Internet Payment Gateway

Linux API Developer Guide

st.george

Table of Contents

Disclaimer

Confidentiality

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 the St. George Bank Linux API and delivers valuable insights into more user-specific deployments.

It is assumed that the parties implementing this API will be experienced in implementing web applications in a Linux development environment, and this document is written principally toward parties with this expertise.

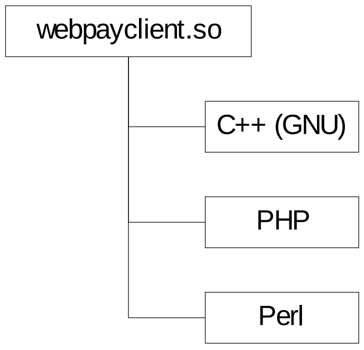
Related Documents

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

1 Introduction

The St. George Bank Linux API is a shared object containing methods for use with the *St. George Bank Internet Payment Gateway*. The API enables web site and application developers to efficiently add credit card transaction processing capabilities to their products.

The St. George Bank Linux API can be used in Linux-based applications in many programming environments:



This document should be used in conjunction with the St. George Bank *Generic API Developer Guide* document, which includes details on parameters required for credit card transactions, implementation guidelines and a list of response codes.

This document also includes common configuration settings, test applications and protocols, and sample code that will assist developers in incorporating the API into their chosen application.

Communication between the St. George Bank API and the St. George Bank Transaction Server Gateway uses SSL connectivity through server addresses and defined ports.

NOTE: If your application is located behind a firewall or proxy server, you should ensure that the relevant addresses and ports are not being filtered or blocked.

It is important to note that the St. George Bank Transaction Server utilises two distinct transaction-processing environments. These environments are defined as **LIVE** and **TEST**. The **LIVE** environment connects directly to the live banking processing system and performs real-time transactions. The **TEST** environment is a simulated environment controlled within the transaction server that effectively mimics the live processing environment; **TEST** is principally used for integration testing and training.

Using the parameters provided by your St. George Bank representative you must define and test your application and API integration against the **TEST** environment to ensure compliance.

1.1 Minimum System Requirements

The following minimum system configuration is required for implementation of the St. George Bank Linux API.

Operating System	**Redhat Enterprise Linux 4 (kernel 2.6.9-22.0.1.EL)** with glibc 2.3.4 (**With FULL ADMIN Access**) **– Important: Must be 32bit**
Processor	Pentium 4
Processor Speed	1.5Ghz
Memory	512Mb
Additional Requirements	Open SSL 0.9.7a **/bin/sh** (rpm package ash 0.3.7-2) C++ compiler like GCC **make** development utility

Please note:

- The Linux C++ API has been certified for a machine built exactly as above and is not supported for any other configurations.
- It is assumed that the parties implementing this API will meet the above requirements and be experienced in implementing web applications in a Linux C++ development environment, and this document is written principally toward parties with this expertise.

1.1.1 OpenSSL libraries

All St. George Bank API depends on the OpenSSL libraries. These libraries should be OpenSSL version 0.9.7a or greater, and must be correctly installed on your machine before the Linux Webpay client software will execute successfully.

These files will be installed as part of a default RHEL install.

The OpenSSL libraries are named **libssl.so.2** and **libcrypto.so.2**. They may be installed as part of the operating system and usually reside in the **/lib** folder. If your system lacks these files, please download pre-compiled versions from a trusted site or download the OpenSSL source and compile it on your machine.

1.1.2 OpenSSL libraries

A C++ compiler is required to compile the Webpay C++ BaseTest object.

GCC is installed with the optional Development Tools package of RHEL.

The latest version of gcc can also be obtained from http://gcc.gnu.org

2 Installation

The St. George Bank Linux API has been thoroughly tried and tested under **Redhat Enterprise Linux 4 (kernel 2.6.9-22.0.1.EL).** The API will also work under other versions of Linux; however, these are **not supported** and consequently not recommended.

The installation is delivered as a gzip tarball, named **webpayC++-2.0.tar.gz**, which contains all the necessary components including sample source code, documentation, and supporting libraries.

