



# JetPay Authorization XML Specification

© 2012 JetPay LLC. All rights reserved.

# Contents

- 1 Introduction
- 2 Element Overview
  - ◆ 2.1 JetPay Transaction Request Elements
  - ◆ 2.2 Request Transaction Element Definitions
  - ◆ 2.3 JetPay Transaction Response Elements
  - ◆ 2.4 Response Transaction Element Definitions
- 3 General Examples
  - ◆ 3.1 "AUTHONLY" Transaction Example
  - ◆ 3.2 "CAPT" Transaction Example
  - ◆ 3.3 "SALE" Transaction Example
  - ◆ 3.4 "FORCE" Transaction Example
  - ◆ 3.5 "VOID" Transaction Example
  - ◆ 3.6 "REVERSEAUTH" Transaction Example
    - ◇ 3.6.1 What is the Difference Between "VOID" and "REVERSEAUTH"?
  - ◆ 3.7 "CREDIT" Transaction Example
  - ◆ 3.8 "ENQ" Transaction Example
  - ◆ 3.9 "PING" Transaction Example
  - ◆ 3.10 Error Response
  - ◆ 3.11 Additional Examples
    - ◇ 3.11.1 Address Verification (AVS) Example
    - ◇ 3.11.2 Card Validation (CVV2) Example
    - ◇ 3.11.3 Cardholder Authentication (CAVV) Example
    - ◇ 3.11.4 Recurring/Bill Payment Examples
- 4 Application-Specific Examples
  - ◆ 4.1 Explicitly Invoking Alternatives, Using RoutingCode
  - ◆ 4.2 POS Features for the Retail Industry Type
    - ◇ 4.2.1 POS Request Message
    - ◇ 4.2.2 Request Transaction Element Definitions
    - ◇ 4.2.3 POS Response Message
    - ◇ 4.2.4 Response Transaction Element Definitions
    - ◇ 4.2.5 Credit Card Examples
      - 4.2.5.1 Magnetic Stripe POS Retail Sale Transaction Example
      - 4.2.5.2 Magnetic Stripe Transaction Example for Restaurant
      - 4.2.5.3 Manually Entered POS Transaction Example
      - 4.2.5.4 Contactless Track POS Transaction Example
    - ◇ 4.2.6 Debit and EBT Card Examples
      - 4.2.6.1 Debit POS Sale Transaction Example
      - 4.2.6.2 EBT POS Sale Transaction Example
  - ◆ 4.3 Industry-Specific Examples
    - ◇ 4.3.1 Ecommerce Industry-Specific Example

- ◊ 4.3.2 MOTO Industry-Specific Examples
- ◊ 4.3.3 Hotel AUTHONLY Example
- ◊ 4.3.4 Hotel INCREMENTAL Example
- ◊ 4.3.5 Hotel PARTIALREVERSAL Example
- ◊ 4.3.6 Hotel REVERSEAUTH Example
- ◊ 4.3.7 Hotel CAPT Example
- ◆ 4.4 Handling Non-U.S. Card Types (Switch/Solo)
  - ◊ 4.4.1 Special note about Credit with Switch and Solo cards
- 5 ACH processing
  - ◆ 5.1 ACH Request Messages
  - ◆ 5.2 Request Transaction Element Definitions
  - ◆ 5.3 ACH Response Message
  - ◆ 5.4 Response Transaction Element Definitions
  - ◆ 5.5 ACH Examples
    - ◊ 5.5.1 "CHECK" Transaction Example (ACH)
    - ◊ 5.5.2 "REVERSAL" Transaction Example (ACH)
    - ◊ 5.5.3 "VOIDACH" Transaction Example (ACH)
- 6 Memo Transactions
- 7 Data Security Features
  - ◆ 7.1 CryptoPAN Secure Credit Card Number Storage
    - ◊ 7.1.1 Sending Card Numbers Securely, Using CryptoPAN
    - ◊ 7.1.2 CryptoPAN Encoding
    - ◊ 7.1.3 Sample CryptoPAN Program
  - ◆ 7.2 Jetpay Account Safe tokens
  - ◆ 7.3 Passwords
- 8 Communicating with JetPay
  - ◆ 8.1 Methods of Securely Posting an XML file
  - ◆ 8.2 JetPay URLs
- 9 Certification
  - ◆ 9.1 Standard Transaction Processing Test Suite
    - ◊ 9.1.1 AUTHONLY, SALE, and CAPT Test Cases
    - ◊ 9.1.2 Magnetic Stripe Test Cases
    - ◊ 9.1.3 Contactless Magnetic Stripe Test Cases
    - ◊ 9.1.4 CREDIT Test Cases
    - ◊ 9.1.5 VOID Test Cases
  - ◆ 9.2 Validation, Verification, and Authentication Test Suite
    - ◊ 9.2.1 CVV2/CVC2/CID Test Cases
    - ◊ 9.2.2 Address Verification Test Cases
    - ◊ 9.2.3 UCAF Test Cases (Verified-by-Visa and SecureCode)
  - ◆ 9.3 ACH Transactions
    - ◊ 9.3.1 Checking Account and Savings Account Transactions
- 10 Verification and Validation Features
  - ◆ 10.1 Address Verification Services
    - ◊ 10.1.1 Visa AVS Result Codes
    - ◊ 10.1.2 MasterCard AVS Responses

- ◊ 10.1.3 American Express AAV Responses
- ◆ 10.2 Card Validation Using CVV2, CVC2, and CID
  - ◊ 10.2.1 JetPay CVV2 Result Codes
- 11 JetPay Test System
  - ◆ 11.1 Connecting to the JetPay Test System
  - ◆ 11.2 The "TESTTERMINAL" TerminalID
  - ◆ 11.3 Simulating "APPROVED" Transactions
  - ◆ 11.4 Simulating "DECLINED" Transactions
    - ◊ 11.4.1 List of Amount and Action codes
  - ◆ 11.5 Other Responses
  - ◆ 11.6 Simulating AVS Responses
    - ◊ 11.6.1 Postal Code Values That Affect the AVS Result
  - ◆ 11.7 Simulating CVV2/CVC2/CID Responses
    - ◊ 11.7.1 CVV2 Values That Affect the CVV2 Result
- 12 Result Codes
  - ◆ 12.1 AVS Result Codes
  - ◆ 12.2 CVV2 Result Codes
  - ◆ 12.3 CAVV Result Codes
- 13 The JetPay Authorization XML Schema

# 1 Introduction

The JetPay Authorization XML interface provides a flexible and easy to use interface for processing transactions with the JetPay system. Through one interface, a merchant can perform credit card, debit card and ACH transactions whether they be E-COMMERCE, MOTO, or POS in origin.

This document is for merchants or third-party developers who will be generating their own XML for transmission to JetPay. This document does not cover the Windows DLL, Java, or Batch File interfaces. This document does assume that the reader is familiar with XML generation and parsing, using the https protocol, and the rudiments of credit card and/or ACH processing.

Any questions or problems with this document should be sent to JetPay's customer service, [assist@jetpay.com](mailto:assist@jetpay.com).

## 2 Element Overview

The following table shows the logical structure of the JetPay transaction request message. The **Document Structure** column lists the element name and its place in the message structure in order and indentation. The **Qualifications** column explains the conditions for supplying that element to a transaction. The condition column lists any conditions that apply to the element.

### 2.1 JetPay Transaction Request Elements

Element in Document Structure	Qualifications
JetPay	Required root element.
TransactionType	Required
TerminalID	Required
TransactionID	Required, exactly 18-characters in length
Approval	Required for FORCE transactions.
RoutingCode	Optional
BatchID	Optional
Origin	Optional (default is INTERNET), but recommended
Password	Optional
OrderNumber	Optional
CardNum	Required for SALE, AUTHONLY, & FORCE transactions, unless Track1 or Track2 is supplied.
CardNum CardPresent	Attribute is boolean, default is false
CVV2	Optional
Issue	Optional, only for certain card types
CardExpMonth	

JetPay Authorization XML Specification © 2012 JetPay LLC.

	Required for SALE, AUTHONLY, & FORCE transactions, unless Track1 or Track2 is supplied.
CardExpYear	Required for SALE, AUTHONLY, & FORCE transactions, unless Track1 or Track2 is supplied.
CardStartMonth	Optional
CardStartYear	Optional
Track1	Required for SALE, AUTHONLY, & FORCE transactions unless CardNum or Track2 is supplied.
Track2	Required for SALE, AUTHONLY, & FORCE transactions unless CardNum or Track1 is supplied.
ACHAccountType	Optional Attribute contains value of Checking, Savings,and BusinessCk
AccountNumber	Required, but only as part of the optional ACH element.
ABA	Required, but only as part of the optional ACH element.
CheckNumber	Required, but only as part of the optional ACH element.
CardName	Optional, required for ACH
DispositionType	Optional
TotalAmount	Required for SALE, AUTHONLY, & FORCE transactions.
FeeAmount	Optional
TaxAmount	Optional
BillingAddress	Optional
BillingCity	Optional
BillingStateProv	Optional
BillingPostalCode	Optional
BillingCountry	Optional
BillingPhone	Optional
Email	Optional
UserIPAddress	Optional
UserHost	Optional
UDField1	Optional
UDField2	Optional
UDField3	Optional
ActionCode	Required for ACK transactions only.
IndustryInfoType	Optional, but recommended.Attribute contains value of ECOMMERCE, RETAIL, MOTO, HOTEL, RESTAURANT, AUTORENTAL, AIRLINE, PARKING, or QUASICASH.
VerificationType	Optional Attribute contains value of VbV.
CavvEncoding	Optional Attribute contains value of HEX or BASE64
XidEncoding	Optional Attribute contains value of HEX or BASE64
Eci	Optional
ShippingInfo	Optional
CustomerPO	Optional
ShippingMethod	Optional, value of SAME DAY, OVERNIGHT, PRIORITY, GROUND, or ELECTRONIC

ShippingName	Optional
ShippingAddr	Optional
Address	Optional
City	Optional
StateProv	Optional
Country	Optional
Phone	Optional

## 2.2 Request Transaction Element Definitions

The following properties describe the request transactions.

- **JetPay** - root element, enclosing the following transaction elements:
- **TransactionType** - Required for all transactions. Must be one of the following values:
  - ◆ SALE - Authorizes and captures a credit card charge in a single transaction.
  - ◆ AUTHONLY - The credit card limit is checked to verify that a certain amount is available (and to reserve that amount), but the card is not charged. Either a FORCE or CAPT is used to complete the transaction.
  - ◆ CAPT - A credit card charge using an amount equal to or less than the amount of a previous AUTHONLY transaction. The AUTHONLY transaction is required to be present in the JetPay system for the capture to complete.
  - ◆ FORCE - The JetPay system will capture the transaction using the given amount and authorization code. This does not require that a corresponding AUTHONLY transaction be present in the JetPay system.
  - ◆ VOID - The VOID transaction removes a credit card transaction from the host before the transaction settles. If a transaction has already settled, it cannot be VOIDed (when a transaction is not VOIDed before settlement, a CREDIT transaction is required to reverse the charge). See also CREDIT.
  - ◆ ENQ - Query for a credit card charge in the JetPay system. ResponseText is:"AUTHORIZED" when a transaction was an AUTHONLY, or"ACCEPTED" when the transaction was a FORCE, or"APPROVED" for a SALE or AUTHONLY/FORCE or AUTHONLY/CAPT,or TRANS NOT FOUND.
  - ◆ CREDIT - The CREDIT transaction submits a credit card transaction reversal for settlement. The CREDIT reverses a transaction regardless of the settlement status of the transaction it reverses.
  - ◆ REVERSEAUTH - The REVERSEAUTH transaction attempts to free a cardholder's open-to-buy for a prior AUTHONLY transaction. The REVERSEAUTH transaction is supported unevenly by issuers, so this transaction is of questionable usefulness.
  - ◆ CHECK - ACH only.
  - ◆ REVERSAL - ACH only.
  - ◆ VOIDACH - ACH only.
  - ◆ PING - Verify active communication with the acquirer software. ResponseText reads "PING" to acknowledge this transaction.
  - ◆ ACK - Acknowledge receipt of a response message.
- **TerminalID** - Required for all transactions. This ID is issued by JetPay LLC when an account is set up with the merchant bank.
- **TransactionID** - Required for all transactions. The TransactionID is an 18-character value. Allowed characters are "a" through "z" , "A" through "Z", "0" through "9" and "-".
- **RoutingCode** - A mechanism for declaring alternate transaction routes. The value of the RoutingCode must be pre-arranged.

- **Approval** - Required for FORCE transactions. Supply an authorization code from an earlier AUTHONLY response.
- **BatchID** - Enable Merchant-Initiated Terminal Reconciliation feature. The BatchID allows a merchant to tag a transaction for manual settlement. Any transaction having a BatchID does not settle until the merchant manually triggers the settlement of transactions having matching BatchIDs.
- **Origin** - The manner in which this transaction was communicated to the merchant. Possible values are "INTERNET" (ecommerce merchants), "POS" (retail merchants with magnetic card readers), "RECURRING" (ecommerce and MOTO merchants), "**PHONE ORDER**" (MOTO merchants), and "**MAIL ORDER**" (MOTO merchants).
- **Password** - The password (coordinated with JetPay LLC) is used to verify that a submitted XML transaction originates from a specific merchant.
- **OrderNumber** - A string referencing an order.
- **CardNum** - The credit card number or PAN ("personal account number") submitted for a transaction. The PAN is thirteen to nineteen digits long, depending on the type of the credit card.
- **CVV2** - The Card Verification Value printed on the back of a credit card. The CVV2 number is a three- or four-digit value. These CVV2 digits provide additional security for a transaction, assuring the physical presence of a credit card. Please note the terminology differences between credit card companies, in that Visa uses the CVV2 term (meaning "Card Verification Value 2"), MasterCard uses CVC2 term (meaning "Card Validation Code 2"), and American Express uses the CID term (meaning "Card Identifier code").
- **Issue** - Two-digit value (European credit cards).
- **CardExpMonth** - Two-digit card expiration month.
- **CardExpYear** - Two-digit card expiration year.
- **CardStartMonth** - Two-digit start-up month.
- **CardStartYear** - Two-digit start-up year.
- **ACH** - Banking transaction information: The **AccountType** attribute may have a value of "Checking", "Savings", or "BusinessCk" (defaulting to "Checking").
- **AccountNumber** - The cardholder's bank account number.
- **ABA** - The ABA routing number for the cardholder's bank.
- **CheckNumber** - The check number for the banking transaction.
- **CardName** - The customer name, that is, the name of the cardholder.
- **DispositionType** - Custom host extension disposition model on the transaction-processing engine. Do not change the value of the DispositionType unless instructed to do so.
- **TotalAmount** - Purchase price to be transacted, including taxes and fees. The currency type assumed for the amount is dependent on the merchant. *Digits only: no decimal point, no dollar sign, no plus/minus sign.* Value must be non-zero.
- **FeeAmount** - The surcharge applied to a transaction. The surcharge value within this element is assumed to be already included in total amount. *Digits only: no decimal point, no dollar sign, no plus/minus sign.*
- **TaxAmount** - Tax applied to a transaction. The tax value within this element is assumed to be already included in total amount. *Digits only: no decimal point, no dollar sign, no plus/minus sign.*
- **BillingAddress** - Cardholder's address.
- **BillingCity** - Cardholder's city.
- **BillingStateProv** - Cardholder's state code abbreviation.
- **BillingPostalCode** - Cardholder's zip code.
- **BillingCountry** - Cardholder's country, a valid ISO Country Code (consisting of either three alphabetic characters or three digits).
- **BillingPhone** - Cardholder's telephone number.
- **Email** - Cardholder's email address.
- **UserIPAddr** - Customer's IP address.
- **UserHost** - Customer's host name.
- **UDField1** - User-defined field, available for reporting purposes.
- **UDField2** - User-defined field.

- **UDField3** - User-defined field.
- **ActionCode** - For "ACK" transactions, the ActionCode from a previous transaction response.
- **IndustryInfo** - A declaration of the general marketing category served by the merchant. Different merchants fall into different categories, including "ECOMMERCE" (for web-based sales through Internet merchants), "RETAIL" (for merchants with brick-and-mortar storefronts), "MOTO" (for merchants selling through mail-order or phone-order), "HOTEL", "RESTAURANT", "AUTORENTAL", "AIRLINE", "PARKING", and "QUASICASH".
- **Verification** - Support for transactional security, for ecommerce merchants who subscribe to Verified By Visa or SecureCode.
- **ShippingInfo** - When a product or service is shipped to a customer, this category enables collection of the shipping data. This is best applied to cardholder-not-present transactions. Ecommerce merchants and MOTO merchants always submit cardholder-not-present transactions; they may submit shipping information to improve their transaction qualification if permitted.
  - ◆ **CustomerPO** - The cardholder's purchase order number.
  - ◆ **ShippingMethod** - The manner of delivering a purchase to the recipient. One of five shipping methods may be specified: **SAME DAY**- courier delivery **OVERNIGHT**- next day delivery. **PRIORITY**- two to three days shipping time. **GROUND**- four or more days shipping time. **ELECTRONIC**- delivery of purchase by electronic means.
  - ◆ **ShippingName** - The name of the recipient of the order.
  - ◆ **ShippingAddr** - The destination shipping address data.

## 2.3 JetPay Transaction Response Elements

The following table describes the logical structure of the JetPay transaction response message, sent in response to an initiating JetPay transaction request message.

Element in Document Structure	Qualifications
JetPayResponse	Required root element.
TransactionID	Present, matching the request's TransactionID. Absent only when ErrMsg is present.
ActionCode	Present in all responses. "000" = <i>approved</i> .
Approval	Present in many transactions. Absent when the issuer bank declines a transaction. Absent for PING. Absent when ErrMsg is present.
CVV2	Present when a CVV2 or CVC2 or CID value is submitted in JetPay request.
VerificationResult	Present when a CAVV value is submitted in JetPay request.
ResponseText	Present, except when ErrMsg is present.
AddressMatch	Present only when address info is submitted.
ZipMatch	Present only when address info is submitted.
AVS	Present only when address info is submitted.
ErrMsg	Present only when a JetPay Authorization XML message completely fails, indicating that the incoming message cannot be processed. Usually contains a message pointing out XML syntax errors. Useful for debug feedback during development.

## 2.4 Response Transaction Element Definitions

The following properties describe the request messages:

- **JetPayResponse**- root element, enclosing the following transaction elements:
- **TransactionID** - This 18-character value matches the TransactionID submitted in the corresponding JetPay request transaction message. This value can be used to correlate the transaction's acknowledgment information with its submitted request data.
- **ActionCode** - The ActionCode is three (3) characters. The ActionCode contains a "000" value to indicate that a transaction has been approved by the issuer bank. The ActionCode will be some other value when either the issuer bank declines the transaction or if the JetPay server host detects an error.
- **Approval** - The issuer bank returns an authorization code, six (6) alphanumeric characters long. A "PING" transaction does not return an Approval value.
- **ResponseText** - Message from the JetPay server, describing to the ActionCode. Note that a "000" ActionCode may have multiple ResponseText values, including "APPROVED", "ACCEPTED", "CAPTURED", and "PING". Other ActionCodes will contain a ResponseText value of "DECLINED" or some similar phrase.
- **AVS** - A code indicating the AVS ("Address Verification Service") results for a transaction. Refer to [#AVS Result Codes](#) in this reference for a explanation of the valid result codes.
- **AddressMatch** - Code indicating the address match results for a credit card transaction. Returns "Y", "N", or "X", where:  
 Y - The address matches.  
 N - The address does not match.  
 X - No result available (AVS requested but not performed).
- **ZipMatch** - Code indicating the postal code match results for a credit card transaction. Returns "Y", "N", or "X", where:  
 Y - The postal code matches.  
 N - The postal code does not match.  
 X - No result available (AVS requested but not performed).
- **CVV2** - The response code to a Visa CVV2 or a MasterCard CVC2 or an American Express CID submission. Refer to [#JetPay CVV2 Result Codes](#) in this reference for a explanation of the valid result codes.
- **VerificationResult** - The response code to a Visa CAVV or a MasterCard UCAF submission. Refer to [#CAVV Result Codes](#) in this reference for a explanation of the valid result codes.
- **ErrMsg** - Error message, if any. This error message usually describes an XML syntax error in the submitted JetPay transaction. This element will usually also contain a message that points out the specific XML syntax errors within the submitted XML document. You should only expect to see this **ErrMsg** element during development. If you see this element in your production software system, analyze your software immediately and make corrections accordingly.

## 3 General Examples

The following examples illustrate JetPay transaction messages, showing a sample request message with a corresponding sample response message.

### 3.1 "AUTHONLY" Transaction Example

An "AUTHONLY" transaction is the most basic type of transaction. In an "AUTHONLY" transaction, JetPay accepts the XML transaction block containing the credit card transaction information, retrieves an authorization response from the issuer of the credit card, and responds back to the original sender with that resulting authorization information.

Required data for an "AUTHONLY" transaction request are:

<TransactionType>	AUTHONLY
<TerminalID>	your terminal ID, assigned by JetPay LLC
<TransactionID>	character string, exactly eighteen characters long, preferably a unique value.
<CardNum>	valid credit card number
<CardExpMonth>	credit card expiration month (two digits)
<CardExpYear>	credit card expiration year (two digits)
<TotalAmount>	transaction amount, (digits only)

Minimum data returned from a successful "AUTHONLY" transaction response are:

<TransactionID>	matching the value in the request, above
<ActionCode>	000
<Approval>	character string exactly six characters long
<ResponseText>	the transaction's outcome status

The following is a sample "AUTHONLY" transaction for \$87.99:

```
<JetPay>
<TransactionType>AUTHONLY</TransactionType>
<TerminalID>TESTMERCHANT</TerminalID>
<TransactionID>010327153017T10018</TransactionID>
<CardNum>4000300020001000</CardNum>
<CardExpMonth>12</CardExpMonth>
<CardExpYear>15</CardExpYear>
<TotalAmount>8799</TotalAmount>
</JetPay>
```

The following shows the sample response to the above "AUTHONLY" request.

```
<JetPayResponse>
<TransactionID>010327153017T10018</TransactionID>
<ActionCode>000</ActionCode>
<Approval>502F6B</Approval>
<ResponseText>APPROVED</ResponseText>
</JetPayResponse>
```

### 3.2 "CAPT" Transaction Example

A "CAPT" (or "capture") transaction enables a previously authorized transaction to be settled into a complete sale. In a "CAPT" transaction, JetPay accepts the XML transaction block containing the credit card transaction information, looks up the matching authorized transaction, and responds back to the original sender with the resulting sales information.

## JetPay Authorization XML Specification © 2012 JetPay LLC.

The matching "AUTHONLY" transaction for this capture operation must have been authorized through JetPay in order for the "CAPT" transaction to be successful. If a transaction was not authorized through JetPay, the response to the "CAPT" will indicate that the authorization was not found.

Submitted data for a "CAPT" transaction request are:

<TransactionType>	CAPT (required)
<TerminalID>	your terminal ID, assigned by JetPay LLC (required)
<TransactionID>	character string. exactly eighteen characters long, either matching the target "AUTHONLY" transaction or otherwise containing a unique value. (required)
<CardNum>	target credit card number
<CardExpMonth>	credit card expiration month (two digits)
<CardExpYear>	credit card expiration year (two digits)
<Approval>	six characters long, matching the code from "AUTHONLY" transaction.
<TotalAmount>	transaction amount, (digits only)

Minimum data returned from a successful "AUTHONLY" transaction response are:

<TransactionID>	matching the value in the request, above
<ActionCode>	000
<Approval>	character string exactly six characters long
<ResponseText>	the transaction's outcome status

"Capturing" an authorized transaction gets accomplished in two ways:

1. Submitting the matching transaction ID for the authorization (primary way).
2. Submitting the credit card information along with a matching total amount and authorization code (secondary way, if primary way gets no match).

This first "CAPT" transaction demonstrates the primary way to complete a sale for the "AUTHONLY" transaction example from the previous section:

```
<JetPay>
<TransactionType>CAPT</TransactionType>
<TerminalID>TESTMERCHANT</TerminalID>
<TransactionID>010327153017T10018</TransactionID>
</JetPay>
```

For this first "CAPT" transaction, the following response illustrates how the returning XML would look.

```
<JetPayResponse>
<TransactionID>010327153017T10018</TransactionID>
<ActionCode>000</ActionCode>
<Approval>502F6B</Approval>
<ResponseText>APPROVED</ResponseText>
</JetPayResponse>
```

The second "CAPT" transaction demonstrates the alternative way to complete a sale for the "AUTHONLY" transaction example from the previous section:

```
<JetPay>
<TransactionType>CAPT</TransactionType>
<TerminalID>TESTMERCHANT</TerminalID>
```



The following shows how the successful response would look.

```
<JetPayResponse>
<TransactionID>010327153017T10017</TransactionID>
<ActionCode>000</ActionCode>
<Approval>002F6B</Approval>
<ResponseText>APPROVED</ResponseText>
</JetPayResponse>
```

### 3.4 "FORCE" Transaction Example

A "FORCE" transaction completes an authorized transaction, regardless of whether the transaction was authorized through JetPay or not. The JetPay server will assume that the submitted transaction has been properly authorized and will use the submitted authorization code to settle the transaction. The JetPay server always responds that the transaction is accepted for settlement processing (as long as complete transaction information is submitted in the "FORCE" transaction).

Required data for a "FORCE" transaction request are:

<TransactionType>	FORCE
<TerminalID>	your terminal ID, assigned by JetPay LLC
<TransactionID>	character string. exactly eighteen characters long, preferably a unique value.
<CardNum>	valid credit card number
<CardExpMonth>	credit card expiration month (two digits)
<CardExpYear>	credit card expiration year (two digits)
<Approval>	six characters long, matching the authorization code from the original transaction.
<TotalAmount>	transaction amount (digits only)

Minimum data returned from a successful "FORCE" transaction response are:

<TransactionID>	matching the value in the request, above
<ActionCode>	000
<Approval>	character string exactly six characters long
<ResponseText>	the transaction's outcome status

The following "FORCE" transaction completes a sale:

```
<JetPay>
<TransactionType>FORCE</TransactionType>
<TerminalID>TESTMERCHANT</TerminalID>
<TransactionID>010327153017T10018</TransactionID>
<CardNum>4000300020001000</CardNum>
<CardExpMonth>12</CardExpMonth>
<CardExpYear>15</CardExpYear>
<TotalAmount>8799</TotalAmount>
<Approval>502F6B</Approval>
</JetPay>
```

The following shows how the response to the "FORCE" transaction would look.

```
<JetPayResponse>
<TransactionID>010327153017T10018</TransactionID>
```

```
<ActionCode>000</ActionCode>
<Approval>502F6B</Approval>
<ResponseText>ACCEPTED</ResponseText>
</JetPayResponse>
```

A "FORCE" transaction differs from a "CAPT" transaction inasmuch as the authorization does not have to originate with JetPay when it is "forced."

A "FORCE" transaction will complete the sale for any transaction whether that transaction originated through JetPay or not. JetPay attempts to settle that transaction using the submitted authorization code. When "forcing" a transaction, the successful completion of that transaction gets reported to the merchant through settlement processing (and not directly by the JetPay server).

The successful completion of a "FORCE" transaction ultimately depends on the policies and procedures of the authorizing institution that issued the credit card.

### 3.5 "VOID" Transaction Example

A "VOID" transaction removes an approved transaction before settlement. The JetPay server can only remove unsettled transaction specified by the VOID transaction, so settled transactions are ineligible for VOIDing. The JetPay server responds with either "VOID PROCESSED" or "RECORD NOT FOUND", depending on the outcome of the transaction.

