
```

`User` is a string identifier assigned to a user.

UserDomain names a partner or a vendor under whose auspice a transaction is being submitted.

`Password` is the user's password (also a string).

The XMLPayResponse Document

```
<XMLPayResponse>
      <ResponseData>
            (Vendor)
            (Partner)
            <TransactionResults>
                  (TransactionResult)+
            </TransactionResults>
      </ResponseData>
      (Signature)?
      (TransactionReceipts)?
</XMLPayResponse>
```

`Vendor` identifies the merchant of record for the transaction within the payment processing network. Partner identifies the partner who submitted the transaction on behalf of the vendor.

`TransactionResult` is defined below.

`Signature` is an optional signature over **`ResponseData`**, executed by the payment processing gateway using the W3C XML Signature syntax. This signature may be used to provide integrity protection of the response data and/or authentication of the responder, if needed (transport-level security may also be used to provide these protections).

`TransactionReceipts`, an optional list of receipts from the payment processing network, is defined below.

TransactionResult

```
<TransactionResult Id=?>
        (Result)
        (AVSResult)?
        (CVResult)?
        (Message)?
        (PNRef)?
        (AuthCode)?
        (HostCode)?
        (HostURL)?
        (OrigResult)?
        (Status)?
        (ReceiptURL)?
        (ExtData)*
</TransactionResult>
```

`Result` is a number that indicates outcome of the transaction (see Appendix C,“Transaction Results”). **`AVSResult`** provides the results of the AVS check, if appropriate. **`CVResult`** provides the results of the CV check, if appropriate. Possible values are: “match”, “no match”, “service not available”, or “service not requested”. **`Message`** is a descriptive message describing Result.

`PNRef` is an identifier assigned to the transaction by the payment processing network.

`AuthCode` is the authorization code for the transaction provided by the bank, if any. **`HostCode`** is the result code returned by the payment processor, if any. Whereas **`Result`** provides a normalized view the transaction status, **`HostCode`** passes through the back-end processor status unmodified.

`HostURL` is a URL returned by the payment processor, if any, to use in referring to the transaction; and **`ReceiptURL`** is a URL returned by the payment processor, if any, to use in referring to a receipt for the transaction.

`OrigResult` and **`Status`** give the original result and current status respectively for a transaction queried with GetStatus.

`ExtData` is an optional element that may carry extended data (outside the syntax of the XMLPay schema).

`Id` is the identifier assigned to the transaction by the merchant, if one was provided in the transaction request.

AVSResult

```
<AVSResult>
       (StreetMatch)
       (ZipMatch)
</AVSResult>
```

`StreetMatch` indicates whether or not the billing street address matched the bank's records. Possible values are: "match", "no match", "service not available", or "service not requested".

`ZipMatch` indicates whether or not the billing zip matched the bank's records. Possible values are: "match", "no match", "service not available", or "service not requested".

The XMLPayReceipt Document

```
<XMLPayReceipt>
       <ReceiptData>
              (Vendor)
              (Partner)
              (Transaction)
              (TransactionResult)
       </ReceiptData>
       (Signature)
</XMLPayReceipt>
```

`Vendor` identifies the merchant of record for the transaction within the payment processing network. Partner identifier the partner of record within the payment processing network.

`Transaction` describes a transaction request and TransactionResult describes the results of processing that transaction.

`Signature` is a digital signature over ReceiptData, executed by the payment processing gateway using the W3C XML Signature syntax. This signature, which includes a timestamp applied by the payment gateway, ensures that an XMLPayReceipt is a verifiable record of a transaction.

Core Structures

PayData

```
<PayData>
      (Invoice)
      (Tender)
</PayData>
```

`Invoice` describes the details of a purchase and is defined below.

`Tender` describes the details of a payment instrument and is defined below.

PayDataAuth

```
<PayDataAuth>
      (PKCS7Signature|Signature)
</PayDataAuth>
```

The **`PayDataAuth`** element provides authentication of the payer for an associated PayData, using either a PKCS-7 format or a W3C XML Signature format digital signature.

Invoice

```
<Invoice>
      (InvNum)?
      (Date)?
      <BillFrom>
            (Name)?
            (Address)?
            (EMail)?
            (Phone)?
            (Fax)?
            (URL)?
      </BillFrom>
      <BillTo>
            (Name)?
            (Address)?
            (EMail)?
            (Phone)?
            (Fax)?
            (CustCode)?
            (PONum)?
            (TaxExempt)?
      </BillTo>
      <ShipFrom>
            (Name)?
            (Address)?
            (EMail)?
            (Phone)?
            (Fax)?
      </ShipFrom>
      <ShipTo>
            (Name)?
            (Address)?
            (EMail)?
            (Phone)?
            (Fax)?
      </ShipTo>
      (Description)*
      (Items)?
      (DiscountAmt)?
      (ShippingAmt)?
      (DutyAmt)?
      (TaxAmt)?
      (NationalTaxIncl)?
      (TotalAmt)?
      (Comment)?
      (ExtData)*
</Invoice>
```

InvNum is the invoice number.

Date is the invoice date:

YYYYMMDD	ISO 8601

BillFrom Name, Address, EMail, Phone, Fax, and **URL** provide information about the biller.

BillTo Name, Address, EMail, Phone, and **Fax** provide information about the buyer.

BillTo CustCode is a code, chosen by the merchant, that may be included in the invoice to identify the customer (buyer).

BillTo PONum is the buyer's purchase order number.

BillTo TaxExempt indicates that the buyer is a tax exempt entity.

`ShipFrom` and **`ShipTo`** provide information about the shipping addresses, if different from BillFrom and BillTo respectively.

`Description` provides a summary description of the purchase. This field, in the case of an Amex purchase card, can contain up to four separate descriptions of 40 characters each.

`Items`, defined below, provides a full line item breakdown of the purchase.

`DiscountAmt` is the discount to be applied to the item subtotal.

`ShippingAmt` is the total of all shipping and handling charges.

`DutyAmt` is the total duty amount of this invoice, if applicable.

`TaxAmt` is the total of all taxes.

`NationalTaxIncl` is a boolean which when true, indicates that the national tax in included in the TaxAmt.

`TotalAmt` is the grand total (item subtotal - DiscountAmt + ShippingAmt + TaxAmt).

`Comment` is a free form comment about the purchase.

`ExtData` is an optional element that may carry extended data (outside the syntax of the XMLPay schema).

Items

```
<Items>
        (Item)+
</Item>
```

Items is a list of line item detail records, **`Item`** which is defined below.

Item

