TrustCommerce Developer's Guide 2.5

http://www.trustcommerce.com
developer@trustcommerce.com

March 2004

Table of Contents

I. Introduction

This guide is for developers. If you are seeking to add payment processing functionality to a software package or website using the TrustCommerce, platform and language neutral, transaction processing API, then this guide is for you. If you are connecting via a pre-integrated commerce package, including shopping carts such as StoreForge, Mal's e-Commerce, or osCommerce, then you do not need this document.

An overview of payment processing and information about the TrustCommerce Vault website are presented in the TrustCommerce User's Guide and will not be covered here. If you are

not familiar with the basics of credit card processing (such as the difference between an auth and capture), then you should probably read the User's Guide first. The User's Guide also describes the Vault's reporting capabilities, which you may find useful to confirm that your test transactions are working as expected.

There are two methods of connecting to the TrustCommerce gateway at the programming level: TCLink client software and HTTPS POST to the Vault server. TCLink is HIGHLY recommend, as it offers support for TrustCommerce's advanced features such as automated failover, 2048-bit encryption and server certificate verification. It's available for many platforms and languages, and installation is straightforward. If you are unable to use TCLink, the HTTPS POST method will still allow you to access most features of the TrustCommerce processing engine, but be aware that it cannot match the speed and especially the reliability of TCLink.

II. Connecting via TCLink

The TCLink API is available for download from the Vault website. Be sure to choose the version that matches the platform and language that you are using. Win32 developers will typically want the COM object, which contains support for a number of different languages on the Win32 platform, including ASP, Cold Fusion, Perl, PHP, and Python.

Installation instructions specific to the TCLink version you download can be found in the readme file contained within the archive. It will also contain a version of the TCTest program specific to the language you are using. Review this example, and use it as the basis for your own code. In fact, we recommend that you look over the code sample now to get a feel for the API, and then return to reading this document.

Versions of TCLink for high-level languages (including PHP, Perl, Python, and Java) with built in support for hashes only have one method: Send(). This method accepts a single parameter, a hash of input values, and returns a single parameter, a hash of output values. The only other method available is GetVersion(), which returns the version string for TCLink.

TCLink for other languages (including C/C++, Win32 COM, and CFX) contains four basic methods: Create(), PushParam(), Send(), and GetResponse(). These four methods map to the four stages of running a transaction: initialize, push a number of parameters describing the transaction, send the transaction to the TrustCommerce gateway for processing, and finally get details about the transaction results.

The only method which contacts the TC servers is Send(). This function will block for one or two seconds while the transaction is processed. Afterward, you can retrieve response parameters at your leisure; they are stored internally in TCLink.

C-style TCLink functions are described in the table below.

TCLink Function Names

Function Name	Description
CreateLink()	Create a new transaction handle.
PushParam()	Push a parameter into the transaction.
Send()	Submit a transaction to the gateway.
GetResponse()	Retrieve a response parameter from the transaction.

GetEntireResponse()	Get the entire response for the transaction, primarily use for debugging or logging.
Destroy()	Deallocate any memory associated with the transaction handle.
GetVersion()	Get the version string for the TCLink API in use.

III. Transaction Types

You may wish to read the section of the TC User's Guide which describes the transaction types and the authorization/capture process, especially if you are not already familiar with credit card processing.

In terms of parameters passed, preauths and sales are almost identical. They take account data and personal information about the customer, along with a transaction amount. The transaction is authorized and the response values returned. In the case of a sale, the item is also immediately submitted for capture. Preauths are not captured until they are postauthed.

Postauths and credits are also very similar to one another. They indicate a capture or a return of funds for a previous transaction, and no personal data about the customer or their payment method is passed.

There are three other types of transaction. Store and unstore are covered in the Billing ID section of this document. Walletsale is covered under the TC Wallet section .

IV. Input Parameters

There are a large number of parameters available for describing transactions. Not all parameters are accepted for all transaction types, and most parameters are optional. Most transactions can be sent with only a few parameters. The only parameters always required are custid, password, and action.

Parameters are carefully checked for format and size before the transaction is sent. If there are any errors in the parameters you've entered, the transaction will be aborted and detailed information about the errors will be returned. These return values are described in more detail in the next section.

All transaction types use the parameters presented in the following tables.

Required Parameters

Parameter Name	Description
amount	Amount of the transaction, in cents. (example: "500" is $5.00)
custid	The customer ID number assigned to you by TrustCommerce.
password	The password for your custid as assigned by TrustCommerce.
action	The transaction type; can be one of preauth, sale, postauth,credit, store, unstore, walletsale, or chargeback

Optional Parameters

Parameter Name	Description
demo	When set to "y", the transaction will be a test transaction only. This parameter is useful for testing. Demo transactions allow you to submit transactions from your processing system without charging real money to a card and without incurring any transaction fees. Demo transactions are also flagged separately in the Vault database and will not appear in the regular accounting reports.
ticket	Free-form text field intended for storing ticket number or order number associated with the transaction, to aid merchants in order tracking and reporting. The ticket parameter is passed to the acquiring bank in the settlement process as the "purchase identifier." This may aid in reconciling the TrustCommerce transactions with the deposits made into the merchant bank account. If no ticket is passed, then the TrustCommerce transID will be sent to the bank instead.
operator	Free-form text field intended for recording the operator that entered the transaction, to aid merchants in order track and reporting. Enterprise-class customers will note that this field corresponds to the username assigned to operators in the Vault.

V. Return Parameters

Any transaction sent will return several parameters describing the success or failure of the transaction. Properly formatted transactions will always return a transaction ID (or "transID".) The transaction ID is the unique identifier for this transaction, and can be used to retrieve the transaction from the Vault website, or otherwise access the transaction in the future. (For example, in order to credit a previous transaction, you will need to send the transID of the original transaction.)

Transactions always return a status parameter which describes the success or failure of the transaction. Status can be set to one of the following:

Status

Status Value	Description
approved	The transaction was successfully authorized.
accepted	The transaction has been successfully accepted into the system.
decline	The transaction was declined, see declinetype for further details.
baddata	Invalid parameters passed, see error for further details.
error	System error when processing the transaction, see errortype for details.

The difference between approved and accepted is subtle, but important. Approved means that the transaction was an authorization of some sort, and has been successfully cleared

with the bank. Accepted only means that the transaction was queued into the system without errors, but may be rejected at some later time. An example of the difference is a sale versus a credit. A sale is a real-time authorization, so it returns approved on success. A credit is not real-time; there is no such thing as a "credit authorization." Instead it is queued up for processing by the bank and there is the small but non-zero possibility that it will be rejected at a later time. In practice, however, this rarely happens, so the developer need not worry unduly.

