Secure Java API Integration Guide

Table of Contents

Document Control

This is a control document

DESCRIPTION	Secure Java API Integration Guide		
CREATION DATE	02/05/2007	CREATED BY	SecurePay
VERSION	1.4	DATE UPDATED	22/09/2010

1   Introduction

1.1   About this Guide

This guide provides technical information about installing and configuring Secure Java within your environment.

Secure Java is written in the Sun Java programming language, and can run on any computer with the Java Runtime Environment installed. Messages are transported via a HTTP protocol using SSL.

Secure Java integrates into a web site via the Java programming language. The merchant's web site captures the credit card information and then posts the details in a Java message format over a secure connection to the SecurePay Payment Gateway for authorisation. The authorisation response is then returned as a Java message over the same secure connection.

1.2   Intended Audience

This document is intended for developers, integrating SecurePay's Secure Java interface into their own Java applications, Java-based websites (such as servlets or JSP), or applications that are able to instantiate a Java object and invoke its methods.

Knowledge of the Java programming language and the Java Virtual Machine is required for some sections of this document.

2   System Overview

SecurePay's Payment Gateway provides merchants with the ability to process credit card and direct entry payments in a secure environment.

SecurePay partners with the following major banks and financial institutions in the provision of the SecurePay Payment Gateway:

- ANZ
- American Express
- BankWest
- Commonwealth Bank
- Diners Club
- National Australia Bank
- St George (including Bank of SA)
- Westpac (including Challenge Bank and Bank of Melbourne)

Direct entry payments are not processed in real time; they are stored in SecurePay's database and processed daily at 4.30pm EST.

Secure Java supports four payment transaction types:

- Payments
- Refunds
- Preauthorise
- Preauthorise Complete

Secure Java is written in the Sun Java programming language, and can run on any computer with the Java Runtime Environment installed. Messages are transported via a HTTP protocol using SSL.
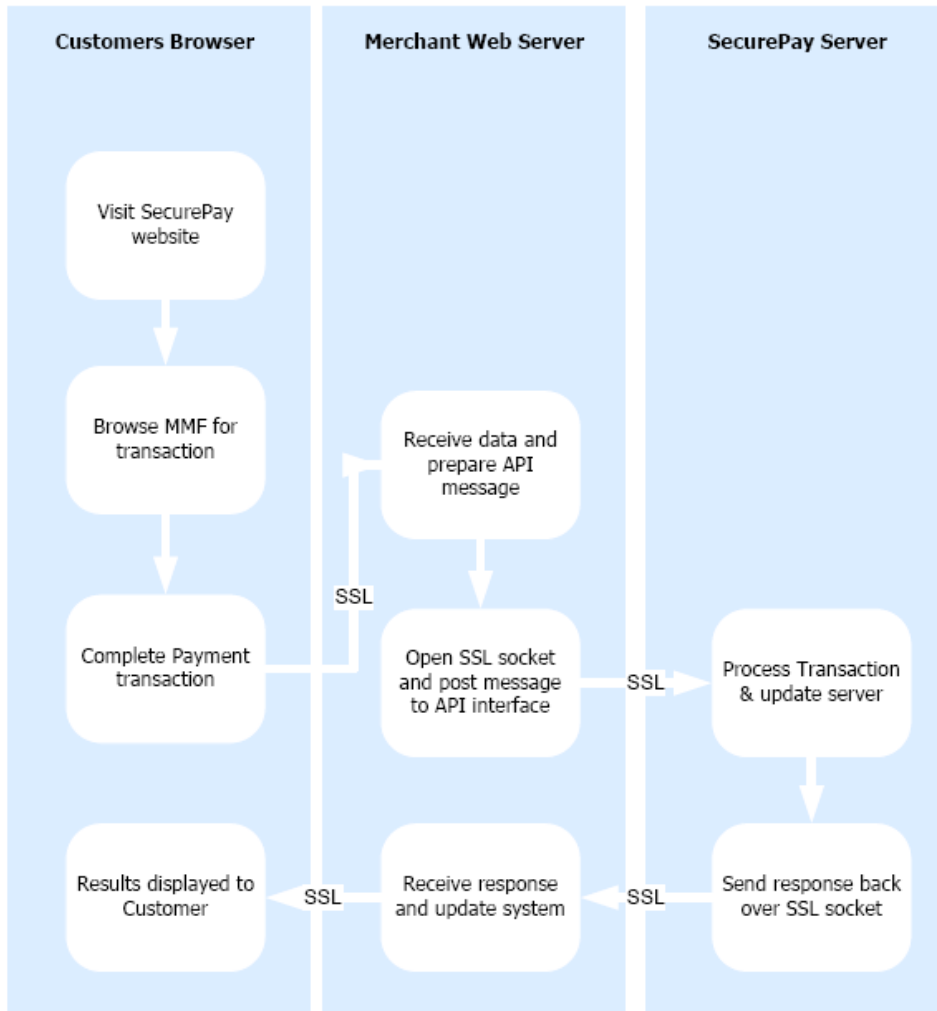
3 Secure Java Payments

3.1 Payment

The Payment command is used to pass financial credit card transaction messages to SecurePay's payment server, which will authorise the transaction with the merchant's bank and customer's card issuer, and produce a response based on the banks' authorisation of the transaction.

The Payment command can be used to send following credit card transaction types:

- Credit Card Payment
- Credit Card Refund
- Credit Card Reversal (Void)
- Credit Card Preauthorise
- Credit Card Preauthorise Complete (Advice)

SecurePay's payment server is also capable of processing direct entry transactions. Direct entry transactions are not processed real time. The SecurePay payment server stores direct entry transaction in a database and processes them daily at 4.30pm Melbourne time.

The Payment command can be used to send following direct entry transaction types:

- Direct Debit
- Direct Credit

3.2 Authentication, Communication & Encryption

To ensure security, each merchant is issued with transaction password. This password is authenticated with each request before it is processed. This makes sure that unauthorised users will be unable to use the interface.

> *The password can be changed by the merchant via SecurePay's Merchant Management facility.*

The Secure Java interface uses HTTP protocol and SSL for communication with SecurePay's servers.

Merchants using Secure Java will automatically use SecurePay's security certificate to encrypt requests and decrypt responses from SecurePay.

4   Java Installation

Secure Java has been built using Sun's Java programming language. Java boasts full software portability, allowing the same compiled software package to be run independently of the platform on which it is installed. The Secure Java software package provided can therefore be run on Microsoft Windows, Unix, or Macintosh platforms without recompiling.

4.1   Prerequisites

1. The installation machine must have the Java Runtime Environment (JRE) installed. Secure Java currently runs on JRE versions 1.4 and later. The JRE can be downloaded free of charge from the Sun website: http://java.sun.com

2. The installation machine, and any firewall, etc, in front of it, must be able to create an outbound socket connection on the communications port, to the SecurePay Payment Server host. Standard web communication ports will be used initially, however are subject to change in future releases. Port 80 will be used on SecurePay's Test Server, and port 443 on SecurePay's Live Server.

4.2   Java Inventory

The Secure Java Software package contains the following files:

- securepayxmlapi_obf.jar
- xmlParserAPIs-2.4.0.jar
- xercesImpl-2.4.0.jar
- Base64.jar

4.3   Installation

Secure Java

1. Create a directory on the installation machine called "SecurePayAPI" in a location of your choice.
2. Copy all files listed in the Software Inventory (see 4.2) into this new directory.

