Internet Payment Gateway

Generic API Developer Guide

st.george

Table of Contents

Disclaimer

This document, including attachments, is not for general circulation, but for distribution to a limited number of specific corporations and individuals that are known to St.George Bank ("Company"). It is in draft form and its contents are subject to change, including changes of substance. Any person choosing to act on information contained in this document is advised to verify the information.

Confidentiality

By accepting this document, the recipient agrees to keep the information contained in it permanently confidential. This document is intended for use by the recipient only and may not be copied, reproduced or distributed to others at any time without the prior written consent of the Company. It cannot be used for any purpose except that for which it is given to you i.e. as user documentation.

If you do not agree with any of the above conditions, you must return this document immediately.

Document Purpose / Intended Audience

This guide is one component of the St. George Bank Transaction Server document set. It intended to act a reference point for developers seeking to implement the St. George Bank transaction functionality into merchant web applications. It provides specific technical information about the typical deployment of credit card transactions using the St. George Bank APIs.

This document contains information applicable to all APIs and should be used in conjunction with the relevant API Developer guide to suit your environment.

Related Documents

- **St. George Bank API Developer Guide**
  - **St. George Bank Win32 Developer Guide**
  - **St. George Bank Java Developer Guide**
  - **St. George Bank Perl Developer Guide**
  - **St. George Bank Linux Developer Guide**
  - **St. George Bank PHP Developer Guide**

1 Overview

The St.George Internet Payment Gateway (IPG) is a secure digital e-Commerce business solution that is used by merchants to process financial transactions.

The system can be divided into two layers: Client and Gateway. The two layers are connected using a proprietary messaging protocol, with the messages being encrypted and then sent using TCP/IP (Transmission Control Protocol/Internet Protocol). Diagram 1 illustrates the two layers and the separation of processing.

The St. George API's reside within the Client layer and gather information from merchant applications. This information is then compiled into a "bundle" and transmitted to the Gateway using the proprietary messaging protocol. After processing the information is "bundled" and returned to the API for display to the customer and reconciliation with the merchant's records.

Throughout each stage of operation, the IPG authorises, logs, authenticates, and validates all transactions. This ensures that transactions are never lost. Comprehensive real-time reporting and monitoring is available through a simple browser-based administration console. The console even allows merchants to remotely manage reporting and their own users.

1.1 Security

The St. George Bank Payment Gateway maintains secure transactions through the use of SSL (Secure Sockets Layer). The SSL Handshake Protocol was originally developed by Netscape Communications Corporation to provide security and privacy over the Internet. Using SSL, we can provide client and server authentication, and ensure 128-bit encryption of all transaction details.

NOTE: Only the connection between API and IPG is secured. This means for example, that the collection of credit card details from the customer has to be secured separately.

St. George bank provides developers with a certificate keystore for the Java API or a certificate file for all other APIs.

1.2 Performance

The St. George Bank IPG is replicated to provide high availability, fail-over, and fast processing under load. The average time taken to complete a transaction is 6-8 seconds, but merchants may experience faster or slower transaction completion times due to a number of factors (see section 4.6)

1.3 Connectivity

St. George Bank is currently using two gateway machines for communication with the client. This allows for load balancing and replication for fail over. The API utilises round-robin DNS (2 IP addresses) to ensure robustness.

1.4 Interface Description

The St.George Bank client API uses a “transaction bundle” to communicate with the IPG. Before sending a transaction, “request fields” in the bundle (such as TRANSACTIONTYPE, CLIENTID, TOTALAMOUNT, etc.) need to be set to appropriate values. The transaction is then sent off by calling the execute() method of the API. The Transaction Server’s response can be obtained by extracting the values of “response fields” (such as RESPONSECODE, RESPONSETEXT, etc.) from the bundle. The request and response fields are discussed in chapter 3.

2 Available APIs

St. George Bank provides a range of APIs that integrate with most popular platforms, operating systems and programming languages.

The St. George Bank APIs provide an interface for use with the St. George Bank Internet Payment Gateway. They enable web site and application developers to safely and efficiently add credit card transaction processing capabilities to their products.

A comprehensive Software Development Kit is delivered with each API. Contained in the Development Kit is everything a developer requires to develop code to integrate the API into a merchant application, including; system libraries, test applications, documentation, sample code, a description of all methods and functions, a troubleshooting guide, and all reference material.