When status is set to decline, the parameter declinetype will contain one of the following:

Decline Type

Declinetype Value	Description
decline	This is a "true" decline, it almost always is a result of insufficient funds on the card.
avs	AVS failed; the address entered does not match the billing address on file at the bank.
cvv	CVV failed; the number provided is not the correct verification number for the card. (See CVV section for more details.)
call	The card must be authorized manually over the phone. You may choose to call the customer service number listed on the card and ask for an offline authcode, which can be passed in the offlineauthcode field.
expiredcard	The card has expired. Get updated expiration date from cardholder.
carderror	Card number is invalid, which could be a typo, or sometimes a card reported stolen.
authexpired	Attempt to postauth an expired (more than 14 days old) preauth.
fraud	CrediGuard fraud score was below requested thershold.
blacklist	CrediGuard blacklist value was triggered.
velocity	CrediGuard velocity control was triggered.
dailylimit	Daily limit in transaction count or amount as been reached.
weeklylimit	Weekly limit in transaction count or amount as been reached.
monthlylimit	Monthly limit in transaction count or amount as been reached.

A status of baddata indicates that no transaction was attempted because one or more parameters was invalid. In this case, the parameter error will indicate the problem, and the offender's parameter will list the offending input fields. The error parameter may be set to one of the following:

Bad Data

Error Value	Description

missingfields	One or more parameters required for this transaction type were not sent.
extrafields	Parameters not allowed for this transaction type were sent.
badformat	A field was improperly formatted, such as non-digit characters in a number field.
badlength	A field was longer or shorter than the server allows.
merchantcantaccept	The merchant can't accept data passed in this field. If the offender is "cc", for example, it usually means that you tried to run a card type (such as American Express or Discover) that is not supported by your account. If it was "currency", you tried to run a currency type not supported by your account.
mismatch	Data in one of the offending fields did not cross-check with the other offending field.

A status of error indicates the an error occurred while processing the transaction. These are almost always networking errors; see the Troubleshooting section for more details. If the status is error, then the errortype parameter will be set to one of the following:

Errortype

Errortype Value	Description
cantconnect	Couldn't connect to the TrustCommerce gateway. Check your Internet connection to make sure it is up.
dnsfailure	The TCLink software was unable to resolve DNS hostnames. Make sure you have name resolving ability on the machine.
linkfailure	The connection was established, but was severed before the transaction could complete.
failtoprocess	The bank servers are offline and unable to authorize transactions. Try again in a few minutes, or try a card from a different issuing bank.

Other parameters (such as avs or billingid) may be returned by the transaction depending on the action; see sections covering the transaction type you're running for detailed information on the specialized return values.

VI. Credit Card Preauths and Sales

The 'preauth' and 'sale' actions are identical in terms of parameters and function. The only difference is that a sale submits the item for capture as well, while preauths run the authorization only and can be postauthed at another time.

The parameters that can be passed with these action types are described in the following two tables. The first table list required fields that you must pass in order to send the transaction. The second table lists optional fields that you can pass if you wish, but are not required.

Required Parameters

Parameter Name	Description
amount	Amount of the transaction, in cents. (example: "500" is $5.00)
cc	Credit card number, digits only (no spaces or dashes)
exp	Credit card expiration date, in MMYY format.

Optional Parameters

Parameter Name	Default Value	Description
media	cc	"cc" for credit card or "ach" for ACH.
currency	usd	Currency code (see the Currency Table for a list of allowed codes.)
avs	n	AVS requested, "y" or "n". NOTE: The AVS system is a useful way to help screen for fraud. Unfortunately, card issuers do not provide uniform support for AVS; approximately 30% of US-based credit cards do not have AVS capability, and AVS is not available with any non-US cards at all. The ONLY time that a transaction will be declined due to AVS is in cases where AVS is available and the address data submitted with the transaction does not match the data on file at the bank. If AVS is unavailable, it will not cause the transaction to fail. For this reason, you may wish to screen AVS results closely. Only the numeric parts of the address (street address and zip code) are verified. It should also be noted that on occasion, AVS data is incorrect or not entirely up-to-date. Treat AVS as a helpful tool which can be used in combination with other methods (dependent upon your business model) for screening transactions. If you trust AVS too blindly, you may risk losing legitimate sales. Some merchants (again, dependent upon business model) choose to turn AVS off altogether. You should analyze your customer base and make the decision that will work best for your business.
name		Cardholder's name.
address1		First line of cardholder's street address.
address2		Second line of cardholder's street address.
city		Cardholder's city.
state		Two-character code for the cardholder's state, or the full region/province for international addresses.

zip		Cardholder's zip code, five or nine digits with no spaces or dashes, or the full postal code for international addresses.
country		Cardholder's country, leave blank for US.
phone		Cardholder's phone number.
email		Cardholder's email address.
ip		Customer's originating TCP/IP address (e-commerce only). This is not your server's IP, but that of the customer's computer, as reported by the web server. For example, in PHP the global variable $_SERVER{'REMOTE_ADDR'} contains the incoming IP address which should be passed in this field.
offlineauthcode		Six-digit numeric code used to "force" the transaction. NOTE: Some cards returned a declinetype of call. This indicates that the card cannot be automatically authorized. If you call the card issuer, they may be able to give you a manual authorization, which will consist of a six-digit auth code. You may then push the transaction through by entering that code into the offlineauthcode field. Offline authorizations via offlineauthcode can only be used with sale transactions. Since offlineauthcode skips the authorization phase altogether, it is not guaranteed in the same way as a normal authorization. Be careful with the use of this parameter.

AVS Returns Codes

Code	Description
X	Exact match, 9 digit zip code.
Y	Exact match, 5 digit zip code.
A	Street address match only.
W	9 digit zip code match only.
Z	5 digit zip code match only.
N	No match on street address or zip code.
U	AVS unavailable on this card.
G	Non-US card issuer, AVS unavailable.
R	Card issuer system currently down, try again later.
E	Error, ineligible - not a mail/phone order.

S	Service not supported.
0	General decline or other error.

Preauth/Sale Example

```
params = {
        'custid':    'mycustid',
        'password':  'mypasswd',
        'action':    'preauth',
        'amount':    '500',
        'cc':        '4111111111111111',
        'exp':       '0404',
        'name':      'Jennifer Smith',
        'address1':  '123 Test St.',
        'city':      'Somewhere',
        'state':     'CA',
        'zip':       '90001',
        'avs':       'y'
}

result = tclink.send(params)

if (result['status'] == 'approved'):
        print  'Transaction was successful'
elif (result['status'] == 'decline'):
        print 'Transaction declined.  Reason: ', result['declinetype']
elif (result['status'] == 'baddata'):
        print 'Improperly formatted data.  Offending fields: ', result['offenders']
else:
        print 'An error occurred: ', result['errortype']

print 'Here are the full details:'
print result
```

VII. Card Verification Value (CVV)

Another method for cardholder verification is the CVV system. Visa cards use what is called a CVV(II) or CVV2 code; MasterCard calls it CVC2, and American Express calls it CID. CVV(II) and CVC2 (Visa/MC) are located on the back of the credit card, printed in ink on the signature strip. The number is the last three digits on the right hand side of the long string of numbers on the strip. In the case of CID (AmEx), the number is a four-digit value located on the front of the card, printed just above the card number on the left hand side. This value is passed to TCLink through the following parameter.