Secure Java Software - Periodic and Triggered add in

1. Create a directory on the installation machine called "SecurePayPeriodicAPI" in a location of your choice.
2. Copy all files listed in the Software Inventory (see 3.2) into this new directory

SecurePay Java API integration Guide Version 1.3

5 Java Integration

5.1 Classpath Setup

Before running Secure Java, your Java classpath must be set to each of the JAR archives included in this package. This can be done when executing the Java from the command line, e.g.:

> ➤ `%JAVA_HOME%\bin\java  -cp  abc.jar;def.jar  YourMainClass  arg1 arg2`

or, by setting the system environment variable "CLASSPATH" before running, e.g.:

Windows:
```
> set CLASSPATH=abc.jar;def.jar
```

Unix:
```
> setenv CLASSPATH abc.jar:def.jar
```

These commands may also be written in a batch file or shell script for your application.

5.2 Interface

5.2.1 Payments

5.2.1.1 securepay.jxa.api.Payment

The Payment object should be created first, and a Txn object added to it, containing details of the financial transaction to be processed. The Payment object is then "submitted" to SecurePay to process the transactions it contains.

5.2.1.1.1 Constructor

public Payment()

Create an empty `Payment` object to which `Txns` may be added.

Parameters:

No parameters required by this constructor.

5.2.1.1.2 Public Methods

5.2.1.1.3 getMessageId

```
public String getMessageId()
```

Returns the unique message identifier created by the API.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	Unique identifier of this Payment object.

5.2.1.1.3.1   getMessageTimestamp

```
public Date getMessageTimestamp()
```

Returns the timestamp created by the API for the `Payment` in a `Date` object.

Input Parameters:

> No input parameters are required by this method.

Return Parameter:

Type	Description
Date	Timestamp of this Payment object.

5.2.1.1.3.2   getMessageTimestampAsString

```
public String getMessageTimestampAsString()
```

Returns the timestamp created by the API for the `Payment` as a string. Refer to Appendix E for the format returned.

Input Parameters:

> No input parameters are required by this method.

Return Parameter:

Type	Description
String	Timestamp of this Payment object.

5.2.1.1.3.3   getApiVersion

```
public String getApiVersion()
```

Returns the version of the API being used to create this object.

Input Parameters:

> No input parameters are required by this method.

Return Parameter:

Type	Description
String	API version used to create this Payment object.

5.2.1.1.3.4   setMerchantId

public void setMerchantId(String id)

Sets the merchant id to be used when processing the payments. The merchant id must be set in order for SecurePay to determine the bank account to which the funds should be settled.

Input Parameters:

Name	Type	Description	Allowed Values
id	String	Merchant ID allocated to the merchant by SecurePay.	7-character string in format XXXDDDD, where X is a letter (A-Z) and D is a digit (0-9). SecurePay will supply this value to you upon application. **For Direct Entry:** 5-7 character string in format XXXDDDD, where X is a letter (A-Z) and D is a digit (0-9). Last two digits can be ignored.

Return Parameter:

No object returned by this method.

5.2.1.1.3.5 setServerURL

```
public void setServerURL(String url)
```

Sets the URL of the SecurePay Payment Server to which this object will be sent when `process()` is called.

Input Parameters:

Name	Type	Description	Allowed Values
url	String	URL of SecurePay's Payment Server or Direct Entry Server in case of Direct Entry payments.	**Test:** https://test.securepay.com.au/xmlapi/payment http://test.securepay/com.au/xmlapi/payment **Live:** https://api.securepay.com.au/xmlapi/payment **For Direct Entry:** **Test:** https://test.securepay.com.au/xmlapi/directentry http://test.securepay.com.au/xmlapi/directentry **Live:** https://api.securepay.com.au/xmlapi/directentry **For Antifraud:** **Test:** https://test.securepay.com.au/antifraud/payment http://test.securepay.com.au/antifraud/payment **Live:** https://www.securepay.com.au/antifraud/payment (Values are subject to change in future releases.)

Return Parameter:

No object returned by this method.

5.2.1.1.3.6 setProcessTimeout

```
public void setProcessTimeout(int timeout)
```

Sets the timeout in seconds to wait for a response message from SecurePay's server. If no response is received in this time, a timeout response is returned, and transaction results may be queried at a later time.

If value is not set, or is set to an integer < 0, default timeout of 80 seconds is used.

Input Parameters:

Name	Type	Description	Allowed Values
timeout	int	Seconds to wait for SecurePay response.	Int < 0: default value is used Int >= 0: supplied value is used.

Return Parameter:

No object returned by this method.

5.2.1.1.3.7   addTxn

```
public Txn addTxn(int txnType, String ponum)
```

Adds a Txn object to the Payment and returns the object for configuration. If a Txn already exists with the same txnType and ponum combination, NULL is returned.

The current version of SecurePay's Payment Server allows only Payments containing a single Txn object. Payments submitted with more than one Txn will be rejected with Status Code "577". Multiple transaction batching will be included in a future release, and will be advertised by SecurePay when available. It is expected that no change will be required to the API to support multi-Txn batches.

As of Secure Java V4.0.3a, it is possible to create a `Payment` object, add a `Txn` to it, call `process()`, then add another `Txn` and `process()` again, as many times as required on a single `Payment` object.

Since Direct Entry payments are sent to a different URL than credit card payments a separate instance of Payment Class should be used for processing those payments.

For refunds and reversals, the `ponum` field must be the same as the `ponum` used for the original payment.

For completes the `ponum` field must be the same as the `ponum` used for the original preauthorisation.

Input Parameters:

Name	Type	Description	Allowed Values
TxnType	int	Integer representing the transaction type to be processed.	Refer to Appendix A
Ponum	String	Unique merchant transaction identifier, typically an invoice number.	Any alphanumeric string, up to 60 characters in length. **For Direct Entry:** Any string using characters from, Appendix I up to 18 characters in length.

Return Parameter:

Type	Description
Txn	Instance of the Txn object added to the Payment. If a Txn with the same identification already exists, then this object is returned, otherwise a new instance is returned.

5.2.1.1.3.8   getTxn

```
public Txn getTxn(int txnType, String ponum)
```

Returns the Txn object from the Payment's request queue (before calling process()) with a matching txnType and ponum value. If no such Txn exists, NULL is returned. This method will always return NULL after a call to process(), as the request queue will be empty.

Input Parameters:

Name	Type	Description	Allowed Values
txnType	int	Integer representing the transaction type to be processed.	
ponum	String	Unique merchant transaction identifier, typically an invoice number.	Any alphanumeric string, up to 60 characters in length.

Return Parameter:

SecurePay Java API integration Guide Version 1.3

Type	Description
Txn	Txn request object matching the txnType and ponum supplied, or NULL if no such Txn exists.

5.2.1.1.3.9   Process

```
public boolean process(String password)
```

Sends this Payment object and all contained Txns to SecurePay's Payment Server for processing by the bank. Method blocks until all transactions contained are processed and a response message is received, or the process timeout expires (refer to 5.2.1.1.3.6).

As of Secure Java V4.0.3a, the `password` parameter <u>must</u> be passed to this method. Previous releases of Secure Java V4.0.x which did not pass a password will no longer be compatible with the SecurePay Payment Server, as it is now required that a password is used for all financial transactions and echoes. The password can be changed via SecurePay's Merchant Login website by logging in as the "admin" user.

If the merchant Id and password sent do not match those stored in SecurePay's database, the response message will have a status code of "550".

Input Parameters:

Name	Type	Description	Allowed Values
password	String	The password is configured in SecurePay's database per Merchant ID, and must match the value passed by this method for the message to be accepted and processed.	The password is allocated to you by SecurePay when you receive this software, and can be changed via the Merchant Login website.

Return Parameter:

Type	Description
boolean	**true** if message was sent to server correctly and processed (regardless of the transactions' outcome), and **false** if client-side errors occurred preventing the object from being sent.

5.2.1.1.3.10 getStatusCode

```
public long getStatusCode()
```

Returns the status code of the Payment. Refer Appendix F for more information.