3 Credit Card Transactions

This chapter deals exclusively with **creditcard** transactions and the parameters required to complete them.

Please note: As additional interfaces and transaction types become available, St. George Bank will issue merchants and developers with the relevant information regarding the new parameters and other settings needed to adopt these transaction types. If the Merchant decides to implement this new functionality, it is simply a matter of setting new parameters in the existing interface, calling execute and getting the returned values.

3.1 Supported Credit Card Transaction Types

This section describes the valid types of transactions available within the CREDITCARD interface.

Credit card transaction types are specified in the transaction bundle and are defined by setting the **TRANSACTIONTYPE** parameter. See the transaction request table on the following page for element requirements.

Please note: The transaction type must be specified in uppercase.

3.1.1 PURCHASE

The PURCHASE transaction type performs a 'standard' credit card purchase transaction. It is the most commonly used transaction type within the St. George Bank System.

3.1.2 REFUND

The REFUND transaction type performs a refund of any PURCHASE or COMPLETION transaction within the St. George Bank System. Partial refunds are allowed, with cumulative partial refunds being allowed up to the value of the original transaction. The transaction reference of the original transaction is the catalyst utilised for the REFUND transaction to be processed.

3.1.3 PREAUTH

PREAUTH transactions put a "hold" on funds, for completion some time later. The PREAUTH transaction will return an AUTHCODE & TXNREFERENCE, which must be used when completing the transaction. PREAUTH transactions are completed using the COMPLETION transaction type.

Please note that PREAUTH/COMPLETION transactions have to be enabled by St. George Bank upon request.

3.1.4 COMPLETION

The COMPLETION transaction type finalises a transaction initiated using the PREAUTH transaction type. The AUTHCODE returned by the original transaction needs to be provided as AUTHORISATIONNUMBER or AUTHCODE or AUTHORISATIONCODE for the completion. The TXNREFERENCE returned by the original transaction needs to be provided as PREAUTHNUMBER or ORIGINALTXNREF for the completion.

Depending on the financial institution used to process the transaction, the completion amount may vary from the amount originally authorised (currently +/- 15%). Please check with the St. George Bank Support Team to confirm this amount range. Additionally, the COMPLETION must use the same financial institution that completed the original authorisation.

Please note that PREAUTH/COMPLETION transactions have to be enabled by St. George Bank upon request.

3.1.5 STATUS

The STATUS transaction type is used to query the status of any transaction. It is useful when the transaction returns with "IP" (in progress), or fails after the client has received the transaction reference. The transaction reference returned is used to query the IPG to determine the current status of the transaction.

3.2 Credit Card Transaction Request

The matrix below shows the request parameters required, and the transaction types to which they apply. These parameters need to be supplied from the merchant application to complete the transaction request. Full descriptions of the Request fields are also included below.

NOTE: All parameters used must be entered in UPPERCASE.

Request Field/ Transaction Type	PURCHASE	REFUND	PREAUTH	COMPLETION	STATUS
INTERFACE	✓	✓	✓	✓	✓
TRANSACTIONTYPE	✓	✓	✓	✓	✓
TOTALAMOUNT	✓	✓	✓	✓	✗
TAXAMOUNT	*	*	*	*	✗
CARDDATA	✓	✗	✓	✗	✗
CARDEXPIRYDATE	✓	✗	✓	✗	✗
TXNREFERENCE	✗	✗	✗	✗	✓
ORIGINALTXNREF	✗	✓	✗	✓	✗
ORIGINALTXNREF or PREAUTHNUMBER	✗	✗	✗	✓	✗
AUTHNUMBER or AUTHORISATIONCODE or AUTHORISATIONNUMBER	✗	✗	✗	✓	✗
CLIENTREF	*	*	*	*	✗
COMMENT	*	*	*	*	✗
MERCHANT_CARDHOLDERNAME	*	*	*	*	✗
MERCHANT_DESCRIPTION	*	*	*	*	✗
TERMINALTYPE	*	*	*	*	✗
CVC2	*	*	*	*	✗

✓ – Mandatory, * – Optional, ✗ – Not Available

3.2.1 INTERFACE

The INTERFACE parameter determines the interface to be used for transaction processing. Currently only two values are supported; 'TEST' - used for testing client-server connectivity, and 'CREDITCARD' - for processing credit card transactions on the Live and Test systems. Please only use interface type '**CREDITCARD**".