CVV Parameters

Parameter Name	Description
cvv	A three or four digit CVV number.
checkcvv	Set to "n" to disable CVV check, even if the value is passed. Default is "y".

The CVV value will be checked if anything is passed in this parameter. (If you don't include the parameter, no CVV test will be run.) If the CVV matches, the transaction will run as normal. If it does not, you will receive a status of decline, and a declinetype of cvv.

In some cases you may wish to pass the CVV code without actually checking to confirm that it is correct. In that case set checkcvv=n.

CVV Example

```
params = {
        'custid':    'mycustid',
        'password':  'mypasswd',
        'action':    'sale',
        'amount':    '500',
        'cc':        '4111111111111111',
        'exp':       '0404',
        'cvv':       '123'
}

result = tclink.send(params)

if (result['status'] == 'decline'):
        if (result['declinetype'] == 'cvv'):
                print 'The CVV number is not valid.'
```

VIII. Card-Present Transactions

In a retail environment with a card swiper, you may wish to pass magnetic stripe data along with the other credit card information on a preauth or sale. There are two parameters which you can use to send this data, named track1 and track2. Each one is for a different type of track data, and depends on the type of card reader being used.

Card-Present Parameters

Parameter Name	Description
track1	Up to 79 bytes of Track 1 data.
track2	Up to 40 bytes of Track 2 data.

You can include all data read from the track, but only data between the start sentinel (a '%' character for track1 and a ';' for track2) and the end sentinel ('?') will be used. Everything outside the start and end sentinels (such as the trailing LRC character) will be discarded.

The cc and exp fields may be passed, but are not required, since they will be extracted from the track data. If you do pass one or both of these fields, however, and it does not match the data extracted from the track data, a status of baddata and an error mismatch will be returned.

If both track1 and track2 data are passed, the system will choose track1 by default for the authorization, and track2 will be discarded.

Generally AVS and address data are not necessary for swiped transactions, but you may pass them anyway if you choose. All data passed will be saved in the vault and can be downloaded for reports and later analysis.

Card Present Example

```
params = {
        'custid':    'mycustid',
        'password':  'mypasswd',
        'action':    'sale',
```

```
        'amount':    '500'
}

params['track1'] = CardReader.readMagStripe()

result = tclink.send(params)
```

IX. ACH

ACH, also known as electronic checks, is very different from credit cards in that it allow a direct debit or credit to a checking account. The concept of "authorization" does not exist; it is purely a money transfer. For this reason, the only transaction types available for ACH are sale and credit. ACH credits are identical to all other types of credits in the TrustCommerce system, so please refer to postauths, credits and chargebacks for details on issuing credits.

ACH sales take the same parameters as credit card sales, with the exception of the "cc" and "exp" fields. Instead, use the following parameters.

ACH Parameters

Parameter Name	Description
routing	The routing number of the bank being debited.
account	The account number of the person or business being debited.

AVS is not available for ACH transactions, so the AVS setting is ignored.

There is only one declinetype returned from unsuccessful ACH sales, and that is "decline."

ACH Example

```
params = {
        'custid':    'mycustid',
        'password':  'mypass',
        'action':    'sale',
        'media':     'ach',
        'amount':    '1000',
        'routing':   '789456124',
        'account':   '55544433221'
}

tclink.send(params)
```

X. Postauths, Credits, and Chargebacks

These three transaction types are similar in that they reference a successful transaction previously executed through the TrustCommerce system. These transaction types will not accept credit card information or other personal information fields. There is only one required parameter.

Required Parameters

Parameter Name	Description
	The transaction ID (format: nnn-nnnnnnnnnn) of the referenced

transid	transaction.

In addition, postauths and credits can take an optional amount parameter.

Optional Parameters

Parameter Name	Description
amount	The amount, in cents, to credit or postauth if it is not the same as the original amount.

If no amount is passed, the default will be to take the amount from the original transaction. If, however, you wish to credit or postauthorize less than the original amount, you may pass this parameter.

Credit Example

```
params = {
        'custid':    'mycustid',
        'password':  'mypass',
        'action':    'credit',
        'transid':   '001-0000111101'
}

tclink.send(params)

if (result['status'] != 'accepted'):
        print 'Credit unsuccessful.'
```

XI. Citadel (Billing IDs)

The Citadel enables you to store customer billing information in an encrypted TrustCommerce database. This information can then be recalled with a single six-character alphanumeric code known as a billing ID. Besides offloading the liability of storing sensitive data such as credit card numbers from your business to TrustCommerce, the Citadel can also simplify application development on your servers.

The Citadel has two unique actions, store and unstore. These transction types allow you to create, update, and deactivate billing IDs. Once stored, the billing ID will be passed in place of the many billing information fields for preauth and sale transactions.

Store Parameters

The store transaction looks very similar to a preauth or sale. You may pass credit card information , billing and shipping address, or even ACH information.

Parameter Name	Description
verify	When set to "y", a $1.00 preauth will be run on the card to ensure that it is valid.
billingid	Pass if you wish to update values of an existing billing ID.

You can update information on existing billing IDs by passing the billingid field along with the store transaction. If you wish to erase the data contained in a field, pass the exact string "null" (four characters, all lower case) as the parameter value.

The verify parameter is useful for ensuring that the card number is valid if you are not planning to bill the customer right away. It has no effect for ACH billing IDs, because there is no such thing as an ACH preauth. It is recommended that you pass avs=y for further verification of the cardholder's data. Please note: normally, billing ID storage returns a status of accepted, because the card is unverified. If you use the verify parameter, however, it will return approved upon success.

Unstore Parameters

The unstore action takes a single parameter:

Parameter Name	Description
billingid	The six-character alphanumeric billing ID returned by a previous store transaction.

Unstore removes the billing ID from active use, and you won't be able to run further transactions on it. For your convenience, however, it does remain visible in Vault (flagged as an inactive ID), so you can still look up old IDs if needed.

Billing ID Example

To run a billing ID transaction, pass the billingid parameter to a preauth or a sale in place of all of the usual billing info fields (name, cc, exp, routing, account, etc). That's all there is to it! You'll find that your sale or preauth transactions are now only a few parameters: custid, password, billingid, and amount.

```
# First, store a new ID.
params = {
        'custid':    'mycustid',
        'password':  'mypass',
        'action':    'store',
        'cc':        '4111111111111111',
        'exp':       '0404',
        'name':      'Jennifer Smith'
}

result = tclink.send(params)

if (result['status'] == 'accepted'):
        # It was successfully stored.  Now try running a transaction on the new ID.
        billingid = result['billingid']
        params2 = {
                'custid':    'mycustid',
                'password':  'mypass',
                'action':    'sale',
                'billingid': billingid,
                'amount':    '995'
        }
        tclink.send(params2);
        # Unstore the ID now that we are done.
        params3 = {
                'custid':    'mycustid',
                'password':  'mypass',
```

```
        'action':    'unstore',
        'billingid': billingid
}
tclink.send(params3)
```

XII. Recurring Billing

Recurring billing (also called "subscription billing") is an extension of the billing ID system. There are four parameters that are added to a store which turn the billing ID into a recurring billing ID. They are described below.