Input Parameters:

> No input parameters are required by this method.

Return Parameter:

Type	Description
long	Code representing the status of the Payment object after processing.

5.2.1.1.3.11 getStatusDesc

```
public String getStatusDesc()
```

Returns the status description of the Payment. Refer to <u>Appendix F</u> for more information.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	Textual description of the status of the Payment object after processing.

5.2.1.1.3.12 toString

public String toString()

Returns the object with its parameters formatted in a readable string.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	Textual representation of the Payment object.

5.2.1.1.3.13 getCount

```
public int getCount()
```

Returns the number of `Txn` objects contained in this `Payment` object. Before `process()` is called, the number of `Txn` requests is returned. The number of `Txn` responses is given after a call to `process()`.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
Int	Number of Txn objects

5.2.1.1.3.14 getTxn

```
public Txn getTxn(int index)
```

Returns the `Txn` object from the `Payment` at the index[th] position. This method returns a `Txn` request <u>before</u> calling `process()` or a `Txn` response <u>after</u> calling `process()`.

Input Parameters:

Name	Type	Description	Allowed Values
index	int	Index to the Txn object required.	int between 0 and getCount() – 1

Return Parameter:

Type	Description
Txn	Txn object from requested position.

5.2.1.1.3.15 getAdditionalInfo (Available in software v4.1.2, but not currently in use.)

```
public String getAdditionalInfo(int index, String fieldName)
```

This field is not currently in use, and will return a `null` value if called. SecurePay will release a list of available fields when they become available.

Returns the value of the specified field from the `Txn` object at the `index`[th] position. If `index` is –`1`, the field returned is from the `Payment` object itself.

Input Parameters:

Name	Type	Description	Allowed Values
index	int	Index to the Txn object required, or –1 to reference the Payment object itself.	int between –1 and getCount() – 1
fieldName	String	Name of field to return from the object specified by index.	Valid field name.

Return Parameter:

Type	Description
String	Value of field requested, or `null` if the field does not exist in the message.

5.2.1.2   securepay.jxa.api.Txn

The `Txn` object is returned when `addTxn(…)` is called from a `Payment` object. The required fields of this object must be set using "set" functions provided, as described below. When the `Payment` is submitted and processed, the result parameters of the `Txn` can be retrieved using the "get" methods described.

5.2.1.2.1 Transaction Type Required Field Map

The following table specifies which transaction parameters must be set for each available Txn type. Parameters are mandatory, optional, or not required.

METHOD \ TXN TYPE	Standard Payment	Refund	Reversal	Preauthorise	Complete (Advice)	Recurring Payment	Card-Present Payment	Direct Entry Debit Payment	Direct Entry Credit Payment	Antifraud Payment	Antifraud Only Request
	0	4	6	10	11	14	19	15	17	21	22
setTxnSource	O	O	O	O	O	O	O	O	O	O	O
setAmount	M	M	M	M	M	M	M	M	M	M	M
setCurrencyCode	O	X	X	O	X	O	O	X	X	O	O
setCardNumber	M	X	X	M	X	M	M	X	X	M	M
SetCVV	O	O	O	O	O	O	O	X	X	O	O
setExpiryDate	M	O	O	M	O	O	M	X	X	M	M
setTrack2Data	X	X	X	X	X	X	M	X	X	X	X
SetXID	O	O	O	O	O	O	O	X	X	X	X
SetCAVV	O	O	O	O	O	O	O	X	X	X	X
SetSLI	O	O	O	O	O	O	O	X	X	X	X
SetTxnId	X	M	M	X	X	X	X	X	X	X	X
setPreauthCode	X	X	X	X	M	X	X	X	X	X	X
setBsbNumber	X	X	X	X	X	X	X	M	M	X	X
setAccountNumber	X	X	X	X	X	X	X	M	M	X	X
setAccountName	X	X	X	X	X	X	X	M	M	X	X
setFirstName	X	X	X	X	X	X	X	X	X	O	O
setLastName	X	X	X	X	X	X	X	X	X	O	O
setZipCode	X	X	X	X	X	X	X	X	X	O	O
setTown	X	X	X	X	X	X	X	X	X	O	O
setBillingCountry	X	X	X	X	X	X	X	X	X	O	O
setDeliveryCountry	X	X	X	X	X	X	X	X	X	O	O
setEmailAddress	X	X	X	X	X	X	X	X	X	O	O
setIp	X	X	X	X	X	X	X	X	X	M	M

M = mandatory   O = optional   X = not required (ignored if set)

> For refunds and reversals the ponum field must be the same as the ponum used for the original payment.
>
> For completes the ponum field must be the same as the ponum used for the original preauthorisation.
>
> The ponum field is set using addTxn method on the Payment object.

5.2.1.2.2 Constructor

The `Txn` object cannot be constructed directly. It must be created by calling `addTxn(…)` in the `Payment` object.

5.2.1.2.3 Public Methods

5.2.1.2.3.1 setTxnSource

```
public void setTxnSource(int value)
```

Sets the transaction source for this transaction. Source is used for transaction origin tracking. This field is <u>optional</u>, and is set to 0, "Unknown", by default.

Input Parameters:

Name	Type	Description	Allowed Values
value	int	Transaction source	Refer to Appendix B

Return Parameter:

No object returned by this method.

5.2.1.2.3.2 setAmount

```
public void setAmount(String value)
```

Sets the amount associated with this financial transaction. Amount is supplied with no currency formatting, in the currency's smallest denomination. If using the default currency of Australian Dollars (AUD), the amount is set in Australian <u>cents</u>. E.g. an amount of AU$125.40 would be set calling:

```
txn.setAmount("12540");
```

Note for Multi-Currency Users: For examples of how other currency codes affect the value set in this field, and how many minor units to pass for each currency type, see Appendix HError! Reference source not found.. (You must fulfil certain requirements with your bank and SecurePay before using the multi-currency features. Contact SecurePay Support for further details.)

Input Parameters:

Name	Type	Description	Allowed Values
Value	String	Transaction amount in Australian cents.	String representing a number > 0, with no whitespace or currency formatting.

Return Parameter:

No object returned by this method.

5.2.1.2.3.3 setCurrencyCode (Approved Multi-Currency merchants only)

```
public void setCurrencyCode(String value)
```

Before setting this value, check with SecurePay to find out if we support multi-currency transactions through your bank, and what currency codes they allow. You may need to open a special multi-currency account with your bank to allow payments in other currencies. (Contact SecurePay Support for further details.)

> *When doing Refunds, Reversals, and Advice (Preauth Complete), the currency code is not required. The currency will be determined from the original currency used for the Payment or Preauthorisation transaction.*

Sets the 3-character currency code associated with this financial transaction. If this field is not set, the default currency will be "AUD" (Australian Dollars). To modify the transaction to use Euro, for example, use the following code:

```
txn.setCurrencyCode("EUR");
```

Currency codes are in all uppercase. The complete list of supported currencies, and the appropriate amount field formatting for each currency, is supplied in the Appendix H.

Input Parameters:

Name	Type	Description	Allowed Values
Value	String	Transaction currency code.	One of the valid 3-letter codes provided in the Appendix H to this document.

Return Parameter:

   No object returned by this method.

5.2.1.2.3.4 setCardNumber

```
public void setCardNumber(String value)
```

Only for Credit Card transactions.

Sets the credit card number for this financial transaction. The card number is not required for refunds, reversals and complete transactions. As these transaction types can only be performed on an existing transaction (payment or preauthorisation) SecurePay already has a record of the card number used.

Input Parameters:

Name	Type	Description	Allowed Values
value	String	Customer's credit card number.	String of digits, no whitespace. 13-16 characters. String must pass Luhn algorithm.

Return Parameter:

   No object returned by this method.