3.2.2 TRANSACTIONTYPE

The TRANSACTIONTYPE parameter controls the type of transaction performed. Valid TRANSACTIONTYPE values are:

- PURCHASE
- REFUND
- PREAUTH
- COMPLETION
- STATUS

3.2.3 TOTALAMOUNT

TOTALAMOUNT is the total amount to be transacted, inclusive of any taxes. This value must be in decimal format and may contain numeric characters and a decimal point '.' character, for example: 20.00 for a $20.00 transaction, 3099.95 for a transaction to the value of $3099.95. Do not include the dollar symbol '$'.

NOTE: Using the **Test** environment the RESPONSECODE returned may be varied by selecting a different cents value in this field. It is essential that you test for all responses and handle each appropriately within the merchant application.

3.2.4 TAXAMOUNT

The TAXAMOUNT parameter contains the tax component of the total amount. This field must have the same format as the TOTALAMOUNT field. (This field is optional).

3.2.5 CARDDATA

The CARDDATA parameter holds the credit card number that the transaction will apply to. This number must not contain any white space, or characters other than the numbers 0 to 9. It must be between 13 and 16 characters long or it will be deemed invalid. The card type is automatically determined by the first 4 digits of the card data therefore no additional field is required.

3.2.6 CARDEXPIRYDATE

The CARDEXPIRYDATE parameter is the expiry date of the credit card to which the transaction applies. This number must not contain any white space, or characters other than the numbers 0 to 9, and must be exactly 4 characters (format MMYY) in length.

3.2.7 TXNREFERENCE

The TXNREFERENCE input field applies only to STATUS transaction types. It is used to hold the Unique St. George Bank Transaction reference for the original transaction.

3.2.8 ORIGINALTXNREF / PREAUTHNUMBER

The ORIGINALTXNREF input field applies only to REFUND & COMPLETION transaction types. It is used to hold the Unique St. George Bank Transaction reference for the original transaction. (The value must correlate to the transactions originally processed by the St. George Bank system.) For COMPLETIONs, the ORIGINALTXNREF or PREAUTHNUMBER can be used, but not both.

3.2.9 AUTHORISATIONNUMBER / AUTHCODE / AUTHORISATIONCODE

The AUTHORISATIONNUMBER is used to finalise PREAUTH transactions (COMPLETION). This parameter needs to be set to the AUTHCODE returned by the PREAUTH transaction to which the COMPLETION relates. Only one value should be used, not all three.

3.2.10 CLIENTREF

The CLIENTREF field can be used by merchants for storing their own reference number. Its use is optional, but is primarily used for transaction reconciliation purposes. This field is alphanumeric and has a maximum size of 50 characters.

3.2.11 COMMENT

Merchants can use this optional alphanumeric field for storing additional data to help with transaction reconciliation. For example card holder's name, description of purchased item and other information. The maximum length of this field is 255 characters.

3.2.12 TERMINALTYPE

The TERMINALTYPE is an optional field used to specify the terminal type or transactional indicator used to identify the transaction source. The **default** terminal type may be set through the System Administration Console; however, this default value may be overridden for each individual transaction by stipulating an alternate value through this field.

The TERMINALTYPE value is a single-digit numeric field with a value between 0 and 5.

Permissible TERMINALTYPE values are:

0 = Internet (eCommerce)

1 = Telephone Order

2 = Mail Order

3 = Customer Present

4 = Recurring Payment

5 = Instalment

3.2.13 CVC2

CVC2 is an optional 3-6-digit alphanumeric value is used as secondary card validation for e-commerce transactions. The value is generally defined as the suffix value printed on the back of all credit cards. St. George Bank currently only validates this value for Visa and Mastercard. The value may be supplied for other credit cards, but will not affect the success of a transaction.

3.3 Credit Card Transaction Response

The matrix below shows the parameters returned, and the transaction types to which they apply. These parameters may need to be conveyed back to the merchant application or cardholder to advise to success or failure of the transaction request. A full description of the Response fields is also included below.