Recurring Parameters

Parameter Name	Required?	Default Value	Description
cycle	X		Repeat cycle in days, weeks, months, or years.
amount	X		Amount to bill in cents.
start			Date to start, or else offset from current date (defaults to now).
payments		0	Number of payments to make (defaults to infinite).
authnow		n	If 'y', performs the card authorization immediately. Funds will be captured upon first payment cycle. *(requires start date)*
lastpaymentunstore		y	If 'y', the final payment in the billing cycle will result in the billing ID being unstored.

Cycle is in the format of a number between 1 and 99 followed by a character representing the time frame: 'd' for days, 'w' for weeks, 'm' for months or 'y' for years. For example, a value of "3d" in this field would cause the charge to recur every three days, whereas a value of "2m" would cause it to recur every two months.

Amount is identical to the amount passed to a sale or a preauth, and indicates the amount of each charge made during the subscription.

Start is an optional field which allows the delay of the cycle's start. By default, the cycle starts the day the billing ID is stored. If it is specified in the format YYYY-MM-DD, it indicates a date when the first payment should be made. (For example, "2005-02-01" would be February 1, 2005.) If it is in the same format as the cycle (a number followed by a character), it indicates an offset from the current date. (For example, "1d" would start the billing tomorrow.)

If there is no start parameter, the first transaction is run immediately (and can cause the store action to return a "decline" or other result you would expect from a sale). Other return parameters associated with auths, such as avs code, will be returned as well.

If you pass authnow=y, an immediate authorization will be performed for the payment amount, but the funds will not be captured until the start date is reached. This is handy for free trial accounts: because the funds are authorized immediately, you are assured the

money. If they cancel sometime between the time the billing ID is stored and the start date of the billing cycle, you can unstore the ID and they will never be charged. Their free trial will be the time between the billing ID store and the billing cycle start date. Authnow requires that a start date be specified.

The payments field is also optional; left blank or set to zero it will continue the billing cycle until it is manually interrupted by an unstore, or by updating the billing ID with cycle set to "null". Once the final payment is made, the billing ID will be unstored, unless lastpaymentunstore is set to 'n'. lastpaymentunstore should only be used when the number of payments is specified.

There is only one instance in which the cycle parameter is not required, and that is a one-time future payment. By setting payments to "1" and passing a start date, the payment will be made on that date and then the billing ID unstored. The cycle parameter may be included with a one-time future payment, but it will have no effect.

Recurring Billing Example

```
params = {
        'custid':    'mycustid',
        'password':  'mypass',
        'action':    'store',
        'cc':        '4111111111111111',
        'exp':       '0404',
        'name':      'Jennifer Smith',
        'amount':    '1200'
}

mode = chooseSubscriptionMode()

if (mode == 1):
        # Make a payment every day, infinitely (or until someone manually disables it)
        params['cycle'] = '1d';
elif (mode == 2):
        # Make a payment once a month for one year
        params['cycle'] = '1m';
        params['payments'] = '12';
elif (mode == 3):
        # Make a payment every six weeks, starting one week from now
        params['cycle'] = '62';
        params['start'] = '1w';
elif (mode == 4):
        # Make annual payments, and don't start until September 1, 2004
        params['cycle'] = '1y';
        params['start'] = '2004-09-01';
elif (mode == 5):
        # Make a one time payment in 45 days
        params['start'] = '45d';
        params['payments'] = '1';

tclink.send(params)
```

XIII. TC Wallet

The TC Wallet is an additional service offered by the TrustCommerce gateway that allows you to run many small micro payments which are later lumped together and submitted as a single, large payment. This can offer a substantial savings for the merchant in transaction fees, as merchant account providers tend to penalize merchants that run small payments.

Wallet Parameters

The Wallet is an extension of the billing ID system. It adds three new parameters to the store action, described below.

Parameter Name	Description
wallet	"y" to enable the wallet.
walletsize	Accumulated amount, in cents, before the wallet is submitted for capture.
walletexposure	Length of time to wait before capturing the wallet.

In order to enable wallets for the billing ID that you are storing, set the "wallet" parameter to "y". The other two parameters are optional, as there are default values attached to your TC account. If you wish to change these default values (which are normally $10.00 and 72 hours) please contact TrustCommerce. If you wish to change the values for an individual billing ID without changing your defaults, include these parameters.

Once the billing ID is stored, it is possible to use a new action type on the ID called walletsale. It is identical to a sale in all ways, except that it can only be used on wallet-enabled billing IDs, and it can accept amounts anywhere from $0.01 up to the size of the wallet. Walletsale will usually return "approved", but it may return "decline" if the stored credit card fails to re-authorize when the current wallet is exceeded. Except for these two modifications (the wallet parameters on the store, and changing sales to walletsales), there is no extra development to be done client-side to make wallets work. Captures happen automatically after walletexposure has expired or the walletamount has been used up; reauthorizations also happen automatically.

Wallet Example

```
# First, store a new ID with the wallet enabled.
params = {
        'custid':    'mycustid',
        'password':  'mypass',
        'action':    'store',
        'cc':        '4111111111111111',
        'exp':       '0404',
        'name':      'Jennifer Smith',
        'wallet':    'y'
}

result = tclink.send(params)

if (result['status'] == 'approved'):
        # We have a new wallet at our disposal.  Run a micropayment on it.
        billingid = result['billingid']
        params2 = {
                'custid':    'mycustid',
                'password':  'mypass',
                'action':    'walletsale',
                'billingid': billingid,
                'amount':    '995'
        }
        tclink.send(params)
```

XIV. CrediGuard - Neural Network Fraud Scoring

When the CrediGuard Neural Network is enabled, a fraud score is assigned to each

transaction based on its appearance of fraud or legitimacy. This score is determined by running each transaction through an advanced artificial intelligence neural network that is trained to recognize fraudulent transactions.

The fraud score represents an evaluation of how legitimate or fraudulent the transaction appears. A fraud score close to 0 means that the transaction is likely to be fraudulent, while a fraud score close to 100 means that the transaction is likely to be legitimate.

The CrediGuard Score Threshold is a number from 0 to 100 that controls how restrictive the CrediGuard fraud screening system should be. Any transaction with a fraud score less than the CrediGuard score threshold will be declined. All other transactions will be accepted as per usual. The following chart sums up the effects of the possible CrediGuard score threshold ranges:

Score Threshold Range	Blocking Effect
0	Allow All
1 - 25	Allow Most
26 - 50	Normal
51 - 75	Restrictive
76 - 100	Highly Restrictive

The default score threshold can be configured within the Vault web interface. To override the default score threshold, you can optionally pass in an extra parameter to set a different score threshold for the current transaction only:

CrediGuard Parameters

Parameter Name	Description
threshold	The fraud threshold for the current transaction. If the transaction receives a Fraud Score below this threshold, it will be declined.