- Please check that your system meets the minimum system requirements listed in section 1.1.
- Before commencing installation, ensure that you have appropriate write permissions to the **/usr/lib** directory
- You may be required to be logged in as root to perform this install
- All commands are case-sensitive

Follow these steps to install the API:

1. Copy the **webpayC++-2.0.tar.gz** file to your workspace
2. In your workspace, decompress the file by typing: `tar –xzvf webpayC++-2.0.tar.gz`
3. The file will now decompress all necessary files including the **Makefile** which is used to compile the base test and to install the library object into the /usr/lib directory. For a full description of all files included in the install, refer to Appendix A.
4. To install the Webpay library, type: `make install`
5. To compile the base test, type: `make`

The install is now complete.

2.1 St. George Bank Specific Information and Files

Your St. George Bank representative will have supplied you with information and files specific to the St. George Bank Gateway you will be using, including:

- St. George Bank Gateway address (www.gwipg.stgeorge.com.au)
- Gateway port numbers: Test (3007) and Live (3006)

  NOTE: If your application is located behind a firewall or proxy server, you should ensure that the relevant addresses and ports are not being filtered or blocked
- Your St. George Bank Client ID
- A St. George Bank client certificate file (e.g., “client.cert”) appropriate for your St. George Bank gateway and the password protecting this certificate file

You must place the St. George Bank client certificate file somewhere on your machine so that the St. George Bank client software can access it. It is recommended that you place the file in the same folder with all your other St. George Bank client files.

2.2 The St. George Base Test Program

Extracting the file archive will install a test application (**baseTest**) into the specified folder. It can be run by using the shell script: **run_base_test.sh**.

You should run this test program to ensure that:

- The OpenSSL libraries are correctly installed
- The basic St. George Bank Client library has been installed
- The necessary ports in your firewall are opened to allow communications
- There are no other basic connectivity problems with your machine

Supplying Input to the Test Program

Edit the **run_base_test.sh** file using any standard text editor and replace all the command line arguments to the test program with the data specific to your setup and St. George Bank gateway. The test program takes seven possible command line arguments:

baseTest <gateway_address> <gateway_port> <client_id> <certificate_file> <certificate_password> <trusted_certificates_file> <number_of_transactions>

Argument	Description
<gateway_address>	The address of your St. George Bank gateway, "www.gwipg.stgeorge.com.au". Multiple gateways may be separated by commas.
<gateway_port>	The port number provided to you. Ensure that you specify the correct port number for either the **test (3007)** or **live (3006)** environment.
<client_id>	The Client ID issued to you (1000…). Note: If you need to separate different business entities or income streams, your St. George Bank representative can set up multiple Client Ids under your IPG Customer ID (5000….) for you.
<certificate_file>	The name of your St. George Bank client certificate file, e.g. "/home/StGeorge/client.cert". You can use the full path or a relative path, as long as the name resolves correctly.
<certificate_password>	The password protecting your St. George Bank client certificate file.
<trusted_certificates_file>	The name of a certificate file containing certificates for all the certificate authorities to be trusted. If left blank (""), all certificate authorities are trusted.
<number_of_transactions>	The number of transactions to submit to the St. George Bank gateway before terminating the test.
<debug_flag>	Used for testing purposes only. Detailed information is written to the file ./webpay.log. Values: debug (turns debugging on); any other value turns debugging off.

The **run_base_test.sh** file should look something like this:

```
baseTest " www.gwipg.stgeorge.com.au " "3007" "10000001" "client.cert" "<password>" "" 5
```

Running the Test Program

Run the test program by typing **`./run_base_test.sh`** at the command prompt.

NOTE: The test may take some time to establish the first SSL connection but all subsequent transactions will occur within seconds.

The output of the test program should include the some of the following responses …

- Response Code = "00"
- Response Text = "Approved (TEST TRANSACTION ONLY)"
- Error Message = "(null)"
- Transaction Reference = 0301000000123469

If the output of the test program contains the word "APPROVED", the installation of the API has been successful.

The following gives a brief overview of the fields involved:

Field	Explanation
RESPONSECODE	A two digit code used to determine the outcome of the transaction. The code number matches the RESPONSETEXT field. "**00**", "**08**" or "**77**" designate an **approved** transaction. A complete list of response can be found in the IPG Response Code Table.
RESPONSETEXT	A meaningful text message that explains the response code
ERROR	Any error additional messages received from the gateway.
TXNREFERENCE	A unique reference number generated for each transaction.

NOTE: Please follow the guidelines in the Generic API Developer Guide on how to determine the outcome of a transaction.