```
<Item Number=>
       (SKU)?
       (UPC)?
       (Description)?
       (Quantity)?
       (UnitOfMeasurement)?
       (UnitPrice)?
       (ExtAmt)?
       (DiscountAmt)?
       (ShippingAmt)?
       (TaxAmt)?
       (TotalAmt)?
       (ExtData)*
</Item>
```

`Number` is the line number for the item in the invoice.

`SKU` is the merchant's product code for the item (stock keeping unit).

`UPC` is the item's universal product code.

`Description` is the item's description.

`Quantity` is the number of units of this item. **`UnitOfMeasurement`** provides the units for Quantity (ISO 31).

`UnitPrice` is the cost of each unit.

`ExtAmt` is the extended amount, `Quantity` * `UnitPrice`.

`DiscountAmt` is the discount to be applied to this line item.

`ShippingAmt` is the shipping and handling cost for this line item.

`TaxAmt` is the total of all taxes for this line item.

`TotalAmt` is the grand total for this line item: `ExtAmt` - `DiscountAmt` + `ShippingAmt` + `TaxAmt`.

`ExtData` is an optional element that may carry extended data (outside the syntax of the XMLPay schema).

Address

```
<Address>
        (Street)?
        (City)?
        (State)?
        (Zip)?
        (Country)?
</Address>
```

`Street` is the street address, including number.

`City` is the city name.

`State` is the state or province. For US addresses, two character state codes should be used.

`Zip` is the postal code.

`Country` is the country code (ISO 3166). Default is US.

Tender

```
<Tender>
        (ACH|Card|Check)
</Tender>
```

ACH

```
<ACH>
        (AcctType)
        (AcctNum)
        (ABA)
        (Prenote)?
        (ExtData)*
</ACH>
```

ACH (Automated Clearing House) tender detail.

`AcctType` indicates the type of the bank account: checking or savings.

`AcctNum` is account number.

`ABA` is the bank routing number.

`Prenote` is a boolean ('Y'/'N'). If 'Y', it indicates that the purpose of this transaction is not to move money, but to establish authorization for future transactions to be submitted on a recurring basis.

`ExtData` is an optional element that may carry extended data (outside the syntax of the XMLPay schema).

Card

```
<Card>
      (CardType)
      (CardNum)
      (ExpDate)
      (CVNum)?
      (MagData)?
      (NameOnCard)?
      (ExtData)*
</Card>
```

Retail Credit/Debit and Corporate Purchase Card tender detail.

`CardType` indicates the type of the card: Amex, DinersClub, Discover, JCB, MasterCard, or Visa.

`CardNum` is account number.

`ExpDate` is card expiration date:

YYYYMM	ISO 8601

`CVNum` is card verification number (typically printed on the back of the card, but not embossed on the front).

`MagData` is the data located on the magnetic strip of a credit card.

`NameOnCard` is card holder's name as printed on the card.

`ExtData` is an optional element that may carry extended data (outside the syntax of the XMLPay schema).

Check

```
<Check>
        (CheckType)
        (CheckNum)
        (MICR)
        (DL)?
        (SS)?
        (DOB)?
        (ExtData)*
</Check>
```

Check tender detail.

`CheckType` indicates the type of the check: corporate, personal, or government.

`CheckNum` is account holder's next unused check number.

`MICR` (Magnetic Ink Check Reader) is the entire line of numbers at the bottom of the check. It includes the transit number, account number, and check number.

`DL` is the account holder's driver's license number:

XxNnnnnnnn	Xx is the state code; Nnnnnnnn is the number

`SS` is the account holder's social security number.

`DOB` is the account holder's date of birth:

YYYYMMDD	ISO 8601

`ExtData` is an optional element that may carry extended data (outside the syntax of the XMLPay schema).

ExtData

```
<ExtData Name= Value= />
```

The **`Name`** attribute is the name of the extended data element.

The **`Value`** attribute is the value of the extended data element.

VeriSign CHAPTER 4

XMLPay Transaction Profiles

The following tables document the transactions supported for each tender, along with the data elements used for each of those transactions.

ACH Transactions

ACH supports the following transactions: Sale, Credit, and Void.

The following data are used to process ACH transactions (required elements for sale are in **bold**):

- **Invoice.TotalAmt**
- **ACH.AcctType**
- **ACH.AcctNum**
- **ACH.ABA**
- ACH.Prenote
- BillTo.Name

Card Transactions

Retail credit and debit cards and corporate purchase cards support the following transactions: Sale, Authorization, Delay Capture, Credit, and Void.

The following data are used to process card transactions (required elements for sale/authorization are in **bold**):

- **Invoice.TotalAmt**
- **Card.CardNum**
- **Card.ExpDate**
- Card.NameOnCard (Note: Defaults to BillTo.Name)
- BillTo.Street, ZIP (Note: Required for AVS)
- CardInfo.CVNum

Purchase Card Level I

Level I purchase cards use the following additional data:

- BillTo.PONum
- Invoice.TaxAmt

Purchase Card Level II

Level II purchase cards use the following additional data:

- BillTo.TaxExempt
- ShipTo.ZIP

Purchase Card Level III

Level III purchase cards use the following additional data:

- Invoice.Date
- BillTo.CustCode
- ShipTo.Country
- ShipFrom.ZIP
- Invoice.ShippingAmt
- Invoice.DiscountAmt
- Item.SKU, UPC, Description, Quantity, UnitOfMeasurement, UnitPrice, ExtAmt, DiscountAmt, TaxAmt, TotalAmt, DutyAmt, NationalTaxIncl

Note Minimum requirements for describing an item are Description, Quantity, and UnitPrice.
ExtAmt = Quantity * UnitPrice.
TotalAmt = ExtAmt – DiscountAmt + TaxAmt

Check Transactions

Checks support the following transaction: Sale. The following data are used to process check transactions (required elements are in **bold**):

- **Invoice.TotalAmt**
- **Check.CheckType**
- **Check.CheckNum**
- **Check.MICR**
- **BillTo.Name**
- **BillTo.Address**
- **BillTo.EMail**
- BillTo.Phone
- **BillTo.DL** (Note: DL is optional if SS is provided)
- **BillTo.SS** (Note: SS is optional if DL is provided)
- BillTo.DOB
- Invoice.InvNum

Notes:

CHAPTER 5

XMLPay Examples

Following are several examples of XMLPay documents.

Note For specific examples of how to submit XML documents using the Payflow Pro client API, see the Payflow Pro SDK download package itself.

Authorization and Response