Required data for a "VOID" transaction request are:

<TransactionType>	VOID
<TerminalID>	your terminal ID, assigned by JetPay LLC
<TransactionID>	character string, exactly eighteen characters long, matching the target "CAPT" or "SALE" or "FORCE" transaction. (required)
<CardNum>	valid credit card number
<Approval>	six characters long, matching the authorization code from the original transaction.
<TotalAmount>	transaction amount (digits only)

Minimum data returned from a successful "VOID" transaction response are:

<TransactionID>	matching the value in the request, above
<ActionCode>	000
<Approval>	character string exactly six characters long
<ResponseText>	the transaction's outcome status

Presuming that the target sale transaction was successfully approved and has not yet settled, the following "VOID" transaction removes that target sale:

```
<JetPay>
<TransactionType>VOID</TransactionType>
<TerminalID>TESTMERCHANT</TerminalID>
<TransactionID>010327153x17T10418</TransactionID>
<CardNum>4000300020001000</CardNum>
<CardExpMonth>12</CardExpMonth>
<CardExpYear>15</CardExpYear>
<TotalAmount>8719</TotalAmount>
```

```
<Approval>502F7B</Approval>
</JetPay>
```

The following shows how the response to the "VOID" transaction would look.

```
<JetPayResponse>
<TransactionID>010327153x17T10418</TransactionID>
<ActionCode>000</ActionCode>
<Approval>502F7B</Approval>
<ResponseText>VOID PROCESSED</ResponseText>
</JetPayResponse>
```

A "VOID" transaction differs from a "CREDIT" transaction inasmuch as the "VOID" transaction doesn't settle while the "CREDIT" transaction does settle. A "VOID" transaction will not appear on a statement, while a "CREDIT" transaction will appear on a statement.

A "VOID" transaction will remove the matching "CAPT", "SALE", or "FORCE" transaction. A "VOID" will not remove an "AUTHONLY" unless a matching "CAPT" transaction has been successfully performed.

If a "VOID" transaction is attempted after the pre-arranged settlement cut-off time, JetPay responds with a "RECORD NOT FOUND" text response. As mentioned above, a settled transaction cannot be VOIDed; a settled transaction can only be reversed using a "CREDIT" transaction.

### 3.6 "REVERSEAUTH" Transaction Example

Because an "AUTHONLY" transaction reserves an "open-to-buy" amount against a cardholder's credit limit, the "REVERSEAUTH" transaction serves to release that open-to-buy amount for the cardholder. A REVERSEAUTH transaction is only effective for uncaptured, unsettled transactions.

A "REVERSEAUTH" transaction releases an approved transaction that has not yet been captured. The JetPay server can only remove uncaptured transaction specified by the REVERSEAUTH transaction, so captured transactions are ineligible for REVERSEAUTHing. The JetPay server responds with "APPROVED", or "RECORD NOT FOUND", or some other "DECLINED" result depending on the outcome of the transaction.

Required data for a "REVERSEAUTH" transaction request are:

<TransactionType>	REVERSEAUTH
<TerminalID>	your terminal ID, assigned by JetPay LLC
<TransactionID>	character string. exactly eighteen characters long, matching the target "AUTHONLY" transaction. (required)
<CardNum>	valid credit card number
<Approval>	six characters long, matching the authorization code from the original transaction.
<TotalAmount>	transaction amount (digits only)

Minimum data returned from a successful "REVERSEAUTH" transaction response are:

<TransactionID>	matching the value in the request, above
<ActionCode>	000
<Approval>	character string exactly six characters long

<ResponseText>	the transaction's outcome status
----------------	----------------------------------

Presuming that the target AUTHONLY transaction was successfully approved and has not yet settled, the following "REVERSEAUTH" transaction removes that target authorization:

```
<JetPay>
<TransactionType>REVERSEAUTH</TransactionType>
<TerminalID>TESTMERCHANT</TerminalID>
<TransactionID>010327153z17T10418</TransactionID>
<CardNum>4000300020001000</CardNum>
<TotalAmount>8719</TotalAmount>
<Approval>502F8B</Approval>
</JetPay>
```

The following shows how the response to the "REVERSEAUTH" transaction would look.

```
<JetPayResponse>
<TransactionID>010327153z17T10418</TransactionID>
<ActionCode>000</ActionCode>
<Approval>502F8B</Approval>
<ResponseText>APPROVED</ResponseText>
</JetPayResponse>
```

### 3.6.1 What is the Difference Between "VOID" and "REVERSEAUTH"?

A "REVERSEAUTH" transaction differs slightly from a "VOID" transaction. Discretely speaking, the "VOID" transaction acts upon *captured* transactions (including "CAPT" and "SALE" and "CREDIT") while the "REVERSEAUTH" acts upon *authorized* transactions (including "AUTHONLY" and "CAPT" and "SALE").

Conceptually speaking, it's easier to distinguish the difference between a "REVERSEAUTH" and a "VOID" by utilizing the following high-level business logic:

- A merchant uses a "VOID" transaction in order to cancel a sale or a refund.
- A merchant uses a "REVERSEAUTH" transaction to release their unused reserve against a cardholder's "open-to-buy". When a cardholder is close their credit limit, it's important for a merchant release any unused reserves on that cardholder's credit limit.

*Note: The card associations plan to institute new rules that promote the release of unused reserves. When a merchant fails to perform a "REVERSEAUTH" on derelict authorizations, the card associations intend to assess fees for failure to release unused reserves. These fees may be assessed by the card association as soon as a few days after an authorization fails to be completed. These new rules may become active soon. JetPay encourages all merchants to perform a "REVERSEAUTH" on their incomplete sales transactions.*

## 3.7 "CREDIT" Transaction Example

The "CREDIT" transaction complements the "SALE" transaction. The "CREDIT" transaction is used to reverse a "SALE", crediting the cardholder's account.

The required data for a "CREDIT" transaction request are the same as for a "SALE" transaction request (except that the TransactionType is "CREDIT" instead of "SALE"). The minimum data returned from a successful "CREDIT" transaction response are almost the same as for a "SALE" transaction response, except that AVS and

CVV2 (and CVC2 and CID) are unavailable.

This particular example also demonstrates submission of a "CREDIT" transaction.

The following is a sample "CREDIT" transaction for \$999.99, using a Visa card:

```
<JetPay>
<TransactionType>CREDIT</TransactionType>
<TerminalID>TESTMERCHANT</TerminalID>
<TransactionID>010327153017T10017</TransactionID>
<CardNum>4000300020001000</CardNum>
<CardExpMonth>12</CardExpMonth>
<CardExpYear>15</CardExpYear>
<TotalAmount>99999</TotalAmount>
</JetPay>
```

The following shows how the successful response would look.

```
<JetPayResponse>
<TransactionID>010327153017T10017</TransactionID>
<ActionCode>000</ActionCode>
<Approval>002F6B</Approval>
<ResponseText>RETURN ACCEPTED</ResponseText>
</JetPayResponse>
```

### 3.8 "ENQ" Transaction Example

An "ENQ" (or "enquire") transaction enables verification and review of a previous transaction, including a previous "AUTHONLY", "SALE", "CAPT", or "FORCE" transaction. In an "ENQ" transaction, JetPay accepts the XML transaction block containing the credit card transaction information, looks up the matching authorized transaction, and responds back to the original sender with the resulting search information.

The matching transaction for this enquire operation must have been authorized or settled through JetPay in order for the "ENQ" transaction to be successful. If a target transaction was not processed through JetPay, the response to the "ENQ" will indicate that the transaction was not found.

Submitted data for a "ENQ" transaction request are:

<TransactionType>	ENQ (required)
<TerminalID>	your terminal ID, assigned by JetPay LLC (required)
<TransactionID>	character string, exactly eighteen characters long, matching the target transaction. (required)
<CardNum>	target credit card number
<CardExpMonth>	credit card expiration month (two digits)
<CardExpYear>	credit card expiration year (two digits)
<Approval>	six characters long, matching the code from the target transaction.
<TotalAmount>	transaction amount, (digits only)

Minimum data returned from a successful "ENQ" transaction response are:

<TransactionID>	matching the value in the request, above
<ActionCode>	000

<Approval>	character string exactly six characters long
<ResponseText>	the transaction's outcome status

You may "ENQ" (enquire) on a JetPay transaction, checking the status of that JetPay transaction:

```
<JetPay>
<TransactionType>ENQ</TransactionType>
<TerminalID>TESTMERCHANT</TerminalID>
<TransactionID>010327153017T10018</TransactionID>
</JetPay>
```

The response would look similar to the original response for the originating transaction.

```
<JetPayResponse>
<TransactionID>010327153017T10018</TransactionID>
<ActionCode>000</ActionCode>
<Approval>502F6B</Approval>
<ResponseText>APPROVED</ResponseText>
</JetPayResponse>
```

### 3.9 "PING" Transaction Example

To check your Internet connection with the JetPay server, you can send a null transaction and receive back an innocuous response. This transaction only verifies the integrity of the Internet connection between the sender and JetPay's interface.

The "PING" transaction allows you to judge the status of your communications link with the JetPay server:

```
<JetPay>
<TransactionType>PING</TransactionType>
<TerminalID>TESTMERCHANT</TerminalID>
<TransactionID>010327153017T10052</TransactionID>
</JetPay>
```

The response to the "PING" request simply returns .

```
<JetPayResponse>
<TransactionID>010327153017T10052</TransactionID>
<ActionCode>000</ActionCode>
<ResponseText>PING</ResponseText>
</JetPayResponse>
```

### 3.10 Error Response

During testing, errors in building a syntactically correct JetPay request message can occur. The JetPay server parses a request message and determines where the XML syntax error occurs within the message. The ErrMsg element is filled in and sent back, informing developers of problems in building a syntactically correct JetPay message.

The following example shows how a response looks when a JetPay syntax error is detected. To generate an error response, submit a request with a misspelled element; for this example, change the <TotalAmount> tag to <Amount>. Of course, since the element is misspelled, you will get the following response:

```
<JetPayResponse>
<TransactionID></TransactionID>
<ActionCode>900</ActionCode>
<ResponseText>INVALID MESSAGE TYPE</ResponseText>
<ErrMsg>
INPUT XML STREAM FAILS PARSING. Faulty JetPay message.

Error on line 11, column 15

Message: Unknown element 'Amount'

</ErrMsg>
</JetPayResponse>
```

Notice that the TransactionID is empty. Because the XML was faulty, the XML parser becomes uncertain of the contents of the message, including the integrity of other XML data within the same message. The only time you should ever receive a JetPay response with no TransactionID is when a faulty JetPay request gets sent.

## 3.11 Additional Examples

JetPay supports common credit card industry features, including address verification, card validation, and cardholder authentication, and recurring transactions. A merchant who submits transactions that subscribe to any of these features can lower their transactions fees or reduce fraud (or both).

Visa and MasterCard call address verification "AVS"; American Express calls it "AAV." By submitting the billing address and postal code of a customer, the merchant receives a result code that indicates whether the merchant's information matches the address information on file with the cardholder's bank. The risk of a credit card transaction is lower when matching billing address information is submitted in a JetPay transaction.

Card validation is achieved through CVV2, CVC2, and CID, collectively called "CVV2" by JetPay. Every credit card has a three- or four-digit CVV2 number on the back (or front) of every physical credit card. A merchant can submit a CVV2 value along with a transaction to validate a physical credit card. When a validated card is submitted, the merchant receives charge back protection because the the existence of the physical credit card is validated as the origin of the credit card number.

Cardholder authentication is featured by Verified by Visa and SecureCode. To authenticate that a customer is the legal cardholder for a credit card, a website allows their customer to log in to Verified by Visa and SecureCode. Then, a credit card transaction proceeds after the cardholder's identity is authenticated. This offers charge back protection to the subscribing merchant whenever Verified by Visa and SecureCode is invoked.

Recurring transactions are used to declare a payment agreement between a merchant and their customer. The three types of recurring transactions are "recurring" and "standing order" and "bill payment" transactions. The "recurring" and "standing order" transactions are used for subscription-based payment (i.e. - monthly payments that remain the same every month). The "bill payment" transaction is used for service-based payment where the actual amount of a payment might differ between payments (i.e. - your electric bill varies from month-to-month). There will be more explanation on this later.

### 3.11.1 Address Verification (AVS) Example

By verifying billing address information, merchants insure that their customers are representing their financial responsibilities in good faith. For example, an online dating service that advertises a trustworthy reputation may wish to verify their subscribers' billing addresses, or a travel agency might wish to verify that they are delivering some expensive plane tickets to a proven billing address. However, regardless of whether AVS is actually necessary for a business to operate, AVS mitigates lower transaction fees for many participating merchants. This is reason alone for any merchant to participate in AVS.

American Express also performs a name-matching operation. By submitting the cardholder's name on an Amex transaction, additional AVS responses are available.

The following "SALE" example illustrates a transaction invoking AVS:

```
<JetPay>
<TransactionType>SALE</TransactionType>
<TerminalID>TESTMERCHANT</TerminalID>
<TransactionID>010327153017T10017</TransactionID>
<CardNum>4000300020001000</CardNum>
<CardExpMonth>12</CardExpMonth>
<CardExpYear>15</CardExpYear>
<TotalAmount>99999</TotalAmount>
<CardName>Bob Tucker</CardName>
<BillingAddress>8800 Central Dr.</BillingAddress>
<BillingCity>Dallas</BillingCity>
<BillingStateProv>TX</BillingStateProv>
<BillingPostalCode>75251</BillingPostalCode>
<BillingCountry>USA</BillingCountry>
<BillingPhone>214-890-1800</BillingPhone>
</JetPay>
```

The following shows how the successful response would look.

```
<JetPayResponse>
<TransactionID>010327153017T10017</TransactionID>
<ActionCode>000</ActionCode>
<Approval>002F6B</Approval>
<ResponseText>APPROVED</ResponseText>
<AddressMatch>Y</AddressMatch>
<ZipMatch>Y</ZipMatch>
<AVS>Y</AVS>
</JetPayResponse>
```

The above example is applicable to "AUTHONLY" transactions in addition to the "SALE" transactions.

JetPay's AVS is automatically invoked whenever an incoming transaction contains billing address data. Merchants may invoke AVS for "AUTHONLY" and "SALE" transactions only. AVS is not available for "CAPT", "FORCE", "CREDIT", or other transactions.

### 3.11.2 Card Validation (CVV2) Example

For card validation, Visa offers CVV2, MasterCard offers CVC2, and American Express offers CID for detecting the presence of a valid physical credit card. JetPay enables these features under the collective name of CVV2.

All physical credit cards have a three- or four-digit CVV2 value imprinted on them. The CVV2 is calculated as a function of the card number, the expiration date, and a secret key value set by the card's issuer. The formula for

calculating CVV2 is a trade secret known only to the credit card associations and the card issuers. After receiving a transaction have a CVV2 along with the credit card number and expiration date, the credit card companies discern the presence of an authentic physical credit card at authorization time. Fraud and risk factors can be weighed.

The following "SALE" example illustrates a transaction invoking CVV2:

```
<JetPay>
<TransactionType>SALE</TransactionType>
<TerminalID>TESTMERCHANT</TerminalID>
<TransactionID>010327153017T10017</TransactionID>
<CardNum>4000300020001000</CardNum>
<CardExpMonth>12</CardExpMonth>
<CardExpYear>15</CardExpYear>
<TotalAmount>99999</TotalAmount>
<CVV2>465</CVV2>
</JetPay>
```

The following shows how the successful response would look.

```
<JetPayResponse>
<TransactionID>010327153017T10017</TransactionID>
<ActionCode>000</ActionCode>
<Approval>002F6B</Approval>
<ResponseText>APPROVED</ResponseText>
<CVV2>M</CVV2>
</JetPayResponse>
```

The above example is applicable to "AUTHONLY" transactions in addition to the "SALE" transactions.

JetPay's CVV2 is automatically invoked whenever an incoming transaction contains a CVV2 value. Merchants may invoke CVV2 for "AUTHONLY" and "SALE" transactions only. CVV2 is definitely not available for "CAPT", "FORCE", "CREDIT", or other transactions.

Never write down or otherwise store a CVV2 value; it isn't allowed. Merchants are advised never to save CVV2 values on any processing system. All card associations cite rules restricting merchants from saving CVV2 values. JetPay LLC is also subject to these rules, so CVV2 values are also not stored in JetPay's system. The CVV2 is only permitted to exist in a processing system for the duration of the transaction. If a merchant is found to be storing CVV2 values, they are subject to being fined by the card associations.

### 3.11.3 Cardholder Authentication (CAVV) Example

Visa and MasterCard offer a system for verifying the person submitting a credit card in a transaction as the true owner of that credit card. Visa offers Verified by Visa (a.k.a. CAVV) and MasterCard offers SecureCode (a.k.a. UCAF) to accomplish cardholder authentication. This feature is currently only available for transactions sent by ecommerce merchants.

To accomplish cardholder authentication, immediately prior to sending an authorization to JetPay, the ecommerce merchant presents their customer to Cardinal Commerce. Cardinal Commerce splashes up a dialog box that queries for a user account and password from the customer. Upon completion of this authentication process, Cardinal Commerce generates a CAVV token. The CAVV token becomes included in the authorization transaction's data.

JetPay accepts the CAVV token within both the "AUTHONLY" and "SALE" transactions. The token serves as proof that a merchant has attempted cardholder authentication. The ecommerce merchant receives charge back protection in return for their authentication attempt.

The following example illustrates a Verified by Visa "SALE" transaction:

```
<JetPay>
<TransactionType>SALE</TransactionType>
<TerminalID>TESTMERCHANT</TerminalID>
<TransactionID>061117133220DOLNPR</TransactionID>
<CardNum CardPresent="false">4000300020001000</CardNum>
<CardExpMonth>05</CardExpMonth>
<CardExpYear>15</CardExpYear>
<TotalAmount>2398</TotalAmount>
<Verification Type = "VbV">
<Cavv Usage = "2" Encoding = "BASE64">BwAQASdSdwUFERdUZfJ3EENDaa4=</Cavv>
<Eci>06</Eci>
<Xid Encoding = "BASE64">uhzrZEtTk/wsRsa6hV3/dbfJEPk=</Xid>
</Verification>
<IndustryInfo Type="ECOMMERCE"></IndustryInfo>
</JetPay>
```

The default format for the CAVV and XID token is in Base 64. It's generally unnecessary to declare the Usage and Encoding format of these tokens because they're rarely manipulated to be any other format.

The response to a Verified by Visa transaction looks like the following:

```
<JetPayResponse>
<TransactionID>061117133220DOLNPR</TransactionID>
<ActionCode>000</ActionCode>
<Approval>079818</Approval>
<ResponseText>APPROVED</ResponseText>
<VerificationResult Type="VbV">2</VerificationResult>
</JetPayResponse>
```

An "AUTHONLY" transaction for with SecureCode looks similar to the previous example:

```
<JetPay>
<TransactionType>AUTHONLY</TransactionType>
<TerminalID>TESTMERCHANT</TerminalID>
<TransactionID>200611170000477007</TransactionID>
<CardNum>5000400030002001</CardNum>
<CardExpMonth>09</CardExpMonth>
<CardExpYear>15</CardExpYear>
<TotalAmount>7998</TotalAmount>
<Verification Type = "SC">
<Cavv>jJDMZodNcgMiARAC1PFZd/YPT4k=</Cavv>
<Xid>IpM03Vsyp9TQzrGDQBK/9TQT1A=</Xid>
<Eci>02</Eci>
</Verification>
</JetPay>
```

Notice that the implicit defaults for Base 64 encoding for the CAVV and XID were assumed.

The SecureCode result is included in the transaction response, and it looks like this:

```
<JetPayResponse>
<TransactionID>200611170000477007</TransactionID>
```

```
<ActionCode>000</ActionCode>
<Approval>871482</Approval>
<ResponseText>APPROVED</ResponseText>
<VerificationResult Type="SC">C</VerificationResult>
</JetPayResponse>
```

### 3.11.4 Recurring/Bill Payment Examples

Recurring transactions are low-risk transactions submitted on a periodic basis. A customer authorizes periodic payments to a merchant, enabling the merchant to submit authorizations to the credit card companies with prior approval, but without the customer's involvement in performing the periodic transactions. The merchant fulfills a periodic subscription or a standing order according to a merchant-customer agreement, and the merchant fills the customer's order automatically without customer interaction.

An example of a merchant who can use recurring payments is an online dating service having a monthly subscription fee. Their customer subscribes to their service, agreeing to a monthly fee automatically applied to their credit card. The merchant sends monthly transactions to fulfill their customer's subscription to their services.

Bill payment transactions are also low-risk transactions that are submitted on a periodic basis. However, for bill payment, a customer's relationship with their merchant is usage-oriented (instead of subscription-oriented). A customer may need to approve a variable monthly bill for fees accrued during that period, and bill payment fits the description of the transactions made by that customer.

Merchants who use bill payment may include utility companies and other companies who send periodic billing to regular customers. The merchant may maintain a website that enables bill payment by their customer.

An example of a "RECURRING" transaction appears below:

```
<JetPay>
<TransactionType>SALE</TransactionType>
<TerminalID>TESTMERCHANT</TerminalID>
<TransactionID>475835825796NWAYeh</TransactionID>
<CardNum CardPresent="false">4000300020001000</CardNum>
<CardExpMonth>07</CardExpMonth>
<CardExpYear>15</CardExpYear>
<TotalAmount>12500</TotalAmount>
<Origin>RECURRING</Origin>
<IndustryInfo Type="RETAIL"></IndustryInfo>
</JetPay>
```

The transaction response is the same as you've observed in the general examples:

```
<JetPayResponse>
<TransactionID>475835825796NWAYeh</TransactionID>
<ActionCode>000</ActionCode>
<Approval>099852</Approval>
<ResponseText>APPROVED</ResponseText>
</JetPayResponse>
```

An example of a "*BILL PAYMENT*" transaction requires a different Origin:

```
<JetPay>
<TransactionType>SALE</TransactionType>
<TerminalID>YMCA00000060</TerminalID>
```

```
<TransactionID>B00000129708368001</TransactionID>
<CardNum CardPresent='false'>4000300020001000</CardNum>
<CardExpMonth>09</CardExpMonth>
<CardExpYear>15</CardExpYear>
<TotalAmount>7400</TotalAmount>
<Origin>BILL PAYMENT</Origin>
</JetPay>
```

Again, the transaction response appears similar to the general examples:

```
<JetPayResponse>
<TransactionID>B00000129708368001</TransactionID>
<ActionCode>000</ActionCode>
<Approval>109534</Approval>
<ResponseText>APPROVED</ResponseText>
</JetPayResponse>
```

When considering the use of recurring transactions, some merchants start with the wrong idea about recurring transactions. The following explanations are significant.

JetPay supports recurring transactions. Recurring transactions are simply transactions that are automatically authorized by an agreement between a merchant and their customer. The merchant is authorized to submit recurring transactions without the customer's direct involvement subsequent to the initial authorization.

JetPay's recurring transactions are not an automated customer billing service. Some merchants mistakenly think that they only have to set up a repeat billing scheme with JetPay and that JetPay will take care of the rest. This is not so.

The purpose of declaring a recurring transaction is to declare a transaction's risk. Recurring transactions are considered low-risk by the credit card industry, and fee schedules for transactions are lower for recurring transactions. The reason for submitting a recurring transaction is to be awarded lower transaction rates, not to set up automated billing.

There are legal reasons why JetPay does not offer automated customer billing. If you are looking for a billing system that automatically charges customer and simply sends you a check every month, then there are numerous legal documents to sign that would define the merchant's responsibilities to their customers and absolve the billing service of liability for unpaid transactions. Although JetPay's services fit into the business model for such customer billing services, JetPay does not supply that automated customer billing service itself.

JetPay is in the business of lowering your transaction fees. When you submit recurring transactions, you're declaring to the credit card companies that you have a legal agreement between you and your customer for payment. Credit card companies award you a lower transaction fee when you submit a recurring transaction.

## 4 Application-Specific Examples

The following examples illustrate JetPay transaction messages, showing a sample request message with a corresponding sample response message.

## 4.1 Explicitly Invoking Alternatives, Using RoutingCode

Some merchants subscribe to alternative back-end processing, which are different than the default back-end connections configured for default JetPay processing. For example, a merchant might have contractual obligations to use a specific third-party processor for their transactions, or a merchant might wish to send transactions in foreign currencies to a processor specializing in banking with those currencies. By explicitly declaring the `RoutingCode`, a merchant can route their JetPay transaction to a non-default connection.

A merchant having diverse banking partners might need to use the `RoutingCode` to declare the desired destination of a transaction to JetPay.

To declare explicit routing to a specific connection, a merchant simply adds the `RoutingCode` element to their transaction with a predetermined eight-character code value. JetPay will then direct that explicitly labeled transaction to the destination processor specified by the `RoutingCode`.

The following is an example of explicit routing using the `RoutingCode`:

```
<JetPay>
<TransactionType>AUTHONLY</TransactionType>
<TerminalID>TESTMERCHANT</TerminalID>
<TransactionID>210227153217T1X018</TransactionID>
<RoutingCode>ForeignB</RoutingCode>
<CardNum>4000300020001000</CardNum>
<CardExpMonth>12</CardExpMonth>
<CardExpYear>15</CardExpYear>
<TotalAmount>8059</TotalAmount>
</JetPay>
```

The following shows a familiar response to that explicitly routed request.

```
<JetPayResponse>
<TransactionID>210227153217T1X018</TransactionID>
<ActionCode>000</ActionCode>
<Approval>512G6B</Approval>
<ResponseText>APPROVED</ResponseText>
</JetPayResponse>
```

If a merchant declares an invalid `RoutingCode` value, JetPay will time out on the transaction with a response (`ActionCode = "992"`) that describes the failure to reach that non-existent destination.

## 4.2 POS Features for the Retail Industry Type

The retail industry consists of brick-and-mortar merchants that submit card-present transactions to JetPay. Transactions are commonly submitted using a POS ("point-of-sale") terminal device having a magnetic stripe reader. Some merchants have developed and deployed POS terminal emulators (similar to JetPay's Virtual Terminal software) to run on standard desktop computers and submit standard retail transactions. All of these POS transactions are classified as a `RETAIL` industry type.

JetPay can process POS (point-of-sale) terminal transactions via the Internet from brick-and-mortar store locations. In these cases, a brick-and-mortar store is presumed to be any shop, restaurant, or hotel that

consummates credit card sales for purchases made with physical credit cards. This document covers the XML features that support POS terminal transactions via JetPay. The JetPay server, constructed by JetPay LLC, supports the POS features described in this document.

For credit card sales, the credit card is physically presented to the merchant to pay for the sale. The merchant may scan data off of the credit card's magnetic stripe using a magnetic card reader, initiating a credit card transaction. The merchant may also manually key in data off of the face of the card to initiate a credit card transaction.

PIN-based debit card sales are also available through JetPay.

This document is targeted at developers of POS terminal software. A developer may be writing software for a POS terminal device capable of network communications. A developer may be enhancing a merchant's custom POS server, used throughout all their retail stores. A developer may be developing a POS terminal emulation program (also known as a "virtual terminal") destined to run as either a website-based application or perhaps as a standalone application. This document covers the XML specifications necessary for communicating a POS transaction to JetPay.

For debit card processing, the reader of this document is presumed to already understand PIN-based debit card processing using a PIN pad. The reader must already understand encryption techniques associated with PIN pads. This document explains the JetPay interface used in setting up PIN-based debit card processing. This document does **not** explain how to initialize a PIN pad, inject a key, *etc.*.

The reader of this document is also presumed to be familiar with XML. XML is a markup language for building so-called "XML documents." JetPay LLC defines the JetPay transactions through XML documents. As you browse through the examples at the end of this document, please keep in mind that each JetPay transaction is an XML document, that XML is case-sensitive, and that JetPay LLC strives to follow XML document conventions for defining its JetPay transactions. JetPay LLC recommends that the reader review XML's authoring rules for building XML documents, educating themselves with an implicit understanding of the underlying XML syntax rules that underpin all JetPay transactions.

For information about the overall specifications of a JetPay transaction, refer to the *JetPay Authorization XML Specification* document.

#### 4.2.1 POS Request Message

The following two tables show the logical structure of the JetPay POS request messages. The **Document Structure** column lists the element name and its place in the message structure in order and indentation. The **Qualifications** column explains the conditions for supplying that element to a transaction.