With Neural Network scoring enabled, the return parameters for each transaction will contain an additional parameter for the calculated fraud score:

CrediGuard Return Parameters

Parameter Name	Description
fraudscore	The actual numeric score, 0 to 100, that the Crediguard Neural Network gave to the transaction.

CrediGuard Neural Network Example

```
params = {
        'custid':    'mycustid',
        'password':  'mypasswd',
```

```
        'action':    'sale',
        'amount':    '500',
        'cc':        '4111111111111111',
        'exp':       '0404',
        'threshold': '45',
}

result = tclink.send(params)

if (result['status'] == 'decline'):
        if (result['declinetype'] == 'fraud'):
                print 'Transaction declined based on fraud score.'
```

You may also wish to handle the score result yourself, perhaps with fuzzy logic that treats a certain range of low scores as out-and-out declines, a high range as approvals, and a middle range as accepting pending further processing. The last category of transactions could be set aside in your local database or perhaps trigger an email to a customer service rep, who could manually review the transaction, and perhaps call the customer for verification.

CrediGuard Fuzzy Logic Scoring Example

```
score_bottom = 30
score_top = 70

params = {
        'custid':    'mycustid',
        'password':  'mypasswd',
        'action':    'sale',
        'amount':    '500',
        'cc':        '4111111111111111',
        'exp':       '0404',
        'threshold': score_bottom
}

result = tclink.send(params)

if (result['status'] == 'decline'):
        if (result['declinetype'] == 'fraud'):
                print 'Transaction declined, score was too low.'

if (result['status'] == 'approved'):
        if (result['fraudscore'] > score_top):
                print 'Transaction approved'
        else:
                print 'Transaction accepted pending review'
```

XV. CrediGuard - Blacklists

Blacklists are used to block specific cardholder information that you know to be fraudlent. The most common example would be a customer who keeps using a credit card that caused chargebacks in the past. You can block the customer from running charges on the card by entering their credit card number as a blacklist value.

Blacklist rules are configured from within the Vault web interface and apply to all transactions, so there are no blacklist input parameters that you need to pass in with TCLink.

If a transaction is declined because it matches a blacklist rule that you have created, the reason for this decline will be returned via the following parameters:

Blacklist Return Parameters

Parameter Name	Description
blacklistfield	The name of the input field that contained the blacklisted value. This can be one of "name", "cc", "zip", "address1", "state", "country", "email", "phone", or "ip".
blacklistvalue	The input value that triggered the blacklist decline. For instance, if the decline was due to a blacklisted IP address, the offending IP address would be returned here.

Blacklist Example

```
params = {
        'custid':    'mycustid',
        'password':  'mypasswd',
        'action':    'sale',
        'amount':    '500',
        'cc':        '4111111111111111',
        'exp':       '0404',
}

result = tclink.send(params)

if (result['status'] == 'decline'):
        if (result['declinetype'] == 'blacklist'):
                print 'Transaction declined based on blacklist rule.'
                print 'The offending field was: ' + result['blacklistfield']
                print 'And the blacklisted value was: ' + result['blacklistvalue']
```

XVI. CrediGuard - Velocity Control

In the world of payment processing, 'velocity' refers to limiting the number or amount of transactions which occur over a set period of time. This allows you, the merchant, to control your risk. The most common use of velocity would be to limit the dollar amount spent per day on a single credit card. But the CrediGuard package supports a wide variety of velocity parameters, allowing you to tailor them to match the usage patterns of your customers and thereby control your risk factor.

CrediGuard can perform four different types of velocity checks:

Velocity Types

Velocity Type	Description
Global	All transactions
Zipcode	Transactions on the same zipcode
IP Address	Transactions sent from the same Internet address
Credit Card	Transactions on the same credit card

Each of the four velocity types offers a **count** (number of transactions) and a **total** (dollar amount sum). For each of those two values, there are four time periods, ranging from one

day to one month. All of these velocity settings are configured from within the Vault web interface and apply to all transactions, so there are no velocity input parameters that you need to pass in with TCLink.

If a transaction is declined on the basis of exceeding one of these velocity settings, the reason for this decline will be returned via the following parameters:

Velocity Return Parameters

Parameter Name	Description
velocitytype	A brief string that contains the precise reason for the velocity decline. The first part of this string contains a velocity type, either "global", "zip", "ip", or "cc". The next part of the string contains a time period, either "1day", "3day", "15day", or "30day". The end of the string is "count" if the transaction count for the velocity type and time period was exceeded, or "total" if the dollar amount sum was exceeded. For example, "zip3daycount" would be returned if the velocity transaction count was exceeded for a particular zip code in a three day period.
velocityvalue	The input value that triggered the velocity decline. For instance, if the decline was due to a zip code velocity setting, the offending zip code would be returned here.

Velocity Example

```
params = {
        'custid':    'mycustid',
        'password':  'mypasswd',
        'action':    'sale',
        'amount':    '500',
        'cc':        '4111111111111111',
        'exp':       '0404',
}

result = tclink.send(params)

if (result['status'] == 'decline'):
        if (result['declinetype'] == 'velocity'):
                print 'Transaction declined based on ' + result['velocitytype'] + ' velocity controls.'
```

XVII. Purchase Level II

Purchase Level II is required by some merchant banks to achieve a qualified discount rate. It is only used for B2B or B2G transactions. If you are selling products directly to consumers, you do not need to use PL2. Additionally, PL2 can only be used if the cardholder is using one of the three types of commercial cards: a corporate card, a business card, or a purchasing card.

PL2 Input Parameters

PL2 has three new parameters:

Parameter Name	Description

purchaselevel	Specify as "2".
purchaseordernum	The purchasing order number from their Visa purchasing card. This is either a 16 or a 17 digit number. If they are not using a Visa purchasing card, do not pass this field.
tax	The amount of tax, in cents, charged for the order. If the order is tax exempt omit this field or set it to 0.

PL2 Output Parameter

A successful PL2 transaction will return one extra response parameter

Parameter Name	Description
commercialcard	Set to "S" for if it is a purchasing card, "R" if it is corporate card, and "B" if it is a business card.

If you don't care what kind of card they used (which you probably don't), there is no reason to store or otherwise do anything with the commercialcard return value. TrustCommerce internally handles all the details for you. However, if you want this information for some reason, you can access it from the result parameters.

Purchase Level II Example

```
params = {
        'custid':           'mycustid',
        'password':         'mypass',
        'action':           'sale',
        'cc':               '4111111111111111',
        'exp':              '0404',
        'amount':           '1000'
        'name':             'Jennifer Smith',
        'purchaselevel':    '2'
        'purchaseordernum': '12345678901234567'
        'tax':              '83'    # $10.00 x 8.25% sales tax = $0.83

}

result = tclink.send(params)

if (result['status'] == 'approved'):
        print 'Approved'
```

XVIII. Automated Fulfillment

TrustCommerce offers the unique feature of automated fulfillment, by which orders submitted through the TCLink API can contain parameters describing product information. This information is then passed on to your fulfillment house, which ships the products automatically. Even if you aren't using automated fulfillment, you can use the order line item specification parameters to store information in the TrustCommerce transaction database about the type of products ordered.