```
<XMLPayRequest Timeout="40">
    <RequestData>
    <Vendor>M1234</Vendor>
    <Partner>Ariba</Partner>
    <Transactions>
        <Transaction Id="004003">
            <Authorization>
                <PayData>
                    <Invoice>
                        <InvNum>001</InvNum>
                        <Date>2000-02-03</Date>
                        <BillTo>
                            <CustomerId>C100</CustomerId>
                            <Name>John Smith</Name>
                            <Address>
                                <Street>123 4th street</Street>
                                <City>San Jose</City>
                                <State>CA</State>
                                <Zip>95032</Zip>
                                <Country>USA</Country>
                            </Address>
                            <EMail>jsmith@signio.com</EMail>
                        </BillTo>
                        <ShipFrom>
            <Address>
                <Street>5 Warehouse Street</Street>
                <City>San Jose</City>
                <State>CA</State>
                <Zip>95030</Zip>
            </Address>
        </ShipFrom>
        <ShipTo>
            <Address>
                <Street>123 4th street</Street>
                <City>San Jose</City>
                <State>CA</State>
                <Zip>95032</Zip>
                <Country>USA</Country>
            </Address>
        </ShipTo>
        <Items>
            <Item Number="1">
                <SKU>pc012</SKU>
                <UPC>upc811818</UPC>
                <Description>Electric Toothbrush</Description>
                <Quantity>1</Quantity>
                <UnitPrice>114.00</UnitPrice>
                <DiscountAmt>14.00</DiscountAmt>
                <TaxAmt>8.25</TaxAmt>
                <TotalAmt>118.25</TotalAmt>
            </Item>
            <Item Number="2">
                <SKU>pc013</SKU>
                <UPC>upc811819</UPC>
                <Description>Toothpaste</Description>
                <Quantity>2</Quantity>
                <UnitPrice>2.30</UnitPrice>
                <ExtAmt>4.60</ExtAmt>
                <TaxAmt>0.37</TaxAmt>
                <TotalAmt>6.97</TotalAmt>
            </Item>
        </Items>
        <DiscountAmt>14.00</DiscountAmt>
        <ShippingAmt>12.00</ShippingAmt>
        <TaxAmt>8.62</TaxAmt>
        <TotalAmt>125.22</TotalAmt>
    </Invoice>
    <Tender>
        <Card>
            <CardType>MasterCard</CardType>
```

```
                            <CardNum>5499576040500004</CardNum>
                            <ExpDate>200011</ExpDate>
                            <CVNum>828</CVNum>
                            <NameOnCard>Signio</NameOnCard>
                        </Card>
                    </Tender>
                </PayData>
            </Authorization>
        </Transaction>
    </Transactions>
    </RequestData>
</XMLPayRequest>


<XMLPayResponse>
    <ResponseData>
    <Vendor>M1234</Vendor>
    <Partner>Ariba</Partner>
    <TransactionResults>
        <TransactionResult Id="004003">
            <Result>0</Result>
            <AVSResult>
                <StreetMatch>match</StreetMatch>
                <ZipMatch>match</ZipMatch>
            </AVSResult>
            <CVResult>match</CVResult>
            <Message>Authorization Approved</Message>
            <PNRef>PN123412345</PNRef>
            <AuthCode>12345678</AuthCode>
            <HostCode>0</HostCode>
        </TransactionResult>
    </TransactionResults>
    </ResponseData>
</XMLPayResponse>
```

Multiple Captures

```
<XMLPayRequest>
   <RequestData>
   <Vendor>M1234</Vendor>
   <Partner>Ariba</Partner>
   <Transactions>
      <Transaction Id="004004">
         <Capture>
            <PNRef>PN123412345</PNRef>
         </Capture>
      </Transaction>
      <Transaction Id="004005">
         <Capture>
            <PNRef>PN123412346</PNRef>
            <Invoice>
               <TotalAmt>75.00</TotalAmt>
            </Invoice>
         </Capture>
      </Transaction>
   </Transactions>
   </RequestData>
</XMLPayRequest>
```

Sale

```
<XMLPayRequest Timeout="40">
   <RequestData>
   <Vendor>M1234</Vendor>
   <Partner>Ariba</Partner>
   <Transactions>
      <Transaction Id="004006">
         <Sale>
            <PayData>
               <Invoice>
                  <InvNum>002</InvNum>
                  <Date>2000-02-03</Date>
                  <BillTo>
                     <CustomerId>C101</CustomerId>
                     <Name>Tooth Boutique</Name>
                     <Address>
                        <Street>123 4th street</Street>
                        <City>San Jose</City>
                        <State>CA</State>
                        <Zip>95032</Zip>
                        <Country>USA</Country>
                     </Address>
                     <EMail>jsmith@signio.com</EMail>
                  </BillTo>
                  <ShipFrom>
                     <Address>
                        <Street>5 Warehouse Street</Street>
                        <City>San Jose</City>
                        <State>CA</State>
                        <Zip>95030</Zip>
                     </Address>
                  </ShipFrom>
                  <ShipTo>
                     <Address>
                        <Street>123 4th street</Street>
                        <City>San Jose</City>
                        <State>CA</State>
                        <Zip>95032</Zip>
                        <Country>USA</Country>
                     </Address>
                  </ShipTo>
                  <Items>
                     <Item Number="1">
                        <SKU>pc012</SKU>
                        <UPC>upc811818</UPC>
                        <Description>Electric Toothbrush</Description>
                        <Quantity>100</Quantity>
                        <UnitPrice>50.00</UnitPrice>
                        <ExtAmt>5000.00</ExtAmt>
                        <TotalAmt>5100.00</TotalAmt>
                     </Item>
                  </Items>
                  <ShippingAmt>100.00</ShippingAmt>
                  <TotalAmt>5100.00</TotalAmt>
               </Invoice>
               <Tender>
                  <ACH>
                     <AcctType>Checking</AcctType>
                     <AcctNum>10012340505</AcctNum>
                     <ABA>1004</ABA>
                  </ACH>
               </Tender>
            </PayData>
         </Sale>
      </Transaction>
   </Transactions>
   </RequestData>
</XMLPayRequest>
```

APPENDIX A

XMLPay Schemas

XMLPay Schema

```
<?xml version ="1.0"?>
<schema targetNamespace = "http://www.verisign.com/XMLPay"
    xmlns = "http://www.w3.org/1999/XMLSchema">

<include schemaLocation = "XMLPayTypes.xsd"/>

<element name = "XMLPayRequest">
   <complexType content = "elementOnly">
      <sequence>
         <element ref = "RequestData"/>
         <element ref = "RequestAuth"/>
      </sequence>
      <attribute name = "version" type = "string" />
      <attribute name = "Timeout" type = "int" />
      <attribute name = "UniqueTransactionIdentifier" type = "string" />
   </complexType>
</element>

<element name = "RequestData">
   <complexType content = "elementOnly">
      <sequence>
         <element name = "Vendor" type = "UserIdType"/>
         <element name = "Partner" type = "UserIdType"/>
         <element name = "Transactions">
            <complexType content = "elementOnly">
               <element ref = "Transaction" minOccurs = "1" maxOccurs =
"unbounded" />
            </complexType>
         </element>
      </sequence>
   </complexType>
</element>

<element name = "RequestAuth">
   <complexType content = "elementOnly">
      <choice>
         <element ref = "UserPass"/>
         <element ref = "Signature"/>
      </choice>
   </complexType>
</element>