Element in Document Structure	Qualifications
JetPay	Required root element.
TransactionType	Required
TerminalID	Required
TransactionID	Required, exactly 18-characters in length
CardNumCardPresent=	Required, if Track1 and Track2 are absent.Default is "false".
CardExpMonth	Required, if Track1 and Track2 are absent
CardExpYear	Required, if Track1 and Track2 are absent

Track1	Required, if CardNum and Track2 is absent
Track2	Required, if CardNum and Track1 is absent, or if a contactless magnetic stripe reader is used.
Origin	Required, value is "POS"
ReaderUsed	Conditionally required, values are "KEYPAD", "CHIP", "MAGNETIC STRIPE", and "CONTACTLESS MS".
TotalAmount	Required
IndustryInfoType=	Required when Origin is "POS". "RETAIL" or "HOTEL" or "RESTAURANT" or "PARKING"
Element in Document Structure	Qualifications
IndustryInfoType=	Required root element. when Origin is "POS". "RETAIL" or "HOTEL" or "RESTAURANT" or "PARKING"
TicketNumber	Optional, for Type="RESTAURANT" or "PARKING"
BaseAmount	Optional, for Type="RESTAURANT"
TipAmount	Optional, for Type="RESTAURANT"
ServerID	Optional, for Type="RESTAURANT"
TableNumber	Optional, for Type="RESTAURANT"
FolioNumber	Optional, for Type="HOTEL"
CheckInDate	Optional, for Type="HOTEL"
CheckOutDate	Optional, for Type="HOTEL"
OrigAuthAmount	Optional, for Type="HOTEL"
TotalAuthAmount	Optional, for Type="HOTEL"
RoomRate	Optional, for Type="HOTEL"
RoomNumber	Optional, for Type="HOTEL"
PrestigiousProp	Optional, for Type="HOTEL"
SpecialCode	Optional, for Type="HOTEL"
Extra	Optional, for Type="HOTEL"
PlazaID	Optional, for Type="PARKING"
BoothID	Optional, for Type="PARKING"
ShiftID	Optional, for Type="PARKING"
ClerkID	Optional, for Type="PARKING"

#### 4.2.2 Request Transaction Element Definitions

The following properties describe the root **JetPay** request transactions.

- **JetPay** - root element, enclosing the following transaction elements:
- **TransactionType** - Required for all transactions. Must be one of the following values: "SALE", "AUTHONLY", "CAPT", "VOID", "CREDIT".
  - SALE - Authorize and capture a credit card transaction.
  - AUTHONLY - Authorize a credit card transaction.
  - CAPT - Capture an authorized credit card transaction (see "AUTHONLY").
  - VOID - Remove a captured credit card transaction before settlement.
  - CREDIT - Send funds from the merchant to the cardholder.

- **TerminalID** - Required for all transactions. This ID is issued by JetPay LLC when an account is set up with the merchant's bank.
- **TransactionID** - Required for all transactions. The TransactionID is an 18-character alphanumeric value.
- **CardNum** - The PAN ("personal account number") printed on the face of the credit card used in the credit card transaction.
- **CardExpMonth** - The two-digit month of expiration printed on the face of the credit card. Valid range of values is "01" to "12".
- **CardExpYear** - The two-digit year of expiration printed on the face of the credit card. Valid range of values is "00" to "99".
- **Track1** - The track 1 data gleaned from the magnetic stripe on the back of the credit card.
- **Track2** - The track 2 data gleaned from the magnetic stripe on the back of the credit card.
- **Origin** - Always has a value of "POS" for terminal-based transactions. Other valid Origin values will not designate the transaction as a terminal-based credit card transaction.
- **ReaderUsed** - Description of the specific medium used for collecting the terminal-based data. Types of data-collection media include "KEYPAD", "CHIP", "MAGNETIC STRIPE", and "CONTACTLESS MS". A dual default applies: when the JetPay server detects manually entered data is detected, the default is "KEYPAD", and when track data is detected, the default is "MAGNETIC STRIPE". For a contactless magnetic stripe reader or a chip reader to be designated, the reader must be declared explicitly.
- **TotalAmount** - Total amount of the credit card transaction, including fees, surcharges, and taxes. Digits only. No periods, commas, plus signs, minus signs, dollar signs, or other non-numeric characters.

The following properties describe the root **IndustryInfo** request transactions.

- **IndustryInfo** - root element, with the following attribute and elements:**Type** - A declaration of the "market type" of the merchant's industry. Values of "RETAIL", "HOTEL", "RESTAURANT" or "PARKING" are appropriate for transactions having an Origin with value "POS".
- **TicketNumber** - (Restaurant or parking only) The ticket ID of the customer's restaurant check or parking stub.
- **BaseAmount** - (Restaurant only) The amount of the customer's restaurant check, without tip, taxes, or other service charges.
- **TipAmount** - (Restaurant only) The wait staff's service fee applied to the customer's order.
- **ServerID** - (Restaurant only) The ID of the wait staff employee filling the customer's order.
- **TableNumber** - (Restaurant only) The ID of a customer's dining station.
- **FolioNumber** - (Hotel only) The account ID for a customer hotel stay.
- **CheckInDate** - (Hotel only) The date that a hotel account is to become active. Format is standard XML date format, "CCYY-MM-DD".
- **CheckOutDate** - (Hotel only) The date that a hotel account is to become closed. Format is standard XML date format, "CCYY-MM-DD".
- **OrigAuthAmount** - (Hotel only) The preliminary amount authorized for covering the cost of the entire hotel stay, at the beginning of a hotel stay. Digits only. No periods, commas, plus signs, minus signs, dollar signs, or other non-numeric characters.
- **TotalAuthAmount** - (Hotel only) The actual amount authorized for covering the cost of the entire hotel stay, including incremental authorizations. Digits only. No periods, commas, plus signs, minus signs, dollar signs, or other non-numeric characters.
- **RoomRate** - (Hotel only) The base rate for calculating the cost of a room for a hotel stay.
- **RoomNumber** - (Hotel only) The room at a hotel associated with a hotel stay.
- **PrestigiousProp** - (Hotel only) Member of prestigious property program, guaranteeing the floor limit. Value "D", "B", "S", or "N", where:
  - D - Prestigious property with a \$500 floor limit (in USD).
  - B - Prestigious property with a \$1000 floor limit (in USD).
  - S - Prestigious property with a \$1500 floor limit (in USD).

N - Non-participant.

- **SpecialCode** - (Hotel only) Value "N".
- **PlazaID** - (Parking only) Three-character value identifying a general location for parking booths associated with a transaction.
- **BoothID** - (Parking only) Three-character value identifying a parking booth associated with a transaction.
- **ShiftID** - (Parking only) Two-character value denoting the shift associated with a transaction (i.e. - morning shift, evening shift, graveyard shift).
- **ClerkID** - (Parking only) Six-character value identifying a parking attendant.

#### 4.2.3 POS Response Message

The following table describes the logical structure of the JetPay transaction response message, sent in response to an initiating JetPay POS request message.

Element in Document Structure	Qualifications
JetPayResponse	Required root element.
TransactionID	Present, matching the request's TransactionID.
ActionCode	Present in all responses. "000" = <i>accepted</i> .
Approval	Only present when the ActionCode is "000"
ResponseText	Present.

#### 4.2.4 Response Transaction Element Definitions

The following properties describe the response messages:

- **JetPayResponse** - root element, enclosing the following transaction elements:
- **TransactionID** - This 18-character value matches the TransactionID submitted in the corresponding JetPay request transaction message. This value can be used to correlate the transaction's acknowledgment information with its submitted request data.
- **ActionCode** - The ActionCode is exactly three (3) digits. The ActionCode contains a "000" value to indicate that a transaction has been accepted. The ActionCode will be some other value when the JetPay server host detects an error.
- **Approval** - The Approval is six (6) alphanumeric characters or less. This Approval value is frequently called the "authorization code" or "auth code." The Approval is only present if the ActionCode is "000".
- **ResponseText** - Message from the JetPay server, describing to the ActionCode. The ResponseText is intended to be terse and brief, suitable for display in the presence of a customer. Possible ResponseText values include "APPROVED", "ACCEPTED", "DECLINED", *etc.*

#### 4.2.5 Credit Card Examples

The following examples help to illustrate typical JetPay POS messages, showing sample POS request messages with a corresponding sample response messages.

#### 4.2.5.1 Magnetic Stripe POS Retail Sale Transaction Example

A magnetic stripe reader can collect basic credit card information. By passing the magnetic stripe data in a "SALE" or "AUTHONLY" transaction, the issuer examines the magnetic stripe data to determine its authenticity and approves or declines a transaction accordingly. The integrity of magnetic stripe data is generally high, so these magnetic stripe POS transactions are preferred over other methodologies; there is a lower transactional cost for submitting the magnetic stripe data within a transaction.

Either track 1 or track 2 information must be submitted with a magnetic stripe transaction. At least one of the two tracks is required. The credit card must always be assumed to be present whenever track 1 or track 2 information is submitted, so there is no CardPresent attribute in a magnetic stripe transaction. The Origin defaults to "POS", the IndustryInfo Type defaults to "RETAIL", and the ReaderUsed defaults to "MAGNETIC STRIPE", but it's a good programming practice to explicitly declare these values instead of relying on the defaults.

If you need to submit a RESTAURANT sale transaction which includes support for tips and additional charges, advance to the [next sub-section](#) for an example.

A basic magnetic stripe POS transaction request contains the following data:

<TransactionType>	SALE, AUTHONLY, CREDIT, or VOID.
<TerminalID>	your terminal ID, assigned by JetPay LLC
<TransactionID>	character string, exactly eighteen alphanumeric characters long, preferably a unique value.
<Track1> <Track2>	the character information contained in either the track 1 or track 2 magnetic stripes, including the beginning and ending sentinel characters.
<Origin>	POS.
<ReaderUsed>	<b>MAGNETIC STRIPE</b> (optional).
<TotalAmount>	total amount of the transaction, including any taxes, fees, and surcharges (digits only).
<IndustryInfo Type = "RETAIL" >	A declaration of the Type attribute for this market segment is needed. Appropriate values for the Type are "RETAIL", "HOTEL", "RESTAURANT" and "PARKING".

Data returned in a magnetic stripe transaction response are:

<TransactionID>	matching the value in the request, above
<ActionCode>	000 or other three-digit value
<Approval>	six alphanumeric characters or less (or absent).
<ResponseText>	APPROVED or DECLINED

This sample retail magnetic stripe transaction is a sale for \$12.99:

```
<JetPay>
<TransactionType>SALE</TransactionType>
<TerminalID>TESTMERCHANT</TerminalID>
<TransactionID>MS0327153017T10038</TransactionID>
<Track1>%B4000300020001000^MAVERICK INTERNATIONAL^0712101000000000000000?</Track1>
<Track2>;4000300020001000=07121011139300000378?</Track2>
<Origin>POS</Origin>
<ReaderUsed>MAGNETIC STRIPE</ReaderUsed>
<TotalAmount>1299</TotalAmount>
<IndustryInfo Type='RETAIL'></IndustryInfo>
</JetPay>
```

Do not rely on default values for Origin, ReaderUsed, and IndustryInfo Type. When track data is sent, the Origin value defaults to "POS", the ReaderUsed value defaults to "MAGNETIC STRIPE", and the IndustryInfo's Type defaults to "RETAIL". When track data is absent, these defaults change to internet-biased values. When building a POS terminal program, do not rely on default values for relating terminal information; better results are obtained when values are explicitly declared.

Note that the above example shows no CardPresent attribute. When sending magnetic stripe data, a physical credit card is automatically presumed to be present. Therefore, it is unnecessary to explicitly operate any CardPresent attribute because its value cannot ever be "false". Saving track data is a card association violation, so it is patently invalid to declare the CardPresent attribute as "false" (and therefore redundant to declare it to be "true"). When sending track data, ignore any CardPresent attribute because its value is constant.

The following shows a sample response to the above sale request.

```
<JetPayResponse>
<TransactionID>MS0327153017T10038</TransactionID>
<ActionCode>000</ActionCode>
<Approval>1234A6</Approval>
<ResponseText>APPROVED</ResponseText>
</JetPayResponse>
```

In the above example, note that both the track 1 and track 2 data were sent within the transaction request. Only one of the two tracks is required, so when both tracks are sent, the JetPay server will use the track data from track 1.

If the above transaction had been declined, the sample response might be:

```
<JetPayResponse>
<TransactionID>MS0327153017T10038</TransactionID>
<ActionCode>005</ActionCode>
<ResponseText>DECLINED</ResponseText>
</JetPayResponse>
```

#### 4.2.5.2 Magnetic Stripe Transaction Example for Restaurant

As mentioned in the previous example, the issuer examines the magnetic stripe data to determine its authenticity and approves or declines a transaction accordingly. The "AUTHONLY" transaction enables the allocation of open-to-buy funds without actually finalizing the sale. A subsequent "CAPT" transaction is used to finalize the transaction by submitting the adjusted total amount.

Many restaurants need this two-step authorize-and-capture procedure, providing a means for adjusting the final authorized total after adding a tip for the wait staff. By sending an "AUTHONLY" transaction with the magnetic stripe information, the restaurant obtains an initial authorization for the subtotal. The subsequent "CAPT" transaction contains the tip amount and the total amount signed for by the cardholder.

Refer to the previous example for a description of tags and values to be used in a magnetic stripe transaction.

Suppose that a restaurant transaction is initiated on a \$35.00 subtotal, such that a tax of 8.25% is added onto the bill. This sample retail restaurant magnetic stripe transaction is the resulting "AUTHONLY" authorization request for \$37.89:

```
<JetPay>
```

## JetPay Authorization XML Specification © 2012 JetPay LLC.

```
<TransactionType>AUTHONLY</TransactionType>
<TerminalID>TESTMERCHANT</TerminalID>
<TransactionID>MS0327153017R10030</TransactionID>
<Track1>%B4000300020001000^MAVERICK INTERNATIONAL^071210100000000000000000?</Track1>
<Track2>;4000300020001000=07121011139300000378?</Track2>
<Origin>POS</Origin>
<ReaderUsed>MAGNETIC STRIPE</ReaderUsed>
<TotalAmount>3789</TotalAmount>
<TaxAmount>289</TaxAmount>
<IndustryInfo Type='RESTAURANT'>
<TicketNumber>3333</TicketNumber>
<BaseAmount>3500</BaseAmount>
<TipAmount>0</TipAmount>
<ServerID>CHRIS5</ServerID>
<TableNumber>25</TableNumber>
</IndustryInfo>
</JetPay>
```

The following shows a sample approval response for the above authorization request.

```
<JetPayResponse>
<TransactionID>MS0327153017R10030</TransactionID>
<ActionCode>000</ActionCode>
<Approval>1234B6</Approval>
<ResponseText>APPROVED</ResponseText>
</JetPayResponse>
```

At this point, a restaurant presents the authorized transaction to the cardholder, intending to add the tip amount and receive back a signature from the cardholder. The tip amount gets reported along with the adjusted total amount in the "CAPT" transaction.

The cardholder adds a \$6.00 tip to the authorization. This "CAPT" request finalizes the preceding authorization example to an adjusted total amount of \$43.89:

```
<JetPay>
<TransactionType>CAPT</TransactionType>
<TerminalID>TESTMERCHANT</TerminalID>
<TransactionID>MS0327153017R10030</TransactionID>
<TotalAmount>4389</TotalAmount>
<TaxAmount>289</TaxAmount>
<IndustryInfo Type='RESTAURANT'>
<TicketNumber>3333</TicketNumber>
<BaseAmount>3500</BaseAmount>
<TipAmount>600</TipAmount>
<ServerID>CHRIS5</ServerID>
<TableNumber>25</TableNumber>
</IndustryInfo>
</JetPay>
```

The following shows a sample approval response for the above authorization request.

```
<JetPayResponse>
<TransactionID>MS0327153017R10030</TransactionID>
<ActionCode>000</ActionCode>
<Approval>1234B6</Approval>
<ResponseText>APPROVED</ResponseText>
</JetPayResponse>
```

Note that the TransactionID of the "CAPT" transaction is the same as that of the preceding "AUTHONLY" transaction. In cases where the "CAPT" amount is different from the "AUTHONLY" amount, the TransactionIDs of these paired transactions must match.

Throughout these "AUTHONLY" and "CAPT" examples, the TotalAmount always reflects the grand total of the total purchase, including the base amount plus tax and plus tip. Although you may omit the BaseAmount, TaxAmount, and TipAmount, the TotalAmount still must always be submitted as the grand total for a complete transaction.

The "CAPT" amount will be the settled amount of a transaction. The "AUTHONLY" amount represents the authorized amount. Except for restaurants, JetPay LLC advises that the "CAPT" amount should be the same as the "AUTHONLY" amount (because of increased transactional fees when the amounts are not the same); restaurants avoid these increased fees because tip-adjusted amounts are a common business model for restaurant sales.

#### 4.2.5.3 Manually Entered POS Transaction Example

Sometimes the magnetic stripe data cannot be read on a credit card. When that happens, manual data entry on a keypad becomes the alternative. It's practical to build a POS terminal capable of manual data entry.

A manually entered POS transaction looks similar to the internet-based examples within this document. *The difference between a POS transaction and an E-commerce transaction is the declaration of non-default values for the transaction.* The JetPay server will interpret a manually entered transaction as a POS transaction only when the Origin, ReaderUsed, and IndustryInfo Type are explicitly declared. For a manually entered POS transaction, you are required to explicitly add the Origin, ReaderUsed, and IndustryInfo Type.

A basic manually entered POS transaction request contains the following data:

<TransactionType>	SALE, AUTHONLY, CREDIT, or VOID.
<TerminalID>	your terminal ID, assigned by JetPay LLC
<TransactionID>	character string, exactly eighteen alphanumeric characters long, preferably a unique value.
<CardNum CardPresent='true'>	the credit card number, or PAN.
<CardExpMonth>	the expiration month, "01" through "12".
<CardExpYear>	the last two digits of the expiration year
<Origin>	POS.
<ReaderUsed>	KEYPAD.
<TotalAmount>	total amount of the transaction, including any taxes, fees, and surcharges (digits only).
<IndustryInfo Type = "RETAIL" >	A declaration of the Type attribute for this market segment is needed. Appropriate values for the Type are "RETAIL", "HOTEL", "RESTAURANT" and "PARKING".

Data returned in a keyed transaction response are:

<TransactionID>	matching the value in the request, above
<ActionCode>	000 or other three-digit value
<Approval>	six alphanumeric characters or less (or absent).
<ResponseText>	APPROVED or DECLINED

This sample retail transaction is a manually entered sale for \$23.98:

```
<JetPay>
<TransactionType>SALE</TransactionType>
<TerminalID>TESTMERCHANT</TerminalID>
<TransactionID>ME0327153017T10039</TransactionID>
<CardNum CardPresent='true'>4000300020001000</CardNum>
<CardExpMonth>12</CardExpMonth>
<CardExpYear>15</CardExpYear>
<Origin>POS</Origin>
<ReaderUsed>KEYPAD</ReaderUsed>
<TotalAmount>2398</TotalAmount>
<IndustryInfo Type='RETAIL'></IndustryInfo>
</JetPay>
```

The following shows a sample response to the above sale request.

```
<JetPayResponse>
<TransactionID>ME0327153017T10039</TransactionID>
<ActionCode>000</ActionCode>
<Approval>Q2E4A6</Approval>
<ResponseText>APPROVED</ResponseText>
</JetPayResponse>
```

In the above example, please note that the CardPresent attribute is set to "true". This means that the credit card is in the presence of the merchant. Because the default value of the CardPresent attribute is "false", it becomes important to declare CardPresent explicitly in every transaction. For all retail POS transactions, the CardPresent attribute should always be explicitly set to either "true" or "false".

Note: The value of the CardPresent attribute indicates tangible availability of a physical credit card to a merchant during a transaction. The CardPresent attribute can be set to "true" whenever a merchant can easily handle and view a credit card during a transaction. The CardPresent attribute must be set to "false" whenever only the PAN (that is, the credit card number) is available to a merchant.

For manually entered transactions, the default value for Origin is "INTERNET" and the default value for the IndustryInfo Type is "ECOMMERCE". You must always explicitly declare the Origin (as "POS") and IndustryInfo Type (as "RETAIL", "HOTEL", "RESTAURANT", or "PARKING") in every manually entered POS transaction.

The default value of ReaderUsed for manually entered card numbers is "KEYPAD", but JetPay LLC advises to explicitly declare the value of ReaderUsed anyway for the sake of consistency, regardless of the default.

#### 4.2.5.4 Contactless Track POS Transaction Example

The contactless track readers can collect credit card information when an operator touches the reader with a credit card. The issuer examines the contactless track data in a similar manner to the tracks on a magnetic stripe; the issuer determines a card's authenticity and approves or declines a transaction accordingly.

Track 2 data must be submitted as the contactless track transaction. In a coming release of the JetPay server, both contactless tracks will be able to be submitted.

A basic contactless track POS transaction request contains the following data:

<TransactionType>	SALE, AUTHONLY, CREDIT, or VOID.
<TerminalID>	your terminal ID, assigned by JetPay LLC

<TransactionID>	character string, exactly eighteen alphanumeric characters long, preferably a unique value.
<Track2>	the character information contained in track 2 magnetic stripes, including the beginning and ending sentinel characters.
<Origin>	POS.
<ReaderUsed>	<b>CONTACTLESS MS.</b>
<TotalAmount>	total amount of the transaction, including any taxes, fees, and surcharges (digits only).
<IndustryInfo Type="RETAIL">	A declaration of the Type attribute for this market segment is needed. Appropriate values for the Type are "RETAIL", "HOTEL", "RESTAURANT" and "PARKING".

Data returned in a magnetic stripe transaction response are:

<TransactionID>	matching the value in the request, above
<ActionCode>	000 or other three-digit value
<Approval>	six alphanumeric characters or less (or absent).
<ResponseText>	APPROVED or DECLINED

This sample retail contactless track transaction is a sale for \$10.23, and it's very similar to a magnetic stripe transaction:

```
<JetPay>
<TransactionType>SALE</TransactionType>
<TerminalID>TESTMERCHANT</TerminalID>
<TransactionID>CS0327153017T10040</TransactionID>
<Track2>;4000300020001000=07121011139300000378?</Track2>
<Origin>POS</Origin>
<ReaderUsed>CONTACTLESS MS</ReaderUsed>
<TotalAmount>1023</TotalAmount>
<IndustryInfo Type='RETAIL'></IndustryInfo>
</JetPay>
```

The following shows a sample response to the above sale request.

```
<JetPayResponse>
<TransactionID>CS0327153017T10040</TransactionID>
<ActionCode>000</ActionCode>
<Approval>5234A6</Approval>
<ResponseText>APPROVED</ResponseText>
</JetPayResponse>
```

In the above example, only track 2 data was sent within the transaction request. Only track 2 is enabled, so it's important that JetPay server receives the track data from track 2.

For contactless track transactions, the Origin, ReaderUsed, and IndustryInfo Type must always be explicitly submitted. The Origin value is "POS", the ReaderUsed value is "CONTACTLESS MS", and the IndustryInfo's Type should be "RETAIL", "HOTEL", "RESTAURANT", or "PARKING".

#### 4.2.6 Debit and EBT Card Examples

Debit card transactions feature PIN-based operation, where a cardholder is allowed to enter a secret PIN on a PIN pad as part of submitting a debit card transaction. Without a PIN, a debit card transaction is just like any other credit card transaction. In order to submit a debit card transaction, a developer must implement PIN support as part of the submission process.

Debit card transactions also require the submission of track 2 data. Without track 2 data, it becomes impossible to encrypt or decrypt PIN numbers (because the magnetic stripe data is used as part of PIN encryption/decryption). If track data is not submitted, a debit card transaction becomes impossible and the transaction becomes a manually-entered credit card transaction.

Debit card transactions require the availability of two hardware devices: a PIN pad and a magnetic stripe reader. Unless a PIN pad (which is configurable with a working key) and a magnetic stripe reader are both available and operational, these debit card transaction examples cannot be applied. A debit card transaction requires both track data and an encrypted PIN to proceed.

The following examples help to illustrate JetPay POS messages for debit card sales, showing sample POS request messages with a corresponding sample response messages.

#### 4.2.6.1 Debit POS Sale Transaction Example

When submitting a debit sale transaction, an encrypted PIN block is submitted with the authorization. The encrypted binary PIN block is communicated by a PIN pad as a sixteen-byte string of hexadecimal text.

The PIN block is submitted along with the track data for the transaction:

```
<JetPay>
  <TransactionType>SALE</TransactionType>
  <TerminalID>TESTMERCHANT</TerminalID>
  <TransactionID>KEAUWSZPDFRQNJYXIG</TransactionID>
  <Track2>;4123456789012349=16121011000012345678?</Track2>
  <Origin>POS</Origin>
  <ReaderUsed>MAGNETIC STRIPE</ReaderUsed>
  <TotalAmount>1200</TotalAmount>
  <Debit Type="DIRECT">
    <PinBlock Encoding="HEX" Management="DUKPT">FAFBF12344565667</PinBlock>
    <Ksn>4432566374900000009</Ksn>
  </Debit>
  <IndustryInfo Type="RETAIL"></IndustryInfo>
</JetPay>
```

The response appears similar to that of any other "SALE" transaction.

```
<JetPayResponse>
  <TransactionID>KEAUWSZPDFRQNJYXIG</TransactionID>
  <ActionCode>000</ActionCode>
  <Approval>5234A6</Approval>
  <ResponseText>APPROVED</ResponseText>
</JetPayResponse>
```

Remember that all PIN-based transactions require the submission of valid track data in order for the PIN to be properly decrypted. Without track data, PIN submission becomes unnecessary and the transaction is better submitted as a manually-entered credit card transaction (without a PIN).

#### 4.2.6.2 EBT POS Sale Transaction Example

When submitting an EBT sale transaction, an encrypted PIN block is submitted with the authorization similar to a debit card. The encrypted binary PIN block gets communicated by a PIN pad as a sixteen-byte string of hexadecimal text.

The PIN block and FCS identifier is submitted along with the track data for the transaction:

```
<JetPay>
  <TransactionType>SALE</TransactionType>
  <TerminalID>TESTMERCHANT</TerminalID>
  <TransactionID>JJYT3456DFRQNJUECN</TransactionID>
  <Track2>;1234123412341234=16121011000012345678?</Track2>
  <Origin>POS</Origin>
  <ReaderUsed>MAGNETIC STRIPE</ReaderUsed>
  <TotalAmount>1500</TotalAmount>
  <Debit Type="EBT FOOD">
    <PinBlock Encoding="HEX" Management="DUKPT">FAFBF89064563215</PinBlock>
    <Ksn>44325663748000000003</Ksn>
    <FcsID>9876543</FcsID>
  </Debit>
  <IndustryInfo Type="RETAIL"></IndustryInfo>
</JetPay>
```

The response appears similar to that of any other "SALE" transaction.

```
<JetPayResponse>
  <TransactionID>JJYT3456DFRQNJUECN</TransactionID>
  <ActionCode>000</ActionCode>
  <Approval>5234A6</Approval>
  <ResponseText>APPROVED</ResponseText>
</JetPayResponse>
```

Remember that all PIN-based transactions require the submission of valid track data in order for the PIN to be properly decrypted.

## 4.3 Industry-Specific Examples

JetPay accepts some industry-specific data to be submitted for many market verticals. By default, JetPay supports the retail and ecommerce verticals; all other verticals must explicitly declare their industry type within a transaction.

The verticals recognized and supported by JetPay include:

- **RETAIL** This retail industry type is presumed when magnetic track data is submitted, or when the merchant declares a "card present" condition for a transaction.
- **ECOMMERCE** This ecommerce industry type becomes the default for all "card not present" transactions.
- **MOTO** The mail order and telephone order industry vertical must declare this industry type in all their transactions.
- **HOTEL** The hotel and lodging industry submit numerous industry-specific data for this vertical, including room number, room rates, *etc.*
- **RESTAURANT** Restaurants and food preparers declare this vertical and may submit industry-specific data, including tip amount and food order data.
- **PARKING** Paid-parking companies can receive reports for their submitted parking transaction data.
- **AUTORENTAL** The auto rental industry submits industry-specific data for their vertical.
- **QUASICASH** Cash advance. Used by the loan industry and the gaming industry at brick-and-mortar (non-internet) locations.

### 4.3.1 Ecommerce Industry-Specific Example

Ecommerce merchants transact card-not-present transactions via an Internet website and the default industry type for card-not-present transactions is ecommerce. Although JetPay assumes several default values for ecommerce transactions, it's useful for merchants to explicitly declare those defaults.

Billing and shipping information is also applicable in ecommerce transactions. Although all are optional, JetPay recommends that at least the BillingAddress and BillingPostalCode should always be submitted to enable AVS.