Response Field/ Transaction Type	PURCHASE	REFUND	PREAUTH	COMPLETION	STATUS
RESPONSECODE	✓	✓	✓	✓	✓
RESPONSETEXT	✓	✓	✓	✓	✓
ERROR	✓	✓	✓	✓	✓
ERROR_DETAIL	✓	✓	✓	✓	✓
TXNREFERENCE	✓	✓	✓	✓	✓
AUTHCODE	✓	✓	✓	✓	✓
STAN	✓	✓	✓	✓	✓
SETTLEMENTDATE	✓	✓	✓	✓	✓

✓ – Always Available, * – Optional, X – Not Available

3.3.1 RESPONSECODE

The RESPONSECODE signifies whether the transaction was approved, declined, or failed. This field must be displayed to the customer, together with the RESPONSETEXT. Appendix A contains a full listing of Response Codes.

NOTE: Using the **Test** environment the response code may be varied by selecting a different cents value in the TOTALAMOUNT field of the transaction request. It is essential that you test for all responses and handle each appropriately within the merchant application.

3.3.2 RESPONSETEXT

The RESPONSETEXT corresponds directly to the RESPONSECODE and is a brief text description of the response.

3.3.3 ERROR / ERROR_DETAIL

The ERROR / ERROR_DETAIL response contains a brief error message for diagnostic purposes (debug information). These fields can optionally be displayed to the customer.

3.3.4 TXNREFERENCE

The TXNREFERENCE is the unique St. George Bank Transaction Reference assigned to the transaction. Every transaction accepted for processing by the St. George Bank System is assigned a unique Transaction Reference.

3.3.5 AUTHCODE

The AUTHCODE is an identifier assigned to the transaction by the financial institution. The AUTHCODE returned must be included in the transaction request as AUTHORISATIONNUMBER when performing a COMPLETION transaction.

3.3.6 STAN

The STAN is an identifier assigned to the transaction by the financial institution.

3.3.7 SETTLEMENTDATE

SETTLEMENTDATE is the settlement date of the transaction. i.e. the date the actual funds transfer will occur. Depending on whether your merchant account is with St. George Bank or not, you should receive settlement within 1-2 business days, in rare cases within 3-4 business days.

3.4 Determining the Outcome of a Creditcard Transaction

The flowchart below shows the process involved in determining the outcome of a transaction.

1. Execute() returns “true”

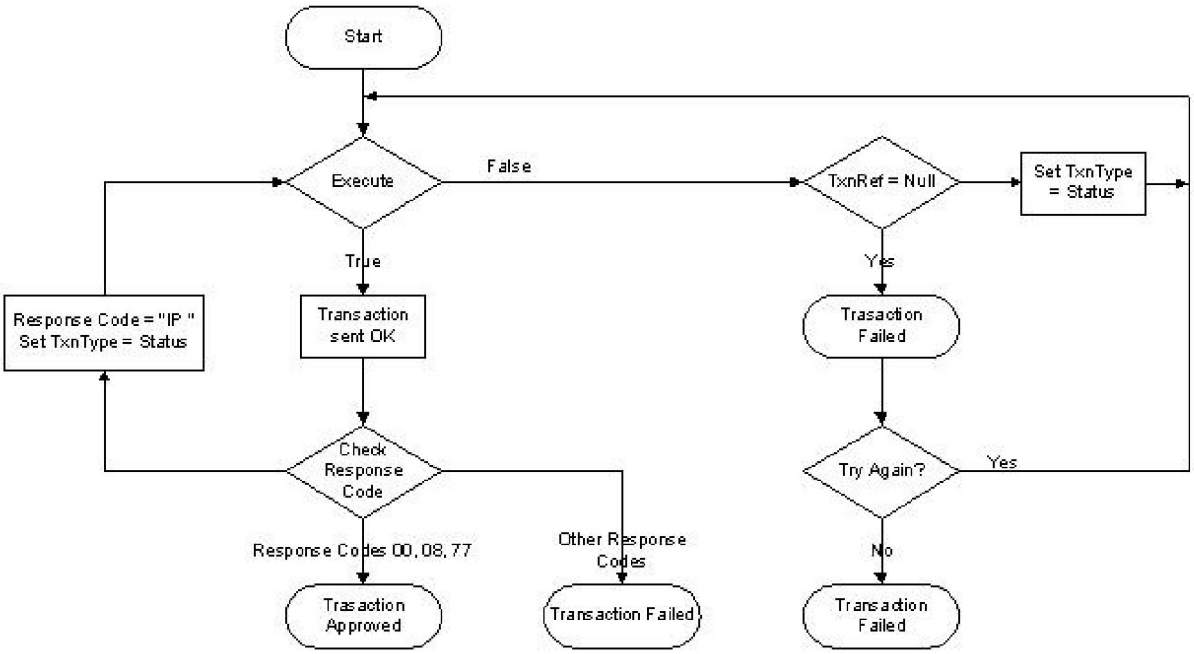
This means that the API has communicated with the gateway successfully. The RESPONSECODE field determines the outcome of the transaction and which action is to be taken.