First, you may wish to specify the shipping address for the customer. If it is the same as the billing address, you need not duplicate it, but rather just send the shiptosame parameter set to y. The table below lists the shipping parameters.

Ship To Parameters

Parameter Name	Description
shiptosame	y or n (defaults to n). If "y", then no other shipto_ parameters should be passed.
shipto_name	Name of the product recipient.
shipto_address1	First line of shipping address.
shipto_address2	Second line (if any) of shipping address.
shipto_city	City.
shipto_state	State.
shipto_zip	Zip code, five or nine digits (no spaces or dashes).
shipto_country	Country, leave blank for US.

Order Header Parameters

Second, some additional data about the order as a whole should be passed in the form of the following parameters.

Parameter Name	Description
shippingcode	Three character code indicating shipping method to use.
shippinghandling	The total cost of shipping and handling, in cents.
numitems	Total number of distinct items (product codes) in the order.

Order Detail Parameters

Finally, a number of parameters which describe the details of each type of item must be passed. In the table below, the 'X' character in the parameter name should be replaced with a digit indicating which item it applies to. For example, if numitems is set to three, then you should pass three product codes, in the form of productcode1, productcode2, and productcode3.

Parameter Name	Description
productcodeX	The alphanumeric product code defined by the fulfillment house.
quantityX	The number of items of this type to be shipped.
priceX	The price of all items of this type not including sales tax or shipping, in cents.
taxX	Total tax charged for all items of this product code.

shippinghandlingX	Total shipping and handling charged for all items of this product code.

Please note: the shippinghandling field and the shippinghandlingX fields are mutually exclusive. Use one or the other, but not both. If you're not sure which to use, please contact your fulfillment house.

Fulfillment Example

```
params = {
        'custid':           'mycustid',
        'password':         'mypass',
        'action':           'preauth',
        'cc':               '4111111111111111',
        'exp':              '0404',
        'amount':           '1979',  # Total of items, tax, and shipping
        'name':             'Jennifer Smith',
        'address1':         '123 Test St.',
        'city':             'Somewhere',
        'state':            'CA',
        'zip':              '90001',
        'shiptosame':       'y',     # Shipping address is same as billing address

        'numitems':         '2',     # Two total product codes will be described
        'shippingcode':     'OVRNGT',
        'shippinghandling': '695',

        'productcode1':     'PCODE1',
        'quantity1':        '3',
        'price1':           '600',   # Item 1 costs $2.00, and there are 3 of them
        'tax1':             '144',   # Tax of 8%

        'productcode2':     'PCODE2',
        'quantity2':        '1',
        'price2':           '500',   # Item 1 costs $5.00, there's only one
        'tax2':             '40'     # Tax of 8%
}

tclink.send(params)
```

XIX. Batch Processing

The TrustCommerce Batch Processing service allows programmers to submit an unlimited number of transactions in a single file for offline processing. The file can contain a mixture of transaction types (preauths and postauths, credits, sales, stores and unstores for example) and can use Citadel Billing IDs, ACH Routing codes or credit card information to identify the account to be charged. In fact, any field found in Appendix E – Input Field List of this guide can be included.

Naming Conventions

While there is no required naming convention for the transaction file sent to TrustCommerce, we recommend the convention shown in the table below.

The results file that we return to you will use your file name with the prefix "result-".

Recommended File Name	Description
	This is the input file containing the list of transactions to be processed.

batch-[custid]-[mmddyy][x].txt	This file MUST be in comma separated values (CSV) format and the first row MUST contain the field names for all fields that you choose to include (See Appendix E .) Individual records are not required to contain values for each field. [custid] is your numeric TrustCommerce custid. [mmddyy] is a numeric date. [x] is an optional alpha identifier to distinguish batches if multiple batches are sent in one day. Do not include the "[" or "]"characters in the file name.
result-[your file name].txt	This file contains a response record for each transaction record sent in the batch file.

NOTE: The records in the results file will be sequenced in exactly the same order as the original input file. If you require a more positive identifier for each record (invoice number for example) include the required value in the "ticket" field of your input file. This value will be echoed back to you in the "ticket" field of the results file. See the examples below.

Sample Batch Upload File

To invoke the Batch Processing service use your programming language's HTTPS POST function and point it at https://batch.trustcommerce.com/submit.php .

A sample batch upload file might look like this:

```
ticket,action,cc,exp,amount,name,email
01-1224,sale,4111111111111111,0404,999,Test Person,person@example.com
00-1175,preauth,4111111111111111,0404,333,Test Person,person@example.com
```

Sample Batch Response File

The response file will contain one response record for each uploaded transaction record.

The results file can then be downloaded interactively from the Vault or by invoking your language's HTTPS POST function pointed at https://batch.trustcommerce.com/download.php .

A sample response file might look like this:

```
ticket,status,transid,billingid,declinetype,errortype,offenders,avs
 01-1224,approved,010-0001364786,,,,,N
 00-1175,approved,010-0001364785,,,,,N
```

Status Reporting

Email is the primary means used by the Batch Processing service to provide status updates. Status updates will be sent to the same email address used for Daily Reports emails. This email address can be modifed on the "Settings" page of the Vault's web interface.

After uploading your file, the system will respond immediately with an "Acknowledged" message and then follow by sending an email with a subject similar to the following:

Subject: Batch File [your file name] Picked Up

The content of the mail will provide processing statistics including the number of records found in the file.

When processing is complete, a second e-mail will be sent. The subject this time will look like:

Subject: Batch File [your file name] Complete

The body of this email will contain relevant information including the number of approvals, declines, accepts and settlements.

Sample Code

The sample HTML code shown below illustrates the functionality of the Batch Processing service invocation of the TCLink API. It can be pasted into a file and loaded on your web browser in order to experiment with batch processing before you even begin to write your code.

Batch Upload Example

```
<html>
<head> <title> TrustCommerce Batch Upload Interface </title> </head>
<body>
<h2> TrustCommerce Batch Upload Interface </h2>

<form action=https://batch.trustcommerce.com/submit.php method=post
enctype=multipart/form-data>
        <table align=center border=1>
        <tr>
                <td> custid </td>
                <td> <input type=text name=custid> </td>
                <td> This is your TrustCommerce custid (required) </td>
        </tr>
        <tr>
                <td> password </td>
                <td> <input type=password name=password> </td>
                <td> This is your TrustCommerce password (required) </td>
        </tr>
        <tr>
                <td> batch file </td>
                <td> <input name=file type=file> </td>
                <td> This is the batch file to upload (required) </td>
        </tr>
        <tr>
                <td colspan=3> <input type=submit> </td>
        </tr>
        </table>
</form>
</body>
</html>
```

Batch Results Download Example

```
<html>
<head> <title> TrustCommerce Batch Results Download Interface </title> </head>
<body>
<h2> TrustCommerce Batch Results Download Interface </h2>

<form action=https://batch.trustcommerce.com/download.php method=post>
        <table align=center border=1>
        <tr>
                <td> custid </td>
                <td> <input type=text name=custid> </td>
                <td> This is your TrustCommerce custid (required) </td>
```

```
        </tr>
        <tr>
                <td> password </td>
                <td> <input type=password name=password> </td>
                <td> This is your TrustCommerce password (required) </td>
        </tr>
        <tr>
                <td> result file name </td>
                <td> <input name=file type=text> </td>
                <td> This is the result file name to download (required) </td>
        </tr>
        <tr>
                <td colspan=3> <input type=submit> </td>
        </tr>
        </table>
</form>
</body>
</html>
```

XX. Vault Query API

The Vault website serves as a web-based interface for merchant reconciliation. On the back end, it is a complex database that tracks all transactional data for your TrustCommerce account. The information contained in the Vault can be accessed at the API level using a standard CGI query over HTTPS.