2.3 Troubleshooting

If your output does not look as expected, please review the steps you have taken so far. The following questions may assist in determining common API problems:

2.4.1 Installation

- Are the OpenSSL libraries in the expected location (/lib) and symbolically linked?
- Are you using OpenSSL version 0.9.7 or higher?
- Is the basic St. George Bank client library (libwebpayclient.so) in the expected location (/usr/lib)?
- Do you have appropriate permissions ? Make sure that you have read access to all critical files (libraries etc.) Also, make sure that you have write permissions for your workspace and for the directory the log file is in.

2.4.2 SSL/Security

If you encounter the error "UNABLE TO INITIALISE SSL" or similar, please check the following:

- Do you have the correct certificate file?
- Are you using the correct certificate pass phrase ?
- Path to certificate file: Have you specified the full path, eg. /home/stg/cert.cert
- Trusted certificate file: leave this parameter blank
- Do you have read access to the cert file ?

2.4.3 Connectivity

- Do you have connectivity to the gateway? Try the command "`telnet www.gwipg.stgeorge.com.au 3006`". If it lets you connect, you can talk to the gateway.
- Are you behind a firewall? You may need to contact your Security Administrator to ensure the API ports are open for traffic in both directions.
- Are you using any packet shapers or bandwidth management products? If not configured properly, these might affect connectivity to the gateway.
- Commercial and freeware tools (eg. Packet Sniffers) can be found on the Internet, to assist you in testing your network connectivity to the St. George Bank the Secure Transaction Servers.

2.4.4 System

- Are you using the latest drivers for you network interface card?
- Are your TCP/IP libraries up to date?
- Have you applied the latest patches to your Operating System?
- Do you have enough RAM on your machine to accommodate for concurrency and/or other applications running on that machine?
- Do you have any applications running that might interfere with the API?

2.4.5 Parameter values / Transaction data

- Is the right location of the St. George Bank certificate file specified? You may need to specify the file's full path.
- Is the password correct for the St. George Bank client certificate?
- Is the correct address and test port number specified for the St. George Bank gateway? You may want to use "nslookup" to ensure that the address resolves correctly. If it does resolve correctly, the IP address of the gateway will be displayed on the screen. If it does not, you will get an "unknown host" message.
- Are you using the correct Client ID?
- Are you sending all the required fields for a transaction? Refer to the generic developer guide for required fields.
- Do all the fields in the transaction bundle contain valid data? Refer to the generic developer guide for field restrictions.

2.4.6 Response codes

- In some cases, the system may return errors when you are expecting an approved or other code during testing. Please check the response code you receive against those contained in the Generic API Developer Guide.
- In the Test environment, the response code is determined by the cents amount, eg. $1.51 = "51 INSUFFICIENT FUNDS"
- Some response codes may be generated at any time and this behaviour is perfectly normal, for example, the system may return the code 'IP' for 'In progress' if the transaction is still being processed.
-

2.4.7 Logging

- To facilitate debugging, it is recommended to log the values of all fields in the transaction bundle prior to executing the transaction. The transaction response (including any errors that might occur) should be logged as well. Refer to chapter 4.3 for instructions on how to enable API logging.

If you are still unable to identify the problem, call your St. George Bank representative. Refer to chapter 6, "Getting Help" for contact details.

3 API Development/Integration

3.1 Implementation Guidelines

- Please carefully read the **Generic API Developer Guide** document before implementing your solution. It contains useful instructions on how to avoid common mistakes when implementing the API.

3.2 Directions for compiling and linking

Here are directions for building the test program in certain development environments. If you use a development environment not listed below, please refer to its documentation for instructions on how to build programs.

- For GNU C++ …
  - Place the shared object file in the appropriate folder (e.g., /usr/lib)
  - Ensure that parameter LINUX is defined for the pre-compiler (-DLINUX or #define LINUX).
  - Build the test application with a command similar to the following…

    “gcc -DLINUX BaseTest.cpp /usr/lib/webpayclient.so -o baseTest”

  See the example **Makefile** for further direction.

If you are building a St. George Bank client component that directly relies upon the basic native St. George Bank client library, you can build your component by following the directions given above. Merely replace the “BaseTest.cpp” file with all your source files and place all your header files in an appropriate folder (such as the folder containing the “webpayclient.h” file).