<element name = "XMLPayResponse">
   <complexType content = "elementOnly">
      <sequence>
         <element ref = "ResponseData"/>
         <element ref = "Signature" minOccurs = "0" maxOccurs = "1"/>
         <element name = "TransactionReceipts" minOccurs = "0" maxOccurs =
"1">
            <complexType content = "elementOnly">
               <element ref = "XMLPayReceipt" minOccurs = "1" maxOccurs =
"unbounded"/>
            </complexType>
         </element>
      </sequence>
      <attribute name = "version" type = "string" />
      <attribute name = "UniqueTransactionIdentifier" type = "string" />
   </complexType>
</element>

<element name = "ResponseData">
   <complexType content = "elementOnly">
      <sequence>
         <element name = "Vendor" type = "UserIdType"/>
         <element name = "Partner" type = "UserIdType"/>
         <element name = "TransactionResults">
            <complexType content = "elementOnly">
               <element ref = "TransactionResult" minOccurs = "1" maxOccurs
= "unbounded"/>
            </complexType>
         </element>
      </sequence>
   </complexType>
</element>
```

```
<element name = "XMLPayReceipt">
    <complexType content = "elementOnly">
        <sequence>
            <element ref = "ReceiptData" />
            <element ref = "Signature" minOccurs = "0" maxOccurs = "1"/>
        </sequence>
    </complexType>
</element>

<element name = "ReceiptData">
    <complexType content = "elementOnly">
        <sequence>
            <element name = "Vendor" type = "UserIdType"/>
            <element name = "Partner" type = "UserIdType"/>
            <element ref = "Transaction"/>
            <element ref = "TransactionResult"/>
        </sequence>
    </complexType>
</element>


<element name = "Transaction">
    <complexType content = "elementOnly">
        <sequence>
            <choice>
                <element ref = "Authorization"/>
                <element ref = "Capture"/>
                <element ref = "Sale"/>
                <element ref = "Credit"/>
                <element ref = "Void"/>
                <element ref = "ForceCapture"/>
                <element ref = "RepeatSale"/>
                <element ref = "GetStatus"/>
            </choice>
        </sequence>
        <attribute name = "Id" type = "ID"/>
        <attribute name = "CustRef" type = "ReferenceIdType"/>
    </complexType>
</element>

<element name = "Authorization">
    <complexType content = "elementOnly">
        <sequence>
            <element ref = "PayData"/>
            <element ref = "PayDataAuth" minOccurs = "0" maxOccurs = "1"/>
            <element ref = "ExtData" minOccurs = "0" maxOccurs = "unbounded"/>
```

```
      </sequence>
   </complexType>
</element>

<element name = "Capture">
   <complexType content = "elementOnly">
      <sequence>
         <element name = "PNRef" type = "PNRefType" />
         <element ref = "Invoice" minOccurs="0" maxOccurs="1"/>
         <element ref = "ExtData" minOccurs = "0" maxOccurs = "unbounded"/>
      </sequence>
   </complexType>
</element>

<element name = "Sale">
   <complexType content = "elementOnly">
      <sequence>
         <element ref = "PayData"/>
         <element ref = "PayDataAuth" minOccurs = "0" maxOccurs = "1"/>
         <element ref = "ExtData" minOccurs = "0" maxOccurs = "unbounded"/>
      </sequence>
   </complexType>
</element>

<element name = "Credit">
   <complexType content = "elementOnly">
      <sequence>
         <choice>
            <element name = "PNRef" type = "PNRefType"/>
            <element ref = "Tender"/>
         </choice>
         <element ref = "Invoice"  minOccurs="0" maxOccurs="1"/>
         <element ref = "ExtData" minOccurs = "0" maxOccurs = "unbounded"/>
      </sequence>
   </complexType>
</element>

<element name = "Void">
   <complexType content = "elementOnly">
      <sequence>
         <element name = "PNRef" type = "PNRefType"/>
         <element ref = "ExtData" minOccurs = "0" maxOccurs = "unbounded"/>
      </sequence>
   </complexType>
</element>
```

```
<element name = "ForceCapture">
    <complexType content = "elementOnly">
        <sequence>
            <element ref = "PayData"/>
            <element ref = "PayDataAuth" minOccurs = "0" maxOccurs = "1"/>
            <element name = "AuthCode" type = "AuthCodeType"/>
            <element ref = "ExtData" minOccurs = "0" maxOccurs = "unbounded"/>
        </sequence>
    </complexType>
</element>

<element name = "RepeatSale">
    <complexType content = "elementOnly">
        <sequence>
            <element name = "PNRef" type = "PNRefType"/>
            <element ref = "Invoice" minOccurs="0" maxOccurs="1"/>
            <element ref = "ExtData" minOccurs = "0" maxOccurs = "unbounded"/>
        </sequence>
    </complexType>
</element>

<element name = "GetStatus">
    <complexType content = "elementOnly">
       <sequence>
            <element name = "PNRef" type = "PNRefType"/>
            <element ref = "ExtData" minOccurs = "0" maxOccurs = "unbounded"/>
       </sequence>
    </complexType>
</element>


<element name = "TransactionResult">
    <complexType content = "elementOnly">
        <sequence>
            <element name = "Result" type = "int"/>
            <element ref = "AVSResult" minOccurs="0" maxOccurs="1"/>
            <element name = "CVResult" type = "MatchResultEnum" minOccurs="0"
maxOccurs="1"/>
            <element name = "Message" type = "string" minOccurs="0"
maxOccurs="1"/>
            <element name = "PNRef" type = "PNRefType" minOccurs="0"
maxOccurs="1"/>
            <element name = "AuthCode" type = "AuthCodeType" minOccurs="0"
maxOccurs="1"/>
            <element name = "HostCode" type = "HostCodeType" minOccurs="0"
maxOccurs="1"/>
```

```
            <element name = "HostURL" type = "uriReference" minOccurs="0"
maxOccurs="1"/>
            <element name = "OrigResult" type = "byte" minOccurs="0"
maxOccurs="1"/>
            <element name = "TrStatus" type = "StatusType" minOccurs="0"
maxOccurs="1"/>
            <element name = "ReceiptURL" type = "uriReference" minOccurs="0"
maxOccurs="1"/>
            <element ref = "ExtData" minOccurs = "0" maxOccurs = "unbounded"/>
        </sequence>
        <attribute name = "Id" type = "ID" />
    </complexType>
</element>

<element name = "AVSResult">
    <complexType content = "elementOnly">
        <sequence>
            <element name = "StreetMatch" type = "MatchResultEnum"/>
            <element name = "ZipMatch" type = "MatchResultEnum"/>
        </sequence>
    </complexType>
</element>

<simpleType name = "AuthCodeType" base = "string">
    <maxLength value = "6"/>
</simpleType>

<simpleType name = "HostCodeType" base = "string">
    <maxLength value = "6"/>
</simpleType>

<simpleType name = "StatusType" base = "string">
    <maxLength value = "3"/>
</simpleType>