Implicit defaults for an ecommerce transaction request are in **bold** type:

- <Origin>INTERNET</Origin>
- <CardNum CardPresent=**"false"**>
- <IndustryInfo Type=**"ECOMMERCE"**>

Optional data included in an ecommerce transaction are:

<BillingAddress> <BillingCity> <BillingStateProv> <BillingPostalCode> <BillingCountry> <BillingPhone>	the billing address information and billing postal code information enable AVS processing (recommended).
<Email>	the customer's email address.
<UserIPAddress>	the customer's IP address from which the transaction originated.
<UserHost>	the name of the customer's server.
<ShippingInfo> <CustomerPO>	the customer's purchase order number.
<ShippingInfo> <ShippingMethod>	delivery method for the purchase.
<ShippingInfo> <ShippingName>	the recipient's name.
<ShippingInfo> <ShippingAddr>	
<ShippingInfo> <Address>	the customer's shipping address info.
<ShippingInfo> <City>	
<ShippingInfo> <StateProv>	
<ShippingInfo> <PostalCode>	
<ShippingInfo> <Country>	
<ShippingInfo> <Phone>	

<IndustryInfo Type="ECOMMERCE"> <UserAgent>	the customer's HTTP browser type.
---	-----------------------------------

The following example illustrates a "SALE" transaction while explicitly declaring all default values. The explicitly declared defaults and optional data appear in bold print in the following example:

```

<JetPay>
<TransactionType>SALE</TransactionType>
<TerminalID>TESTMERCHANT</TerminalID>
<TransactionID>010327153017T10017</TransactionID>
<Origin>INTERNET</Origin>
<CardNum CardPresent="false">4000300020001000</CardNum>
<CVV2>1234</CVV2>
<CardExpMonth>12</CardExpMonth>
<CardExpYear>15</CardExpYear>
<CardName>Bob Tucker</CardName>
<TotalAmount>99999</TotalAmount>
<BillingAddress>8800 Central Dr.</BillingAddress>
<BillingCity>Dallas</BillingCity>
<BillingStateProv>TX</BillingStateProv>
<BillingPostalCode>75251</BillingPostalCode>
<BillingCountry>USA</BillingCountry>
<BillingPhone>214-890-1800</BillingPhone>
<Email>btucker@jetpay.com</Email>
<UserIPAddress>123.456.789.000</UserIPAddress>
<UserHost>PHX.QW.AOL.COM</UserHost>
<ShippingInfo>
<CustomerPO>ABC12345</CustomerPO>
<ShippingMethod>OVERNIGHT</ShippingMethod>
<ShippingName>John Public</ShippingName>
<ShippingAddr>
<Address>3150 139th Avenue SE</Address>
<City>Bellevue</City>
<StateProv>WA</StateProv>
<PostalCode>98005</PostalCode>
<Country>USA</Country>
<Phone>123456789</Phone>
</ShippingAddr>
</ShippingInfo>
<IndustryInfo Type="ECOMMERCE">
<UserAgent>MOZILLA/4.0 (COMPATIBLE; MSIE 5.0; WINDOWS 95)</UserAgent>
</IndustryInfo>
</JetPay>

```

Above, notice that the Origin, the CardPresent attribute, and the IndustryInfo Type attribute contain explicit values. For ecommerce transactions, the Origin is always "INTERNET", the CardPresent attribute always has a "false" value, and the IndustryInfo Type attribute always has a value of "ECOMMERCE".

Please note that the above example includes AVS ("Address Verification Service") data, including the BillingAddress, BillingPostalCode, and so on. Although the submission of AVS information is always optional, JetPay recommends that ecommerce merchants submit all billing information in every ecommerce transaction. Discounted transaction fees and enhanced charge back protection may be awarded to merchants submitting AVS data.

It is considered a good programming practice to always explicitly declare your default values and not depend on implicit defaults for any contingency.

### 4.3.2 MOTO Industry-Specific Examples

MOTO is an industry term that stands for "Mail Order / Telephone Order" merchant. MOTO transactions must explicitly declare certain industry-specific data within every submitted transaction. The only default that a MOTO transaction may presume is that the CardPresent attribute is always "false."

Billing and shipping information is also applicable in MOTO transactions. Although all are optional, JetPay recommends that at least the BillingAddress and BillingPostalCode should always be submitted to enable AVS.

The explicit values require for declaring a MOTO transaction request appear below:

- <CardNum CardPresent="false">
- <IndustryInfo Type="MOTO">

The Origin values must be one of two possible values:

- <Origin>MAIL ORDER</Origin>
- <Origin>PHONE ORDER</Origin>

Optional data included in an ecommerce transaction are:

<BillingAddress> <BillingCity> <BillingStateProv> <BillingPostalCode> <BillingCountry> <BillingPhone>	the billing address information and billing postal code information enable AVS processing (recommended).
<Email>	the customer's email address.
<ShippingInfo> <CustomerPO>	the customer's purchase order number.
<ShippingInfo> <ShippingMethod>	delivery method for the purchase.
<ShippingInfo> <ShippingName>	the recipient's name.
<ShippingInfo> <ShippingAddr>	
<ShippingInfo> <Address>	the customer's shipping address info.
<ShippingInfo> <City>	
<ShippingInfo> <StateProv>	
<ShippingInfo> <PostalCode>	
<ShippingInfo> <Country>	

<ShippingInfo> <Phone>	
<IndustryInfo Type="MOTO"> <PhoneANI>	ANI phone number for the phone order.
<IndustryInfo Type="MOTO"> <PhoneII>	information identifier coding digits.

The following example illustrates a typical phone order sale. The sale illustrated here was transacted via telephone. MOTO-specific data appears in bold print:

```

<JetPay>
<TransactionType>SALE</TransactionType>
<TerminalID>TESTMERCHANT</TerminalID>
<TransactionID>010327153017T10017</TransactionID>
<Origin>PHONE ORDER</Origin>
<CardNum CardPresent="false">4000300020001000</CardNum>
<CardExpMonth>12</CardExpMonth> <CardExpYear>15</CardExpYear>
<TotalAmount>99999</TotalAmount>
<BillingAddress>8800 Central Dr.</BillingAddress>
<BillingCity>Dallas</BillingCity>
<BillingStateProv>TX</BillingStateProv>
<BillingPostalCode>75251</BillingPostalCode>
<BillingCountry>USA</BillingCountry>
<BillingPhone>214-890-1800</BillingPhone>
<Email>btucker@jetpay.com</Email>
<ShippingInfo>
<CustomerPO>XYZ54321</CustomerPO>
<ShippingMethod>GROUND</ShippingMethod>
<ShippingName>John Public</ShippingName>
<ShippingAddr>
<Address>3150 139th Avenue SE</Address>
<City>Bellevue</City>
<StateProv>WA</StateProv>
<PostalCode>98005</PostalCode>
<Country>USA</Country>
</ShippingAddr>
</ShippingInfo>
<IndustryInfo Type="MOTO">
<PhoneANI>9725038900</PhoneANI>
<PhoneII>00</PhoneII>
</IndustryInfo>
</JetPay>

```

Alternatively, a mail order sale looks like this.

```

<JetPay>
<TransactionType>SALE</TransactionType>
<TerminalID>TESTMERCHANT</TerminalID>
<TransactionID>010327153017T10017</TransactionID>
<Origin>MAIL ORDER</Origin>
<CardNum CardPresent="false">4000300020001000</CardNum>
<CardExpMonth>12</CardExpMonth> <CardExpYear>15</CardExpYear>
<TotalAmount>99999</TotalAmount>
<BillingAddress>8800 Central Dr.</BillingAddress>
<BillingCity>Dallas</BillingCity>
<BillingStateProv>TX</BillingStateProv>
<BillingPostalCode>75251</BillingPostalCode>
<BillingCountry>USA</BillingCountry>
<BillingPhone>214-890-1800</BillingPhone>
<Email>btucker@jetpay.com</Email>

```

```

<ShippingInfo>
<CustomerPO>XYZ54321</CustomerPO>
<ShippingMethod>GROUND</ShippingMethod>
<ShippingName>John Public</ShippingName>
<ShippingAddr>
<Address>3150 139th Avenue SE</Address>
<City>Bellevue</City>
<StateProv>WA</StateProv>
<PostalCode>98005</PostalCode>
<Country>USA</Country>
<Phone>5556667777</Phone>
</ShippingAddr>
</ShippingInfo>
<IndustryInfo Type="MOTO"></IndustryInfo>
</JetPay>

```

Notice that, for the "MAIL ORDER" transaction, the PhoneANI and PhoneII tags are not used. The PhoneANI and PhoneII values are used only for "PHONE ORDER" transactions.

Although the submission of AVS information is always optional, JetPay recommends that MOTO merchants submit all billing information in every ecommerce transaction. Discounted transaction fees and enhanced charge back protection may be awarded to merchants submitting AVS data.

AVS and billing information is omitted in the above examples solely for the sake of brevity. Although AVS participation is optional, MOTO merchants are urged to always submit AVS data in every transaction. Refer to the prior "SALE" examples in earlier sections for an example of an AVS submission.

### 4.3.3 Hotel AUTHONLY Example

JetPay supports lodging merchant with an array of multiple transaction types. These transactions differ from others in that additional charges can be incrementally added and reversed. This example describes a scenario for submitting an initial authorization:

Hotel-specific data for an "AUTHONLY" transaction request include:

<IndustryInfo Type="HOTEL"> <FolioNumber>	maximum length of ten characters, preferably a unique value. Required.
<IndustryInfo Type="HOTEL"> <CheckInDate>	date format: <b>yyyy-mm-dd</b> . Required.
<IndustryInfo Type="HOTEL"> <CheckOutDate>	date format: <b>yyyy-mm-dd</b> . Required.
<IndustryInfo Type="HOTEL"> <RoomRate>	base room rate, digits only.
<IndustryInfo Type="HOTEL"> <RoomNumber>	room number, characters permitted.
<IndustryInfo Type="HOTEL"> <ExtraCharges>	valid values are "RESTAURANT", "GIFTSHOP", "MINIBAR", "TELEPHONE", "LAUNDRY", "OTHER".
<IndustryInfo Type="HOTEL"> <NoShow>	"true" indicates a failure to check-in."false" indicates a successful check-in.

Following the rules for submitting a hotel authorization the XML "AUTHONLY" example becomes the following:

```
<JetPay>
```

```

<TransactionType>AUTHONLY</TransactionType>
<TerminalID>TESTTERMINAL</TerminalID>
<TransactionID>TransactionID22222</TransactionID>
<CardExpMonth>12</CardExpMonth>
<CardExpYear>15</CardExpYear>
<TotalAmount>10000</TotalAmount>
<CardNum CardPresent="true">4000300020001000</CardNum>
<IndustryInfo Type="HOTEL" >

<FolioNumber>Folio12345</FolioNumber>
<CheckInDate>2007-01-01</CheckInDate>
<CheckOutDate>2007-01-03</CheckOutDate>
<RoomRate>5000</RoomRate>
<RoomNumber>123</RoomNumber>
<ExtraCharges>RESTAURANT</ExtraCharges>
<ExtraCharges>MINIBAR</ExtraCharges>
<ExtraCharges>LAUNDRY</ExtraCharges>
<NoShow>>false</NoShow>
</IndustryInfo>
</JetPay>

```

#### 4.3.4 Hotel INCREMENTAL Example

An "INCREMENTAL" transaction enables a hotel to increase the authorized amount of an existing "AUTHONLY" transaction. The "INCREMENTAL" transaction requires the submission of a matching FolioNumber to successfully complete the transaction.

The following example increases an authorization, matching on the FolioNumber of the initial authorization in the previous "AUTHONLY" example. The TotalAmount field is the additional amount to be added to an existing authorized amount. This example INCREMENTAL transaction will add \$75.65 to the existing authorization for a total authorized amount of \$175.65:

```

<JetPay>
<TransactionType> INCREMENTAL </TransactionType>
<TerminalID> TESTTERMINAL </TerminalID>
<TransactionID> TransactionID22222 </TransactionID>
<CardExpMonth>12</CardExpMonth>
<CardExpYear>15</CardExpYear>
<TotalAmount>7565</TotalAmount>
<CardNum CardPresent="false">4000300020001000</CardNum>
<IndustryInfo Type="HOTEL">
<FolioNumber>Folio12345</FolioNumber>
</IndustryInfo>
</JetPay>

```

If a FolioNumber is found not to match any previous transaction, a "LODGING TRANSACTION NOT FOUND" response is returned.

#### 4.3.5 Hotel PARTIALREVERSAL Example

The "PARTIALREVERSAL" transaction enables a hotel to decrease the authorized amount of an existing "AUTHONLY" transaction. The "PARTIALREVERSAL" transaction requires the submission of a matching FolioNumber to successfully complete the transaction.

The following is a "PARTIALREVERSAL" transaction, matching by FolioNumber. The TotalAmount value contained in the transaction becomes the final authorization amount, not the amount to be deducted from the existing authorization:

```
<JetPay>
<TransactionType>PARTIALREVERSAL</TransactionType>
<TerminalID>TESTTERMINAL</TerminalID>
<TransactionID>TransactionID22222</TransactionID>
<CardExpMonth>12</CardExpMonth>
<CardExpYear>15</CardExpYear>
<TotalAmount>5495</TotalAmount>
<CardNum CardPresent="false">4000300020001000</CardNum>
<IndustryInfo Type="HOTEL">
<FolioNumber>Folio12345</FolioNumber>
</IndustryInfo>
</JetPay>
```

The TotalAmount in the "PARTIALREVERSAL" transaction must be less than the amount of the existing authorization. If the TotalAmount is greater than or equal to the amount of the existing authorization, an "INVALID REVERSAL AMOUNT" response will be returned.

If a matching FolioNumber cannot be found in a previous transaction, a "LODGING TRANSACTION NOT FOUND" response is returned. If the transaction is already reversed, then an "AUTHORIZATION TRANSACTION NOT FOUND" response is returned.

### 4.3.6 Hotel REVERSEAUTH Example

A "REVERSEAUTH" transaction enables a hotel to clear an authorized amount of an existing "AUTHONLY" transaction. After performing a "REVERSEAUTH", a customer's open-to-buy on their credit card may be cleared for the amount of a matching "AUTHONLY" transaction. The "REVERSEAUTH" transaction requires the submission of a matching FolioNumber to successfully complete the transaction.

The following is a "REVERSEAUTH" transaction, matching by FolioNumber.

```
<JetPay>
<TransactionType>REVERSEAUTH</TransactionType>
<TerminalID>TESTTERMINAL</TerminalID>
<TransactionID>TransactionID22222</TransactionID>
<CardExpMonth>12</CardExpMonth>
<CardExpYear>15</CardExpYear>
<CardNum CardPresent="false">4000300020001000</CardNum>
<IndustryInfo Type="HOTEL">
<FolioNumber>Folio12345</FolioNumber>
</IndustryInfo>
</JetPay>
```

If the FolioNumber does not match a previous transaction a "LODGING TRANSACTION NOT FOUND" error is returned. If the transaction has been already reversed, then an "AUTHORIZATION TRANSACTION NOT FOUND" response is returned.

### 4.3.7 Hotel CAPT Example

A hotel "CAPT" transaction operates similarly to other "CAPT" transactions. However, the additional hotel-specific data within the hotel "CAPT" transaction enables the merchant to update the industry-specific information associated with the original "AUTHONLY" transaction.

Hotel-specific data for an "CAPT" transaction request include:

<IndustryInfo Type="HOTEL"> <FolioNumber>	maximum length of ten characters, must match the initial authorization's FolioNumber. Required.
<IndustryInfo Type="HOTEL"> <CheckInDate>	date format: <b>yyyy-mm-dd</b> . Required.
<IndustryInfo Type="HOTEL"> <CheckOutDate>	date format: <b>yyyy-mm-dd</b> . Required.
<IndustryInfo Type="HOTEL"> <RoomRate>	base room rate, digits only.
<IndustryInfo Type="HOTEL"> <RoomNumber>	room number, characters permitted.
<IndustryInfo Type="HOTEL"> <ExtraCharges>	valid values are "RESTAURANT", "GIFTSHOP", "MINIBAR", "TELEPHONE", "LAUNDRY", "OTHER".
<IndustryInfo Type="HOTEL"> <NoShow>	"true" indicates a failure to check-in. "false" indicates a successful check-in.

The following example shows a CAPT transaction able to finalize a sale and begin the settlement process for the sequence of hotel transactions:

```
<JetPay>
<TransactionType>CAPT</TransactionType>
<TerminalID>TESTTERMINAL</TerminalID>
<TransactionID>TransactionID22222</TransactionID>
<CardExpMonth>12</CardExpMonth>
<CardExpYear>15</CardExpYear>
<TotalAmount>10000</TotalAmount>
<CardNum CardPresent="false">4000300020001000</CardNum>
<IndustryInfo Type="HOTEL">
<FolioNumber>Folio12345</FolioNumber>
<CheckInDate>2006-02-03</CheckInDate>
<CheckOutDate>2006-03-04</CheckOutDate>
<RoomRate>12300</RoomRate>
<RoomNumber>456</RoomNumber>
<ExtraCharges>TELEPHONE</ExtraCharges>
<ExtraCharges>OTHER</ExtraCharges>
<NoShow>>false</NoShow>
</IndustryInfo>
</JetPay>
```

If you already have ExtraCharges on the original authorization, such as, for example, "RESTAURANT" and "MINIBAR", and you want to add "TELEPHONE," you must resubmit "RESTAURANT," "MINIBAR" and "TELEPHONE" in the CAPT transaction.

If the FolioNumber does not match a previous transaction, a "LODGING TRANSACTION NOT FOUND" response is returned. If the transaction has been fully reversed, then an "AUTHORIZATION TRANSACTION NOT FOUND" response is returned.

## 4.4 Handling Non-U.S. Card Types (Switch/Solo)

Transactions on non-U.S. cards, especially Switch and Solo debit cards, may have additional required fields you must submit to JetPay for obtaining authorizations. A merchant must collect this information and pass it to JetPay with the transaction if it exists.

The Switch and Solo debit cards require a merchant to submit the "start date" of the credit card and the "issue number" of the credit card, if present. Be advised that the start date and the issue number information is not necessarily present on a given Switch or Solo debit card, depending on the independent policies of the issuing bank, but these values are nevertheless present on many Switch and Solo cards. A merchant must supply this information if it is available. **Note:** JCB credit cards do not require this additional information.

JetPay provides separate CardStartMonth and CardStartYear elements, allowing the merchant to submit a two-digit month and a two-digit year for the target credit card. Rules are similar to those for the expiration date, except that the start date always designates a time in the past.

The Issue element enables the merchant to submit the two-digit number that might be printed on the Switch or Solo card. The issue number is a numeric value ranging from "00" to "99".

A merchant might need to submit a RoutingCode (see previous example).

The following is an example of a Switch credit card transaction submitted to JetPay:

```
<JetPay>
<TransactionType>AUTHONLY</TransactionType>
<TerminalID>TESTMERCHANT</TerminalID>
<TransactionID>210227153217T1X014</TransactionID>
<RoutingCode>GBPdebit</RoutingCode>
<CardNum>4000300020001000</CardNum>
<CardExpMonth>12</CardExpMonth>
<CardExpYear>15</CardExpYear>
<CardStartMonth>07</CardStartMonth>
<CardStartYear>00</CardStartYear>
<Issue>00</Issue>
<TotalAmount>7028</TotalAmount>
</JetPay>
```

The following response looks similar to many other JetPay responses.

```
<JetPayResponse>
<TransactionID>210227153217T1X014</TransactionID>
<ActionCode>000</ActionCode>
<Approval>412W6B</Approval>
<ResponseText>APPROVED</ResponseText>
</JetPayResponse>
```

### 4.4.1 Special note about Credit with Switch and Solo cards

JetPay enables European debit card transactions. When working with the European debit cards, particularly Switch and Solo cards, the merchant must adjust to slightly different rules. The merchant must take additional precautions in order to insure that they are complying with the rules associated with these debit cards.

The CREDIT operation for Switch and Solo debit cards is slightly stricter than for credit cards. For Switch and Solo, a CREDIT cannot be submitted unless a pre-existing sale transaction exists. The merchant must have already submitted a corresponding SALE or CAPT transaction before submitting the CREDIT transaction for Switch/Solo.

A CREDIT transaction without a corresponding SALE or CAPT transaction is not permitted. If the merchant has not submitted a prior SALE or CAPT transaction using that card number, the merchant must not submit a CREDIT transaction having that card number. Doing so violates association rules for European debit cards. By looking up the original sale, the merchant complies with association rules for European debit cards that enable a CREDIT transaction.

When building a CREDIT transaction, the merchant must submit a unique transaction ID. The CREDIT's transaction ID must not duplicate the transaction ID of the original SALE/CAPT transaction (normally, JetPay does not disallow non-unique transaction IDs).

The transaction ID for the CREDIT transaction must be unique in comparison to transactions already submitted by the merchant to JetPay.

The merchant must not attempt to submit a CREDIT without insuring that the above conditions are met.

## 5 ACH processing

JetPay can process ACH (Automated Clearing House) transactions. This section covers the XML processing necessary to submit ACH transactions through JetPay.

### 5.1 ACH Request Messages

The following two tables show the logical structure of the JetPay ACH request messages. The **Document Structure** column lists the element name and its place in the message structure in order and indentation. The **Qualifications** column explains the conditions for supplying that element to a transaction. The condition column lists any conditions that apply to the element.

Element in Document Structure	Qualifications
JetPay	Required root element.
TransactionType	Required (value is CHECK or REVERSAL or VOIDACH)
TerminalID	Required
TransactionID	Required, exactly 18-characters in length
RoutingCode	Optional
Password	Optional
CardName	Required
TotalAmount	Required
FeeAmount	Optional
ACHType SEC	Required for ACH processing only. Attribute is CHECKING, SAVINGS, or BUSINESSCK. Attribute is Standard Entry Class code, three characters.

ACH Only Elements

Element in Document Structure	Qualifications
ACHType SEC	Required root element. Attribute is CHECKING, SAVINGS, or BUSINESSCK. Attribute is Standard Entry Class code, three characters.
AccountNumber	Required
ABA	Required
CheckNumber	Required

## 5.2 Request Transaction Element Definitions

The following properties describe the root **JetPay** request transactions.

- **JetPay** - root element, enclosing the following transaction elements:
- **TransactionType** - Required
  - ◆ CHECK - Authorize and capture an ACH transaction. \*\*REVERSAL - Ship out an ACH transaction.
  - ◆ VOIDACH - Remove an ACH transaction before settlement.
- **TerminalID** - Required for all transactions. This ID is issued by JetPay LLC when an account is set up with the merchant bank.
- **TransactionID** - Required for all transactions. The TransactionID is an 18-character alphanumeric value.
- **RoutingCode** - A mechanism for declaring alternate transaction routes. The value of the RoutingCode must be pre-arranged.
- **Password** - The password (coordinated with JetPay LLC) is used to verify that a submitted XML transaction originates from a specific merchant. See [#Passwords](#).
- **CardName** - The name of the target bank account owner.
- **TotalAmount** - Total amount of the ACH transaction, including the ACH amount, fees, surcharges, and taxes. *Digits only: no decimal point, no dollar sign, no plus/minus sign.*
- **FeeAmount** - Surcharge applied to the ACH transaction. Note that this surcharge is already included as part of the TotalAmount. *Digits only: no decimal point, no dollar sign, no plus/minus sign.*
- **ACH** - Target bank account information (detailed in the ensuing descriptions).

The following properties describe the root ACH request transactions.

- **ACH** - root element, with the following attributes and elements:
  - ◆ **Type** attribute having a value of CHECKING, SAVINGS, or BUSINESSCK.
  - ◆ **SEC** attribute describes the ACH payment application type. Other values conform with Standard Entry Class codes, described at, [http://www.nacha.org/OtherResources/buyers2003/Understanding\\_the\\_ACH.pdf](http://www.nacha.org/OtherResources/buyers2003/Understanding_the_ACH.pdf)
- **AccountNumber** - The account number for the target bank account.
- **ABA** - The nine-digit bank routing identifier for the target bank account.
- **CheckNumber** - A numeric identifier for the banking transaction.

## 5.3 ACH Response Message

The following table describes the logical structure of the JetPay transaction response message, sent in response to an initiating JetPay Merchant-Initiated Batch request message.

Element in Document Structure	Qualifications
JetPayResponse	Required root element.
TransactionID	Present, matching the request's TransactionID.
ActionCode	Present in all responses. "000" = <i>accepted</i> .
ResponseText	Present.
AdditionalInfo	Conditionally present.

## 5.4 Response Transaction Element Definitions

The following properties describe the response messages:

- **JetPayResponse** - root element, enclosing the following transaction elements:
- **TransactionID** - This 18-character value matches the TransactionID submitted in the corresponding JetPay request transaction message. This value can be used to correlate the transaction's acknowledgment information with its submitted request data.
- **ActionCode** - The ActionCode is three (3) characters. The ActionCode contains a "000" value to indicate that a transaction has been accepted. The ActionCode will be some other value when the JetPay server host detects an error. For ACH transactions an ActionCode of "000" does *not* mean that the merchant will receive the funds requested, it only means that JetPay accepted the message and will include the transaction the next ACH transactions are processed.
- **ResponseText** - Message from the JetPay server, describing to the ActionCode. Possible ResponseText values include "ACCEPTED" and "DECLINED".

## 5.5 ACH Examples

The following examples illustrate JetPay ACH messages, showing sample ACH request messages with a corresponding sample response messages.

### 5.5.1 "CHECK" Transaction Example (ACH)

A "CHECK" transaction captures an ACH transaction for settlement. The AccountNumber and ABA values define the target account to be debited for the transaction. In a "CHECK" transaction, JetPay initiates the transfer of monies out of the bank account defined within the transaction and into the merchant's account.

Required data for a "CHECK" transaction request are:

<TransactionType>	CHECK
<TerminalID>	your terminal ID, assigned by JetPay LLC
<TransactionID>	character string, exactly eighteen characters long, preferably a unique value.
<CardName>	name of the owner of the bank account.

<TotalAmount>	total amount of the transaction, including any taxes, fees, and surcharges (digits only).
<ACH Type>	CHECKING, SAVINGS, or BUSINESSCK.
<ACH SEC>	ACH payment application type.
<AccountNumber>	the account identifier for the transaction.
<ABA>	nine-digit bank routing id for the bank.
<CheckNumber>	unique numeric identifier for the transaction.

Minimum data returned from a "CHECK" transaction response are:

<TransactionID>	matching the value in the request, above
<ActionCode>	000 or other three-digit value
<ResponseText>	<b>CHECK ACCEPTED</b> or DECLINED

The following is a sample "CHECK" transaction for \$12.99 along with a one-dollar service charge tacked onto it, accessing a savings account:

```
<JetPay>
<TransactionType>CHECK</TransactionType>
<TerminalID>TESTMERCHANT</TerminalID>
<TransactionID>SC0327153017T10018</TransactionID>
<CardName>Mickey Mouse</CardName>
<TotalAmount>1399</TotalAmount>
<FeeAmount>100</FeeAmount>
<ACH Type = "SAVINGS" SEC = "PPD">
<AccountNumber>11111999</AccountNumber>
<ABA>123456789</ABA>
<CheckNumber>15</CheckNumber>
</ACH>
</JetPay>
```

The following shows a sample "successful response" to the above "CHECK" request.

```
<JetPayResponse>
<TransactionID>SC0327153017T10018</TransactionID>
<ActionCode>000</ActionCode>
<ResponseText>CHECK ACCEPTED</ResponseText>
</JetPayResponse>
```

## 5.5.2 "REVERSAL" Transaction Example (ACH)

The "REVERSAL" transaction captures an ACH transaction for settlement. The AccountNumber and ABA values define the target account to be credited for the transaction. In a "REVERSAL" transaction, JetPay initiates the transfer of monies out of the merchant's account and into the bank account defined within the transaction.

Submitted data for a "REVERSAL" transaction request are:

<TransactionType>	REVERSAL
<TerminalID>	your terminal ID, assigned by JetPay LLC
<TransactionID>	character string, exactly eighteen characters long, preferably a unique value.
<CardName>	name of the owner of the bank account.
<TotalAmount>	total amount of the transaction, including any taxes, fees, and surcharges (digits only).