The latest version of gcc can be obtained from http://gcc.gnu.org

4 Linux C++ API Reference

This section contains reference material, functions, methods and source code used to implement the Linux API

For information on Credit Card transactions and parameters refer to the separate *Generic API Developer Guide.*

4.1 Available Functions/Methods

init_client

Synopsis

```
BOOL init_client ( )
```

Parameters

There are no parameters for this method.

Return Type

This method returns true when the client is successfully initialised.

Usage

Initialise the St. George Bank client. This method can be called explicitly or it will be called automatically when other methods are invoked.

free_client

Synopsis

```
BOOL free_client ( )
```

Parameters

There are no parameters for this method.

Return Type

This method returns true when all system resources allocated to the client are cleared.

Usage

Free the resources of the St. George Bank client. This method should be called when use of the St. George Bank client is complete to ensure that all system resources are freed.

cleanup

Synopsis

```
void cleanup ( void * txn )
```

Parameters

The only parameter required for the cleanup method is the transaction context.

Return Type

The cleanup method returns void (no return value).

Usage

The cleanup method should be called at the end of a transaction in order to de-allocate memory and other resources used by St. George Bank Linux, to complete the transaction.

execute

Synopsis

```
BOOL execute ( void * txn )
```

Parameters

The only parameter needed by the execute method is a void pointer to the transaction context.

Return Type

Execute will return true if the transaction was processed successfully. This does not imply approval of the transaction. Execute returns false if the transaction request is incomplete for any reason.

Usage

Ordinarily, execute is called after all values have been set, and should only be called once per transaction. If execute fails, the appliation may call execute again, depending on the rules outlined in the section ‘Determining the Outcome of a Transaction’

flushBundle

Synopsis

```
void * flushBundle ( void * txn )
```

Parameters

The flushBundle requires only the transaction context.

Return Type

This method returns a cleared bundle that may be different from the original.

Usage

This method is used to re-create a context/bundle. All values, including server address, port, certificate password and path, and the licence key will need to be reset after calling this method.

get

Synopsis

```
char * get ( void * txn, LPCTSTR name )
```

Parameters

The get method requires the transaction context and a string containing the name of the value to be returned.

Return Type

The get method returns a character string containing the value requested.

Usage

The get method can be used at any time to return values from the transaction context. It is typically used after the execute method has been called. See the section ‘Transaction Responses’ for a full listing of values available for the transaction type being used.

newBundle

Synopsis

```
void * newBundle ( void );
```

Parameters

newBundle has no required parameters.

Return Type

A void pointer is returned to the application.

Usage

newBundle is used to create a new transaction context. The application must maintain reference to this pointer, and should not attempt to modify it in any way, except through the libraries methods. This method should be the first library method called by the application, as the transaction context or bundle is required for nearly all of the libraries methods.

put

Synopsis

```
void put ( void * txn, LPCTSTR name, LPCTSTR value )
```

Parameters

The put method requires the transaction context, a string containing the name of the parameter to be set, and a string containing the value of the parameter.

Return Type

There is no return value for this method.

Usage

This method sets values in the transaction context. For a full list of the parameters that can be set, see the ‘Transaction Requests’ section above.

put_ClientID

Synopsis

```
void put_ClientID ( void * txn, LPCTSTR newVal )
```

Parameters

The put_ClientID method requires the transaction context and a string containing the ClientID.

Return Type

There is no return value for this method.

Usage

This method sets the ClientID in the transaction context/bundle.

put_CertificatePassword

Synopsis

```
void put_CertificatePassword ( void * txn, LPCTSTR newVal )
```

Parameters

The put_CertificatePassword method requires the transaction context and a string containing the Certificate Password.

Return Type

There is no return value for this method.

Usage

This method sets password for the Certificate in the transaction context/bundle.

put_CertificatePath

Synopsis

```
void put_CertificatePath ( void * txn, LPCTSTR newVal )
```

Parameters

The put_CertificatePath method requires the transaction context and a string containing the Certificate Path.

Return Type

There is no return value for this method.

Usage

This method sets path and location for the Certificate in the transaction context/bundle.

setPort

Synopsis

```
void setPort ( void * txn, LPCTSTR Port )
```

Parameters

The setPort method requires the transaction context and a string containing the server port.

Return Type

There is no return value for this method.