2. Execute() returns “false”

This means there was a problem during communication with the gateway.

- If no transaction reference has been obtained, the transaction can safely be reprocessed.
- If a transaction reference has been obtained, its status can be determined and appropriate action can be taken.

Diagram 3: Determining the outcome of a transaction



Please note: This diagram is based on the C++ API. The execute() method of the Java API for example does not return a value, but uses exceptions to deal with failures. The principle however stays the same.

A more detailed discussion on how to determine the outcome of a transaction can be found in chapter 4.

4 API Implementation Guidelines

This chapter contains key points that should be followed during API implementation and some advice on how to deal with failures.

4.1 "Test" and "Live" Merchant Status

St. George Bank provides two environments for merchant use:

LIVE environment:

- Connects directly to the live banking processing system and performs real-time transactions.

TEST environment:

- A simulated environment controlled within the transaction server that effectively mimics the live processing environment.
- In the Test environment, the cents amount of the transaction determines the response code. For example:

  A transaction of $10.00 returns '00' – APPROVED.

  A transaction of $10.51 returns '51' – INSUFFICIENT FUNDS.
- Exception: A cents amount of .88 returns "IP – IN PROGRESS", not "88 – SYSTEM ERROR"

The St. George Bank Servers produce a wide range of transactional responses. To fully test your system's ability to handle and control varying responses, you must test all cents values between 00 and 99.

A complete list of all creditcard response codes is contained in Appendix A.

Initially, you will develop and test your code using the Test system. When you believe you are ready to "Go Live", you must contact the St. George Bank Support Team to be migrated to the Live system.

NOTE: When changing from "Test" to "Live", you will need to change the **port number** that the API is talking to. The port numbers for the Live and Test system will be sent to you along with other St George Bank specific account information.

The Test system will still accessible after you have been migrated to live.

4.2 Transaction protocol

The diagram below shows a transaction from beginning to end (= single call of the "execute" method).

1. The first step is an initialisation step.

If this is successful the API obtains a reference number (TXNREFERENCE), which is then used in all communication with the IPG. This appears in the debug log as

Preparing Message...
Message Sent...
Reading Response...

The IPG logs the transaction with status "IN" (initialised) at this stage.

Once a TXNREFERENCE has been received, the transaction is traceable through the system and its status can be obtained from the gateway by doing a STATUS transaction.

2. The second step is the confirmation step.

The API passes the reference number to the IPG and the transaction is then processed.

The IPG updates the status of the transaction to "IP" (in progress) and then to whatever is returned by the financial network or the cardholder's bank.