<ACH Type>	CHECKING, SAVINGS, or BUSINESSCK.
<ACH SEC>	ACH payment application type.
<AccountNumber>	the account identifier for the transaction.
<ABA>	nine-digit bank routing id for the bank.
<CheckNumber>	unique numeric identifier for the transaction.

Minimum data returned from a successful "REVERSAL" transaction response are:

<TransactionID>	matching the value in the request, above
<ActionCode>	000 or other three-digit value
<ResponseText>	<b>CHECK ACCEPTED</b> or DECLINED

The following is a sample "REVERSAL" transaction for \$15.00, depositing merchant monies into a target checking account:

```
<JetPay>
<TransactionType>REVERSAL</TransactionType>
<TerminalID>TESTMERCHANT</TerminalID>
<TransactionID>SC0327153017T10019</TransactionID>
<CardName>Donald Duck</CardName>
<TotalAmount>1500</TotalAmount>
<ACH Type = "CHECKING" SEC = "CCD">
<AccountNumber>22222999</AccountNumber>
<ABA>987654321</ABA>
<CheckNumber>1005</CheckNumber>
</ACH>
</JetPay>
```

The following shows a sample "successful response" to the above "REVERSAL" request.

```
<JetPayResponse>
<TransactionID>SC0327153017T10019</TransactionID>
<ActionCode>000</ActionCode>
<ResponseText>CHECK ACCEPTED</ResponseText>
</JetPayResponse>
```

### 5.5.3 "VOIDACH" Transaction Example (ACH)

The "VOIDACH" transaction removes an ACH transaction before settlement occurs. In a "VOIDACH" transaction, JetPay searches its system for a matching ACH record and removes it before settlement can occur. If settlement has already occurred, the "VOIDACH" transaction will complete unsuccessfully and returns a non-zero ActionCode in the response.

Submitted data for a "VOIDACH" transaction request are:

<TransactionType>	VOIDACH
<TerminalID>	your terminal ID, assigned by JetPay LLC
<TransactionID>	character string. exactly eighteen characters long, matching the target transaction to be removed from settlement.
<TotalAmount>	total amount of the transaction, including any taxes, fees, and surcharges (digits only).
<ACH Type>	CHECKING, SAVINGS, or BUSINESSCK.

<AccountNumber>	the account identifier for the transaction.
<ABA>	nine-digit bank routing id for the bank.
<CheckNumber>	unique numeric identifier for the transaction.

Minimum data returned from a successful "VOIDACH" transaction response are:

<TransactionID>	matching the value in the request, above
<ActionCode>	000 or other three-digit value.
<ResponseText>	ACCEPTED or other descriptive text.

The following is a sample "VOIDACH" transaction is to remove an ACH transaction for \$15.00, preempting the settlement of that transaction:

```
<JetPay>
<TransactionType>VOIDACH</TransactionType>
<TerminalID>TESTMERCHANT</TerminalID>
<TransactionID>SC0327153017T10019</TransactionID>
<CardName>Mickey Mouse</CardName>
<TotalAmount>1500</TotalAmount>
<ACH Type = "CHECKING">
<AccountNumber>22222999</AccountNumber>
<ABA>987654321</ABA>
<CheckNumber>1005</CheckNumber>
</ACH>
</JetPay>
```

The following shows a sample "successful response" to the above "VOIDACH" request.

```
<JetPayResponse>
<TransactionID>SC0327153017T10019</TransactionID>
<ActionCode>000</ActionCode>
<ResponseText>ACCEPTED</ResponseText>
</JetPayResponse>
```

## 6 Memo Transactions

Memo transactions (also called "Memo Post") are transactions that do not have have a direct authorization or clearing impact, but that do have a financial impact on the merchant. The primary use of the memo transaction is going to be to capture the information on debit transactions made by Jetpay merchant but not using Jetpay's debit system. By sending Jetpay a memo transaction, these externally authorized debit transactions can be properly funded, billed, and reported. Other uses for memo transactions include charging service fees to merchants or recording transactions that could not otherwise be parsed.

All memo transactions have a `TransactionType` of "MEMO". The `<Memo>` tag is used for all additional information specific to memo transactions. The `Memo` tag has an the `Type` attribute that is used to specify the exact memo type from the following:

SALE	Transaction was a credit card sale
DEBIT SALE	Transaction was a pin based debit card sale
REFUND	Transaction was a credit card refund
DEBIT REFUND	Transaction was a debit card refund
SYNTAX ERROR	Transaction is a xml syntax error

FEE	Transaction is a fee to be applied to the merchant's bill
-----	---

There are also two tags present in the Memo block.

<AuthAuthority>	The organization that was responsible for the original authorization of the transaction (no more than 24 bytes)
<Description>	An optional description associated with the transaction (no more than 80 bytes)

The following is an example of a Memo transaction for a debit sale:

```
<JetPay>
<TransactionID>RTST00001301411019</TransactionID>
<TransactionType> MEMO </TransactionType>
<TerminalID> TESTTERMINAL </TerminalID><
CardExpMonth> 12 </CardExpMonth>
<CardExpYear> 15 </CardExpYear>
<TotalAmount> 100 </TotalAmount>
<CardNum CardPresent='false' Tokenize='false'> 4000300020001000 </CardNum>
<Memo Type="DEBIT SALE">
<AuthAuthority> REGRESSION TESTER </AuthAuthority>
<Description> this is the desc </Description>
</Memo>
</JetPay>
```

And this is the response:

```
<JetPayResponse><TransactionID>RTST00001301411019</TransactionID>
<ActionCode>000</ActionCode>
<ResponseText>ACCEPTED</ResponseText>
</JetPayResponse>
```

## 7 Data Security Features

### 7.1 CryptoPAN Secure Credit Card Number Storage

Merchants who store customers' credit card numbers must also mitigate liability against theft of those credit card numbers. JetPay offers CryptoPAN, enabling merchants to store and use customer credit card numbers while maintaining secure customer records.

CryptoPAN processing enables secure data transfer easily and enhances merchant security. A CryptoPAN-secured credit card number can be used directly in JetPay credit card transactions, even though the credit card number is not decrypted within the transaction. A CryptoPAN value is encoded using an asymmetric public key and the resulting encrypted credit card number becomes safe to store in insecure or compromised environments. After encrypting a credit card number using CryptoPAN's system, a merchant can treat that CryptoPAN number similar to a credit card number in a JetPay transaction. Unauthorized intruders on a merchant's computer network cannot decrypt CryptoPAN data; sensitive credit card numbers remain secure when they're encrypted using CryptoPAN. In fact, even merchant personnel cannot decrypt a CryptoPAN value back into a card number, which enhances a merchant's overall data processing security.

CryptoPAN processing is particularly useful in recurring transactions, where a merchant receives permission from a customer to apply charges to a customer's credit card account on a repetitive basis. The merchant must save the customer's sensitive credit card data in order to enable recurring transactions. The merchant encrypts the customer's credit card using the CryptoPAN system, saves the CryptoPAN result, and then uses the CryptoPAN value in JetPay transactions. JetPay uses an asymmetric private key to decrypt the CryptoPAN value, converting it back into the original credit card number within JetPay's secure PCI-compliant environment.

### 7.1.1 Sending Card Numbers Securely, Using CryptoPAN

When sending a CryptoPAN-secured card number, only a few extra flags need to be set within a JetPay transaction. By designating the CardNum as encrypted as well as designating the name of the public key used to encode the CardNum, JetPay uses asymmetric private keys to convert the CryptoPAN value back into the original credit card number. The CryptoPAN value is sent in an XML-compatible format, including a base 64 format or a hexadecimal format.

Merchants can send a CryptoPAN value in any JetPay transaction that contains a CardNum tag. The syntax for the CardNum entry should look similar to the following example:

The following is a sample "SALE" transaction for \$998.99 using a CryptoPAN-secured credit card number:

```
<JetPay>
<TransactionType>SALE</TransactionType>
<TerminalID>TESTMERCHANT</TerminalID>
<TransactionID>510327153017T10017</TransactionID>
<CardNum Encrypted = "true" KeyName = "PUBLICKEY"
  Encoding = "BASE64">
Abc+EasyAs+123+OrSimpleAs+DoughRayMe+Abc+123+BabyYouAndMe==
</CardNum>
<CardExpMonth>12</CardExpMonth>
<CardExpYear>15</CardExpYear>
<TotalAmount>99899</TotalAmount>
</JetPay>
```

The following shows how the successful response would look.

```
<JetPayResponse>
<TransactionID>510327153017T10017</TransactionID>
<ActionCode>000</ActionCode>
<Approval>212F6C</Approval>
<ResponseText>APPROVED</ResponseText>
</JetPayResponse>
```

### 7.1.2 CryptoPAN Encoding

To encode credit card numbers using the CryptoPAN system, JetPay distributes a few hundred lines of source code to merchants written in C++ under Open Source distribution rules. A developer implements this CryptoPAN software in enabling CryptoPAN processing on their system. A knowledgeable developer can easily find and use Open Source resources to complete this encryption portion of the CryptoPAN system.

Each resulting CryptoPAN value will be formatted in base 64 and constitute around 400 bytes of printable character data. The resulting data is sufficiently secured for distribution and storage in a non-secure environment.

This document contains sample C++ source code that illustrates how to build CryptoPAN data. In implementing this scheme, a merchant encrypts credit card numbers immediately, keeping the customer credit card numbers safe thereafter.

For an example of the XML needed to send encoded CryptoPAN values to JetPay see [#Sending Card Numbers Securely, Using CryptoPAN](#). Remember that the CryptoPAN value is equivalent to a credit card number and only JetPay can decrypt it.

### 7.1.3 Sample CryptoPAN Program

The following software is in the public domain and, along with a public key supplied by JetPay, enables CryptoPAN encryption using the OpenSSL library ([1]) for performing the asymmetric encryption. This program is known to work with OpenSSL 0.9.7a and gcc 3.4.

```
#include <stdio.h>
#include <string.h>
#include <openssl/err.h>
#include <openssl/pem.h>
#include <openssl/rsa.h>
#include <openssl/bio.h>

RSA *
cryptopan_load_key (const char *file, char *pass, int priv_key)
{
    BIO *key = NULL;
    RSA *rsa = NULL;

    if (!file) {
        printf ("no keyfile specified\n");
        goto end;
    }

    key = BIO_new (BIO_s_file ());

    if (key == NULL) {
        ERR_print_errors_fp (stderr);
        goto end;
    }

    if (BIO_read_filename (key, file) <= 0) {
        printf ("Error opening %s\n", file);
        ERR_print_errors_fp (stderr);
        goto end;
    }

    if (priv_key) {
        rsa = PEM_read_bio_RSAPrivateKey (key, NULL, 0, pass);
    } else {
        rsa = PEM_read_bio_RSA_PUBKEY (key, NULL, 0, pass);
    }

end:
    if (key != NULL) {
        BIO_free_all (key);
    }

    if (rsa == NULL) {
        ERR_print_errors_fp (stderr);
    }
}
```

```

    printf ("unable to load %s\n", file);
}

return rsa;
}

/*
 * Allocates memory which _must_ be freed!
 */
size_t
base64_encode (const unsigned char *input, int len, char **output)
{
    size_t size;
    BIO *mbio, *b64, *bio;
    char *p;
    int result;

    mbio = BIO_new (BIO_s_mem ());
    b64 = BIO_new (BIO_f_base64 ());
    BIO_set_flags (b64, BIO_FLAGS_BASE64_NO_NL);

    bio = BIO_push (b64, mbio);

    BIO_write (bio, input, len);
    result = BIO_flush (bio);

    size = BIO_get_mem_data (bio, &p);
    *output = calloc (1, size + 1);
    memcpy (*output, p, size);

    BIO_free_all (bio);

    return size;
}

inline int
is_base64 (char c)
{
    if (c >= 'A' && c <= 'Z')
        return 1;
    if (c >= 'a' && c <= 'z')
        return 1;
    if (c >= '0' && c <= '9')
        return 1;
    if (c == '+')
        return 1;
    if (c == '/')
        return 1;
    if (c == '=')
        return 1;

    return 0;
}

/*
 * Allocates memory which _must_ be freed!
 */
size_t base64_decode(char *unfiltered, int len, unsigned char **output)
{
    int size, i, result;
    char *input;
    BIO *b64, *bmem;

```

```

if (len < 0) {
    len = strlen (unfiltered);
}

input = calloc (1, len + 1);
for (i = len = 0; unfiltered[i]; i++) {
    if (is_base64 (unfiltered[i])) {
        input[len++] = unfiltered[i];
    }
}
input[len + 1] = '\0';

b64 = BIO_new(BIO_f_base64());
BIO_set_flags (b64, BIO_FLAGS_BASE64_NO_NL);
bmem = BIO_new_mem_buf(input, len);
bmem = BIO_push(b64, bmem);
result = BIO_flush (bmem);

*output = calloc (1, len + 1);
size = BIO_read(bmem, *output, len);

BIO_free_all (bmem);
free (input);

return size;
}

int
cryptopan_encrypt (RSA *pub_key, unsigned char *input, int len, char **output)
{
    int keysize;
    unsigned char *out_buf;
    int ret;

    if (!pub_key) {
        fprintf (stderr, "%s %d (%s): No cryptopan public key loaded.\n",
                __FILE__, __LINE__, __func__);
        return -1;
    }
    if (len < 0) {
        len = strlen ((char *) input);
    }

    keysize = RSA_size (pub_key);
    out_buf = calloc (1, keysize);

    ret = RSA_public_encrypt (len, input, out_buf, pub_key,
                             RSA_PKCS1_OAEP_PADDING);
    if (ret < 0) {
        ERR_print_errors_fp (stderr);
        return -1;
    }

    ret = base64_encode (out_buf, ret, output);
    free (out_buf);

    return ret;
}

int
cryptopan_decrypt (RSA *priv_key, char *input, int len, unsigned char **output)

```

```

{
    int keysize;
    unsigned char *out_buf;
    unsigned char *in_buf;
    int ret;

    if (!priv_key) {
        fprintf (stderr, "%s %d (%s): No cryptopan private key loaded.\n",
                __FILE__, __LINE__, __func__);
        return -1;
    }

    if (len < 0) {
        len = strlen ((char *) input);
    }

    len = base64_decode (input, len, &in_buf);

    keysize = RSA_size (priv_key);
    out_buf = calloc (1, keysize);

    ret = RSA_private_decrypt (len, in_buf, out_buf, priv_key,
                              RSA_PKCS1_OAEP_PADDING);

    free (in_buf);

    if (ret < 0) {
        ERR_print_errors_fp (stderr);
        return -1;
    }

    *output = calloc (1, ret + 1);
    memcpy (*output, out_buf, ret);
    free (out_buf);

    return ret;
}

int
main (int argc, char *argv[])
{
    int ret;
    char *buf;

    if (argc != 4) {
        goto usage;
    }

    if (argv[1][0] == 'e') {
        RSA *pub_key;

        pub_key = cryptopan_load_key (argv[2], "", 0);
        if (!pub_key) {
            fprintf (stderr, "Unable to load public key '%s'.\n", argv[2]);
            exit (1);
        }

        ret = cryptopan_encrypt (pub_key, (unsigned char *) argv[3], -1, &buf);
        if (ret < 0) {
            fprintf (stderr, "Unable to encrypt data!\n");
            exit (1);
        }
    }
}

```

```

} else if (argv[1][0] == 'd') {
    RSA *priv_key;

    priv_key = cryptopan_load_key (argv[2], "", 1);
    if (!priv_key) {
        fprintf (stderr, "Unable to load private key '%s'.\n", argv[2]);
        exit (1);
    }

    ret = cryptopan_decrypt (priv_key, argv[3], -1, (unsigned char **) &buf);
    if (ret < 0) {
        fprintf (stderr, "Unable to decrypt data!\n");
        exit (1);
    }
} else {
    goto usage;
}

printf ("%s\n", buf);
free (buf);
exit (0);
usage:
fprintf (stderr, "Usage: %s <e or d> <keyfile> <string to encrypt>\n",
        argv[0]);
exit (1);
}

```

## 7.2 Jetpay Account Safe tokens

JetPay Account Safe tokens are data that can be used in the place of card information when performing transactions. Because the tokens are not card holder information, merchants using them do not have to protect them in the same way they have to protect card holder information. The token takes the place of the card number and the expiration date. No other card holder information is associated with the token.

The first step to using the tokens is getting a token that maps to card number. This can be done in two ways. The first way is to use the TOKENIZE message. This message looks like:

```

<JetPay>
  <TransactionID>TOKENTEST000000002</TransactionID>
  <TransactionType>TOKENIZE</TransactionType>
  <CardNum>4000300020001000</CardNum>
  <CardExpMonth>12</CardExpMonth>
  <CardExpYear>15</CardExpYear>
  <TerminalID>TESTTERMINAL</TerminalID>
</JetPay>

```

And its response looks like:

```

<JetPayResponse>
  <TransactionID>TOKENTEST000000002</TransactionID>
  <ActionCode>000</ActionCode>
  <Approval>TOKEN0</Approval>
  <ResponseText>TOKENIZED</ResponseText>
  <Token>KKHKOIIBJBJBKMKMNBJJJKJK</Token>
</JetPayResponse>

```

## JetPay Authorization XML Specification © 2012 JetPay LLC.

The TOKENIZE message does not send the card to the associations to be validated. So a successful tokenization does not mean that the card is valid.

The second way is to use the Tokenize attribute of the CardNum field to return a token as part of another transaction such as an AUTHONLY or a SALE. To enable this set the Tokenize attribute in the CardNum field to "true":

```
<JetPay>
  <TransactionID>TOKENEST000000003</TransactionID>
  <TransactionType>SALE</TransactionType>
  <CardNum Tokenize="true">4000300020001000</CardNum>
  <CardExpMonth>12</CardExpMonth>
  <CardExpYear>15</CardExpYear>
  <TotalAmount>100</TotalAmount>
  <TerminalID>TESTTERMINAL</TerminalID>
</JetPay>
```

The response will look like:

```
<JetPayResponse>
  <TransactionID>TOKENEST000000003</TransactionID>
  <ActionCode>000</ActionCode>
  <Approval>TEST47</Approval>
  <ResponseText>APPROVED</ResponseText>
  <Token>KKHKOIIBJBJBKMKMNBJJJKJK</Token>
</JetPayResponse>
```

Once you have a token then it can be used in place of the CardNumber, CardExpMonth, and CardExpYear fields:

```
<JetPay>
  <TransactionID>TOKENEST000000003</TransactionID>
  <TransactionType>AUTHONLY</TransactionType>
  <Token>KKHKOIIBJBJBKMKMNBJJJKJK</Token>
  <TotalAmount>100</TotalAmount>
  <TerminalID>TESTTERMINAL</TerminalID>
</JetPay>
```

And the response will be as normal:

```
<JetPayResponse>
  <TransactionID>TOKENEST000000003</TransactionID>
  <ActionCode>000</ActionCode>
  <Approval>TEST47</Approval>
  <ResponseText>APPROVED</ResponseText>
</JetPayResponse>
```

If the token is not present in our system, an ActionCode of 912 is returned.

Tokens are also available for ACH transactions. In an ACH transaction they take the place of the both the ABA (routing code) and DDA (account number). To request a token with an ACH transaction set the Tokenize attribute of the AccountNumber tag to true as in the following example:

```
<JetPay>
  <TransactionID>RTST00001301336246</TransactionID>
  <TransactionType> CHECK </TransactionType>
  <TerminalID> TESTTERMINAL </TerminalID>
```

## JetPay Authorization XML Specification © 2012 JetPay LLC.

```
<Origin> INTERNET </Origin>
<TotalAmount> 599500 </TotalAmount>
<CardName> Fred Furtz </CardName>
<ACH>
  <AccountNumber Tokenize='true'> 1982597 </AccountNumber>
  <ABA > 061120767 </ABA>
  <CheckNumber > 1234 </CheckNumber>
</ACH>
</JetPay>
```

And the response for this transaction will look like:

```
<JetPayResponse>
  <TransactionID>RTST00001301336246</TransactionID>
  <ActionCode>000</ActionCode>
  <Approval>000000</Approval>
  <ResponseText>CHECK ACCEPTED</ResponseText>
  <Token>KKHIHCOIKJJHHCKCKLHMJKJK</Token>
</JetPayResponse>
```

And the token can then be used in a later transaction like so:

```
<JetPay>
  <TransactionID>RTST00001301336260</TransactionID>
  <TransactionType> CHECK </TransactionType>
  <TerminalID> TESTTERMINAL </TerminalID>
  <Origin> INTERNET </Origin>
  <TotalAmount> 599500 </TotalAmount>
  <CardName> Fred Furtz </CardName>
  <Token> KKHIHCOIKJJHHCKCKLHMJKJK </Token>
  <ACH>
    <CheckNumber > 1234 </CheckNumber>
  </ACH>
</JetPay>
```

Please notice that the Token tag does not go in the ACH block and that the CheckNumber is still required.

If you only want to tokenize an ABA/AccountNumber pair without doing an actual transaction the TOKENIZE transaction type can be used:

```
<JetPay>
  <TransactionID>RTST00021301516309</TransactionID>
  <TransactionType> TOKENIZE </TransactionType>
  <TerminalID> TESTTERMINAL </TerminalID>
  <CardName> Fred Furtz </CardName>
  <ACH>
    <AccountNumber> 1772105 </AccountNumber>
    <ABA > 061120767 </ABA>
    <CheckNumber > 1234 </CheckNumber>
  </ACH>
</JetPay>
```

And the response will be

```
<JetPayResponse>
  <TransactionID>RTST00021301516309</TransactionID>
  <ActionCode>000</ActionCode>
  <Approval>TOKEN0</Approval>
  <ResponseText>TOKENIZED</ResponseText>
```

```
<Token>KKHJMMIMKJJHIKHKKCHBJKJI</Token>  
</JetPayResponse>
```

## 7.3 Passwords

As a secondary security feature, JetPay LLC supports an optional password string for each submitted transaction. Submitting passwords with each transaction is completely optional and must be pre-arranged with JetPay LLC.

Setting up and changing passwords requires preparation. The following procedure documents the steps to follow for setting up or changing passwords:

The password will be a string, case-sensitive, exactly sixteen characters in length. The password string can be either a pronounceable word or it can be a "nonsense-string" of characters, depending on preference. Regardless, the password must be correctly sequenced; since the password must be set up using a telephone call, any pronounceable password must have an agreed-upon spelling. Mis-sequenced or misspelled passwords will cause incoming transactions to be rejected with an action code of "926".

Passwords associate with Terminal IDs. Each of the Terminal IDs can be set up with the same password, or each of the Terminal IDs could also get set up with a different password.

JetPay LLC will not require periodic password changes, but JetPay LLC reserves the right to advise a merchant that a password change is necessary. A merchant may either choose to never change passwords or to change passwords periodically. Under normal conditions, it's the merchant's responsibility to initiate requests for all password changes. Ultimately, each merchant determines a reasonable policy for changing passwords (including a schedule for password changes), while JetPay LLC retains the right to affirm the reasonableness of that policy.

Passwords will be coordinated through a phone call between JetPay LLC's customer service department and the merchant. A merchant supplies a contact number where JetPay LLC can call to coordinate password changes. Although a merchant can request a password change by contacting JetPay LLC, the password change can only be expedited after JetPay LLC telephones the merchant's pre-arranged contact number. A password will only be changed after JetPay LLC talks to a person answering the phone at the merchant's pre-arranged contact number.

Password changes on the test system can be arranged informally between engineers.

## 8 Communicating with JetPay

The XML is sent to JetPay inside of an HTTPS POST message. Each post can contain only one message. The XML itself must contain only valid UTF-8 characters. Transactions that use the non-ASCII portions of character sets such as ISO-8859-1, ISO-8859-15 or Windows-1252 will be rejected. For best results, it is strongly recommended that merchants restrict themselves to using only the US-ASCII character set.

### 8.1 Methods of Securely Posting an XML file

If you are using Java, you can download the Java Secure Socket Extension package available at <http://java.sun.com>.

If you are using Microsoft technologies, you can use the XMLHttpRequest object of msxml.dll. Please note that this solution is built upon wininet.dll, which is not scalable. Another solution is to use the rope.dll that is included with the SOAP toolkit. This is a newly emerging technology that at the time of this writing was not able to successfully post data without code changes (the source is freely available) but may be useful in the future.

For Windows 2000 and Windows XP platforms, the Microsoft WinHTTP Service SDK is useful for posting secure XML documents. This solution is more scalable. A simple snip of Visual Basic code illustrates the use of the Microsoft objects available in WinHTTP:

```
Dim strRequest As String
Dim strURL As String
Dim objXml As DOMDocument
Dim objHttp As WinHttp.WinHttpRequest

strRequest = _
"<JetPay>" & _
"<TransactionType>PING</TransactionType>" & _
"<TerminalID>TESTMERCHANT</TerminalID>" & _
"<TransactionID>010327153017T10017</TransactionID>" & _
"</JetPay>"

strURL = "https://test1.jetpay.com/jetpay"

Set objHttp = New WinHttp.WinHttpRequest

objHttp.SetTimeouts 0, 60000, 60000, 60000

objHttp.Open "POST", strURL, False

objHttp.Send strRequest

Set objXml = New DOMDocument

objXml.loadXML objHttp.ResponseText

Dim node As IXMLDOMNode

For Each node In objXml.selectSingleNode("JetPayResponse").childNodes
Debug.Print txtOut.Text & node.nodeName & ": " & node.Text & vbNewLine
Next
```

There are also third party products available for this purpose. IP\*Works is available at <http://www.dev-soft.com>.

## 8.2 JetPay URLs

The Internet connection to JetPay can be found at the following URLs:

For XML development, testing XML-generating software, or for testing applications linked with the JetCom DLL:

<https://test1.jetpay.com/jetpay>

**Production gateway for live transactions**, sending XML according to this specification, or using applications linked with the JetCom DLL:

<https://gateway17.jetpay.com/jetpay>

The production gateway should not be used for test purposes (unless a test of live transaction data is deliberately planned with intent to settle that transaction data).

Stress testing is not permitted over the Internet connections without the specific consent of JetPay LLC. Any entity discovered to be stress testing over the Internet without consent of JetPay LLC will be categorized as a Denial Of Service attacker and be prohibited from using the JetPay system without notice.

These URL's are subject to change, please contact your account representative to get the current URL's before starting testing.

## 9 Certification

JetPay certification testing is intended to prepare a merchant for processing transactions through JetPay. This document contains test suites and test cases that enable a merchant to prepare and submit test JetPay transactions and verify compliance with JetPay submission rules.

There are several test suites, including standard credit card transactions, address verification, card validation, Verified-By-Visa and SecureCode test suites. A merchant might need to perform all test cases within a test suite, or perhaps a subset of the test cases, or perhaps none of the test cases within a test suite. The type of testing that must be performed will depend on the merchant's functional needs.

### 9.1 Standard Transaction Processing Test Suite

The following test cases consist of common JetPay transactions that many merchants will need to be able to perform. Some test cases are specific to ecommerce merchants; some test cases are specific to brick-and-mortar retail merchants. To certify, the merchant needs to submit those test cases that apply to the merchant's anticipated requirements. At the very least, every merchant should anticipate that they'd need to submit some combination of sale and credit test cases.

#### 9.1.1 AUTHONLY, SALE, and CAPT Test Cases

The following test cases will certify standard sales transactions. The credit card used for these test cases will follow the format, 4111 1111 1111 xxxx.

Test Case	Type	PAN	Exp. Date	Amount	Expected Result
STND001	SALE	1111	12/13	\$10.00	Approved
STND002	SALE	1129	12/09	\$10.00	Expired card

STND003	SALE	1128	12/13	\$10.00	Invalid card number
STND004 thru STND104	SALE or AUTHONLY	1137	12/13	amounts between \$1.00 and \$2.00	Assorted responses, both approved and declined. These test cases are always required.
STND105a	AUTHONLY	1145	12/13	\$10.00	Approved
STND105b	CAPT	1145	12/13	\$10.00	Approved
STND106a	AUTHONLY	1152	12/13	\$10.00	Approved
STND106b	CAPT	1152	12/13	\$5.00	Depends on Trans ID
STND107a	AUTHONLY	1160	12/13	\$11.00	Approved
STND107b	CAPT	1178	12/13	\$11.00	Depends on Trans ID
STND108a	AUTHONLY	1186	12/13	\$12.00	Approved
STND108b	CAPT	1186	12/13	\$12.00	Approved
STND108c	CAPT	1186	12/13	\$12.00	Authorization not found
STND109a	AUTHONLY	1194	12/13	\$13.05	Declined
STND109b	CAPT	1194	12/13	\$13.05	Authorization not found

### 9.1.2 Magnetic Stripe Test Cases

The following test cases will certify standard sales transactions. The credit card used for these test cases will follow the format, 4222 2222 2222 xxxx.