Usage

This method sets Port through which the Servers communicate in the transaction context/bundle.

setServers

Synopsis

```
void setServers ( void * txn, LPCTSTR ServerList )
```

Parameters

This method takes the transaction context as a void pointer, a string containing a comma separated list of server names or IP addresses, and a string containing the TCP Port number to connect on.

Return Type

This method has no return type.

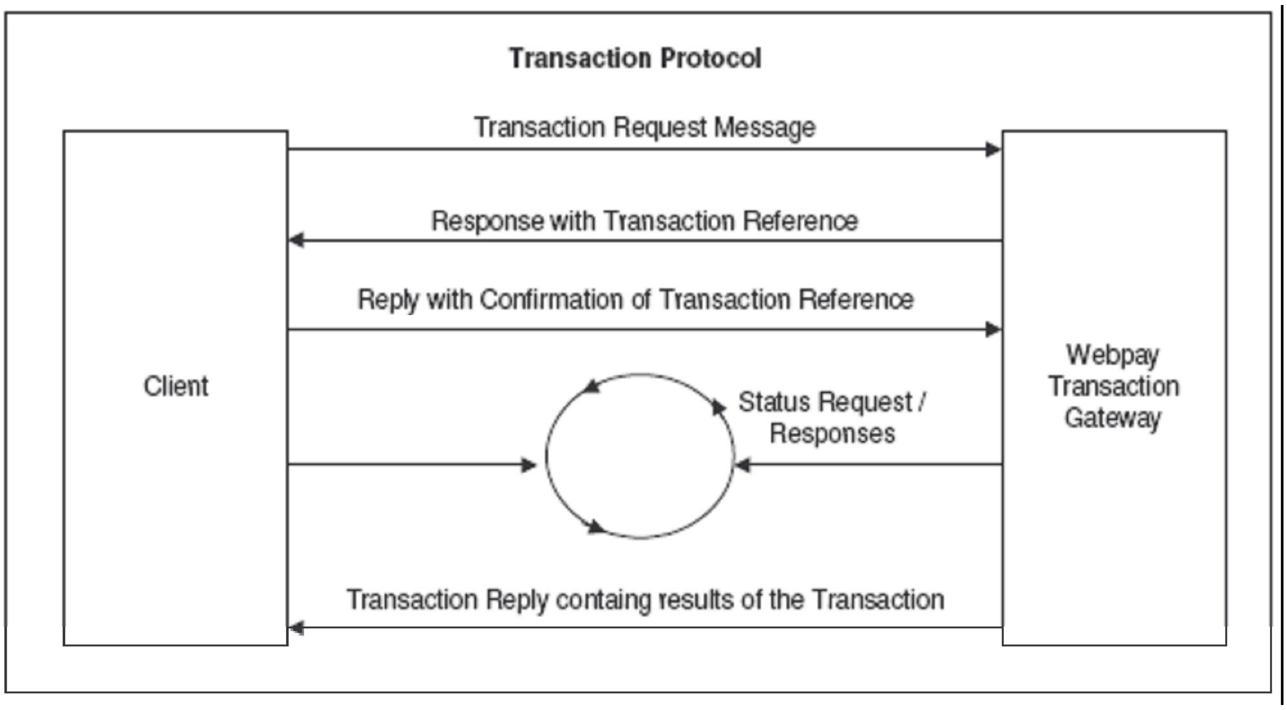
Usage

This method must always be used to set the names or IP addresses and the port number of the servers with which to communicate. Multiple servers may be specified, but they may only use the one port.

4.2 Error/Exception Handling

The St. George Bank Linux API handles all errors internally. Any errors encountered during processing, cause the transaction to fail. Error messages are viewed by investigating the ERROR field.

If a transaction attempt fails and a Transaction Reference has been returned by the St. George Bank Servers, the execute method will automatically query the status up to three more times. Beyond this, execute fails, and the transaction reference is available to the application. The merchant's application should be enabled to query the status of such a transaction, and be able to do so at any time.