<simpleType name = "MatchResultEnum" base = "string">
    <enumeration value = "Match"/>
    <enumeration value = "No Match"/>
    <enumeration value = "Service Not Available"/>
    <enumeration value = "Service Not Requested"/>
</simpleType>

<element name = "UserPass">
    <complexType content = "elementOnly">
        <sequence>
            <element name = "User" type = "UserIdType"/>
```

```
          <element name = "UserDomain" type = "UserIdType" minOccurs="0"
maxOccurs="1"/>
          <element name = "Password" type = "PasswordType"/>
      </sequence>
   </complexType>
</element>

</schema>
```

XMLPay Types Schema

```
<?xml version ="1.0"?>
<schema targetNamespace = "http://www.verisign.com/XMLPay"
    xmlns = "http://www.w3.org/1999/XMLSchema">

<!-- stub out xmldsig
<import schemaLocation = "xmldsig-core-schema.xsd"/>
-->
<element name = "Signature">
   <complexType content = "elementOnly">
      <any/>
   </complexType>
</element>


<element name = "PayData">
   <complexType content = "elementOnly">
      <sequence>
         <element ref = "Invoice"/>
         <element ref = "Tender"/>
      </sequence>
   </complexType>
</element>

<element name = "PayDataAuth">
   <complexType content = "elementOnly">
      <choice>
         <element name = "PKCS7Signature" type = "Base64BinaryData"/>
         <element ref = "Signature"/>
      </choice>
   </complexType>
</element>


<element name = "Invoice">
   <complexType content = "elementOnly">
      <sequence>
         <element name = "InvNum" type = "InvNumType"  minOccurs = "0"
maxOccurs = "1"/>
         <element name = "Date" type = "date" minOccurs = "0" maxOccurs =
"1"/>
         <element name = "BillFrom" minOccurs = "0" maxOccurs = "1">
            <complexType context = "elementOnly">
```

```
                <element name = "Name" type = "NameType" minOccurs = "0"
maxOccurs = "1"/>
                <element ref = "Address" minOccurs = "0" maxOccurs = "1"/>
                <element name = "EMail" type = "EMailType" minOccurs = "0"
maxOccurs = "1"/>
                <element name = "Phone" type = "PhoneNumType" minOccurs = "0"
maxOccurs = "1"/>
                <element name = "Fax" type = "PhoneNumType" minOccurs = "0"
maxOccurs = "1"/>
                <element name = "URL" type = "uriReference" minOccurs = "0"
maxOccurs = "1"/>
             </complexType>
          </element>

          <element name = "BillTo" minOccurs = "0" maxOccurs = "1">
             <complexType content = "elementOnly">
                <element name = "CustomerId" type = "UserIdType" minOccurs =
"0" maxOccurs = "1"/>
                <element name = "Name" type = "NameType" minOccurs = "0"
maxOccurs = "1"/>
                <element ref = "Address"  minOccurs = "0" maxOccurs = "1"/>
                <element name = "EMail" type = "EMailType" minOccurs = "0"
maxOccurs = "1"/>
                <element name = "Phone" type = "PhoneNumType" minOccurs = "0"
maxOccurs = "1"/>
                <element name = "Fax" type = "PhoneNumType" minOccurs = "0"
maxOccurs = "1"/>
                <element name = "CustCode" type = "CustCodeType" minOccurs =
"0" maxOccurs = "1"/>
                <element name = "PONum" type = "PONumType" minOccurs = "0"
maxOccurs = "1"/>
                <element name = "TaxExempt" type = "boolean" minOccurs = "0"
maxOccurs = "1"/>
            </complexType>
          </element>

          <element name = "ShipFrom" minOccurs = "0" maxOccurs = "1">
            <complexType content = "elementOnly">
                <element name = "Name" type = "NameType" minOccurs = "0"
maxOccurs = "1"/>
                <element ref = "Address"/>
                <element name = "EMail" type = "EMailType" minOccurs = "0"
maxOccurs = "1"/>
                <element name = "Phone" type = "PhoneNumType" minOccurs = "0"
maxOccurs = "1"/>
```

```
                <element name = "Fax" type = "PhoneNumType" minOccurs = "0"
maxOccurs = "1"/>
            </complexType>
          </element>

          <element name = "ShipTo" minOccurs = "0" maxOccurs = "1">
            <complexType content = "elementOnly">
                <element name = "Name" type = "NameType" minOccurs = "0"
maxOccurs = "1"/>
                <element ref = "Address"/>
                <element name = "EMail" type = "EMailType" minOccurs = "0"
maxOccurs = "1"/>
                <element name = "Phone" type = "PhoneNumType" minOccurs = "0"
maxOccurs = "1"/>
                <element name = "Fax" type = "PhoneNumType" minOccurs = "0"
maxOccurs = "1"/>
            </complexType>
          </element>

          <element name = "Description" type = "DescType" minOccurs = "0"
maxOccurs = "unbounded"/>
          <element name = "Items" minOccurs = "0" maxOccurs = "1">
             <complexType content = "elementOnly">
                <element ref = "Item" minOccurs = "0" maxOccurs = "unbounded"
/>
             </complexType>
          </element>
          <element name = "DiscountAmt" type = "CurrencyAmount" minOccurs =
"0" maxOccurs = "1"/>
          <element name = "ShippingAmt" type = "CurrencyAmount" minOccurs =
"0" maxOccurs = "1"/>
          <element name = "DutyAmt" type = "CurrencyAmount" minOccurs = "0"
maxOccurs = "1"/>

          <element name = "TaxAmt" type = "CurrencyAmount" minOccurs = "0"
maxOccurs = "1"/>
          <element name = "NationalTaxIncl" type = "boolean" minOccurs = "0"
maxOccurs = "1"/>
          <element name = "TotalAmt"  type = "CurrencyAmount"/>

          <element name = "Comment" type = "CommentType" minOccurs = "0"
maxOccurs = "1"/>
          <element ref = "ExtData" minOccurs = "0" maxOccurs = "unbounded"/>
      </sequence>
   </complexType>
</element>
```

```
<simpleType name = "InvNumType" base = "string">
   <maxLength value = "20"/>
</simpleType>

<simpleType name = "PONumType" base = "string">
   <maxLength value = "25"/>
</simpleType>

<simpleType name = "DescType" base = "string">
   <maxLength value = "160"/>
</simpleType>

<simpleType name = "CommentType" base = "string">
   <maxLength value = "255"/>
</simpleType>