Test Case	Type	PAN	Exp. Date	Amount	Expected Result
MAG01	SALE	2220	12/13	\$10.00	Approved
MAG02	SALE	2238	12/09	\$10.00	Expired card
MAG03	SALE	2237	12/13	\$10.00	Invalid card number
MAG04	SALE	2246	12/13	\$15.05	Declined
MAG05a	AUTHONLY	2253	12/13	\$10.00	Approved
MAG05b	CAPT		12/13	\$10.00	Approved
MAG06a	AUTHONLY	2261	12/13	\$10.00	Approved
MAG06b	CAPT		12/13	\$5.00	Approved
MAG07a	AUTHONLY	2279	12/13	\$11.00	Approved
MAG07b	CAPT		12/13	\$11.00	Approved
MAG08a	AUTHONLY	2287	12/13	\$12.00	Approved
MAG08b	CAPT		12/13	\$12.00	Approved
MAG08c	CAPT		12/13	\$12.00	Authorization not found
MAG09a	AUTHONLY	2295	12/13	\$13.05	Declined
MAG09b	CAPT		12/13	\$13.05	Authorization not found

### 9.1.3 Contactless Magnetic Stripe Test Cases

The following test cases will certify standard sales transactions for contactless cards. The credit card used for these test cases will follow the format, 4333 3333 3333 xxxx.

Test Case	Type	PAN	Exp. Date	Amount	Expected Result
CMAG01	SALE	3339	12/13	\$10.00	Approved
CMAG02	SALE	3347	12/09	\$10.00	Expired card
CMAG03	SALE	3354	12/13	\$10.00	Invalid card number
CMAG04	SALE	3362	12/13	\$15.05	Declined

### 9.1.4 CREDIT Test Cases

The following test cases will certify standard credit transactions. The credit card used for these test cases will follow the format, 4444 4444 4444 xxxx.

Test Case	Type	PAN	Exp. Date	Amount	Expected Result
CRED01	CREDIT	4448	12/13	\$10.00	Approved
CRED02	CREDIT	4455	12/09	\$10.00	Expired card
CRED03	CREDIT	4462	12/13	\$10.00	Invalid card number
CRED04	CREDIT	4471	12/13	\$15.05	Approved

### 9.1.5 VOID Test Cases

The following test cases will certify standard void transactions. The credit card used for these test cases will follow the format, 4555 5555 5555 xxxx.

Test Case	Type	PAN	Exp. Date	Amount	Expected Result
VOID01a	SALE or AUTH/CAPT	5519	12/13	\$10.00	Approved
VOID01b	VOID	5519	12/13	\$10.00	Successful
VOID02a	SALE or AUTH/CAPT	5527	12/13	\$10.00	Approved
VOID02b	VOID	5527	12/13	\$15.00	Transaction not found
VOID03a	SALE or AUTH/CAPT	5550	12/05	\$10.05	Declined
VOID03b	VOID	5550	12/13	\$10.05	Transaction not found
VOID04a	CREDIT	5568	12/13	\$10.00	Approved
VOID04b	VOID	5568	12/13	\$10.00	Successful
VOID05a	CREDIT	5576	12/13	\$10.00	Approved
VOID05b	VOID	5576	12/13	\$5.00	Transaction not found
VOID06a	SALE or AUTH/CAPT	5584	12/13	\$12.00	Approved
VOID06b	VOID	5584	12/13	\$12.00	Successful
VOID06c	VOID	5584	12/13	\$12.00	Transaction not found
VOID07	VOID	5592	12/13	\$18.00	Transaction not found

## 9.2 Validation, Verification, and Authentication Test Suite

The credit card companies support multiple methods for validating credit cards and verifying a cardholder's identity. JetPay supports CVV2, CVC2, and CID card validation, available from Visa, MasterCard, and American Express, as well as all the other major credit card brands. JetPay supports address verification ("AVS") as implemented by Visa, MasterCard, and American Express (as well as Amex's name verification feature). With

these two common features, a credit card's integrity can be validated and a cardholder's billing address can be verified. Even more importantly, these two features reduce transaction charges and help to protect a merchant against charge backs.

UCAF is the "Universal Cardholder Authentication Field", more commonly branded as Verified-by-Visa (by Visa) and SecureCode (by MasterCard). The UCAF feature authenticates the cardholder of a specific credit card, protecting both the merchant and the cardholder against fraudulent internet sales.

## 9.2.1 CVV2/CVC2/CID Test Cases

The following test cases will certify CVV2 transactions. The credit card used for these test cases use the following format: 4666 6666 6666 xxxx.

Test Case	Type	PAN	Exp. Date	Amount	CVV2	Expected Result
CVV201	SALE or AUTHONLY	6669	12/13	\$10.00	321	Approved,CVV2 matches
CVV202	SALE or AUTHONLY	6677	12/13	\$10.00	432	Approved,CVV2 doesn't match
CVV203	SALE or AUTHONLY	6685	12/13	\$10.00	543	Approved,CVV2 not processed
CVV204	AUTHONLY	6693	12/13	\$0.00	021	Approved,CVV2 matches
CVV205	SALE or AUTHONLY	6610	12/13	\$10.05	123	Declined,CVV2 not processed
CVV206	SALE or AUTHONLY	6628	12/13	\$12.51	089	Declined,CVV2 matches

## 9.2.2 Address Verification Test Cases

The following test cases will certify address verification transactions using AVS. The credit card used for these test cases is 4777 7777 7777 xxxx with an expiration date of December, 2013.

Test Case	Type	PAN	Amount	Address Postal Code	Expected Result
AVS01	SALE or AUTHONLY	7711	\$11.00	1234 Fifth Street 77708	Approved,both address & zip match
AVS02	SALE or AUTHONLY	7729	\$11.00	1234 Fifth Street 11100	Approved,address matches, zip not
AVS03	SALE or AUTHONLY	7737	\$11.00	1234 Fifth Street 88809	Approved,zip matches, address not
AVS04	SALE or AUTHONLY	7745	\$11.00	1234 Fifth Street 44403	Approved,address & zip don't match
AVS05	SALE or AUTHONLY	7752	\$11.00	1234 Fifth Street 55504	Approved,retry of AVS needed
AVS06	SALE or AUTHONLY	7760	\$11.00	1234 Fifth Street 99906	Approved,AVS unavailable
AVS07	SALE or AUTHONLY	7778	\$11.00	1234 Fifth Street 33312	Approved, international non-AVS participant
AVS08	AUTHONLY	7786	\$0.00	1234 Fifth Street 55508	Approved,both address & zip match
AVS09	SALE or AUTHONLY	7794	\$10.05	1234 Fifth Street	Declined,AVS unavailable

				66606	
AVS10	SALE or AUTHONLY	7810	\$12.51	1234 Fifth Street 00006	Declined,AVS unavailable

### 9.2.3 UCAF Test Cases (Verified-by-Visa and SecureCode)

The Verified-by-Visa and SecureCode transaction authentication features help to assure that the actual cardholder is involved in a transaction. UCAF processing is available to ecommerce merchants who add web-based password authentication support, enabling them to submit authentication keys to transaction processing for both Visa and MasterCard transactions. UCAF test cases use card number 4888 8888 8888 xxxx:

Test Case	Type	PAN	Amount	CAVV (in base 64)XID (in base 64)	Expected Result
UCAF01	SALE	8887	\$11.00	AAABAQhkdgAAAAAAGR2AAAAAA= HX1+XXeczE4V4TvPN/A/PIi5oMw=	Approved,UCAF result "D"
UCAF02	SALE	8895	\$11.00	AAAAAAAAAAAAAAAAAAAAAAAAAAAA= djiXD8eGnoNPyd4A7eLnr7c5WTA=	Approved,UCAF result "0"

## 9.3 ACH Transactions

Merchants may submit bank drafts to JetPay. JetPay can enable properly configured merchants to send checks and savings account drafts directly into their bank account. By submitting the ABA routing code listed on every check along with the bank account number and the accountholder's name, JetPay will route these bank drafts directly into the merchant's bank account. Refunds of bank drafts are also available.

### 9.3.1 Checking Account and Savings Account Transactions

Bank accounts submitted through ACH require an ABA number, and account number, an accountholder's name, and an identifying check number.

The following processes prescribe the test cases for ACH processing. Use an ABA routing code of 122000496. For the cardholder name, use "John Q. Public" or the name of your favorite Disney character.

Test Case	Type	Account Number	Check Number	Amount	Expected Result
ACH01a	CHECK	123411	234	\$10.00	Approved
ACH01b	REVERSAL	123411	234	\$10.00	Successful
ACH02a	CHECK	552777	567	\$10.00	Approved
ACH02b	VOIDACH	552777	567	\$10.00	Transaction not found
ACH03a	CHECK	765432	890	\$10.00	Approved
ACH03b	VOIDACH	765432	890	\$15.00	Transaction not found
ACH04	VOIDACH	559277	1122	\$18.00	Transaction not found

## 10 Verification and Validation Features

JetPay offers multiple fraud avoidance features, including address verification (AVS), card verification (CVV2), BIN blocking, and velocity checking.

JetPay supports the Address Verification Service (AVS) and the Automatic Address Verification (AAV) features offered by Visa, MasterCard, and American Express, generically referred to as "AVS". A merchant sends address information in a transaction to JetPay and the address information is automatically forwarded for verification. The merchant receives an AVS response or result code from the issuer, verifying address information either fully or conditionally. Submitting AVS information reduces risk and lowers transaction cost. The merchant can also judge the validity of a customer's billing address information by using AVS through JetPay.

The CVV2/CVC2/CID value printed on the back of most credit cards can also be submitted in a transaction to JetPay. Submitting the CVV2 value to JetPay reduces risk, lowers risk, and reduces a merchant's charge back liability.

Merchants can configure JetPay to automatically react to specific BIN values or country codes, either accepting or rejecting transactions based on a card's BIN or an issuer's acquirer country code. If a merchant wishes to restrict business with a specific issuer or a specific country, the JetPay BIN Blocking service can filter those transactions according to the merchant's wishes.

This document is targeted at both the technical and business staffs that wish to understand AVS results. This document contains three tables, one for each credit card type. The brief explanation of each AVS result in this document is sufficiently described for most business or technical purposes. If an expanded understanding of AVS results is needed, contact JetPay with your specific question, comment, or concern about AVS results.

The information within this document is considered draft material. The details related by this document, especially URL, routing code are subject to alteration without notice. JetPay discourages distributing this document unless explicit permission is granted on a case-by-case basis.

## 10.1 Address Verification Services

JetPay supports the Address Verification Service (AVS) and the Automatic Address Verification (AAV) features offered by Visa, MasterCard, and American Express, generically referred to as "AVS". When a merchant sends address information in a transaction, JetPay automatically submits this information for verification. The merchant will receive an AVS response or result code from the issuer that verifies address information either fully or conditionally. In addition to reducing risk and transaction cost, the merchant can also judge the validity of a customer's billing address information by using AVS through JetPay.

Both Visa and MasterCard refer to address verification as "AVS", while American Express refer to it as "AAV". American Express augments their AAV functionality with an additional name verification feature, allowing a merchant to verify the cardholder's name. The result codes and responses sent by Visa, MasterCard, and American Express are all similar and they overlap without creating irreconcilable discrepancies. The AVS and AAV features combine together compatibly.

A merchant might also wish to develop their own software capable of implementing business decisions based on the AVS result. It's up to the merchant to establish the level of automated support they need for reacting to AVS results.

Finally, JetPay's Automated AVS Rejection feature enables a merchant to be configured to automatically decline any transactions that fail to achieve a satisfactory AVS response. Automated AVS Rejection is useful when the

correct billing address information submitted by a customer is important to the merchant. Merchants can arrange to be configured for this feature after discussing it with an account representative.

The pages in this section include an explanation of AVS rejection as well as tables defining the AVS/AAV responses, as documented by Visa, MasterCard, and American Express.

### 10.1.1 Visa AVS Result Codes

Code	Definition
A	Street addresses match. The street addresses match but the postal/ZIP codes do not, or the request does not include the postal/ZIP code.
B	Street addresses match. Postal code not verified due to incompatible formats.(Acquirer sent both street address and postal code).
C	Street address and postal code not verified due to incompatible formats.(Acquirer sent both street address and postal code).
D	Street address and postal codes match.
F	Street address and postal ode match.Applies to U.K. Only.
G	Address information not verified for international transaction.
I	Address information not verified.
M	Street address and postal code match.
N	No match. Acquirer sent postal/ZIP code only, or street address only, or both postal code and street address.
P	Postal code match. Acquirer sent both postal code and street address, but street address not verified due to incompatible formats.
R	Retry: System unavailable or timed out. Issuer ordinarily performs their own AVS but was unavailable. Available for U.S. issuers only.
S	N/A, replaced with either "U" or "G".
U	Address not verified for domestic transaction. Visa tried to perform check on issuer's behalf but no AVS information was available on record, issuer is not an AVS participant, or AVS data was present in the request but issuer did not return an AVS result.
W	N/A, replaced with "Z".
X	N/A, replaced with "Y".
Y	Street address and postal code match.
Z	Postal/ZIP matches; street does not match or street address not included in request.

### 10.1.2 MasterCard AVS Responses

Code	Definition
A	Address matches, postal code does not.
B	Visa Only. Street address match. Postal code not verified because of incompatible formats. (Acquirer sent both street address and postal code.)
C	Visa Only. Street address and postal code not verified because of incompatible formats. (Acquirer sent both street address and postal code).
D	Visa Only. Street addresses and postal codes match.

G	Visa Only. Non-AVS participant outside the U.S.; address not verified for international transaction.
I	Visa Only. Address information not verified for international transaction.
M	Visa Only. Street addresses and postal codes match.
N	Neither address nor postal code matches.
P	Visa Only. Postal codes match. Street address not verified because of incompatible formats. (Acquirer sent both street address and postal code).
R	Retry, system unable to process.
S	AVS currently not supported.
U	No data from issuer/Authorization System.
W	For U.S. addresses, nine-digit postal code matches, address does not, for address outside the U.S. postal code matches, address does not.
X	For U.S. addresses, nine-digit postal code and address matches, for address outside the U.S., postal code and address match.
Y	For U.S. addresses, five-digit postal code and address matches.
Z	For U.S. addresses, five-digit postal code matches, address does not.

### 10.1.3 American Express AAV Responses

Code	Definition
A	Address only correct.
C	Card Member information and Ship-to information verified - Standard.
G	Card Member information and Ship-to information verified - Fraud Protection Program
K	Card Member Name matches.
L	Card Member Name and Billing ZIP match.
M	Card Member Name, Billing Address and ZIP match.
N	No, Address and ZIP are both incorrect.
O	Card Member Name and Billing Address match.
R	System Unavailable; retry.
S	Address Verification subscribed valid (not subscribed).
U	Information unavailable.
Y	Yes, Address and ZIP are both correct.
Z	ZIP only correct.

## 10.2 Card Validation Using CVV2, CVC2, and CID

CVV2 is the "Card Verification Value 2" feature offered by Visa, CVC2 is the "Card Validation Code" feature offered by MasterCard, and CID is the "Card Identifier Code" offered by American Express. JetPay documentation refers to these three features collectively as "CVV2". When a merchant sends the three- or four-digit CVV2 value in a transaction, JetPay automatically submits this CVV2 value for verification. The merchant will receive a CVV2 result code from the issuer that validates a credit card either fully or conditionally. CVV2 enables a merchant to reduce transaction risk, lower transaction cost, S through JetPay.

The Visa's CVV2 and MasterCard's CVC2 are always three-digit decimal values. American Express's CID is always a four-digit value. The CVV2 value is usually printed within the "signature block" on the back of a credit

card (sometimes accompanied by the last four digits of the credit card's PAN). This value may be different on multiple cards issued upon the same account. Note: The CVV2/CVC2/CID value might conceivably begin with a zero; if a leading zero digit is present, it is obviously explicit, never implicit, and must never be presumed to be either insignificant or optional.

The CVV2, CVC2, and CID result codes sent by Visa, MasterCard, and American Express are similar and overlap without creating conflict. Although American Express doesn't actually generate a separate CID result code, JetPay will generate a compatible CVV2 result code that properly represents the actual CID result.

A merchant might also wish to develop their own software capable of implementing business decisions based on the CVV2 result. It's up to the merchant to establish the level of automated support they need for reacting to CVV2 results.

Finally, JetPay's Automated CVV2 Rejection feature enables a merchant to be configured to automatically decline any transactions that fail to achieve a satisfactory CVV2 response. Automated CVV2 Rejection is useful when attempting to insure that the customer can only submit a valid credit card. Merchants can arrange to be configured for this feature after discussing it with an account representative.

### 10.2.1 JetPay CVV2 Result Codes

Code	Definition
M	Match on CVV2/CVC2/CID. Valid CVV2/CVC2/CID
N	No match on CVV2/CVC2/CID. Invalid CVV2/CVC2/CID
P	Not processed or unable to process.
S	CVV2/CVC2 should be on the card, but the merchant indicates that it is not.
U	Issuer not certified for CVV2/CVC2.

Note: American Express enforces a policy of declining any transaction which submits a non-matching CID value. A few of the Visa and MasterCard issuers may also enforce a similar policy.

## 11 JetPay Test System

The JetPay Test System allows developers to simulate the submission and subsequent approval (or denial) of financial transactions. Merchants can system-test their financial transaction programs, building and submitting their JetPay transactions in a safe environment. The Test System operates in a non-production environment; any transaction submitted to the Test System does not actually enter the world of live financial transactions. Using the Test System, merchants can debug their custom programs without worrying about bogus test transactions being processed as live transactions.

The Test System uses virtually the same JetPay software as the production environment. Use the Test System with the knowledge that it will operate in the same manner as the production system. The minor differences between the Test System and the production system are:

1. the back end simulators
2. enhancements being introduced for test purposes prior to deployment in production
3. the test database, a separate database from the production database.

The Test System is disconnected from the financial institutions. Instead of communicating with financial institutions, the JetPay Test System accesses a suite of back-end simulators that send artificial responses back to JetPay, thereby simulating the connections with Visa, MasterCard, American Express, and numerous other financial organizations. Few organizations provide any system for simulating real-time responses for financial transactions, but the JetPay Test System does.

Due to ongoing development, the newest JetPay features will always appear on the Test System before their deployment into production. Some merchants might require the deployment of brand new JetPay features. The Test System serves as the test bed for the newest features. By deploying new features on the Test System, this also allows JetPay to rigorously test the backward compatibility of those enhancements prior to deployment.

The Test System has features that allow merchants to test both approved transactions and declined transactions. Methods for configuring such tests are explained in this document.

This document lists the various features of the Test System. This document assumes that the reader is capable of posting an XML transaction to JetPay. This document concentrates on relating the testing methodology available to developers through the Test System.

## 11.1 Connecting to the JetPay Test System

For XML development, testing XML-generating software, or for testing applications linked with the JetCom DLL, the Internet connection to the JetPay Test System is at the following URL:

<https://test1.jetpay.com/jetpay>

The JetPay Test System's URL should be used for all testing. The production gateway should never be used for test purposes (unless a test of live transaction data is deliberately planned with intent to settle that transaction data).

JetPay prohibits stress testing over its Internet connections without explicit prior consent. Any entity discovered to be stress testing over the Internet without consent of JetPay will be categorized as a Denial Of Service operator and barred from using JetPay's URLs without notice.

## 11.2 The "TESTTERMINAL" TerminalID

The JetPay Test System recognizes all the terminal IDs that are already boarded on the production system plus a few anonymous terminal IDs. Merchants should use their actual terminal IDs on the Test System. However, the Test System also recognizes an anonymous terminal ID, "TESTTERMINAL", which can be used in the absence of a properly boarded merchant account.

The "TESTTERMINAL" terminal ID should only be used when a merchant account is otherwise unavailable. Many different developers at diverse companies use the TESTTERMINAL account. The ubiquity of TESTTERMINAL transactions on the Test System complicates JetPay's technical support for your development team. Avoid using the TESTTERMINAL account, if possible.

## 11.3 Simulating "APPROVED" Transactions

The JetPay Test System will approve any transaction having an whole-dollar amount submitted in the transaction. By submitting a transaction with some dollar amount plus zero cents, the JetPay Test System will approve the transaction.

It's that simple: submit a transaction for an whole number of dollars (and no pennies) and the Test System will send back a JetPay response with a "000" ActionCode, an "APPROVED" ResponseText, and a six-character Approval value.

An example of a JetPay request that the Test System approves is this test transaction for \$99.00:

```
<JetPay>
<TransactionType>SALE</TransactionType>
<TerminalID>TESTTERMINAL</TerminalID>
<TransactionID>010327153017T10017</TransactionID>
<CardNum>4000300020001000</CardNum>
<CardExpMonth>12</CardExpMonth>
<CardExpYear>15</CardExpYear>
<TotalAmount>9900</TotalAmount>
</JetPay>
```

The following shows how the successful "APPROVED" response would look.

```
<JetPayResponse>
<TransactionID>010327153017T10017</TransactionID>
<ActionCode>000</ActionCode>
<Approval>002F6B</Approval>
<ResponseText>APPROVED</ResponseText>
</JetPayResponse>
```

Other Amount values will also generate an "APPROVED" response. By experimenting with other penny amounts, you can find other values that will also be "APPROVED".

## 11.4 Simulating "DECLINED" Transactions

To generate a "DECLINED" response, the transaction should be sent with a fractional dollar amount. Not all fractional dollar amount will generate a decline.

The so-called "hard decline" response is the most common type of "DECLINED" transaction. In the response, the ActionCode is "005" (for Visa and MasterCard) or "100" (for American Express). You can simulate a "DECLINED" transaction by inserting some dollar value plus five cents in the Amount.

Observe the following example of a JetPay transaction for \$52.05 that causes the Test System to react with a "hard decline" result:

```
<JetPay>
<TransactionType>SALE</TransactionType>
<TerminalID>TESTTERMINAL</TerminalID>
<TransactionID>010327153x17T10418</TransactionID>
<CardNum>4000300020001000</CardNum>
<CardExpMonth>12</CardExpMonth>
<CardExpYear>15</CardExpYear>
```

```
<TotalAmount>5205</TotalAmount>
</JetPay>
```

The following shows how the response to that transaction.

```
<JetPayResponse>
<TransactionID>010327153x17T10418</TransactionID>
<ActionCode>005</ActionCode>
<ResponseText>DECLINED</ResponseText>
</JetPayResponse>
```

The key observation in the above example is the last two digits within the TotalAmount of the initial JetPay request transaction. The back-end simulators for Visa, MasterCard, and American Express examine the penny-amount of the TotalAmount, generating a predictable response based on that value. Since the TotalAmount is "5205" and the last two digits are "05", the simulators will return an ActionCode of either "005" (for Visa and MasterCard) or "100" (for Amex).

### 11.4.1 List of Amount and Action codes

The JetPay Test System generates a "DECLINED" transaction for many different penny amounts. Each card type had a different set of action codes that is equivalent to the action codes returned by that card association.

Pennies	Amex	Discover	MC	Visa
00	000	000	000	000
01	100	001	001	001
02	000	002	000	002
03	003	003	003	003
04	200	004	004	004
05	100	005	005	005
06	181	000	000	006
07	200	007	000	007
08	000	008	000	000
09	000	000	000	000
10	000	010	000	000
11	000	011	000	000
12	104	012	012	012
13	110	013	013	013
14	111	014	014	014
15	111	015	015	015
16	000	000	000	000
17	000	000	000	000
18	000	000	000	000
19	103	019	000	019
20	000	000	000	000

JetPay Authorization XML Specification © 2012 JetPay LLC.

21	182	000	000	000
22	000	000	000	000
23	000	000	000	000
24	000	000	000	000
25	000	000	000	000
26	000	000	000	000
27	000	000	000	000
28	109	000	000	000
29	000	000	000	809
30	000	030	030	805
31	000	031	000	047
32	000	000	000	049
33	000	033	000	806
34	000	034	000	807
35	000	035	000	045
36	000	036	000	045
37	000	037	000	808
38	000	038	000	808
39	111	039	000	808
40	000	040	000	999
41	200	041	041	041
42	000	000	000	807
43	200	043	043	043
44	000	000	000	899
45	000	000	000	000
46	000	000	000	000
47	000	000	000	000
48	000	000	000	000
49	000	000	000	000
50	000	000	000	000
51	110	051	051	051
52	105	000	000	052
53	105	053	000	053
54	101	054	054	054
55	183	055	055	055
56	000	056	000	000
57	115	057	057	057
58	103	058	058	058
59	000	059	000	059
60	000	060	000	000
61	110	061	061	000

JetPay Authorization XML Specification © 2012 JetPay LLC.

62	189	062	062	062
63	000	063	063	000
64	000	064	000	000
65	188	065	065	000
66	000	066	000	000
67	000	067	000	000
68	000	068	000	000
69	000	000	000	000
70	000	000	000	000
71	000	000	000	000
72	000	000	000	000
73	000	000	000	000
74	000	000	000	000
75	000	000	075	075
76	000	076	076	000
77	000	077	077	807
78	187	078	078	078
79	000	000	000	000
80	125	000	000	080
81	000	000	000	081
82	122	000	000	082
83	000	000	000	083
84	000	000	084	000
85	400	000	000	000
86	000	000	000	000
87	000	087	000	000
88	000	000	000	000
89	000	000	000	000
90	992	992	992	000
91	107	091	091	091
92	092	092	092	092
93	000	093	000	093
94	000	094	094	000
95	000	000	000	000
96	000	096	096	096
97	000	000	000	000
98	000	000	000	000
99	000	000	000	000

## 11.5 Other Responses

The responses on the previous table are not the only ActionCode values you can obtain. Some responses are generated internal to the JetPay server, relying on information already configured within JetPay and not relying on external communications.

An "025" ActionCode is generated when a record cannot be found. For example, if a merchant sends a "CAPT" transaction to JetPay, attempting to capture a previously authorized transaction, JetPay sends a "025" ActionCode whenever that previously authorized transaction cannot be found in the system. Because a merchant can only capture an authorized transaction that has been sent previously, it's important to recognize that this "025" ActionCode will be sent whenever that authorization is absent.

The "981" ActionCode is generated by JetPay for merchant accounts that are configured to reject transactions based on AVS results. By observing the AVS results, the merchant can verify that transactions are being properly rejected by JetPay whenever the configured AVS responses appear. Unless a merchant's account is boarded to reject certain AVS results, this "981" ActionCode will not appear.

For merchants who do not accept certain credit card types, the "913" ActionCode gets returned when those card types are detected. For example, a merchant who does not wish to accept any Discover cards can be boarded to automatically decline that card type; whenever a Discover transaction is sent by that merchant, a "913" ActionCode is returned. Only a merchant account that is boarded to reject certain card types will receive this "913" ActionCode.

If you receive a "900" ActionCode, your incoming XML transaction contains an XML syntax error. All XML documents that are not "well-formed" cause the JetPay server to return a "900" ActionCode. Additionally, data values submitted in your XML might be invalid JetPay syntax, and this will also return a "900" ActionCode. Within a "900" response will be an ErrMsg tag delimiting an error message; this will explain the technical reason (and frequently the location) for a syntax error. By comparing the ErrMsg text to your submitted XML, you can figure out how to correct all "900" responses. You should correct all "900" syntax errors on the test system before sending any transactions to the production server.

## 11.6 Simulating AVS Responses

AVS stands for "Address Verification Service." The back-end simulators will trigger specific AVS responses that react to the incoming transaction information.

First of all, keep in mind that all AVS responses from the JetPay Test System are simulated. If the JetPay Test System actually returned a true AVS result that accurately related the status of cardholder address information, issues concerning privacy laws arise. Support personnel at JetPay occasionally hear complaints from merchants concerning their misconceptions about simulated AVS responses. JetPay asks the indulgence of merchants in this matter.