The query API returns data in CSV (comma separated value) format, which is just a flat text file with fields separated by commas, and records separated by newlines. (You can also request the data returned in an HTML table, which is useful for debugging.) The first line of the file contains the name of each field that will be returned in subsequent records.

The parameters that the query interface accepts are described in the table below.

Required Parameters

Parameter Name	Required?	Description
custid	X	TrustCommerce customer ID number.
password	X	The password for your custID.
format		Set to "html" for human-readable HTML output, or "text" (default) for CSV text output.
querytype	X	Possible values are: chain, transaction, summary, or billingid, corresponding to the equivalent reports on the Vault website. Read the User's Guide for further description of each report type.
media		Set to "cc" (default) or "ach" for media type.
begindate		Begin date for query, format: MM-DD-YYYY HH:MM:SS
enddate		End date for query, format: MM-DD-YYYY HH:MM:SS
chain		Narrow search to a single chain of transactions.
transid		Narrow search to a single transaction ID.

billingid		Narrow search to a single billing ID.
pastdue		Use 'y' or 'n'. For billing ID search. Show only recurring billing IDs that have been unable to capture the requested funds.
action		Narrow search by action (preauth, sale, credit, etc).
status		Narrow search by status (approved, accepted, decline, etc).
name		Narrow search by cardholder name, partial or complete.
cc		Narrow search by credit card number.
ticket		Narrow search by contents of the ticket field.
limit		Do not allow size of result set to exceed this number of records.
offset		Report result set starting from this offset. Use this parameter in conjunction with limit in order to page through a large result set.
showcount		Show the number of records returned on the last line of the result set.

The sample HTML code shown below illustrates the functionality of the query API. It can be pasted into a file and loaded on your web browser in order to try out some queries before you even begin to write your code.

Query Example HTML

```
<html>
<head> <title> TrustCommerce Query Interface </title> </head>
<body>
<h2> TrustCommerce Query Interface </h2>
<form action=https://vault.trustcommerce.com/query/>
        <table align=center border=1>
        <tr>
                <td> custid </td>
                <td> <input type=text name=custid> </td>
                <td> This is your TrustCommerce custid (required) </td>
        </tr>
        <tr>
                <td> password </td>
                <td> <input type=password name=password> </td>
                <td> This is your TrustCommerce password (required) </td>
        </tr>
        <tr>
                <td> format </td>
                <td> <select name=format> <option value=text>text</option>
                                <option value=html>html</option></select></td>
                <td> Human readable (html) or computer readable (text) results</td>
        </tr>
        <tr>
                <td> Query type </td>
                <td> <select name=querytype>
                <option value=chain>chain</option>
                <option value=transaction>transaction</option>
                <option value=summary>summary</option>
                <option value=billingid>billingid</option>
                </select></td>
```

```
        <td> Type of query </td>
</tr>
<tr>
        <td> media </td>
        <td> <input type=text name=media value=cc> </td>
        <td> For now this must be cc </td>
</tr>
<tr>
        <td> begindate MM-DD-YYYY HH:MM:SS </td>
        <td> <input type=text name=begindate> </td>
        <td> Query begins at this date</td>
</tr>
<tr>
        <td> enddate MM-DD-YYYY HH:MM:SS </td>
        <td> <input type=text name=enddate> </td>
        <td> Query ends at this date</td>
</tr>
<tr>
        <td> chain </td>
        <td> <input type=text name=chain> </td>
        <td> Narrow search to a single chain of transactions </td>
</tr>
<tr>
        <td> transid </td>
        <td> <input type=text name=transid> </td>
        <td> Narrow search to a single transactions </td>
</tr>
<tr>
        <td> billingid </td>
        <td> <input type=text name=transid> </td>
        <td> Narrow search to a single billingid </td>
</tr>
<tr>
        <td> pastdue </td>
        <td> <select name=pastdue>
        <option value=y>y</option>
        <option value=n>n</option>
        </select></td>
        <td> Use 'y' or 'n'. For billing ID search.  Show only recurring billing IDs
             that have been unable to capture the requested funds. </td>
</tr>
<tr>
        <td> action </td>
        <td> <input type=text name=action> </td>
        <td> Narrow search by action. (example: preauth,postauth) </td>
</tr>
<tr>
        <td> status </td>
        <td> <input type=text name=status> </td>
        <td> Narrow search by status. (example: approved,accepted) </td>
</tr>
<tr>
        <td> name </td>
        <td> <input type=text name=name> </td>
        <td> Narrow search by name. </td>
</tr>
<tr>
        <td> cc </td>
        <td> <input type=text name=cc> </td>
        <td> Narrow search by credit card field. </td>
</tr>
<tr>
        <td> limit </td>
        <td> <input type=text name=limit value=20> </td>
        <td> Limit results to this number of fields (not used for summary)</td>
</tr>
```

```
        <tr>
                <td> offset </td>
                <td> <input type=text name=offset value=0> </td>
                <td> Report results at this offset (used with limit to page through results)</td>
        </tr>
        <tr>
                <td> showcount </td>
                <td> <select name=showcount>
                <option value=y>yes</option>
                <option value=n>no</option>
                </select></td>
                <td> Show the number of not-limited rows on the last line of the result</td>
        </tr>
        <tr>
                <td colspan=3> <input type=submit> </td>
        </tr>
        </table>
</form>
</body>
</html>
```

Appendix A - Test Data

While testing, you may wish to experiment with the different responses that the gateway can generate. The following test card numbers will produce an approval, and have address data as listed, for testing AVS. If you wish to test CVV, the code listed int he right-hand column is the correct CVV code, Other valid credit cards will work, but will produce a 'U' AVS code.

Please note: these cards ONLY work on transactions flagged as demo, or while your account is in "test" mode! For a live transaction, they will all return a decline with a declinetype of carderror.