<element name = "Item">
   <complexType content = "elementOnly">
      <sequence>
         <element name = "SKU" type = "SKUType" minOccurs = "0" maxOccurs =
"1"/>
         <element name = "UPC" type = "UPCType" minOccurs = "0" maxOccurs =
"1" />
         <element name = "Description" type = "CommentType" minOccurs = "0"
maxOccurs = "1"/>
         <element name = "Quantity" type = "int" minOccurs = "0" maxOccurs =
"1"/>
         <element name = "UnitOfMeasurement" type = "UnitOfMeasurementType"
minOccurs = "0" maxOccurs = "1"/>
         <element name = "UnitPrice"type = "CurrencyAmount" minOccurs = "0"
maxOccurs = "1"/>
         <element name = "ExtAmt" type = "CurrencyAmount" minOccurs = "0"
maxOccurs = "1"/>
         <element name = "DiscountAmt" type = "CurrencyAmount" minOccurs =
"0" maxOccurs = "1"/>
         <element name = "TaxAmt" type = "CurrencyAmount" minOccurs = "0"
maxOccurs = "1"/>
         <element name = "TotalAmt" type = "CurrencyAmount" minOccurs = "0"
maxOccurs = "1"/>
         <element ref = "ExtData" minOccurs = "0" maxOccurs = "unbounded"/>
      </sequence>
      <attribute name = "Number" type = "int" use = "required"/>
   </complexType>
</element>
```

```
<simpleType name = "UnitOfMeasurementType" base = "string">
   <maxLength value = "12"/>
</simpleType>

<simpleType name = "SKUType" base = "string">
   <maxLength value = "18"/>
</simpleType>

<simpleType name = "UPCType" base = "string">
   <maxLength value = "18"/>
</simpleType>


<element name = "Address">
   <complexType content = "elementOnly">
      <sequence>
         <element name = "Street"  type = "NameType" minOccurs = "0" maxOccurs
= "1"/>
         <element name = "City" type = "NameType" minOccurs = "0" maxOccurs =
"1"/>
         <element name = "State" type = "StateType" minOccurs = "0" maxOccurs
= "1"/>
         <element name = "Zip" type = "ZipType" minOccurs = "0" maxOccurs =
"1"/>
         <element name = "Country" type = "CountryCode" minOccurs = "0"
maxOccurs = "1"/>
      </sequence>
   </complexType>
</element>

<simpleType name = "NameType" base = "string">
   <maxLength value = "30"/>
</simpleType>

<simpleType name = "StateType" base = "string">
   <maxLength value = "2"/>
</simpleType>

<simpleType name = "ZipType" base = "string">
   <maxLength value = "10"/>
</simpleType>

<simpleType name = "CountryCode" base = "string">
   <maxLength value = "3"/>
</simpleType>
```

```
<simpleType name = "EMailType" base = "string">
   <maxLength value = "40"/>
</simpleType>


<simpleType name = "PhoneNumType" base = "string">
   <maxLength value = "20"/>
</simpleType>


<complexType name = "CurrencyAmount" base = "Decimal9_2">
   <attribute name = "Currency" type = "CurrencyCode"/>
</complexType>

<simpleType name = "Decimal9_2" base = "string">
   <maxLength value = "12"/>
</simpleType>

<simpleType name = "CurrencyCode" base = "string">
   <maxLength value = "3"/>
</simpleType>


<element name = "Tender">
   <complexType content = "elementOnly">
      <choice>
         <element ref = "ACH"/>
         <element ref = "Card"/>
         <element ref = "Check"/>
      </choice>
   </complexType>
</element>

<simpleType name = "TenderTypeEnum" base = "string">
   <enumeration value = "ACH"/>
   <enumeration value = "Card"/>
   <enumeration value = "Check"/>
</simpleType>

<element name = "ACH">
   <complexType content = "elementOnly">
      <sequence>
         <element name = "AcctType" type = "AcctTypeEnum"/>
         <element name = "AcctNum" type = "AcctNumType"/>
```

```
            <element name = "ABA" type = "ABAType"/>
            <element name = "Prenote" type = "boolean" minOccurs = "0" maxOccurs
= "1"/>
            <element ref = "ExtData" minOccurs = "0" maxOccurs = "unbounded"/>
        </sequence>
    </complexType>
</element>

<simpleType name = "AcctTypeEnum" base = "string">
    <enumeration value = "Saving"/>
    <enumeration value = "Checking"/>
</simpleType>

<simpleType name = "AcctNumType" base = "string">
    <maxLength value = "19"/>
</simpleType>

<simpleType name = "ABAType" base = "string">
    <maxLength value = "9"/>
</simpleType>


<element name = "Card">
    <complexType content = "elementOnly">
        <sequence>
            <element name = "CardType" type = "CardTypeEnum"  minOccurs = "0"
maxOccurs = "1"/>
            <element name = "CardNum" type = "CardNumType"/>
            <element name = "ExpDate" type =  "ExpDateType"/>
            <element name = "CVNum" type = "CVType" minOccurs = "0" maxOccurs =
"1"/>
            <element name = "MagData" type = "MagDataType" minOccurs = "0"
maxOccurs = "1"/>
            <element name = "NameOnCard" type = "NameType" minOccurs = "0"
maxOccurs = "1"/>
            <element ref = "ExtData" minOccurs = "0" maxOccurs = "unbounded"/>
        </sequence>
    </complexType>
</element>

<simpleType name = "CardTypeEnum" base = "string">
    <enumeration value = "Amex"/>
    <enumeration value = "DinersClub"/>
    <enumeration value = "Discover"/>
    <enumeration value = "JCB"/>
    <enumeration value = "MasterCard"/>
```

```
   <enumeration value = "Visa"/>
</simpleType>

<simpleType name = "CardNumType" base = "string">
   <maxLength value = "19"/>
</simpleType>

<simpleType name = "ExpDateType" base = "string">
   <maxLength value = "6"/>
</simpleType>

<simpleType name = "CVType" base = "string">
   <maxLength value = "4"/>
</simpleType>

<simpleType name = "MagDataType" base = "string">
   <maxLength value = "80"/>
</simpleType>


<element name = "Check">
   <complexType content = "elementOnly">
      <sequence>
         <element name = "CheckType" type = "CheckTypeEnum"/>
         <element name = "CheckNum" type = "CheckNumType"/>
         <element name = "MICR" type = "MICRType"/>
         <element name = "DL" type = "DLType" minOccurs = "0" maxOccurs =
"1"/>
         <element name = "SS" type = "SSType" minOccurs = "0" maxOccurs =
"1"/>
         <element name = "DOB" type = "date" minOccurs = "0" maxOccurs = "1"/>
         <element ref = "ExtData" minOccurs = "0" maxOccurs = "unbounded"/>
      </sequence>
  </complexType>
</element>

<simpleType name = "CheckTypeEnum" base = "string">
   <enumeration value = "Personal"/>
   <enumeration value = "Corporate"/>
   <enumeration value = "Government"/>
</simpleType>

<simpleType name = "CheckNumType" base = "string">
   <maxLength value = "8"/>
</simpleType>
```

```
<simpleType name = "MICRType" base = "string">
   <maxLength value = "35"/>
</simpleType>