Within the JetPay Test System, the last two digits of the postal code (or Zip code) determine the AVS result sent by the back-end simulator. The simulator looks at the last two digits of the postal code and returns an AVS result that corresponds with the digits.

To obtain a completely positive AVS result, the last two digits of the postal code should always be a "08". The following XML example applies:

```

<JetPay>
<TransactionType>SALE</TransactionType>
<TerminalID>TESTTERMINAL</TerminalID>
<TransactionID>010327153017T10a17</TransactionID>
<CardNum>4000300020001000</CardNum>
<CardExpMonth>12</CardExpMonth>
<CardExpYear>15</CardExpYear>
<CardName>Bob Tucker</CardName>
<TotalAmount>99900</TotalAmount>
<BillingAddress>3361 Boyington Ste. 180</BillingAddress>
<BillingCity>Dallas</BillingCity>
<BillingStateProv>TX</BillingStateProv>
<BillingPostalCode>75008</BillingPostalCode>
<BillingCountry>USA</BillingCountry>
<BillingPhone>214-890-1800</BillingPhone>
<Email>btucker@jetpay.com</Email>
</JetPay>
    
```

The following shows how the response would look.

```

<JetPayResponse>
<TransactionID>010327153017T10a17</TransactionID>
<ActionCode>000</ActionCode>
<Approval>002F6B</Approval>
<ResponseText>APPROVED</ResponseText>
<AddressMatch>Y</AddressMatch>
<ZipMatch>Y</ZipMatch>
<AVS>Y</AVS>
</JetPayResponse>
    
```

### 11.6.1 Postal Code Values That Affect the AVS Result

By varying the last digit of the postal code, you can get multiple different AVS results. The following table lists the various AVS responses sent by the back-end simulator for specific values:

Last Two Digits	AVS Result	Description
00	A(all)	Address matches, postal code absent or does not match. (All)
01	D(all)	Street address and postal codes match. (Visa/MC)
		CM Name incorrect, postal code matches. (Amex)
02	M(all)	Street address and postal code match. (Visa/MC)
		CM Name, street address and postal code match. (Amex)
03	N(all)	Neither address nor postal code matches. (All)
04	R(all)	Retry (All)
05	S(all)	AVS unavailable. (All)
06	U(all)	AVS unavailable. (All)
07	Z(Visa)	W was replaced by Z. (Visa)
	W(MC/Amex)	Postal code matches, address absent or does not match (MC)

		CM Name, street address and postal code are all incorrect. (Amex)
08	Y(all)	Street address and postal code match. (All)
09	Z(all)	Postal code matches, address absent or does not match. (All)
10	B(Visa/MC)	Street address match. Postal code has invalid format (Visa/MC)
	E(Amex)	CM Name incorrect. Street address and postal code match. (Amex)
11	C(Visa/MC)	Street address and postal code have invalid formats. (Visa/MC)
	K(Amex)	CM Name matches. (Amex)
12	G(Visa/MC)	Non-AVS participant outside U.S. Address not verified. (Visa/MC)
	L(Amex)	CM Name and postal code match. (Amex)
13	I(Visa/MC)	Street address not verified for international transaction. (Visa/MC)
	O(Amex)	CM Name and address match. (Amex)
14	P(Visa/MC)	Postal code match. Street address has invalid format. (Visa/MC)
	F(Amex)	CM Name incorrect. Street address matches. (Amex)
15	X(Visa/MC)	Nine-digit postal code match in U.S. Postal code and address match for outside U.S. (Visa/MC)
16	F(Visa)	Street address and postal code match for U.K. only (Visa)

## 11.7 Simulating CVV2/CVC2/CID Responses

The term "CVV2" defines a feature that covers Visa's CVV2, MasterCard's CVC2, and Amex's CID. Visa defines CVV2 as "Card Verification Value 2," MasterCard defines CVC2 as "Card Validation Code 2," and Amex defines CID as "Card Identifier."

The back-end simulators will trigger specific CVV2/CVC2/CID responses that react to the incoming transaction information.

All CVV2/CVC2/CID responses from the JetPay Test System are simulated. Support personnel at JetPay occasionally hear complaints from merchants concerning their misconceptions about simulated responses. JetPay asks the indulgence of merchants in this matter.

Within the Test System, the third digit of the submitted CVV2 value determines the CVV2 result sent by the back-end simulator. The simulator looks at the third digit of the CVV2 value and returns a CVV2 result that corresponds with that digit.

To obtain a completely positive CVV2 result, the third digit of the CVV2 value can be a "0". The following XML example applies:

```
<JetPay>
<TransactionType>SALE</TransactionType>
<TerminalID>TESTTERMINAL</TerminalID>
<TransactionID>010327153017T10a18</TransactionID>
<CardNum>4000300020001000</CardNum>
<CardExpMonth>12</CardExpMonth>
<CardExpYear>15</CardExpYear>
<TotalAmount>99900</TotalAmount>
```

```
<CVV2>890</CVV2>
</JetPay>
```

The following shows how the response would look.

```
<JetPayResponse>
<TransactionID>010327153017T10a18</TransactionID>
<ActionCode>000</ActionCode>
<Approval>002F6B</Approval>
<ResponseText>APPROVED</ResponseText>
<CVV2>M</CVV2>
</JetPayResponse>
```

## 11.7.1 CVV2 Values That Affect the CVV2 Result

By varying the third digit of the CVV2 value, you can get multiple different CVV2 results. The following table lists the various CVV2 responses sent by the back-end simulator for specific values:

3 <sup>rd</sup> CVV2 Digit 4 <sup>th</sup> CID Digit	CVV2 Result	Description
0	M(all)	CVV2/CVC2/CID matches
1	M(all)	
2	N(all)	CVV2/CVC2/CID does not match
3	P(all)	CVV2/CVC2/CID not processed
4	U(Visa/MC)P(Amex)	Issuer not certified for CVV2/CVC2 (Visa/MC) CID not processed (Amex)
5	S(Visa/MC)P(Amex)	CVV2/CVC2 should be on the card, but the merchant indicates it is not (Visa/MC) CID not processed. (Amex)
6	M(all)	CVV2/CVC2/CID matches
7	M(all)	
8	M(all)	
9	M(all)	

## 12 Result Codes

### 12.1 AVS Result Codes

JetPay offers cardholder address verification through AVS, a system available widely throughout North America and much of Europe. With address verification, the merchant submits a customer's address and postal code information in a transaction; the credit card issuer verifies and validates the address information against their internal billing address data. The result code sent by an issuer advises the merchant of a customer's billing integrity when completing the credit card transaction.

Visa and MasterCard both call this feature AVS, which stands for "Address Verification Service." American Express calls this feature AAV, which stands for "Address Authentication and Verification."

Almost all North American issuers support AVS using Visa and MasterCard association rules. According to these rules, an issuer compares the numeric digits within a cardholder's address and postal code information with their own stored billing information. All alphabetic information is ignored. The address and postal code match (or fail to match) and an AVS result code is generated by the issuer. This is returned to the merchant during an authorization.

American Express adds cardholder name matching to their AAV. A merchant can submit their customer's name in addition to the billing address information and may receive a number of additional AVS result codes. Shipping address information may also be submitted in an Amex transaction, and additional result codes are anticipated from Amex in the future.

For most issuers, the AVS works independent of the action code. In other words, a transaction may be approved even though billing address information doesn't match. Because transaction approval is independent of billing address integrity, it's up to the merchant to decide the importance of correct billing address information when completing a sale. A merchant may simply allow a transaction to proceed, or a merchant might independently discontinue the sale in spite of an approval.

JetPay, based on the AVS result code, automatically fills the AddressMatch and ZipMatch elements. The possible AddressMatch and ZipMatch values are "Y", "N", or "X", and the value is deterministic with the AVS result code.

JetPay offers an "Automatic AVS Rejection" feature, where merchants may automatically decline any transactions showing degraded AVS results. Automatic AVS Rejection is an optional subscription service that requires a merchant to subscribe before JetPay will automatically reject transactions on behalf of the merchant.

A result code indicating the AVS ("Address Verification Service") results for a transaction may be:

A	Address matches, but postal code does not match.
B	Visa only: Street address matches, postal code not verified because of incompatible format.
C	Visa only: Street address and postal code not verified because of incompatible formats.
D	Visa only: Street address and postal code match. Amex only: Cardholder name incorrect, postal code matches.
E	Amex only: Cardholder name incorrect, address and postal code matches.
F	Amex only: Cardholder name incorrect, billing address matches.
G	Visa only: Global non-AVS participant outside the U.S.; address is not verified for international transaction.
K	Amex only: Cardholder name matches.
L	Amex only: Cardholder name and postal code match.
M	Amex only: Cardholder name, address and postal code match. Visa only: Street address and postal code match.
N	Neither the address nor the postal code matches.
O	Amex only: Cardholder name and address match.
P	Visa only: Postal code matches, street address not verified because of incompatible format.
R	Retry, system unable to process.
S	MasterCard only: AVS currently not supported.
U	Unavailable. Address not verified. AVS attempted, but no AVS information was available(issuer is not an AVS participant or issuer did not return an AVS result).

W	MasterCard only: Nine-digit postal code and address match. Amex only: Cardholder name, address, and postal code are all incorrect.
X	MasterCard only: Five-digit postal code and address match.
Y	Both the address and the postal code match exactly.
Z	Postal code matches, but address does not match.

## 12.2 CVV2 Result Codes

JetPay enables credit card validation through the CVV2 feature, a system widely supported throughout the credit card industry. With this feature, every physical credit card has a three- or four-digit value imprinted on the credit card in addition to the card number. Submitting this three- or four-digit "security code" enhances fraud detection for a transaction, ensuring that a genuine physical credit card is present during that transaction.

Visa cards have a three-digit CVV2 value, MasterCard cards have a three-digit CVC2 value, and American Express uses a four-digit CID value to enable card validation. This three- or four- digit value is found imprinted on the back side of a credit card, usually inside the signature block (it may also be imprinted on the front side of the card).

The response code to a Visa CVV2 or a MasterCard CVC2 or an American Express CID submission may be:

M	CVV2 matches.
N	CVV2 does not match.
P	Not processed.
S	CVV2 should be on the card, but merchant indicates it is not.
U	Issuer is not certified for CVV2 or has not provided encryption keys. Participation in CVV2/CVC2 is optional for issuers. The subscribing issuers submit their CVV2/CVC2 keys to Visa and MasterCard, and these keys are kept secret. The individual CVV2/CVC2 implementation policies of the tens of thousands of issuers are confidential, and issuers may change their internal CVV2/CVC2 policies without notification. Statistics are unavailable as to how many issuers subscribe (or don't subscribe) to CVV2/CVC2.

Issuers may decide to return a "806" action code, indicating that a Visa issuer has declined a transaction due to the submission of an invalid CVV2 value. The "806" action code (a response allowed under Visa's association rules) enables issuers to independently decide a policy of rejecting transactions having invalid CVV2 values. An "806" action code may have a "N" or "U" or "P" response, but never an "M" response.

Another factor that may affect CVV2 result codes is stand-in processing. When Visa or MasterCard performs stand-in processing for an issuer (because the issuer is otherwise unable to respond directly for a transaction), Visa/MC will perform the CVV2 calculation on behalf of any issuers who subscribe to CVV2.

## 12.3 CAVV Result Codes

Cardholder authentication is now available in the credit card industry through Visa's Verified by Visa program and MasterCard's SecureCode program. Using these systems, an ecommerce merchant enables their customer to submit an account ID and a password to authenticate their presence during an online transaction. The authentication process generates a token; the token is submitted within an authorization for a transaction.

JetPay LLC supports these two programs, allowing CAVV and UCAF data to be submitted to JetPay. The merchant receives a CAVV result code after a transaction is approved.

The response code to a Visa CAVV or a MasterCard UCAF submission is "0" thru "9" or "A" thru "D", where:

0	CAVV authentication results invalid.
1	CAVV failed validation - authentication.
2	CAVV passed validation - authentication.
3	CAVV passed validation - attempt.
4	CAVV failed validation - attempt.
5	Not used (reserved for future use).
6	CAVV not validated, issuer not participating in CAVV validation.
7	CAVV failed validation - attempt (U.S.-issued cards only).
8	CAVV passed validation - attempt (U.S.-issued cards only).
9	CAVV failed validation - attempt (U.S.-issued cards only).
A	CAVV passed validation - attempt (U.S.-issued cards only).
B	CAVV passed validation - information only, no liability shift.
C	CAVV was not validated - attempt.
D	CAVV was not validated - authentication.

## 13 The JetPay Authorization XML Schema

The rules designated by the JetPay schema within this section get applied against all incoming XML transactions. For your perusal, JetPay applies the following schema at the interface to the server:

```
<xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      $Id: JetPaySchema.xsd.cpp 3234 2011-03-24 18:53:49Z gscogin $
      JetPay schema for JetPay LLC.
      Copyrights 2006 to 2010 by JetPay LLC. All rights reserved.
    </xs:documentation>
  </xs:annotation>
  <!-- ***** JETPAY TRANSACTION REQUEST ***** -->
  <xs:element name = "JetPay" type = "JetPayRequestType" />
  <xs:complexType name = "JetPayRequestType">
    <xs:all>
      <xs:element ref = "TransactionType" minOccurs = "1" maxOccurs = "1" />
      <xs:element ref = "TerminalID" minOccurs = "1" maxOccurs = "1" />
      <xs:element ref = "TransactionID" minOccurs = "1" maxOccurs = "1" />
      <xs:element ref = "BatchID" minOccurs = "0" maxOccurs = "1" />
      <xs:element ref = "Password" minOccurs = "0" maxOccurs = "1" />
      <xs:element ref = "PinBlock" minOccurs = "0" maxOccurs = "1" />
      <xs:element ref = "Ksn" minOccurs = "0" maxOccurs = "1" />
      <xs:element ref = "Approval" minOccurs = "0" maxOccurs = "1" />
      <xs:element ref = "RoutingCode" minOccurs = "0" maxOccurs = "1" />
      <xs:element ref = "OrderNumber" minOccurs = "0" maxOccurs = "1" />
      <xs:element ref = "ACH" minOccurs = "0" maxOccurs = "1" />
      <xs:element ref = "Verification" minOccurs = "0" maxOccurs = "1" />
      <xs:element ref = "CardNum" minOccurs = "0" maxOccurs = "1" />
      <xs:element ref = "CVV2" minOccurs = "0" maxOccurs = "1" />
      <xs:element ref = "CardExpMonth" minOccurs = "0" maxOccurs = "1" />
      <xs:element ref = "CardExpYear" minOccurs = "0" maxOccurs = "1" />
      <xs:element ref = "CardStartMonth" minOccurs = "0" maxOccurs = "1" /> <!-- EUROPEAN CARDS -->
      <xs:element ref = "CardStartYear" minOccurs = "0" maxOccurs = "1" /> <!-- EUROPEAN CARDS -->
      <xs:element ref = "Issue" minOccurs = "0" maxOccurs = "1" /> <!-- EUROPEAN CARDS -->
    </xs:all>
  </xs:complexType>
</xs:schema>
```

## JetPay Authorization XML Specification © 2012 JetPay LLC.

```

<xs:element ref = "Track1" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "Track2" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "Token" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "CardName" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "TotalAmount" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "TenderedAmount" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "FeeAmount" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "TaxAmount" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "TaxPercentRate" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "CashbackAmount" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "BillingAddress" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "BillingCity" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "BillingStateProv" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "BillingPostalCode" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "BillingCountry" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "BillingPhone" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "Email" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "UserIPAddress" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "UserHost" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "UDField1" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "UDField2" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "UDField3" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "ShippingInfo" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "Origin" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "ReaderUsed" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "IndustryInfo" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "Test" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "Retry" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "Version" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "ActionCode" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "Memo" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "SVPTransactionID" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "Reference" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "FICode" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "ReturnURL" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "CancelURL" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "DeliveryCode" minOccurs = "0" maxOccurs = "1" />
</xs:all>
</xs:complexType>
<!-- ***** JETPAY TRANSACTION RESPONSE ***** -->
<xs:element name = "JetPayResponse" type = "JetPayResponseType" />
<xs:complexType name = "JetPayResponseType">
  <xs:all>
    <xs:element ref = "TransactionID" minOccurs = "1" maxOccurs = "1" />
    <xs:element ref = "ActionCode" minOccurs = "1" maxOccurs = "1" />
    <xs:element ref = "Approval" minOccurs = "0" maxOccurs = "1" />
    <xs:element ref = "BalanceAmount" minOccurs = "0" maxOccurs = "1" />
    <xs:element ref = "AuthorizedAmount" minOccurs = "0" maxOccurs = "1" />
    <xs:element ref = "ResponseText" minOccurs = "0" maxOccurs = "1" />
    <xs:element ref = "BatchID" minOccurs = "0" maxOccurs = "1" />
    <xs:element ref = "AdditionalInfo" minOccurs = "0" maxOccurs = "1" />
    <xs:element ref = "AddressMatch" minOccurs = "0" maxOccurs = "1" />
    <xs:element ref = "ZipMatch" minOccurs = "0" maxOccurs = "1" />
    <xs:element ref = "AVS" minOccurs = "0" maxOccurs = "1" />
    <xs:element ref = "CVV2" minOccurs = "0" maxOccurs = "1" />
    <xs:element ref = "VerificationResult" minOccurs = "0" maxOccurs = "1" />
    <xs:element ref = "IndustryInfo" minOccurs = "0" maxOccurs = "1" />
    <xs:element ref = "ErrMsg" minOccurs = "0" maxOccurs = "1" />
    <xs:element ref = "InstitutionURL" minOccurs = "0" maxOccurs = "1" />
    <xs:element ref = "SVPTransactionID" minOccurs = "0" maxOccurs = "1" />
    <xs:element ref = "BankName" minOccurs = "0" maxOccurs = "1" />
    <xs:element ref = "FICode" minOccurs = "0" maxOccurs = "1" />
  </xs:all>
</xs:complexType>

```

## JetPay Authorization XML Specification © 2012 JetPay LLC.

```
<xs:element ref = "PayDateTime" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "Reference" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "TranState" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "TranStateCode" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "Token" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "Summary" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "TransactionTotalAmount" minOccurs = "0" maxOccurs = "1" />
<xs:element ref = "TransactionTotalCount" minOccurs = "0" maxOccurs = "1" />
</xs:all>
</xs:complexType>
<!-- ***** REQUEST ELEMENTS ***** -->
<xs:element name = "TransactionType" type = "TransType" /> <!-- Required - Type of transaction to p
<xs:element name = "TerminalID" type = "String12Type" /> <!-- Required - Alphanumeric terminal iden
<xs:element name = "TransactionID" type = "TransIdType" /> <!-- Required - Alphanumeric, 18 charact
<xs:element name = "BatchID" type = "StringType" /> <!-- For merchant-initiated settlement purposes
<xs:element name = "ACH" type = "AchType" /> <!-- Banking only, separate from credit card transacti
<xs:element name = "Verification" type = "VerifyType" /> <!-- Verified by Visa, CAVV, etc. -->
<xs:element name = "Approval" type = "AuthCodeType" /> <!-- Approval Code, required for Force. -->
<xs:element name = "RoutingCode" type = "StringType" />
<xs:element name = "CardNum" type = "CardNumType" /> <!-- Credit card number for the transaction. -
<xs:element name = "CardExpMonth" type = "MonthType" /> <!-- Two-digit card expiration month. -->
<xs:element name = "CardExpYear" type = "YearType" /> <!-- Two-digit card expiration year. -->
<xs:element name = "CardStartMonth" type = "MonthType" /> <!-- Two-digit card start month. -->
<xs:element name = "CardStartYear" type = "YearType" /> <!-- Two-digit card start year. -->
<xs:element name = "Issue" type = "YearType" /> <!-- Two-digit card issue number. -->
<xs:element name = "CVV2" type = "Cvv2Type" /> <!-- Reference only - CVV2 digits from the credit ca
<xs:element name = "Track1" type = "TrackType" /> <!-- Magnetic track information, track 1 data -->
<xs:element name = "Track2" type = "TrackType" /> <!-- Magnetic track information, track 2 data -->
<xs:element name = "TotalAmount" type = "Amount12Type" /> <!-- Total amount transacted, integer, no
<xs:element name = "TenderedAmount" type = "Amount12Type" /> <!-- Actual amount received, integer,
<xs:element name = "FeeAmount" type = "Amount12Type" /> <!-- Surcharge for transaction, integer, no
<xs:element name = "TaxAmount" type = "Amount12Type" /> <!-- Tax for transaction, integer, no dolla
<xs:element name = "TaxPercentRate" type = "Amount12Type" /> <!-- Tax percent rate for transaction,
<xs:element name = "CashbackAmount" type = "Amount12Type" /> <!-- Cash for transaction, integer, no
<xs:element name = "Password" type = "StringType" /> <!-- Alphanumeric password corresponding with
<xs:element name = "PinBlock" type = "PinBlockType" />
<xs:element name = "Ksn" type = "KsnType" />
<xs:element name = "InstitutionURL" type = "String1204Type" />
<xs:element name = "SVPTransactionID" type = "Number12Type" />
<xs:element name = "BankName" type = "String50Type" />
<xs:element name = "PayDateTime" type = "DateType" />
<xs:element name = "Reference" type = "String22Type" />
<xs:element name = "TranState" type = "Number1Type" />
<xs:element name = "TranStateCode" type = "Number2Type" />
<xs:element name = "FICode" type = "Number9Type" />
<xs:element name = "ReturnURL" type = "String180Type" />
<xs:element name = "CancelURL" type = "String180Type" />
<xs:element name = "DeliveryCode" type = "DeliveryType" />
<xs:element name = "OrderNumber" type = "StringType" /> <!-- Order number (or any alphanumeric refe
<xs:element name = "CardName" type = "StringType" /> <!-- Cardholder name for credit card, bank acc
<xs:element name = "BillingAddress" type = "StringType" />
<xs:element name = "BillingCity" type = "StringType" />
<xs:element name = "BillingStateProv" type = "StringType" />
<xs:element name = "BillingCountry" type = "CountryType" />
<xs:element name = "BillingPostalCode" type = "String10Type" />
<xs:element name = "BillingPhone" type = "StringType" /> <!-- Cardholder telephone -->
<xs:element name = "Email" type = "StringType" /> <!-- Cardholder email address. -->
<xs:element name = "UserIPAddress" type = "StringType" /> <!-- Cardholder IP address. -->
<xs:element name = "UserHost" type = "StringType" /> <!-- Cardholder host name. -->
<xs:element name = "UDField1" type = "StringType" /> <!-- Data echoed on transaction reports. -->
<xs:element name = "UDField2" type = "StringType" /> <!-- Data echoed on transaction reports. -->
<xs:element name = "UDField3" type = "StringType" /> <!-- Data echoed on transaction reports. -->
```

## JetPay Authorization XML Specification © 2012 JetPay LLC.

```

<xs:element name = "ShippingInfo" type = "ShippingInfoType" />
<xs:element name = "Origin" type = "OriginType" />
<xs:element name = "ReaderUsed" type = "ReaderType" />
<xs:element name = "IndustryInfo" type = "IndustryInfoType" />
<xs:element name = "Test" type = "StringType" />
<xs:element name = "Retry" type = "StringType" />
<xs:element name = "Version" type = "VersionType" />
<xs:element name = "ActionCode" type = "ActionCodeType" />
<xs:element name = "Memo" type = "MemoType" />
<!-- ***** RESPONSE ELEMENTS ***** -->
<xs:element name = "ResponseText" type = "StringType" />
<xs:element name = "AdditionalInfo" type = "StringType" />
<xs:element name = "AddressMatch" type = "String1Type" />
<xs:element name = "ZipMatch" type = "String1Type" />
<xs:element name = "BalanceAmount" type = "Amount12Type" /> <!-- account balance -->
<xs:element name = "AuthorizedAmount" type = "Amount12Type" /> <!-- partial authorization support -->
<xs:element name = "AVS" type = "String1Type" />
<xs:element name = "VerificationResult" type = "CavvResultType" />
<xs:element name = "ErrMsg" type = "StringType" />
<xs:element name = "Token" type = "StringType" />
<xs:element name = "Summary" type = "SummaryType" />
<xs:element name = "TransactionTotalAmount" type = "Amount12Type" />
<xs:element name = "TransactionTotalCount" type = "Int5Type" />
<!-- ***** SIMPLE TYPES (without attributes) ***** -->
<xs:simpleType name = "StringType">
  <xs:restriction base = "xs:string">
    <xs:whiteSpace value = "collapse"/>
    <xs:maxLength value = "1204" />
  </xs:restriction>
</xs:simpleType>
<xs:complexType name = "TrackType">
  <xs:simpleContent>
    <xs:extension base = "xs:string">
      <xs:attribute name = "Tokenize" type = "BooleanType" default = "false" />
      <xs:attribute name = "Encrypted" type = "BooleanType" default = "false" />
      <xs:attribute name = "Encoding" type = "EncodeType" default = "BASE64" />
      <xs:attribute name = "KeyName" type = "StringType" default = "none" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:simpleType name = "BooleanType">
  <xs:restriction base = "xs:boolean">
    <xs:whiteSpace value = "collapse"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "SimpleDateTimeType">
  <xs:restriction base = "xs:string">
    <xs:pattern value = "(-?[0-9]{4}-[0-9]{2}-[0-9]{2}T\d{2}:\d{2}:\d{2}(\.\d+)?)?"/>
    <xs:whiteSpace value = "collapse"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "DateType">
  <xs:restriction base = "xs:date">
    <xs:whiteSpace value = "collapse"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "String350Type">
  <xs:restriction base = "StringType">
    <xs:maxLength value = "350"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "String20Type">

```

```

<xs:restriction base = "StringType">
  <xs:maxLength value = "20" />
</xs:restriction>
</xs:simpleType>
<xs:simpleType name = "String22Type">
  <xs:restriction base = "StringType">
    <xs:maxLength value = "22" />
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "String50Type">
  <xs:restriction base = "StringType">
    <xs:maxLength value = "50" />
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "String15Type">
  <xs:restriction base = "StringType">
    <xs:maxLength value = "15" />
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "String180Type">
  <xs:restriction base = "StringType">
    <xs:maxLength value = "180" />
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "String1204Type">
  <xs:restriction base = "StringType">
    <xs:maxLength value = "1204" />
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "String12Type">
  <xs:restriction base = "StringType">
    <xs:maxLength value = "12"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "String10Type">
  <xs:restriction base = "StringType">
    <xs:maxLength value = "10"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "String8Type">
  <xs:restriction base = "StringType">
    <xs:maxLength value = "8"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "String6Type">
  <xs:restriction base = "StringType">
    <xs:maxLength value = "6"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "String3Type">
  <xs:restriction base = "StringType">
    <xs:maxLength value = "3"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "String2Type">
  <xs:restriction base = "StringType">
    <xs:maxLength value = "2"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "String1Type">
  <xs:restriction base = "StringType">
    <xs:maxLength value = "1"/>
  </xs:restriction>
</xs:simpleType>

```

```

    </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "Number3Type">
    <xs:restriction base = "xs:unsignedShort">
        <xs:maxInclusive value = "999"/>
        <xs:whiteSpace value = "collapse"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "Number5Type">
    <xs:restriction base = "xs:unsignedLong">
        <xs:maxInclusive value = "99999"/>
        <xs:whiteSpace value = "collapse"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "Number12Type">
    <xs:restriction base = "xs:unsignedLong">
        <xs:maxInclusive value = "999999999999"/>
        <xs:whiteSpace value = "collapse"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "Number1Type">
    <xs:restriction base = "xs:unsignedShort">
        <xs:maxInclusive value = "9"/>
        <xs:whiteSpace value = "collapse"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "Number8Type">
    <xs:restriction base = "xs:unsignedLong">
        <xs:maxInclusive value = "999999999"/>
        <xs:whiteSpace value = "collapse"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "Number9Type">
    <xs:restriction base = "xs:unsignedLong">
        <xs:maxInclusive value = "9999999999"/>
        <xs:whiteSpace value = "collapse"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "Number2Type">
    <xs:restriction base = "xs:unsignedShort">
        <xs:maxInclusive value = "99"/>
        <xs:whiteSpace value = "collapse"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "Number4Type">
    <xs:restriction base = "xs:unsignedShort">
        <xs:maxInclusive value = "9999"/>
        <xs:whiteSpace value = "collapse"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "TransIdType">
    <xs:restriction base = "xs:string">
        <xs:pattern value = "[-A-Za-z0-9]{18}"/>
        <xs:length value = "18"/>
        <xs:whiteSpace value = "collapse" />
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "Digit2Type">
    <xs:restriction base = "xs:integer">
        <xs:pattern value = "\d{2}"/>
        <xs:whiteSpace value = "collapse"/>
    </xs:restriction>