5.2.1.2.3.5 setCVV

```
public void setCVV(String value)
```

Only for Credit Card transactions.

Sets the Card Verification Value for this financial transaction. This field is optional. The CVV value assists the bank with detecting fraudulent transactions based on automatically generated card numbers, as the CVV number is printed on the physical card and cannot be generated in conjunction with a card number. If passed, the bank may check the supplied value against the value recorded against the card.

Input Parameters:

Name	Type	Description	Allowed Values
value	String	Customer's credit card CVV number.	3 or 4 digit string, no whitespace. NULL or empty string is also valid.

Return Parameter:

No object returned by this method.

5.2.1.2.3.6 setExpiryDate

```
public void setExpiryDate(String value)
```

Only for Credit Card transactions.

Sets the credit card expiry date for this financial transaction. The expiry date is optional for refunds, reversals, recurring payments and complete transactions. As refunds, reversals and completes are transaction types that can only be performed on an existing transaction (payment or preauthorisation) SecurePay already has a record of the expiry date used. Expiry date can be set if the credit card has been re-issued with a new expiry date.

Input Parameters:

Name	Type	Description	Allowed Values
value	String	Customer's credit card expiry date.	5-character string in format MM/YY, where MM is a 2 digit month value (January = 01), YY is a 2 digit year value (2003 = 03). NULL or empty string is also valid for Recurring transactions.

Return Parameter:

No object returned by this method.

5.2.1.2.3.7 setTrack2Data (Card-Present transactions only)

```
public void setTrack2Data(String value)
```

Only for Credit Card transactions.

Set the data read from Track 2 of the magnetic strip on the customer's physical credit card by a card-reading device. Track 2 data is required only for merchants using a card-reading device to perform card-present transactions. For card-present transactions, the Transaction Type must also be set to Card Present.

Input Parameters:

Name	Type	Description	Allowed Values
value	String	Data retrieved from "track 2" on the credit card's magnetic strip.	

Return Parameter:

No object returned by this method.

5.2.1.2.3.8 setXID (3D-Secure merchants only)

```
public void setXID(String value)
```

Only for Credit Card transactions.

Set the 3D-Secure XID (transacion ID) for this financial transaction. The XID is required only for merchants enrolled in the 3D-Secure program. The XID field must be a 20-byte String, matching the unique XID passed to the card issuer before sending this transaction, using any 3D-Secure-enabled software.

Input Parameters:

Name	Type	Description	Allowed Values
Value	String	Transaction's XID, as supplied	Any 20-byte String

		to card issuer prior to sending the financial transaction.	

Return Parameter:

No object returned by this method.

5.2.1.2.3.9 setCAVV (3D-Secure merchants only)

```
public void setCAVV(String value)
```

Only for Credit Card transactions.

Set the 3D-Secure Cardholder Authorisation Verification Value for this financial transaction. The CAVV is required only for merchants enrolled in the 3D-Secure program. The CAVV field must be a 28-character Base-64-encoded string, matching the CAVV generated by 3D-Secure-enabled software before sending this financial transaction.

Input Parameters:

Name	Type	Description	Allowed Values
Value	String	Transaction's CAVV, as supplied by card issuer prior to sending the financial transaction.	28-character Base-64-encoded string.

Return Parameter:

No object returned by this method.

5.2.1.2.3.10 setSLI (3D-Secure merchants only)

```
public void setSLI(String value)
```

Only for Credit Card transactions.

Set the 3D-Secure Service Level Indicator for this financial transaction. The SLI is required only for merchants enrolled in the 3D-Secure program. The SLI field must be a 2-digit string, matching the SLI (or ECI) returned by the 3D-Secure-enabled software, prior to sending this financial transaction.

Input Parameters:

Name	Type	Description	Allowed Values
Value	String	Transaction's SLI, as supplied by card issuer prior to sending the financial transaction.	2-digit String.

Return Parameter:

No object returned by this method.

5.2.1.2.3.11 setTxnId

```
public void setTxnId()
```

Only for Credit Card transactions.

For Refunds and Reversals only.

Set the original bank transaction ID of a payment in order to refund or reverse the payment. Field must be set for refunds or reversals to work, and must match the original payment transaction ID.

Input Parameters:

Name	Type	Description	Allowed Values
value	String	Bank transaction ID of the	6-30 character string

		original payment transaction.	

Return Parameter:

No object returned by this method.

5.2.1.2.3.12 setPreauthCode

```
public void setPreauthCode()
```

Only for Credit Card transactions.

For Preauth Completes (Advice) only.

Set the original authorisation code of a preauth in order to complete the payment. Field must be set for preauth complete to work, and must match the original preauth authorisation ID.

Input Parameters:

Name	Type	Description	Allowed Values
Value	String	Preauth ID of the original preauth transaction.	6 character string

Return Parameter:

No object returned by this method.

5.2.1.2.3.13 setBsbNumber

```
public void setBsbNumber()
```

Only for Direct Entry transactions.

Sets BSB number for financial institution.

Input Parameters:

Name	Type	Description	Allowed Values
value	String	BSB number	Numerical. 6 character string

Return Parameter:

No object returned by this method.

5.2.1.2.3.14 setAccountNumber

```
public void setAccountNumber()
```

Only for Direct Entry transactions.

Sets account number for financial institution.

Input Parameters:

Name	Type	Description	Allowed Values
value	String	Account number	Numerical. 4-9 character string

Return Parameter:

No object returned by this method.

5.2.1.2.3.15 setAccountName

```
public void setAccountName()
```

Only for Direct Entry transactions.

Sets account name for financial institution.

Input Parameters:

Name	Type	Description	Allowed Values
Value	String	Account name	String using characters from Appendix I up to 32 characters in length.

Return Parameter:

No object returned by this method.

5.2.1.2.3.16 setFirstName

```
public void setFirstName()
```

Only for antifraud transactions.

Sets customer's first name.

Input Parameters:

Name	Type	Description	Allowed Values
Value	String	Customer's first name	String, max length 40.

Return Parameter:

No object returned by this method.

5.2.1.2.3.17 setLastName

```
public void setFirstName()
```

Only for antifraud transactions.

Sets customer's last name.

Input Parameters:

Name	Type	Description	Allowed Values
Value	String	Customer's last name	String, max length 40.

Return Parameter:

No object returned by this method.

5.2.1.2.3.18 setZipCode

```
public void setZipCode()
```

Only for antifraud transactions.

Sets customer's address zip code or postal code.

Input Parameters:

Name	Type	Description	Allowed Values
Value	String	Customer's address zip code or postal code	String, max length 30.

Return Parameter:

No object returned by this method.

5.2.1.2.3.19 setTown

```
public void setTown()
```

Only for antifraud transactions.

Sets billing or delivery town of the buyer.

Input Parameters:

Name	Type	Description	Allowed Values
Value	String	Billing or delivery town of the buyer	String, max length 60.

Return Parameter:

No object returned by this method.

5.2.1.2.3.20 setBillingCountry

```
public void setBillingCountry()
```

Only for antifraud transactions.

Sets ISO country code of the billing address.

Input Parameters:

Name	Type	Description	Allowed Values
Value	String	ISO country code of the billing address.	3 digit numeric ISO code or 2 or 3 alpha character ISO code.

Return Parameter:

No object returned by this method.

5.2.1.2.3.21 setDeliveryCountry

```
public void setDeliveryCountry()
```

Only for antifraud transactions.

Sets ISO country code of the delivery address.

Input Parameters:

Name	Type	Description	Allowed Values
Value	String	ISO country code of the delivery address.	3 digit numeric ISO code or 2 or 3 alpha character ISO code.

Return Parameter:

No object returned by this method.