<simpleType name = "DLType" base = "string">
   <maxLength value = "35"/>
</simpleType>
<simpleType name = "SSType" base = "string">
   <maxLength value = "35"/>
</simpleType>

  <simpleType name = "UserIdType" base = "string">
   <maxLength value = "12"/>
</simpleType>

<simpleType name = "CustCodeType" base = "string">
   <maxLength value = "17"/>
</simpleType>

<simpleType name = "PasswordType" base = "string">
   <maxLength value = "12"/>
</simpleType>


<simpleType name = "PNRefType" base = "string">
   <maxLength value = "12"/>
</simpleType>


<simpleType name="Base64BinaryData" base="binary">
  <encoding value="base64"/>
</simpleType>


<element name = "ExtData">
   <complexType content = "elementOnly">
      <attribute name = "Name" type = "string"/>
      <attribute name = "Value" type = "string"/>
   </complexType>
</element>

</schema>
```

APPENDIX B

XMLPay DTD

A Document Type Definition (DTD) defines the structure of an XML document. With a DTD you can define the set and order of tags, as well as the attributes for each. A well-formed XML document is considered valid when it conforms to its corresponding DTD.

The following is a Document Type Definition (DTD) representation of the XMLPay schema.

```
<!ELEMENT Signature (#PCDATA)>

<!ELEMENT InvNum (#PCDATA)>

<!ELEMENT Date (#PCDATA)>

<!ELEMENT Name (#PCDATA)>

<!ELEMENT Street (#PCDATA)>

<!ELEMENT City (#PCDATA)>

<!ELEMENT State (#PCDATA)>

<!ELEMENT Zip (#PCDATA)>

<!ELEMENT Country (#PCDATA)>

<!ELEMENT Address ((Street?, City?, State?, Zip?, Country?))>

<!ELEMENT EMail (#PCDATA)>

<!ELEMENT Phone (#PCDATA)>
```

```
<!ELEMENT Fax (#PCDATA)>

<!ELEMENT URL (#PCDATA)>

<!ELEMENT BillFrom (Name?, Address?, EMail?, Phone?, Fax?, URL?)>

<!ELEMENT CustomerId (#PCDATA)>

<!ELEMENT CustCode (#PCDATA)>

<!ELEMENT PONum (#PCDATA)>

<!ELEMENT TaxExempt (#PCDATA)>

<!ELEMENT BillTo (CustomerId?, Name?, Address?, EMail?, Phone?,
Fax?, CustCode?, PONum?, TaxExempt?)>

<!ELEMENT ShipFrom (Name?, Address, EMail?, Phone?, Fax?)>

<!ELEMENT ShipTo (Name?, Address, EMail?, Phone?, Fax?)>

<!ELEMENT Description (#PCDATA)>

<!ELEMENT SKU (#PCDATA)>

<!ELEMENT UPC (#PCDATA)>

<!ELEMENT Quantity (#PCDATA)>

<!ELEMENT UnitOfMeasurement (#PCDATA)>

<!ELEMENT UnitPrice (#PCDATA)>
<!ATTLIST UnitPrice Currency CDATA #IMPLIED>

<!ELEMENT ExtAmt (#PCDATA)>
<!ATTLIST ExtAmt Currency CDATA #IMPLIED>

<!ELEMENT DiscountAmt (#PCDATA)>
<!ATTLIST DiscountAmt Currency CDATA #IMPLIED>

<!ELEMENT TaxAmt (#PCDATA)>
<!ATTLIST TaxAmt Currency CDATA #IMPLIED>

<!ELEMENT TotalAmt (#PCDATA)>
<!ATTLIST TotalAmt Currency CDATA #IMPLIED>
```

```
<!ELEMENT ExtData (#PCDATA)>
<!ATTLIST ExtData Name CDATA #IMPLIED>
<!ATTLIST ExtData Value CDATA #IMPLIED>

<!ELEMENT Item ((SKU?, UPC?, Description?, Quantity?,
UnitOfMeasurement?, UnitPrice?, ExtAmt?, DiscountAmt?, TaxAmt?,
TotalAmt?, ExtData*))>
<!ATTLIST Item Number CDATA #REQUIRED>

<!ELEMENT Items (Item*)>

<!ELEMENT ShippingAmt (#PCDATA)>
<!ATTLIST ShippingAmt Currency CDATA #IMPLIED>

<!ELEMENT DutyAmt (#PCDATA)>
<!ATTLIST DutyAmt Currency CDATA #IMPLIED>

<!ELEMENT NationalTaxIncl (#PCDATA)>

<!ELEMENT Comment (#PCDATA)>

<!ELEMENT Invoice ((InvNum?, Date?, BillFrom?, BillTo?, ShipFrom?,
ShipTo?, Description*, Items?, DiscountAmt?, ShippingAmt?,
DutyAmt?, TaxAmt?, NationalTaxIncl?, TotalAmt, Comment?,
ExtData*))>

<!ELEMENT AcctType (#PCDATA)>

<!ELEMENT AcctNum (#PCDATA)>

<!ELEMENT ABA (#PCDATA)>

<!ELEMENT Prenote (#PCDATA)>

<!ELEMENT ACH ((AcctType, AcctNum, ABA, Prenote?, ExtData*))>

<!ELEMENT CardType (#PCDATA)>

<!ELEMENT CardNum (#PCDATA)>

<!ELEMENT ExpDate (#PCDATA)>

<!ELEMENT CVNum (#PCDATA)>

<!ELEMENT MagData (#PCDATA)>
```

```
<!ELEMENT NameOnCard (#PCDATA)>

<!ELEMENT Card ((CardType?, CardNum, ExpDate, CVNum?, MagData?,
NameOnCard?, ExtData*))>

<!ELEMENT CheckType (#PCDATA)>

<!ELEMENT AllianceNum (#PCDATA)>

<!ELEMENT CheckNum (#PCDATA)>

<!ELEMENT MICR (#PCDATA)>

<!ELEMENT DL (#PCDATA)>

<!ELEMENT SS (#PCDATA)>

<!ELEMENT DOB (#PCDATA)>

<!ELEMENT Check ((CheckType, AllianceNum?, CheckNum, MICR, DL?,
SS?, DOB?, ExtData*, Address?))>