```

```

</xs:simpleType>
<xs:simpleType name = "MonthType">
  <xs:restriction base = "xs:integer">
    <xs:minInclusive value = "1"/>
    <xs:maxInclusive value = "12"/>
    <xs:pattern value = "\d{2}|\d{0}"/>
    <xs:whiteSpace value = "collapse"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "YearType">
  <xs:restriction base = "xs:integer">
    <xs:minInclusive value = "0"/>
    <xs:maxInclusive value = "99"/>
    <xs:pattern value = "\d{2}|\d{0}"/>
    <xs:whiteSpace value = "collapse"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "CountryType">
  <xs:restriction base = "StringType">
    <xs:pattern value = "\d{3}|[A-Z]{3}"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "Amount12Type">
  <xs:restriction base = "xs:unsignedLong">
    <xs:maxInclusive value = "99999999999999"/>
    <xs:whiteSpace value = "collapse"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "Amount6Type">
  <xs:restriction base = "Amount12Type">
    <xs:maxInclusive value = "999999"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "Amount5Type">
  <xs:restriction base = "Amount12Type">
    <xs:maxInclusive value = "99999"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "UniqueIDType">
  <xs:restriction base = "StringType">
    <xs:maxLength value = "25"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "AuthCodeType">
  <xs:restriction base = "StringType">
    <xs:maxLength value = "6"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "ActionCodeType">
  <xs:restriction base = "StringType">
    <xs:maxLength value = "3"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "antx" >
  <xs:restriction base = "xs:string">
    <xs:pattern value = "([- 0-9])*"/>
    <xs:maxLength value = "17"/>
    <xs:whiteSpace value = "collapse"/>
  </xs:restriction>
</xs:simpleType>
<xs:complexType name = "AcctNumType">
<xs:simpleContent>

```

## JetPay Authorization XML Specification © 2012 JetPay LLC.

```
<xs:extension base = "antx">
  <xs:attribute name = "Tokenize" type = "BooleanType" default = "false" />
</xs:extension>
</xs:simpleContent>
</xs:complexType>
<xs:simpleType name = "AbaType">
  <xs:restriction base = "xs:unsignedInt">
    <xs:pattern value = "\d{9}" />
    <xs:whiteSpace value = "collapse" />
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "Ordinal3Type">
  <xs:restriction base = "xs:positiveInteger">
    <xs:maxInclusive value = "999" />
    <xs:whiteSpace value = "collapse" />
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "Int5Type">
  <xs:restriction base = "Amount5Type" />
</xs:simpleType>
<xs:simpleType name = "Int6Type">
  <xs:restriction base = "Amount6Type" />
</xs:simpleType>
<!-- ***** SIMPLE TYPES (with attributes) ***** -->
<xs:complexType name = "TransType">
  <xs:simpleContent>
    <xs:extension base = "TransEnumType">
      <xs:attribute name = "Qualification" type = "TransQualifEnumType" default = "NONE" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name = "CardNumType">
  <xs:simpleContent>
    <xs:extension base = "String350Type">
      <xs:attribute name = "CardPresent" type = "BooleanType" default = "false" />
      <xs:attribute name = "Tokenize" type = "BooleanType" default = "false" />
      <xs:attribute name = "Encrypted" type = "BooleanType" default = "false" />
      <xs:attribute name = "Encoding" type = "EncodeType" default = "BASE64" />
      <xs:attribute name = "KeyName" type = "StringType" default = "none" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name = "Cvv2Type">
  <xs:simpleContent>
    <xs:extension base = "String350Type">
      <xs:attribute name = "Encrypted" type = "BooleanType" default = "false" />
      <xs:attribute name = "Encoding" type = "EncodeType" default = "BASE64" />
      <xs:attribute name = "KeyName" type = "StringType" default = "none" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name = "EncodingType">
  <xs:simpleContent>
    <xs:extension base = "StringType">
      <xs:attribute name = "Encoding" type = "EncodeType" default = "BASE64" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name = "WorkingKeyType">
  <xs:simpleContent>
    <xs:extension base = "StringType">
      <xs:attribute name = "Encoding" type = "EncodeType" default = "HEX" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

## JetPay Authorization XML Specification © 2012 JetPay LLC.

```

    <xs:attribute name = "Management" type = "ManagementType" default = "DUKPT" />
  </xs:extension>
</xs:simpleContent>
</xs:complexType>
<xs:complexType name = "PinBlockType">
  <xs:simpleContent>
    <xs:extension base = "WorkingKeyType">
      <xs:attribute name = "Format" type = "PinBlockFormatType" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:simpleType name = "KsnType">
  <xs:restriction base = "StringType">
    <xs:pattern value = "[0-9A-Fa-f]{10,20}" />
  </xs:restriction>
</xs:simpleType>
<xs:complexType name = "CavvType">
  <xs:simpleContent>
    <xs:extension base = "EncodingType">
      <xs:attribute name = "Usage" type = "String1Type" default = "2" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name = "CavvResultType">
  <xs:simpleContent>
    <xs:extension base = "StringType">
      <xs:attribute name = "Type" type = "VerifyEnum" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name = "CheckNumType">
  <xs:simpleContent>
    <xs:extension base = "Int6Type">
      <xs:attribute name = "CheckPresent" type = "BooleanType" default = "false" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name = "DateTimeType">
  <xs:simpleContent>
    <xs:extension base = "SimpleDateTimeType">
      <xs:attribute name = "Min" type = "SimpleDateTimeType" />
      <xs:attribute name = "Max" type = "SimpleDateTimeType" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<!-- ***** COMPOUND TYPES (containing simple types) ***** -->
<xs:complexType name = "AddressType">
  <xs:all>
    <xs:element name = "Address" minOccurs = "0" maxOccurs = "1" type = "StringType" />
    <xs:element name = "City" minOccurs = "0" maxOccurs = "1" type = "StringType" />
    <xs:element name = "StateProv" minOccurs = "0" maxOccurs = "1" type = "StringType" />
    <xs:element name = "PostalCode" minOccurs = "0" maxOccurs = "1" type = "String10Type" />
    <xs:element name = "Country" minOccurs = "0" maxOccurs = "1" type = "StringType" />
    <xs:element name = "Phone" minOccurs = "0" maxOccurs = "1" type = "StringType" />
  </xs:all>
</xs:complexType>
<xs:complexType name = "ShippingInfoType">
  <xs:all>
    <xs:element name = "CustomerPO" minOccurs = "0" maxOccurs = "1" type = "StringType" />
    <xs:element name = "CustomerTaxID" minOccurs = "0" maxOccurs = "1" type = "StringType" />
    <xs:element name = "ShippingMethod" minOccurs = "0" maxOccurs = "1" type = "ShippingMethodType" />
    <xs:element name = "ShippingName" minOccurs = "0" maxOccurs = "1" type = "StringType" />
  </xs:all>

```

## JetPay Authorization XML Specification © 2012 JetPay LLC.

```

<xs:element name = "ShippingAddr" minOccurs = "0" maxOccurs = "1" type = "AddressType" />
<xs:element name = "FreightAmount" minOccurs = "0" maxOccurs = "1" type = "Amount12Type" />
<xs:element name = "DutyAmount" minOccurs = "0" maxOccurs = "1" type = "Amount12Type" />
<xs:element name = "DiscountAmount" minOccurs = "0" maxOccurs = "1" type = "Amount12Type" />
<xs:element name = "AlternateTaxAmount" minOccurs = "0" maxOccurs = "1" type = "Amount12Type" />
<xs:element name = "AlternateTaxId" minOccurs = "0" maxOccurs = "1" type = "StringType" />
</xs:all>
</xs:complexType>
<xs:complexType name = "SummaryDetailType">
  <xs:all>
    <xs:element name = "TransactionSubtotalAmount" minOccurs = "1" maxOccurs = "1" type = "Amount12Type" />
    <xs:element name = "TransactionCount" minOccurs = "1" maxOccurs = "1" type = "Int5Type" />
  </xs:all>
</xs:complexType>
<xs:complexType name = "SummaryType">
  <xs:sequence>
    <xs:element name = "SummaryDetail" minOccurs = "1" maxOccurs = "10" type = "SummaryDetailType" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name = "IndustryInfoType">
  <xs:choice minOccurs = "0" maxOccurs = "unbounded" >
    <xs:element name = "UserAgent" minOccurs = "0" maxOccurs = "1" type = "StringType" /> <!--ECOMMER
    <xs:element name = "PhoneANI" minOccurs = "0" maxOccurs = "1" type = "StringType" /> <!--MOTO, Te
    <xs:element name = "PhoneII" minOccurs = "0" maxOccurs = "1" type = "StringType" /> <!--MOTO, Tel
    <xs:element name = "TicketNumber" minOccurs = "0" maxOccurs = "1" type = "String10Type" /> <!--RE
    <xs:element name = "BaseAmount" minOccurs = "0" maxOccurs = "1" type = "Amount12Type" /> <!--REST
    <xs:element name = "TipAmount" minOccurs = "0" maxOccurs = "1" type = "Amount5Type" /> <!--RESTAU
    <xs:element name = "ServerID" minOccurs = "0" maxOccurs = "1" type = "String10Type" /> <!--RESTAU
    <xs:element name = "TableNumber" minOccurs = "0" maxOccurs = "1" type = "Number4Type" /> <!--REST
    <xs:element name = "FolioNumber" minOccurs = "0" maxOccurs = "1" type = "String10Type" /> <!--HOT
    <xs:element name = "CheckInDate" minOccurs = "0" maxOccurs = "1" type = "DateType" /> <!--HOTEL-
    <xs:element name = "CheckOutDate" minOccurs = "0" maxOccurs = "1" type = "DateType" /> <!--HOTEL-
    <xs:element name = "RoomRate" minOccurs = "0" maxOccurs = "1" type = "Amount12Type" /> <!--HOTEL-
    <xs:element name = "RoomNumber" minOccurs = "0" maxOccurs = "1" type = "String8Type" /> <!--HOTEL
    <xs:element name = "ExtraCharges" minOccurs = "0" maxOccurs = "6" type = "StringType" /> <!--HOTE
    <xs:element name = "CpsTranId" minOccurs = "0" maxOccurs = "1" type = "String20Type" /> <!--HOTEL
    <xs:element name = "AgreementNumber" minOccurs = "0" maxOccurs = "1" type = "String10Type" /> <!--
    <xs:element name = "PickupDate" minOccurs = "0" maxOccurs = "1" type = "DateType" /> <!--AUTORENT
    <xs:element name = "ReturnDate" minOccurs = "0" maxOccurs = "1" type = "DateType" /> <!--AUTORENT
    <xs:element name = "RenterName" minOccurs = "0" maxOccurs = "1" type = "StringType" /> <!--AUTORE
    <xs:element name = "AdjustAmount" minOccurs = "0" maxOccurs = "1" type = "Amount12Type" /> <!--AU
    <xs:element name = "AdjustIndicator" minOccurs = "0" maxOccurs = "1" type = "StringType" /> <!--A
    <xs:element name = "PickupCity" minOccurs = "0" maxOccurs = "1" type = "StringType" /> <!--AUTORE
    <xs:element name = "PickupState" minOccurs = "0" maxOccurs = "1" type = "StringType" /> <!--AUTOR
    <xs:element name = "ReturnCity" minOccurs = "0" maxOccurs = "1" type = "StringType" /> <!--AUTORE
    <xs:element name = "ReturnState" minOccurs = "0" maxOccurs = "1" type = "StringType" /> <!--AUTOR
    <xs:element name = "NoShow" minOccurs = "0" maxOccurs = "1" type = "BooleanType" /> <!--AUTORENTA
    <xs:element name = "ExtraAmount" minOccurs = "0" maxOccurs = "1" type = "Amount6Type" /> <!--AUTO
    <xs:element name = "PlazaID" minOccurs = "0" maxOccurs = "1" type = "String3Type" /> <!--PARKING-
    <xs:element name = "BoothID" minOccurs = "0" maxOccurs = "1" type = "String3Type" /> <!--PARKING-
    <xs:element name = "ShiftID" minOccurs = "0" maxOccurs = "1" type = "String2Type" /> <!--PARKING-
    <xs:element name = "ClerkID" minOccurs = "0" maxOccurs = "1" type = "String6Type" /> <!--PARKING-
    <xs:element name = "DocumentNumber" minOccurs = "0" maxOccurs = "1" type = "String10Type" /> <!--
    <xs:element name = "LanID" minOccurs = "0" maxOccurs = "1" type = "StringType" /> <!--QUASICASH--
  </xs:choice>
  <xs:attribute name="Type" type = "IndustryType" use = "required" />
</xs:complexType>
<xs:complexType name = "MemoType">
  <xs:all>
    <xs:element name = "AuthAuthority" minOccurs = "1" maxOccurs = "1" type = "StringType" />
    <xs:element name = "Description" minOccurs = "0" maxOccurs = "1" type = "StringType" />
  </xs:all>

```

## JetPay Authorization XML Specification © 2012 JetPay LLC.

```

    <xs:attribute name="Type" type = "MemoTranType" use = "required" />
</xs:complexType>
<xs:complexType name = "AchType">
    <xs:all>
        <xs:element name = "AccountNumber" minOccurs = "0" maxOccurs = "1" type = "AcctNumType" />
        <xs:element name = "ABA" minOccurs = "0" maxOccurs = "1" type = "AbaType" />
        <xs:element name = "CheckNumber" minOccurs = "1" maxOccurs = "1" type = "CheckNumType" />
        <xs:element name = "Scrutiny" minOccurs = "0" maxOccurs = "1" type = "HighLowType" default = "HIGH" />
    </xs:all>
    <xs:attribute name="Type" type = "AchAccountType" default = "CHECKING" />
    <xs:attribute name="SEC" type = "SecType" default = "PPD" />
</xs:complexType>
<xs:complexType name = "VerifyType">
    <xs:all>
        <xs:element name = "Cavv" minOccurs = "0" maxOccurs = "1" type = "CavvType" />
        <xs:element name = "Xid" minOccurs = "0" maxOccurs = "1" type = "EncodingType" />
        <xs:element name = "Eci" minOccurs = "0" maxOccurs = "1" type = "Digit2Type" />
    </xs:all>
    <xs:attribute name="Type" type = "VerifyEnum" />
</xs:complexType>
<xs:complexType name = "VersionType">
    <xs:all>
        <xs:element name = "Subscriber" minOccurs = "0" maxOccurs = "1" type = "StringType" />
        <xs:element name = "DLL" minOccurs = "0" maxOccurs = "1" type = "StringType" />
        <xs:element name = "JetPay_fe" minOccurs = "0" maxOccurs = "1" type = "StringType" />
        <xs:element name = "Host" minOccurs = "0" maxOccurs = "1" type = "StringType" />
        <xs:element name = "Auth_be" minOccurs = "0" maxOccurs = "1" type = "StringType" />
    </xs:all>
</xs:complexType>
<xs:complexType name = "SourcesType">
    <xs:all>
        <xs:element name = "Active" minOccurs = "0" maxOccurs = "1" type="StringType" />
        <xs:element name = "Settled" minOccurs = "0" maxOccurs = "1" type="BooleanType" />
        <xs:element name = "Historical" minOccurs = "0" maxOccurs = "1" type="BooleanType" />
    </xs:all>
</xs:complexType>
<xs:complexType name = "ReturnType">
    <xs:all>
        <xs:element name = "ActionCode" minOccurs = "0" maxOccurs = "1" type="BooleanType" />
        <xs:element name = "TotalAmount" minOccurs = "0" maxOccurs = "1" type="BooleanType" />
        <xs:element name = "Amount" minOccurs = "0" maxOccurs = "1" type="BooleanType" />
        <xs:element name = "Approval" minOccurs = "0" maxOccurs = "1" type="BooleanType" />
        <xs:element name = "AuthDateTime" minOccurs = "0" maxOccurs = "1" type="BooleanType" />
        <xs:element name = "OrderNumber" minOccurs = "0" maxOccurs = "1" type="BooleanType" />
        <xs:element name = "ShipDateTime" minOccurs = "0" maxOccurs = "1" type="BooleanType" />
        <xs:element name = "Source" minOccurs = "0" maxOccurs = "1" type="BooleanType" />
        <xs:element name = "TransactionID" minOccurs = "0" maxOccurs = "1" type="BooleanType" />
        <xs:element name = "TransactionType" minOccurs = "0" maxOccurs = "1" type="BooleanType" />
        <xs:element name = "UniqueID" minOccurs = "0" maxOccurs = "1" type="BooleanType" />
        <xs:element name = "UserData1" minOccurs = "0" maxOccurs = "1" type="BooleanType" />
        <xs:element name = "UserData2" minOccurs = "0" maxOccurs = "1" type="BooleanType" />
        <xs:element name = "UserData3" minOccurs = "0" maxOccurs = "1" type="BooleanType" />
    </xs:all>
</xs:complexType>
<!-- ***** ENUMERATED TYPES ***** -->
<xs:simpleType name = "IndustryType">
    <xs:restriction base = "StringType">
        <xs:enumeration value = "ECOMMERCE" />
        <xs:enumeration value = "RETAIL" />
        <xs:enumeration value = "MOTO" />
        <xs:enumeration value = "HOTEL" />
        <xs:enumeration value = "RESTAURANT" />
    </xs:restriction>
</xs:simpleType>

```

## JetPay Authorization XML Specification © 2012 JetPay LLC.

```
<xs:enumeration value = "AUTORENTAL" />
<xs:enumeration value = "AIRLINE" />
<xs:enumeration value = "PARKING" />
<xs:enumeration value = "QUASICASH" />
</xs:restriction>
</xs:simpleType>
<xs:simpleType name = "MemoTranType">
  <xs:restriction base = "StringType">
    <xs:enumeration value = "SALE" />
    <xs:enumeration value = "DEBIT SALE" />
    <xs:enumeration value = "REFUND" />
    <xs:enumeration value = "DEBIT REFUND" />
    <xs:enumeration value = "SYNTAX ERROR" />
    <xs:enumeration value = "FEE" />
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="TransQualifEnumType">
  <xs:restriction base = "StringType">
    <xs:enumeration value = "NONE" />
    <xs:enumeration value = "STIP" />
    <xs:enumeration value = "TRANSACTION VOIDED" />
    <xs:enumeration value = "SCRIP DISTRIBUTED" />
    <xs:enumeration value = "STIP TIMEOUT" />
    <xs:enumeration value = "RESUBMISSION TIMEOUT" />
    <xs:enumeration value = "ONLINE RESUBMISSION" />
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="OriginType">
  <xs:restriction base = "StringType">
    <xs:enumeration value = "INTERNET" /> <!-- default, if absent -->
    <xs:enumeration value = "POS" /> <!-- default when track data is present -->
    <xs:enumeration value = "POS STIP" />
    <xs:enumeration value = "RECURRING" />
    <xs:enumeration value = "RESUBMISSION" />
    <xs:enumeration value = "MAIL ORDER" />
    <xs:enumeration value = "PHONE ORDER" />
    <xs:enumeration value = "BILL PAYMENT" />
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="ShippingMethodType">
  <xs:restriction base = "StringType">
    <xs:enumeration value = "SAME DAY" /> <!-- no default value -->
    <xs:enumeration value = "OVERNIGHT" />
    <xs:enumeration value = "PRIORITY" />
    <xs:enumeration value = "GROUND" />
    <xs:enumeration value = "ELECTRONIC" />
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="ReaderType">
  <xs:restriction base = "StringType">
    <xs:enumeration value = "KEYPAD" /> <!-- no default value -->
    <xs:enumeration value = "MAGNETIC STRIPE" />
    <xs:enumeration value = "CHIP" />
    <xs:enumeration value = "CONTACTLESS MS" />
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="VerifyEnum">
  <xs:restriction base = "StringType">
    <xs:enumeration value = "VbV" />
    <xs:enumeration value = "SC" />
    <xs:enumeration value = "MCSC" />
  </xs:restriction>
</xs:simpleType>
```

```

</xs:simpleType>
<xs:simpleType name="DebitCreditType">
  <xs:restriction base = "StringType">
    <xs:enumeration value = "DEBIT" />
    <xs:enumeration value = "CREDIT" />
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="DeliveryType">
  <xs:restriction base = "StringType">
    <xs:enumeration value = "SHIPPED"/>
    <xs:enumeration value = "PICKUP"/>
    <xs:enumeration value = "DOWNLOAD"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="TranStateType">
  <xs:restriction base = "StringType">
    <xs:enumeration value = "NOT PAID"/>
    <xs:enumeration value = "PAID"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="TranStateCodeType">
  <xs:restriction base = "StringType">
    <xs:enumeration value = "TRANSACTION ESTABLISHED"/>
    <xs:enumeration value = "FINANCIAL INSTITUTION REQUESTED VALIDATION OF SESSION TOKEN"/>
    <xs:enumeration value = "FINANCIAL INSTITUTION INQUIRE"/>
    <xs:enumeration value = "FINANCIAL INSTITUTION CANCELLED TRANSACTION"/>
    <xs:enumeration value = "AUTHORIZATION RECEIVED AND UPDATED WITHIN THE EWISE NETWORK"/>
    <xs:enumeration value = "AUTHORIZATION DENIED AND RECEIVED AND UPDATED WITHIN THE EWISE NETWORK"/>
    <xs:enumeration value = "AUTHORIZED WITH UNKNOWN DETAILS"/>
    <xs:enumeration value = "AUTHORIZATION TOKEN DOES NOT MATCH"/>
    <xs:enumeration value = "TRANSACTION QUERIED BY THE MERCHANT"/>
    <xs:enumeration value = "TRANSACTION QUERIED BY THE MERCHANT USING AUTH TOKEN"/>
    <xs:enumeration value = "TRANSACTION TIMED OUT"/>
    <xs:enumeration value = "MERCHANT ERROR"/>
    <xs:enumeration value = "FINANCIAL INSTITUTION ERROR"/>
    <xs:enumeration value = "INTERNAL EWISE NETWORK ERROR"/>
    <xs:enumeration value = "UNKNOWN STATUS"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="HighLowType">
  <xs:restriction base = "StringType">
    <xs:enumeration value = "HIGH" />
    <xs:enumeration value = "LOW" />
    <xs:enumeration value = "NONE" />
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="TransEnumType">
  <xs:restriction base = "StringType">
    <xs:enumeration value = "authonly" />
    <xs:enumeration value = "capt" />
    <xs:enumeration value = "credit" />
    <xs:enumeration value = "enq" />
    <xs:enumeration value = "force" />
    <xs:enumeration value = "ping" />
    <xs:enumeration value = "sale" />
    <xs:enumeration value = "void" />
    <xs:enumeration value = "AUTHONLY" />
    <xs:enumeration value = "BALANCE" />
    <xs:enumeration value = "CAPT" />
    <xs:enumeration value = "CHECK" />
    <xs:enumeration value = "CREDIT" />
    <xs:enumeration value = "ENQ" />
  </xs:restriction>

```

## JetPay Authorization XML Specification © 2012 JetPay LLC.

```

<xs:enumeration value = "FORCE" />
<xs:enumeration value = "INCREMENTAL" />
<xs:enumeration value = "IOU" />
<xs:enumeration value = "PARTIALREVERSAL" />
<xs:enumeration value = "PING" />
<xs:enumeration value = "REVERSAL" />
<xs:enumeration value = "REVERSEAUTH" />
<xs:enumeration value = "SALE" />
<xs:enumeration value = "SVPAUTH" />
<xs:enumeration value = "SVP_CAPTURE" />
<xs:enumeration value = "SVP_PREFUND" />
<xs:enumeration value = "VOID" />
<xs:enumeration value = "VOIDACH" />
<xs:enumeration value = "CLOSE SALE" />
<xs:enumeration value = "STATUS SALE" />
<xs:enumeration value = "BATCH RELEASE" />
<xs:enumeration value = "BATCH STATUS" />
<xs:enumeration value = "MEMO" />
<xs:enumeration value = "TOKENIZE" />
</xs:restriction>
</xs:simpleType>
<xs:simpleType name = "AchAccountType">
  <xs:restriction base = "StringType">
    <xs:enumeration value = "CHECKING" />
    <xs:enumeration value = "SAVINGS" />
    <xs:enumeration value = "BUSINESSCK" />
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "SecType">
  <xs:restriction base = "StringType">
    <xs:enumeration value = "PPD" /> <!-- "Prearranged Payment and Deposit Entry", default, if absent -->
    <xs:enumeration value = "ARC" /> <!-- "Accounts Receivable Entry" -->
    <xs:enumeration value = "BOC" /> <!-- "Back Office Conversion" -->
    <xs:enumeration value = "CCD" /> <!-- "Cash Concentration or Disbursement" -->
    <xs:enumeration value = "POP" /> <!-- "Point-of-Purchase Entry" -->
    <!-- <xs:enumeration value = "CIE" /> "Customer Initiated Entry" -->
    <!-- <xs:enumeration value = "MTE" /> "Machine Transfer Entry" -->
    <!-- <xs:enumeration value = "PBR" /> "Consumer Cross-Border Payment" -->
    <!-- <xs:enumeration value = "POS" /> "Point-of-Sale Entry" -->
    <!-- <xs:enumeration value = "SHR" /> "Shared Network Transaction" -->
    <!-- <xs:enumeration value = "RCK" /> "Re-presented Check Entry" -->
    <xs:enumeration value = "TEL" /> <!-- "Telephone-Initiated Entry" -->
    <xs:enumeration value = "WEB" /> <!-- "Internet-Initiated Entry" -->
    <!-- <xs:enumeration value = "CBR" /> "Corporate Cross-Border Payment" -->
    <!-- <xs:enumeration value = "CTX" /> "Corporate Trade Exchange" -->
    <!-- <xs:enumeration value = "ACK" /> "Acknowledgment Entry" -->
    <!-- <xs:enumeration value = "ATX" /> "Acknowledgment Entry" -->
    <!-- <xs:enumeration value = "COR" /> "Automated Notification of Change or Refused Notifacation o
    <!-- <xs:enumeration value = "DNE" /> "Death Notification Entry" -->
    <!-- <xs:enumeration value = "ENR" /> "Automated Enrollment Entry" -->
    <!-- <xs:enumeration value = "RET" /> "Return Entry" -->
    <!-- <xs:enumeration value = "TRC" /> "Truncated Entry" -->
    <!-- <xs:enumeration value = "TRX" /> "Truncated Entry" -->
    <!-- <xs:enumeration value = "XCK" /> "Destroyed Check Entry" -->
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "EncodeType">
  <xs:restriction base = "StringType">
    <xs:enumeration value = "BINARY" />
    <xs:enumeration value = "HEX" />
    <xs:enumeration value = "BASE64" />
  </xs:restriction>

```

## JetPay Authorization XML Specification © 2012 JetPay LLC.

```
</xs:simpleType>
<xs:simpleType name = "PinBlockFormatType">
  <xs:restriction base = "StringType">
    <xs:enumeration value = "ANSI" />
    <xs:enumeration value = "IBM3624" />
    <xs:enumeration value = "PINPAD" />
    <xs:enumeration value = "IBMPINPAD" />
    <xs:enumeration value = "BURROUGHS" />
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "ManagementType">
  <xs:restriction base = "StringType">
    <xs:enumeration value = "DUKPT" /> <!-- Derived Unique Key Per Transaction -->
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="SourceEnum">
  <xs:restriction base = "StringType">
    <xs:enumeration value = "Active" />
    <xs:enumeration value = "Settled" />
    <xs:enumeration value = "Historical" />
  </xs:restriction>
</xs:simpleType>
<!-- ***** ABBREVIATIONS, DEPRECATED ALIASES ***** -->
<xs:element name = "MerchantID" substitutionGroup = "TerminalID" type = "String12Type" />
</xs:schema>
```