5.2.1.2.3.22 setEmailAddress

```
public void setEmailAddress()
```

Only for antifraud transactions.

Sets email address of the customer.

Input Parameters:

Name	Type	Description	Allowed Values
Value	String	Customer's email address.	String, max length 100.

Return Parameter:

No object returned by this method.

5.2.1.2.3.23 setIp

```
public void setTown()
```

Only for antifraud transactions.

IP address from which the transaction originated.

Input Parameters:

Name	Type	Description	Allowed Values
Value	String	IP address from which the transaction originated.	String, max length 15, must contain 3 p

Return Parameter:

No object returned by this method.

5.2.1.2.3.24 getTxnType

```
public int getTxnType()
```

Returns the transaction type for the transaction. Refer to Appendix A for values of this field.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
int	Integer representing the transaction type of this Txn.

5.2.1.2.3.25 getTxnSource

```
public int getTxnSource()
```

Returns the transaction source for this transaction. Refer to Appendix B for values of this field.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
int	Integer representing the source of the transaction.

5.2.1.2.3.26 getAmount

```
public String getAmount()
```

Returns the amount associated with this financial transaction. Amount is returned with no currency formatting, in the smallest denomination of the transaction's specified currency. The currency of the transaction is determined using `getCurrencyCode()`, described below. If no currency was

specified when making the payment, the default is Australian Dollars (AUD). E.g. an amount of AUD$125.40 would be returned in Australian <u>cents</u>, as "12540".

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	The amount of the financial transaction with no currency formatting, in the smallest denomination of its specified currency. NULL if the amount has not been set.

5.2.1.2.3.27 getCurrencyCode

```
public String getCurrencyCode()
```

Returns the 3-character currency identifier associated with this financial transaction.

If `process()` has not yet been called on this `Payment`, this method will only return a value if `setCurrencyCode()` has been called; otherwise it will return NULL, implying the default value of "AUD" (Australian Dollars).

If `process()` has been called, this method will return the currency code that was used for the payment. If the currency code was not set, the default of "AUD" (Australian Dollars) will be returned.

See valid currency code table in <u>Appendix H</u>.

> *If `process()` has been called and the payment was approved AND this method still returns NULL, then the payment was sent to a SecurePay URL which does not yet support multi-currency, and the payment has been processed in Australian Dollars. You will need to refund the payment in Australian Dollars if this was done in error.*

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	The 3-letter currency code of the financial transaction. NULL if the currency has not been set and process() has not yet been called, indicating Australian Dollars will be used by default; or if the payment HAS been processed but the SecurePay URL does not support multi-currency yet.

5.2.1.2.3.28 getPonum

```
public String getPonum()
```

Returns the purchase order number for this transaction.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	The purchase order number referring to the transaction.

NULL if the purchase order number has not been set.

5.2.1.2.3.29 getCardNumber

```
public String getCardNumber()
```

<u>For Credit Card transactions:</u>

Returns the truncated credit card number, as stored in SecurePay's database. This is the first 6 and last 3 digits of the original card number, separated by "…". E.g. Setting a card number as:

```
txn.setCardNumber("4444333322221111");
```

then a call to:

```
txn.getCardNumber();
```

returns "`444433...111`".

<u>For Direct Entry transactions in Payment requests:</u>

Returns NULL.

<u>Input Parameters:</u>

No input parameters are required by this method.

<u>Return Parameter:</u>

Type	Description
String	The truncated credit card number or BSB and account numbers of this financial transaction. NULL if the card number has not been set.

5.2.1.2.3.30 getCardType

```
public int getCardType()
```

<u>Only for Credit Card transactions.</u>

Returns an integer representing the type of card used. Refer to <u>Appendix C.</u>

<u>Input Parameters:</u>

No input parameters are required by this method.

<u>Return Parameter:</u>

Type	Description
int	The credit card type code of the card number used.

5.2.1.2.3.31 getCardDescription

```
public int getCardDescription()
```

<u>Only for Credit Card transactions.</u>

Returns a textual description of card used, or NULL if transaction has not been processed yet.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	Description of the type of credit card used.

5.2.1.2.3.32 getCVV

```
public String getCVV()
```

Only for Credit Card transactions.

Returns the Card Verification Value of the financial transaction. After the transaction has been processed, this method will return NULL, as this value should not be logged, and is not stored by SecurePay.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	The Card Verification Value of this financial transaction. NULL if the value has not been set, or if the transaction has been processed.

5.2.1.2.3.33 getExpiryDate

```
public String getExpiryDate()
```

Only for Credit Card transactions.

Returns the credit card expiry date of the financial transaction.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	The credit card expiry date of this financial transaction. NULL if the value has not been set, or if the expiry date was not sent with a recurring payment.

5.2.1.2.3.34 getTrack2Data (Card-Present transactions only)

```
public void getTrack2Data()
```

Only for Credit Card transactions.

Track 2 of the magnetic strip on the customer's physical credit card, as passed in by `setTrack2Data()`. After the transaction has been processed, this method will return NULL, as SecurePay does not store this data.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description

String	The Track 2 data from the magnetic strip on the customer's credit card. NULL if the value has not been set, or if the transaction has already been processed.

5.2.1.2.3.35 getXID (3D-Secure merchants only)

```
public String getXID()
```

Only for Credit Card transactions.

Returns the 3D-Secure XID (transaction id) of the financial transaction. After the transaction has been processed, this method will return NULL, as this value should not be logged, and is not stored by SecurePay.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	The 3D-Secure XID of this financial transaction. NULL if the value has not been set, or if the transaction has already been processed.

5.2.1.2.3.36 getCAVV (3D-Secure merchants only)

```
public String getCAVV()
```

Only for Credit Card transactions.

Returns the 3D-Secure Cardholder Authorisation Verification Value of the financial transaction. After the transaction has been processed, this method will return NULL, as this value should not be logged, and is not stored by SecurePay.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	The 3D-Secure CAVV of this financial transaction. NULL if the value has not been set, or if the transaction has already been processed.

5.2.1.2.3.37 getSLI (3D-Secure merchants only)

```
public String getSLI()
```

Only for Credit Card transactions.

Returns the 3D-Secure Service Level Indicator (or ECI, E-Commerce Indicator) of the financial transaction. After the transaction has been processed, this method will return NULL, as this value should not be logged, and is not stored by SecurePay.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	The 3D-Secure SLI (or ECI) of this financial transaction. NULL if the value has not been set, or if the transaction has already been processed.

5.2.1.2.3.38 getBsbNumber

```
public String getBsbNumber()
```

Only for Direct Entry transactions in Payment request.

Returns the BSB number for this transaction.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	The BSB number referring to the transaction. NULL if the BSB number has not been set.

5.2.1.2.3.39 getAccountNumber

```
public String getAccountNumber()
```

Only for Direct Entry transactions in Payment request.

Returns the account number for this transaction.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	The account number referring to the transaction. NULL if the account number has not been set.

5.2.1.2.3.40 getAccountName

```
public String getAccountName()
```

Only for Direct Entry transactions in Payment request.

Returns the account name for this transaction.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	The account name referring to the transaction. NULL if the account name has not been set.

5.2.1.2.3.41 getAntiFraudResponseCode

`public String getAntiFraudResponseCode()`

Only for antifraud transactions in Payment request.

Returns the response code of the antifraud verification. This will be a 3-digit response from SecurePay's server or API. The method `getAntiFraudResponseText()` provides more details in a textual format.

Refer to the SecurePay Payment Response Codes document for details of codes returned. This document may be downloaded from SecurePay's Merchant Login website, or provided via email by SecurePay Merchant Support.

If the antifraud verification failed, the transaction will not be sent to the bank and therefore there will be no values returned by methods such as getResponseCode(), getResponseText(), getSettlementDate(), isApproved(), getTxnId().