This appears in the debug log as

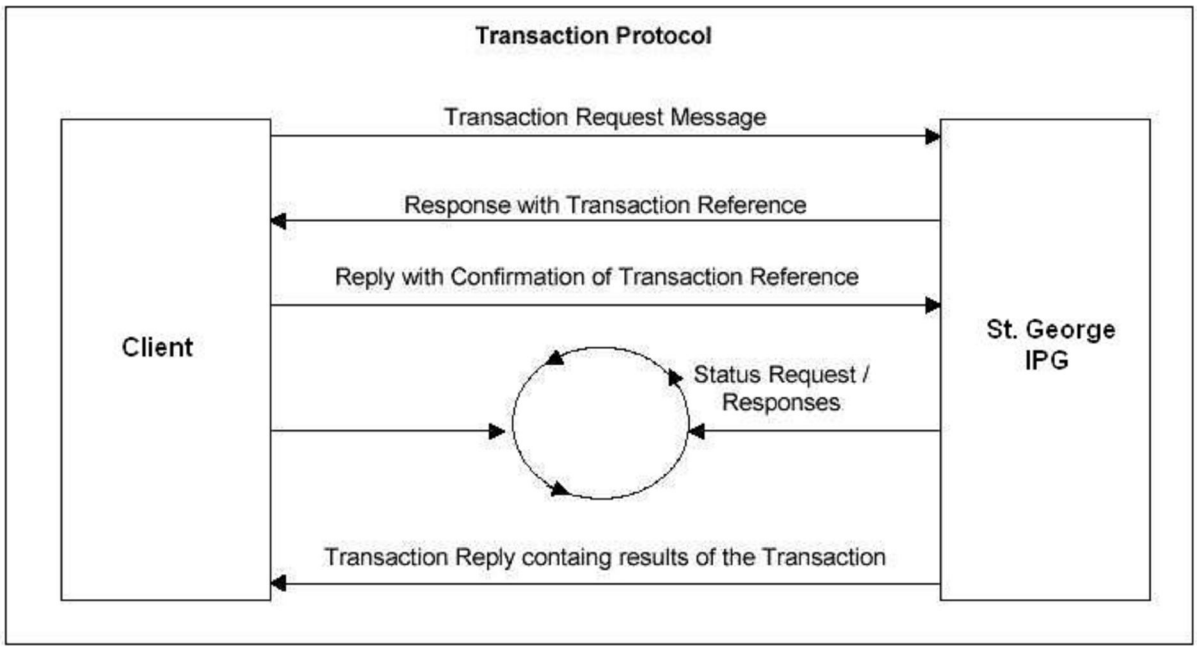
Replying to Server Response...
Reading Final Response...

3. The third step is the return of the transaction results.

This appears in the debug log as

Transaction Execution Finished...

Diagram 4: The protocol implemented between Gateway and API



Please Note: For the Java API, the “debug log” feature is not available. It is recommended to use the JVM logging mechanisms instead.

The API internally retries to communicate with the IPG three times before returning false from the execute command.

As a lot of applications that use the API are time sensitive, the delay between internal retries is quite short so that control can be passed back to the calling application. The calling application then has the capability to retry (as described below) or abort the transaction based on the specific applications business logic.

So in the event of a communication error the first fallback is for the API to internally retry the transaction. The second fallback is to pass control back to the calling application and allow it to decide how many times to retry and how long to delay between attempts.

It is also up to the controlling application to handle things such as network failure and expired SSL certificates.

4.3 Transaction outcomes

The outcome of the transaction is returned in the **RESPONSECODE** field and falls into 3 categories:

1. Approved

Transaction has gone through ok = Codes **00**, **08**, **77**

2. Declined

Card holder's bank has rejected the transaction = All other codes in the range 00-99

3. Error / In Progress

- System/validation errors, connection failures, etc.: Alphanumeric response codes such as "0F", "0A"," –1", etc.
- Initialised / In progress: Response codes "IN", "IP"

Please note: If a transaction returns "IP IN PROGRESS", the gateway should be polled (TRANSACTIONTYPE = STATUS) for at least 1 minute before moving on to the next transaction. This is to avoid transactions being logged as "IP" prematurely. Transactions returning "IN INITIALISED" can safely be reprocessed.

Response codes in the range from 00-99 have been returned by the card holder's bank – all others come from the gateway server or the API itself. A full list of response codes can be found in appendix A.

Please Note: Do not to use the RESULT field or the return value of the execute() function to determine whether the transaction has been approved or declined.

4.4 Dealing with certain types of failures

4.4.1 Recovering from "hanging" transactions and timeouts

Due to the nature of the Internet, connections problems can occur. Fortunately these communication issues are usually fleeting and can be handled programmatically. Examples of such failures are:

- Execute() returns false after a failed attempt to communicate with the gateway
- Error "WP_SOCKET_TIMEOUT_EXCEPTION" is caused by a sluggish communication channel, and is typical of transactions performed across the Internet.
- Occasionally, TCP/IP packets are lost and the API hangs waiting for a response until it eventually times out.