<!ELEMENT Tender ((ACH| Card| Check))>

<!ELEMENT PayData ((Invoice, Tender))>

<!ELEMENT PKCS7Signature (#PCDATA)>

<!ELEMENT PayDataAuth ((PKCS7Signature| Signature))>

<!ELEMENT Vendor (#PCDATA)>

<!ELEMENT Partner (#PCDATA)>

<!ELEMENT Authorization ((PayData, PayDataAuth?, ExtData*))>

<!ELEMENT PNRef (#PCDATA)>

<!ELEMENT Capture ((PNRef, Invoice?, ExtData*))>

<!ELEMENT Sale ((PayData, PayDataAuth?, ExtData*))>

<!ELEMENT Credit (((PNRef| Tender), Invoice?, ExtData*))>

<!ELEMENT Void ((PNRef, ExtData*))>
```

```
<!ELEMENT AuthCode (#PCDATA)>

<!ELEMENT ForceCapture ((PayData, PayDataAuth?, AuthCode,
ExtData*))>

<!ELEMENT RepeatSale ((PNRef, Invoice?, ExtData*))>

<!ELEMENT GetStatus ((PNRef, ExtData*))>

<!ELEMENT Transaction (((Authorization| Capture| Sale| Credit|
Void| ForceCapture| RepeatSale| GetStatus)))>
<!ATTLIST Transaction Id CDATA #IMPLIED>
<!ATTLIST Transaction CustRef CDATA #IMPLIED>

<!ELEMENT Transactions (Transaction+)>

<!ELEMENT RequestData ((Vendor, Partner, Transactions))>

<!ELEMENT User (#PCDATA)>

<!ELEMENT UserDomain (#PCDATA)>

<!ELEMENT Password (#PCDATA)>

<!ELEMENT UserPass ((User, UserDomain?, Password))>

<!ELEMENT RequestAuth ((UserPass| Signature))>

<!ELEMENT XMLPayRequest ((RequestData, RequestAuth))>
<!ATTLIST XMLPayRequest version CDATA #IMPLIED>
<!ATTLIST XMLPayRequest Timeout CDATA #IMPLIED>
<!ATTLIST XMLPayRequest UniqueTransactionIdentifier CDATA
#IMPLIED>

<!ELEMENT Result (#PCDATA)>

<!ELEMENT StreetMatch (#PCDATA)>

<!ELEMENT ZipMatch (#PCDATA)>

<!ELEMENT AVSResult ((StreetMatch, ZipMatch))>

<!ELEMENT CVResult (#PCDATA)>

<!ELEMENT Message (#PCDATA)>
```

```
<!ELEMENT HostCode (#PCDATA)>

<!ELEMENT HostURL (#PCDATA)>

<!ELEMENT OrigResult (#PCDATA)>

<!ELEMENT TrStatus (#PCDATA)>

<!ELEMENT ReceiptURL (#PCDATA)>

<!ELEMENT TransactionResult ((Result, AVSResult?, CVResult?,
Message?, PNRef?, AuthCode?, HostCode?, HostURL?, OrigResult?,
TrStatus?, ReceiptURL?, ExtData*))>
<!ATTLIST TransactionResult Id CDATA #IMPLIED>

<!ELEMENT TransactionResults (TransactionResult+)>

<!ELEMENT ResponseData ((Vendor, Partner, TransactionResults))>

<!ELEMENT ReceiptData ((Vendor, Partner, Transaction,
TransactionResult))>

<!ELEMENT XMLPayReceipt ((ReceiptData, Signature?))>

<!ELEMENT TransactionReceipts (XMLPayReceipt+)>

<!ELEMENT XMLPayResponse ((ResponseData, Signature?,
TransactionReceipts?))>
<!ATTLIST XMLPayResponse version CDATA #IMPLIED>
<!ATTLIST XMLPayResponse UniqueTransactionIdentifier CDATA
#IMPLIED>
```

APPENDIX C

Transaction Results

VeriSign Transaction RESULTs/RESPMSGs

A RESULT greater than '0' indicates a decline or error. Exact wording of the RESPMSG (Response Message-here shown in bold) may vary. Sometimes a colon will appear after the initial RESPMSG followed by more detailed information.

Table C-1 VeriSign Transaction RESULTs/RESPMSGs

RSLT	**RESPMSG**/Explanation
0	**Approved.**
1	**User authentication failed.**
2	**Invalid tender.** Your merchant bank account does not support the following credit card type that was submitted.
3	**Invalid transaction type.** Transaction type is not appropriate for this transaction. For example, you cannot credit an authorization-only transaction.
4	**Invalid amount.**
5	**Invalid merchant information.** Processor does not recognize your merchant account information. Contact your bank account acquirer to resolve this problem.
7	**Field format error**. Invalid information entered.
8	**Not a transaction server.**
9	**Too many parameters or invalid stream.**
10	**Too many line items.**
11	**Client timeout waiting for response.**

Table C-1 VeriSign Transaction RESULTs/RESPMSGs (Continued)

RSLT	**RESPMSG**/Explanation
12	**Declined.** Please check the credit card number and transaction information to make sure they were entered correctly. If this does not resolve the problem, have the customer call the credit card issuer to resolve.
13	**Referral.** Transaction was declined but could be approved with a verbal authorization from the bank that issued the card. Submit a manual Voice Authorization transaction and enter the verbal auth code.
19	**Original transaction ID not found.** The transaction ID you entered for this transaction is not valid.
20	**Cannot find the customer reference number.**
22	**Invalid ABA number.**
23	**Invalid account number.** Please check credit card number and re-submit.
24	**Invalid expiration date.** Please check and re-submit.
25	**Transaction type not mapped to this host.**
50	**Insufficient funds available.**
99	**General error.**
100	**Invalid transaction returned from host.**
101	**Timeout value too small.**
102	**Processor not available.**
103	**Error reading response from host.**
104	**Timeout waiting for processor response.** Please try your transaction again.
105	**Credit error.** Please make sure you have not already credited this transaction, or that this transaction ID is for a creditable transaction. (For example, you cannot credit an authorization.)
106	**Host not available.**
107	**Duplicate supression timeout.**

Table C-1 VeriSign Transaction RESULTs/RESPMSGs (Continued)

RSLT	**RESPMSG**/Explanation
108	**Void error.** Please make sure the transaction ID entered has not already been voided. If not, then look at the Transaction Detail screen for this transaction to see if it has settled. (The Batch field will be set to a number greater than zero if the transaction has been settled). If the transaction has already settled, your only recourse is a reversal (credit a payment or submit a payment for a credit).
109	**Timout waiting for host response.**
111	**Capture error.** Only authorization transactions can be captured.
112	**Failed AVS check.** Address and Zip code do not match.
113	**Cannot exceed sales cap.** For ACH transactions only.
1000	**Generic host error.** This is a generic message returned by your credit card processor. The message itself will contain more information describing the error.

AVS Result Codes

The Address Verification Service (AVS) compares the street address and zip code submitted with that on file at the cardholder's bank. Any one of the following results can appear in the AVS Street Match and AVS Zip Match fields on the transaction detail screen:

Result	Meaning
Y	Information submitted matches information on file with cardholder's bank.
N	Information submitted does not match information on file with the card-holder's bank.
X	Cardholder's bank does not support AVS checking for this information.

Note Results can vary on the same transaction detail screen. In other words, AVS Street Match = Y and AVS Zip Match = N (and vice versa) could appear on the same transaction detail screen. Also note that sometimes when service is unavailable no code at all will be returned.

Index