Transaction Protocol
Transaction Request Message
Response with Transaction Reference
Reply with Confirmation of Transaction Reference
Client
Status Request / Responses
Webpay Transaction Gateway
Transaction Reply containg results of the Transaction

5 Sample Application

5.1 Source Code

The sample included here is C++ code for the most basic **creditcard** implementation possible and contains hard-coded values. It simply displays the responses to the user. It is intended only to demonstrate the basic concepts involved in using the St. George Bank Linux API. Your production application will need to collect user or other input, and log the transaction responses to a database and/or display them to users. By using this sample code you should be able to extrapolate the appropriate parameters required for more complex implementations.

```
#include "webpayclient.h"
#include "BTypes.h"

#define THE_CLIENT_TYPE "webpayC++"
#define THE_VERSION "2.0"

// prototypes
void sendTransaction( int argc, char *argv[] );
boolean doStatusCheck(void * bundle);
boolean approvedTransaction(char * responseCode);
void displayResults(void * bundle);

int main( int argc, char *argv[] )
{
	int status = 0;
	if( argc < 8 ) {
		printf("\r\nNative Webpay Client Test\r\n");
		printf("========================\r\n");
		printf("usage: BaseTest.exe <serverlist> <port> <clientid> <cert path> ");
		printf("<certpass> <ca_file> <num_iters> {\"debug\"or\"no debug\"}>\r\n");
		status = 1;
	} else {
		// send a repeated number of transactions
		int i, numIters = atoi( argv[7] );
		for ( i = 0; i < numIters; i++ ) {
			sendTransaction( argc, argv );
		}

		// free the webpay client resources
		free_client( );
	}

	return 0;
}

// send a transaction given the specified parameters
void sendTransaction( int argc, char *argv[] )
{
	// Create a new context/bundle
	// A void pointer is used here as this application only needs to hold a reference,
	// and does not need to know any of it's specifics.
	void *bundle = newBundle();

	//Add the client type and the version to the bundle
	put(bundle, "CLIENTTYPE", THE_CLIENT_TYPE);
	put(bundle, "VERSION", THE_VERSION);
	printf( "Arguments: %d\r\n", argc );
	if ( argc == 9 ) {
		printf( "Debug set to: [%s]\r\n", argv[8] );
		if ( stricmp( argv[8], "debug" ) == 0 ) {
			printf( "Debug ON\r\n" );
			put ( bundle, "DEBUG", "ON" );
			put ( bundle, "LOGFILE", "webpay.log" );
		} else {
			printf( "Debug OFF\r\n" );
			put ( bundle, "DEBUG", "OFF" );
		}
	} else {
		printf( "Debug OFF\r\n" );
```

```
                put ( bundle, "DEBUG", "OFF" );
        }

        // Set security related parameters
        put_CertificatePath( bundle, argv[4] );
        put_CertificatePassword( bundle, argv[5] );
        put( bundle, "_CAFILE", argv[6] );

        // Set the server address and port number
        setServers ( bundle, argv[1] );
        setPort ( bundle, argv[2] );

        // Set the transaction's parameters.
        // These vary between transaction types and are subject
        // to change with notice, as new types are added.
        put ( bundle, "CLIENTID", argv[3] );
        put ( bundle, "CARDDATA", "4564456445644564" );
        put ( bundle, "CARDEXPIRYDATE", "0215" );
        put ( bundle, "DATA", "Example Transaction" );

        put ( bundle, "INTERFACE", "CREDITCARD" );
        put ( bundle, "TRANSACTIONTYPE", "PURCHASE" );

        put ( bundle, "TOTALAMOUNT", "10.00" );
        put ( bundle, "TAXAMOUNT", "1.00" );

        // Attempt to execute the transaction request...
        if ( execute ( bundle ) ) {
                // If the execute method returns successfully this indicates
                // that communication with the Payment Gateway has been successful.
                // A further test of the Response Code and Response Text will be
                // required to determine if the Payment has been successfully
                // Authorised. Please see the developers guide for more details.

                printf ( "Successfully communicated with the WTS.\r\n" );
        } else {
                // There has been a problem during the execute call.

                // Log message.
                printf ( "Unable to communicate with the WTS.\r\n" );

                //Try transaction recovery
                char *transactionRef = get( bundle, "TXNREFERENCE" );
                if (transactionRef) {
                        //We have a transaction reference so attempt a status transaction.
                        printf("Doing status check with TransRef = [%s]\n", transactionRef);
                        if(doStatusCheck(bundle)) {
                                printf ( "Status Check OK: Details are displayed below.\r\n" );
                        } else {
                                        printf ( "Status check failed: Unknown transaction
status.\nPlease wait a short while and try status check again using Transaction Ref
[%s].\r\n", transactionRef );
                        }
                } else {
                        // There is no transaction reference number so the transaction
                        // has failed completely. It can be safely reprocessed.
                        printf("The transaction can be safely reprocessed as no ");
                        printf("Transaction Reference Number exists.\n");
                }
        }

        // Get the responses and display them...
        displayResults(bundle);

        cleanup ( bundle );
}
```

```
boolean doStatusCheck(void * bundle)
{
        char *transactionRef = get( bundle, "TXNREFERENCE" );
        if (transactionRef) {
                //We have a transaction reference so attempt a status transaction.
                put ( bundle, "TRANSACTIONTYPE", "STATUS" );
                return execute ( bundle );
        } else {
                //No txnref number so we can not do a status check.
                return false;
        }

}

void displayResults(void * bundle)
{
        printf ( "************************************************\n" );
        if(approvedTransaction(get( bundle, "RESPONSECODE"))) {
                printf("************* TRANSACTION APPROVED *************\n");
        } else {
                printf("********** TRANSACTION NOT APPROVED ************\n");
        }
        printf ( "************************************************\n" );
        printf ( "Transaction Reference\t : [%s]\r\n", get( bundle, "TXNREFERENCE") );
        printf ( "Result\t\t\t : [%s]\r\n", get( bundle, "RESULT") );
        printf ( "Auth Code\t\t : [%s]\r\n", get( bundle, "AUTHCODE") );
        printf ( "Response Text\t\t : [%s]\r\n", get( bundle, "RESPONSETEXT") );
        printf ( "Response Code\t\t : [%s]\r\n", get( bundle, "RESPONSECODE") );
        printf ( "Error Message\t\t : [%s]\r\n\n", get( bundle, "ERROR") );
        printf ( "OpenSSL Version\t\t : [%s]\r\n", get( bundle, "_OPENSSL_VERSION") );
        printf ( "************************************************\n" );
}

boolean approvedTransaction(char * responseCode)
{
        //
        // Check the returned response Code against the list of
        // known Approved Response Codes
        //
        // Please check the documentation to ensure that you have the
        // latest list of approved response codes.
        //
        if (responseCode) {
                #define ARRAY_SIZE 3
                char * listOfApprovedResponseCodes[ARRAY_SIZE];
                listOfApprovedResponseCodes[0] = "00"; // Transaction Approved
                listOfApprovedResponseCodes[1] = "08"; // Approved Signature
                listOfApprovedResponseCodes[2] = "77"; // Approved
                for(int i=0; i < ARRAY_SIZE;i++) {
                        if( strcmpi ( responseCode , listOfApprovedResponseCodes[i]) == 0) {
                                //we have a match
                                return true;
                        }
                }
        }
        return false;
}
```