Recovery steps to handle such types of failures:

1. Check for an existing TXNREFERENCE. To counter difficulties with hanging transactions, call execute() in a separate thread and use a reference to the transaction bundle.
2. If no TXNREFERENCE exists, you can be sure that the transaction was not initialised and hence the transaction can be retried. In case of a hanging transaction, the thread can be killed.
3. If a TXNREFERENCE exists, then execute a "STATUS" transaction" (Failure occurred after transaction initialisation and before transaction completion). This status request will provide you with the appropriate information about the state of the transaction.
4. If the Response Code is "IP" this indicates that the transaction is "In Progress" and has been sent out to the financial network, you will need to continue to poll the IPG with status requests until this is resolved.

Following the above steps will normally resolve any failed transactions. In the event of the above failing, it is recommended to wait for a short time and then to repeat the process. After 3 attempts, it is likely that there is a more serious communication problem and the error message should be logged with the TXNREFERENCE (if available) so that it can be checked at a later time.

4.4.2 Avoiding duplicate transactions

1. User errors

To prevent website users from submitting a transaction twice (eg. by clicking "submit" twice, or by hitting the "back" button on the browser, etc.) the following is recommended:

- Use a unique transaction reference as session id
- Let the payment page expire
- If API hangs, kill session (eg after 15 seconds) and display an appropriate message.

2. Reprocessing transactions

Caution should be exerted when reprocessing transactions. It is recommended to double-check the outcome of the transaction by performing a "STATUS" transaction or to manually check the outcome in the Merchant Admin Console prior to resubmitting a transaction.

4.5 Concurrency

When implementing the St. George API, it is important to consider the effect of multiple users accessing your application concurrently (eg. from a web site). A new transaction context must be created for every transaction, to provide a mechanism for maintaining a user's session, which will allow each transaction instance to be processed only once.

4.6 Performance

As a rough guide, the average transaction time on a machine built as specified in the Developer Guide for the selected API is 6-8 seconds. There are many factors causing variances in performance:

- The amount of RAM on the machine running the API needs to be adjusted to accommodate all applications, virtual directories etc. on that machine. A significantly higher amount than what is recommended in the API developer guide might be needed. Keep in mind that additional applications on that machine can affect performance or interfere with the API. It is therefore recommended to install the API on a dedicated machine.
- The C++ API is the fastest, Java the slowest. When using the Java API, keep in mind that Sun's JSSE extensions are fairly slow to initialise. There is a one-time overhead of approx. 13 seconds, subsequent transactions will go through faster. All other APIs are based on the DLL/shared object used by the C++ API, so their performance does not vary much.
- Running concurrent transactions (several instances of the webpay DLL/webpay class running at the same time) will increase memory usage and CPU load, which your machine will have to accommodate.
- The type/speed of your connection and general Internet congestion also have an effect on performance.
- Bandwidth management products (eg. Netguard, Packeteer), packet shapers, and the setting of the MTU size can have a negative impact on performance, if not configured properly.
- Other parts of the infrastructure like Firewalls, Routers, ISPs, and the Network Interface Card, may cause performance variances as well.
- Having additional terminal IDs assigned can improve transaction throughput if there is sufficient transaction volume.

5 American Express & Diners cards

To be able to process Amex/Diners cards, you need to do the following:

1. Contact Amex/Diners and apply for a Merchant ID on their systems.
2. Once Amex/Diners have set this up, their cards can be processed by our gateway.
3. Email your Amex/Diners ID to ebssupp@banksa.com.au for our records.

6 Getting Help

6.1 IPG Web Site

The complete API documentation and software is available from **https://www.ipg.stgeorge.com.au**.

6.2 Contacting St. George Bank

The St. George Bank Technical Support Team can be contacted in the following ways:

- Telephone via the St. George EFTPOS Helpdesk 1300 650 977
- Email to: ipgsupport@stgeorge.com.au
- On-Line at https://www.ipg.stgeorge.com.au

6.3 Other resources

For more specific information about an individual API, check the corresponding API Developers Guide.

Appendix A – Response Codes

The complete list of response codes is now available at

https://www.ipg.stgeorge.com.au/download.asp

Appendix B – Test Credit Card Numbers

The Test System accepts only the following Credit Card numbers as valid card numbers. If used on the Live system they will be rejected as invalid cards. Valid expiry dates (must be in the future) also apply with these card numbers.

Do not use your own credit card numbers within the Test System, as they will reject. Do not include the spaces shown in the examples they are merely used for readability.

Card Type	Card Number
MasterCard	5431 1111 1111 1111
Visa	4111 1111 1111 1111