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	3-digit SecurePay / gateway response.

5.2.1.2.3.42 getAntiFraudResponseText

`public String getAntiFraudResponseText()`

Returns a textual description of the antifraud response code received.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	Description of antifraud response received.

5.2.1.2.3.43 getApproved

`public boolean getApproved()`

Returns a boolean representing the financial result of the transaction.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
boolean	**true** if the payment was authorised by the merchant bank and customer card issuer, or **false** in any other case.

5.2.1.2.3.44 getResponseCode

`public String getResponseCode()`

Returns the response code of the transaction. This is either a 2-digit response (00-99) from the bank, a 3-digit response (100-299) from SecurePay's server or API, or a 3-digit response (900-999)

mapped from SecurePay's bank link responses. The method `getResponseText()` provides more details in a textual format.

Refer to the SecurePay Payment Response Codes document for details of codes returned. This document may be downloaded from SecurePay's Merchant Login website, or provided via email by SecurePay Merchant Support.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	2-digit bank response or 3-digit SecurePay / gateway response.

5.2.1.2.3.45 getResponseText

`public String getResponseText()`

Returns a textual description of the response code received for the transaction.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	Description of response received from API, SecurePay, or bank. NULL if the transaction has not yet been processed.

5.2.1.2.3.46 getThinlinkResponseCode

`public String getThinlinkResponseCode()`

Only for Credit Card transactions.

Maps SecurePay's response code field to a Westpac Thinlink Payment Result Code for ex-Thinlink users. Refer to Appendix G.1 Payment Result Codes for details of codes returned.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	3-digit Thinlink Payment Result Code

5.2.1.2.3.47 getThinlinkResponseText

`public String getThinlinkResponseText()`

Only for Credit Card transactions.

In accordance with Thinlink, this method always returns "000", meaning "No further information available". Additional codes may be added in future.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	3-digit Thinlink Payment Result Description ("000").

5.2.1.2.3.48 getThinlinkEventStatusCode

```
public String getThinlinkEventStatusCode()
```

Only for Credit Card transactions.

Maps SecurePay's response code field to a Westpac Thinlink Event Status Code for ex-Thinlink users. Refer to Appendix

G.2 Event Status Codes for details of codes returned.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	3-digit Thinlink Event Status Code

5.2.1.2.3.49 getThinlinkEventStatusText

```
public String getThinlinkEventStatusText()
```

Only for Credit Card transactions.

Textual description of the Thinlink Event Status Code for ex-Thinlink users.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	Textual description of the Event Status Code.

5.2.1.2.3.50 getSettlementDate

```
public String getSettlementDate()
```

Returns the bank settlement date when the funds will be settled into the merchant's account. This will be the current date mostly, however after the bank's daily cut-off time, or on non-banking days, the settlement date will be the next business day. The format of the settlement date is "YYYYMMDD".

In case of Direct Entry transactions this will be the current date.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	Bank settlement date in "YYYYMMDD" format. NULL if the bank did not receive the transaction. (A settlement date may be returned for declined transactions.) NULL for direct entry transactions if the transaction could not be stored in SecurePay system.

5.2.1.2.3.51 getTxnId

```
public String getTxnId()
```

Returns the Bank Transaction ID of the transaction. The format of this string depends on the merchant's bank, and ranges between 6 and 30 characters.

Note: The transaction ID of a payment transaction must be sent back with a request for a refund or reversal of that transaction.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	Bank transaction ID of the transaction. NULL if the transaction has not been processed, or in some cases if it was not received by the bank.

5.2.1.2.3.52 getPreauthId

```
public String getPreauthId()
```

Only for Credit Card transactions.

Only for Preauth transactions.

Returns the preauthorisation ID of the transaction (for Preauth transaction types only). NULL is returned for any other transaction type, or if the preauthorisation was not received by the bank.

Note: The preauth ID of a preauth transaction must be sent back with a request for a preauth complete (advice) of that transaction.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	Preauth ID of the transaction. NULL if the transaction is not a Preauth type, or has not been processed, or in some cases if the preauth was not received by the bank.

5.2.1.3   Payment Sample Code

```
// Import required classes
import securepay.jxa.api.Payment;
import securepay.jxa.api.Txn;

class SecurePayPaymentTest
{
  public SecurePayPaymentTest()
  {
      // Create Payment container object
      Payment payment = new Payment();

      // Print generated values
      System.out.println("Message ID:  " + payment.getMessageId());
      System.out.println("Timestamp:                            "           +
payment.getMessageTimestampAsString());
      System.out.println("API Version: " + payment.getApiVersion());
```

```java
        // Set Payment values
        payment.setServerURL(
                    "http://fakeurl.com/payment");   // Dummy   payment   URL,
            please use the URL supplied by SecurePay Support
        payment.setProcessTimeout(80);                       // 80 seconds
        payment.setMerchantId("XYZ9999");                    // ID   allocated   by
SecurePay

        // Add a Txn to the container
        //      type = 0 (payment)
        //      ponum = "SecurePayTest"
        Txn txn = payment.addTxn(0, "SecurePayTest");

        // Set the Txn values
        txn.setTxnSource(8);                        // API source
        txn.setAmount("1000");                          // A$10.00
        txn.setCurrencyCode("AUD");                 // Australian Dollars
        txn.setCardNumber("4444333322221111");      // dummy card number
        txn.setCVV("456");                          // dummy CVV (optional)
        txn.setExpiryDate("03/06");                 // March, 2006

        // Send payment to SecurePay for processing
        boolean processed = payment.process("password");

        // If payment was not sent to SecurePay, print status and exit
        if (!processed)
        {
          System.out.println("Payment was not sent to server.");
          System.out.println("Status: " + payment.getStatusCode());
          System.out.println("Desc:   " + payment.getStatusDesc());
        }
        else    // Payment was processed correctly
        {
          System.out.println("Response Received.");

          if (payment.getCount() == 1)
          {
                // Get the Txn response object
                    Txn resp = payment.getTxn(0);    // First  Txn  in  response
            queue

                    // Print response variables
            System.out.println("Transaction response parameters:");
            System.out.println("Status:   " + payment.getStatusCode());
                    System.out.println("Desc:      " + payment.getStatusDesc());
            System.out.println("Approved: " + resp.getApproved());
                    System.out.println("RespCode: " + resp.getResponseCode());
            System.out.println("RespText: " + resp.getResponseText());
                    System.out.println("SettDate: " + resp.getSettlementDate());
            System.out.println("TxnID:     " + resp.getTxnId());
          }
          else
          {
            System.out.println("No Txn object returned.");
            System.out.println("Check status code for error details.");
          }
        }
    }

public static void main(String[] args)
{
  SecurePayPaymentTest test = new SecurePayPaymentTest();
  }
}
```