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

Before reporting errors, **ensure that you are able to replicate them**, so that we are able to diagnose properly.

Be prepared to have available for the Helpdesk or e-mail the following information:

- Operating system + API type (i.e. – Linux + java API)
- Basic Hardware description (i.e. – P3 800 + 512 meg RAM)
- Debug logs & Log file containing the values of all fields in the transaction
- Description of the error (when and how it happened)
- Error code and Error message

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

Appendix A – Files included in this kit

IMPORTANT:

Please note that this kit is delivered on the assumption that GCC and OpenSSL libraries are installed with the Operating System. This is a requirement and the kit might not work correctly if the above said installations are not present.

Documentation

Linux C++ API Developer Guide	This document

Base libraries

libwebpayclient.so	St. George Bank Client shared object library on Linux under gcc, on which the test program is runtime dependent.

Base Test

Makefile	Builds the BaseTest.cpp or StatusCheck.cpp file and performs the library installation. Enter the following cmd to make BaseTest: make. Enter the following cmd to make statusCheck: make doCompStatus.
Run_base_test.sh	The file that executes the BaseTest.exe, providing the required values for the variables. **Note: You need to modify this file to contain St. George Bank specific information.**
BaseTest.cpp	The C++ source code for the base test.
webpayclient.h	The header file contains declaration for public methods.
BPlatform.h	Contains the platform specific declarations and definitions.
BTypes.h	The header file that contains helper declarations and definitions
StatusCheck.cpp	Source code for status test
Run_status_test.sh	Batch file for status test

NOTE: The certificate file (eg. client.cert) is not included in the kit. For security purposes, this file will be supplied separately.