5.2.1.4 Refund Sample Code

```java
// Import required classes
import securepay.jxa.api.Payment;
import securepay.jxa.api.Txn;

class SecurePayRefundTest
{
public SecurePayRefundTest()
     {
     // Create Payment container object
     Payment payment = new Payment();

     // Print generated values
     System.out.println("Message ID:  " + payment.getMessageId());
     System.out.println("Timestamp:                              "        +
payment.getMessageTimestampAsString());
     System.out.println("API Version: " + payment.getApiVersion());

     // Set Payment values
     payment.setServerURL(
                 "http://fakeurl.com/payment");   //   Dummy   payment   URL,
          please use the URL supplied by SecurePay Support
     payment.setProcessTimeout(80);                      // 80 seconds
     payment.setMerchantId("XYZ9999");                   //   ID    allocated    by
SecurePay

     // Add a Txn to the container
     //      type = 4 (refund)
     //      ponum of the original payment = "SecurePayTest"
     Txn txn = payment.addTxn(4, "SecurePayTest");

     // Set the Txn values
     txn.setTxnSource(8);                       // API source
     txn.setAmount("1000");                         // A$10.00
     txn.setCurrencyCode("AUD");                // Australian Dollars
      txn.setTxnId("123123");                   //  Transaction  Id  returned  for
the
                                               // original payment

     // Send refund to SecurePay for processing
     boolean processed = payment.process("password");

     // If refund was not sent to SecurePay, print status and exit
     if (!processed)
     {
       System.out.println("Refund was not sent to server.");
       System.out.println("Status: " + payment.getStatusCode());
       System.out.println("Desc:   " + payment.getStatusDesc());
     }
     else    // Refund was processed correctly
     {
       System.out.println("Response Received.");

       if (payment.getCount() == 1)
       {
             // Get the Txn response object
                 Txn resp = payment.getTxn(0);     //  First  Txn  in  response
           queue

                 // Print response variables
             System.out.println("Transaction response parameters:");
             System.out.println("Status:    " + payment.getStatusCode());
                 System.out.println("Desc:      " + payment.getStatusDesc());
             System.out.println("Approved: " + resp.getApproved());
                 System.out.println("RespCode: " + resp.getResponseCode());
             System.out.println("RespText: " + resp.getResponseText());
                 System.out.println("SettDate: " + resp.getSettlementDate());
```

```
                System.out.println("TxnID:     " + resp.getTxnId());
        }
        else
        {
                System.out.println("No Txn object returned.");
                System.out.println("Check status code for error details.");
        }
        }
    }

  public static void main(String[] args)
  {
    SecurePayRefundTest test = new SecurePayRefundTest();
  }
}
```

5.2.2 Echo

5.2.2.1   securepay.jxa.api.Echo

The Echo object can be created and sent to any of the SecurePay service URLs (Payment or DirectEntry) to ensure the SecurePay service is available, and to confirm the status of the service.

Echo messages must be used appropriately. If you use this method to poll SecurePay services, polls should be no less than 5 minutes apart. An Echo should not be sent if a transaction with a bank response code (00-99) has been processed by your system within the last 5 minutes.

5.2.2.2   Constructor

```
public Echo()
```

Create an empty `Echo` object.

Parameters:

No parameters required by this constructor.

5.2.2.2.1 Public Methods

5.2.2.2.1.1   getMessageId

```
public String getMessageId()
```

Returns the unique message identifier created by the API.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	Unique identifier of this Echo object.

5.2.2.2.1.2   getMessageTimestamp

```
public Date getMessageTimestamp()
```

Returns the timestamp created by the API for the `Echo` in a `Date` object.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
Date	Timestamp of this Echo object.

5.2.2.2.1.3   getMessageTimestampAsString

```
public String getMessageTimestampAsString()
```

Returns the timestamp created by the API for the `Echo` as a string. Refer to Appendix E for the format of the string returned.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	Timestamp of this Echo object.

5.2.2.2.1.4   getApiVersion

```
public String getApiVersion()
```

Returns the version of the API being used to create this object.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	API version used to create this Echo object.

5.2.2.2.1.5   setMerchantId

public void setMerchantId(String id)

Sets the merchant id to be used when processing the echo request.

Input Parameters:

Name	Type	Description	Allowed Values
id	String	Merchant ID allocated to the merchant by SecurePay.	7-character string in format XXXDDDD, where X is a letter (A-Z) and D is a digit (0-9). SecurePay will supply this value to you upon application. **For Direct Entry Echo Request:** 5-7 character string in format XXXDDDD, where X is a letter (A-Z) and D is a digit (0-9).

Page 37 of 49

© SecurePay Pty Ltd   SecurePay Java API integration Guide Version 1.3

			Last two digits can be ignored.

Return Parameter:

No object returned by this method.

5.2.2.2.1.6 setServerURL

`public void setServerURL(String url)`

Sets the URL of the SecurePay service to which this Echo will be sent when `process()` is called.

Input Parameters:

Name	Type	Description	Allowed Values
url	String	URL of SecurePay's server.	**Test:** https://www.securepay.com.au/test /payment **Live:** https://www.securepay.com.au/xml api/payment **For Direct Entry:** **Test:** https://www.securepay.com.au/test /directentry **Live:** https://www.securepay.com.au/xml api/directentry

Return Parameter:

No object returned by this method.

5.2.2.2.1.7 setProcessTimeout

`public void setProcessTimeout(int timeout)`

Sets the timeout in seconds to wait for a response message from SecurePay's server. If no response is received in this time, a timeout response is returned, and transaction results may be queried at a later time.

If value is not set, or is set to an integer < 0, default timeout of 80 seconds is used.

Input Parameters:

Name	Type	Description	Allowed Values
timeout	int	Seconds to wait for SecurePay response.	Int < 0: default value is used Int >= 0: supplied value is used.

Return Parameter:

No object returned by this method.

5.2.2.2.1.8 Process

`public boolean process(String password)`

Sends this Echo object to SecurePay's Server. This method may be called multiple times on the same instance of the `Echo` object.

As of Secure Java V4.0.3a, the `password` parameter <u>must</u> be passed to this method. Previous releases of Secure Java V4.0.x which did not pass a password will no longer be compatible with the SecurePay Payment Servers, as it is now required that a password is used for all financial transactions and echoes. The password can be changed via SecurePay's Merchant Login website by logging in as the "admin" user.

If the merchant Id and password sent do not match those stored in SecurePay's database, the response message will have a status code of "550".

Input Parameters:

Name	Type	Description	Allowed Values
password	String	The password is configured in SecurePay's database per Merchant ID, and must match the value passed by this method for the message to be accepted and processed.	The password is allocated to you by SecurePay when you receive this software, and can be changed via the Merchant Login website.

Return Parameter:

Type	Description
Boolean	**true** if message was sent to server correctly and a response received, and **false** if client-side errors occurred preventing the object from being be sent.

5.2.2.2.1.9 getStatusCode

```
public long getStatusCode()
```

Returns the status code of the Echo. Refer to <u>Appendix F</u> for more details.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
long	Code representing the status of the Echo object after processing.

5.2.2.2.1.10 getStatusDesc

```
public String getStatusDesc()
```

Returns the status description of the Echo response.

Input Parameters:

No input parameters are required by this method.

Return Parameter:

Type	Description
String	Textual description of the status of the Echo object after processing.

5.2.2.3 Echo Sample Code

```
// Import required classes
import securepay.jxa.api.Echo;

class SecurePayEchoTest
{
    public SecurePayEchoTest()
    {
        // Create Payment container object
        Echo echo = new Echo();

        // Print generated values
        System.out.println("Message ID:  " + echo.getMessageId());
        System.out.println("Timestamp:                         "      +
    echo.getMessageTimestampAsString());
        System.out.println("API Version: " + echo.getApiVersion());

        // Set Payment values
        echo.setServerURL(
                    "http://fakeurl.com/payment");    //   Dummy   payment
            URL, please use URL supplied by SecurePay Support
        echo.setProcessTimeout(80);                         // 80 seconds
        echo.setMerchantId("XYZ9999");                      //  ID  allocated  by
    SecurePay

        // Send echo to SecurePay
        boolean processed = echo.process("password");

        if (!processed)
        {
            System.out.println("Payment was not sent to server.");
        }
        else    // Payment was processed correctly
        {
            System.out.println("Response Received.");
        }

        // Print status of Echo
      System.out.println("Status:   " + echo.getStatusCode());
        System.out.println("Desc:     " + echo.getStatusDesc());
    }

    public static void main(String[] args)
    {
        SecurePayEchoTest test = new SecurePayEchoTest();
    }
}
```

SecurePay Java API integration Guide Version 1.3

Appendix A: Transaction Types

Transaction type codes define the type of financial transaction processed by SecurePay.

Codes with **shaded background** are permitted in Payment transactions from this API. All other codes are provided for completeness.

Code	Description
0	Standard Payment
1	Mobile Payment
2	Batch Payment
3	Periodic Payment
4	Refund
5	Error Reversal (Void)
6	Client Reversal (Void)
10	Preauthorise
11	Preauth Complete (Advice)
14	Recurring Payment
15	Direct Entry Debit
17	Direct Entry Credit
19	Card-Present Payment
20	IVR Payment
21	Antifraud payment
22	Antifraud (only) request

SecurePay Java API integration Guide Version 1.3

Appendix B: Transaction Sources

Transaction source codes track the origin of financial transaction processed by SecurePay.

Codes with **shaded background** are permitted in Payment transactions from this API. All other codes are provided for completeness.

Other source codes may be used, with the permission of SecurePay. Contact SecurePay Support for more information.

Code	Description
0	Unknown (default)
1	SecureLink
2	Merchant Login
3	SATM
4	SecureBill Portal
5	SecureBill Link
7	SecurePOS
8	API (Secure Java)
9	Call Centre Payment Switch
10	Batch Server
11	IVR1
12	IVR2
13	SecureMobile
14	Reconciliation Engine
16	Helpdesk Login
18	eSec Interface
19	Periodic Server
23	SecureXML
24	DirectOne Interface
25	Antifraud Server
90	Reserved

Appendix C: Card Types

SecurePay uses numeric codes to refer to credit card types in our system.

Code	Description
0	Unknown
1	JCB
2	American Express (Amex)
3	Diners Club
4	Bankcard
5	MasterCard
6	Visa

Appendix D: Location of CVV

The Card Verification Value is an anti-fraud measure used by some banks to prevent payments from generated card numbers. The CVV number is printed on the physical card, and is randomly assigned, therefore cannot be auto-generated.

The CVV number can be found in the following places:

Card Type	Location
Visa	Signature strip on back of card. Last digits of card number are re-printed in reverse italics, followed by 3-digit CVV.
MasterCard	Signature strip on back of card. Last digits of card number are re-printed in reverse italics, followed by 3-digit CVV.
Bankcard	Signature strip on back of card. Last digits of card number are re-printed in reverse italics, followed by 3-digit CVV.
Amex	4 digit CVV above card number on front of card.
Diners Club	Signature strip on back of card. Last digits of card number are re-printed in reverse italics, followed by 3-digit CVV.
JCB	Not used

Appendix E: Timestamp String Format

The format of the Timestamp or Log Time strings returned by Secure Java is:

YYYYDDMMHHNNSSKKK000sOOO

where:

- YYYY   is a 4-digit year
- DD   is a 2-digit zero-padded day of month
- MM   is a 2-digit zero-padded month of year (January = 01)
- HH   is a 2-digit zero-padded hour of day in 24-hour clock format (midnight =0)
- NN   is a 2-digit zero-padded minute of hour
- SS   is a 2-digit zero-padded second of minute
- KKK   is a 3-digit zero-padded millisecond of second
- 000   is a Static 0 characters, as SecurePay does not store nanoseconds
- sOOO   is a Time zone offset, where s is "+" or "-", and OOO = minutes, from GMT.

E.g. June 24, 2002 5:12:16.789 PM, Australian EST is:

20022406171216789000+600

Appendix F: SecurePay Status Codes

Status Code	Response Text	Description
0	Normal	Message processed correctly (check transaction response for details).
504	Invalid Merchant ID	If Merchant ID does not follow the format XXXDDDD, where X is a letter and D is a digit, or Merchant ID is not found in SecurePay's database.
505	Invalid URL	The URL passed to either Echo, or Payment object is invalid.
510	Unable To Connect To Server	Produced by SecurePay Client API when unable to establish connection to SecurePay Payment Gateway
511	Server Connection Aborted During Transaction	Produced by SecurePay Client API when connection to SecurePay Payment Gateway is lost after the payment transaction has been sent
512	Transaction timed out By Client	Produced by SecurePay Client API when no response to payment transaction has been received from SecurePay Payment Gateway within predefined time period (default 80 seconds)
513	General Database Error	Unable to read information from the database.
514	Error loading properties file	Payment Gateway encountered an error while loading configuration information for this transaction
515	Fatal Unknown Error	Transaction could not be processed by the Payment Gateway due to unknown reasons
516	Request type unavailable	SecurePay system doesn't support the requested transaction type
517	Message Format Error	SecurePay Payment Gateway couldn't correctly interpret the transaction message sent
524	Response not received	The client could not receive a response from the server.
545	System maintenance in progress	The system maintenance is in progress and the system is currently unable to process transactions
550	Invalid password	The merchant has attempted to process a request with an invalid password.
575	Not implemented	This functionality has not yet been implemented
577	Too Many Records for Processing	The maximum number of allowed events in a single message has been exceeded.
580	Process method has not been called	The process() method on either Echo, or Payment object has not been called
595	Merchant Disabled	SecurePay has disabled the merchant and the requests from this merchant will not be processed.

Appendix G: Thinlink Codes

G.1 Payment Result Codes

Code	Description
100	Payment approved
200	Payment declined
300	Payment incomplete – Unable to process, try again later
400	Scheduled – Execution pending
999	Unknown status

G.2 Event Status Codes

Code	Description	Code	Description
000	Normal Completion	986	Error – Invalid Email Address
002	No event to process	988	Error – Unknown Currency
980	Error – Bad Card Number	990	Error – Invalid Amount
981	Error – Expired Card	991	Error – Invalid MTID
982	Error – Invalid Transaction Type	992	Error – Duplicate Invoice Number
984	Error – Formatting Error	999	Error – Unspecified

Appendix H: Currency Codes List

IMPORTNANT NOTICE:

You must meet certain requirements with your bank and SecurePay before using SecurePay's multi-currency features. Please ask SecurePay if we support multi-currency payments through your bank, and if so, what currency types are available. You may also need to open multi-currency accounts with your bank for each currency you propose to transact in. Contact SecurePay Support or your SecurePay Account Manager for full details.

Code	Description	Minor Units	Example*	
			Amount	Pass As
AUD	Australian Dollar	2	$20	2000
CAD	Canadian Dollar	2	$20	2000
CHF	Swiss Franc	2	20	2000
DEM	German Deutschmark	2	20	2000
EUR	Euro	2	€20	2000
FRF	French Franc	2	20	2000
GBP	English Pound	2	£20	2000
GRD	Greek Drachma	0	20	20
HKD	Hong Kong Dollar	2	$20	2000
ITL	Italian Lira	0	L20	20
JPY	Japanese Yen	0	¥20	20
NZD	New Zealand Dollar	2	$20	2000
SGD	Singapore Dollar	2	$20	2000
USD	US Dollar	2	$20	2000

* To pass a multicurrency payment to SecurePay, call `setCurrencyCode(…)` with the value from the Code column, and call `setAmount(…)` with the amount to be charged, ensuring you set the correct number of Minor Units for the selected currency, as shown in the examples.

E.g. For US Dollars, $4,125.90 is set using:

```
txn.setAmount("412590");
txn.setCurrencyCode("USD");
```

or for Japanese Yen, ¥67,925 is set using:

```
txn.setAmount("67925");
txn.setCurrencyCode("JPY");
```

SecurePay Java API integration Guide Version 1.3

Appendix I: Direct Entry Character set

Description	Characters allowed
Numeric	0 - 9
Alphabetic	a – z, A - Z
Oblique slash	/
Hyphen	-
Ampersand	&
Period	.
Asterisk	*
Apostrophe	'
Blank space	

SecurePay Java API integration Guide Version 1.3