Test Cards - Approved

Card Type	Card Number	Exp	Address	City	State	Zip	CVV
Visa	4111111111111111	04/04	123 Test St.	Somewhere	CA	90001	123
MasterCard	5411111111111115	04/04	4000 Main St.	Anytown	AZ	85001	777
American Express	341111111111111	04/04	12 Colorado Blvd.	Elsewhere	IL	54321	4000
Discover	6011111111111117	04/04	6789 Green Ave.	Nowhere	MA	12345	-
Diner's Club	36484444444446	04/04	7390 Del Mar Blvd.	Atown	NY	01101	-
JCB	213122222222221	04/04	350 Madison Ave.	Springfield	OH	400000	-

The following card numbers will generate a decline, with the declinetype listed as follows. You may use this to test code which takes different paths dependent upon the type of decline.

Test Cards - Declined

Card Number	Exp	DeclineType
4012345678909	04/04	decline
5555444433332226	04/04	call
4444111144441111	04/04	carderror

ACH Test Numbers

Routing Number	Account Number
789456124	55544433221

You should also test your code to make sure it properly handles all baddata and error cases. Simply pass bad values in order to generate these situations.

Appendix B - Troubleshooting

Details about installing and troubleshooting TCLink specific to your development platform can be found in the documentation included with the TCLink archive.

Once you are able to connect to the TC gateway, you should be able to diagnose parameter-related issues using the status, declinetype , and errortype parameters returned by the gateway. There is one response which indicates a more generic error, and that's an error of **cantconnect**. First, check the computer's network connection; can you ping machines on the Internet, by IP or by name?

The most common network connectivity error is that your target computer may be behind a firewall. TCLink uses the HTTPS port (443/tcp) for network communications; you can check whether this port is open from the target machine by typing "telnet vault.trustcommerce.com 443" at a UNIX command prompt, or else by loading a web browser on the target machine and attempting to visit https://vault.trustcommerce.com directly. If you timeout attempting to make the connection, but your Internet connection is working otherwise, then you may be firewalled. Speak to your network administrator about allowing outbound TCP traffic on port 443.

Another common problem is the lack of domain name (DNS) resolution. In some cases this will result in an errortype **dnsfailure**. (Not always; the TCLink software will sometimes fall back to hard coded IP addresses for established accounts. But this method is insecure and error-prone, and is used only as a last resort to keep a system with temporary DNS issues up and running during the outage.) The target machine must be able to resolve the trustcommerce.com domain; try typing "host trustcommerce.com" from a UNIX command prompt. If you don't get a response, or get an error, then the machine cannot resolve DNS information and TCLink will not be able to connect to the TC gateway. Speak to your sysadmin about making domain name resolution available to your target host.

Appendix C - Connecting via HTTPS POST

This method should only be used if a TCLink client install is not an option. It does not have the failover capability (and thus the processing uptime is not guaranteed to be 100%), or some of the enhanced security features of TCLink. In addition, transactions may be slightly slower, by an extra half second or so. It does, however support all other features available through TCLink, including all parameters and transaction types.

This is the URL:

https://vault.trustcommerce.com/trans/

The transaction should be sent as a standard POST, with CGI parameters, URL encoded. The parameters are otherwise identical to those used through TCLink.

Response parameters are returned as name-value pairs separated by newlines.

Appendix D - Currency Table

Currency Codes

Code	Currency Type	Code	Currency Type
usd	US Dollars	jpy	Japan Yen
eur	Euro	jod	Jordan Dinar
cad	Canadian Dollars	krw	Korea (South) Won
gbp	UK Pounds	lbp	Lebanon Pounds
dem	German Deutschemarks	luf	Luxembourg Francs
frf	French Francs	myr	Malaysia Ringgit
jpy	Japanese Yen	mxp	Mexico Pesos
nlg	Dutch Guilders	nlg	Netherlands Guilders
itl	Italian Lira	nzd	New Zealand Dollars
chf	Switzerland Francs	nok	Norway Kroner
dzd	Algeria Dinars	pkr	Pakistan Rupees
arp	Argentina Pesos	xpd	Palladium Ounces
aud	Australia Dollars	php	Philippines Pesos
ats	Austria Schillings	xpt	Platinum Ounces
bsd	Bahamas Dollars	plz	Poland Zloty
bbd	Barbados Dollars	pte	Portugal Escudo
bef	Belgium Francs	rol	Romania Leu

bmd	Bermuda Dollars	rur	Russia Rubles
brr	Brazil Real	sar	Saudi Arabia Riyal
bgl	Bulgaria Lev	xag	Silver Ounces
cad	Canada Dollars	sgd	Singapore Dollars
clp	Chile Pesos	skk	Slovakia Koruna
cny	China Yuan Renmimbi	zar	South Africa Rand
cyp	Cyprus Pounds	krw	South Korea Won
csk	Czech Republic Koruna	esp	Spain Pesetas
dkk	Denmark Kroner	xdr	Special Drawing Right (IMF)
nlg	Dutch Guilders	sdd	Sudan Dinar
xcd	Eastern Caribbean Dollars	sek	Sweden Krona
egp	Egypt Pounds	chf	Switzerland Francs
eur	Euro	twd	Taiwan Dollars
fjd	Fiji Dollars	thb	Thailand Baht
fim	Finland Markka	ttd	Trinidad and Tobago Dollars
frf	France Francs	trl	Turkey Lira
dem	Germany Deutsche Marks	gbp	United Kingdom Pounds
xau	Gold Ounces	usd	United States Dollars
grd	Greece Drachmas	veb	Venezuela Bolivar
hkd	Hong Kong Dollars	zmk	Zambia Kwacha
huf	Hungary Forint	eur	Euro
isk	Iceland Krona	xcd	Eastern Caribbean Dollars
inr	India Rupees	xdr	Special Drawing Right (IMF)
idr	Indonesia Rupiah	xag	Silver Ounces
iep	Ireland Punt	xau	Gold Ounces
ils	Israel New Shekels	xpd	Palladium Ounces
itl	Italy Lira	xpt	Platinum Ounces
jmd	Jamaica Dollars		

Appendix E - Input Field List

Input Fields

Name	Type	Minimum Length	Maximum Length	Service
custid	string	1	20	
password	string	1	20	
action	string	1	10	
media	string	1	10	
currency	string	3	3	
amount	number	3	8	
cc	number	13	16	
exp	number	4	4	
cvv	number	3	4	
routing	number	9	9	ACH
account	number	3	17	ACH
billingid	string	6	6	Citadel
verify	y/n	1	1	Citadel
transid	string	14	14	
avs	y/n	1	10	
name	string	1	60	
address1	string	1	80	
address2	string	1	80	
zip	string	1	20	
city	string	1	40	
state	string	1	20	
country	string	1	20	
phone	string	1	30	
email	string	1	50	
ip	string	7	15	
track1	string	1	79	Card Swipe
track2	string	1	40	Card Swipe

ticket	string	1	30	
operator	string	1	20	
shiptosame	y/n	1	1	
shipto_name	string	1	60	
shipto_address1	string	1	40	
shipto_address2	string	1	20	
shipto_city	string	1	20	
shipto_state	string	2	2	
shipto_zip	string	1	20	
shipto_country	string	1	20	
numitems	number	1	3	
price	string	1	20	
shippingcode	string	1	20	
shippinghandling	string	1	20	
productcode#	string	1	20	
quantity#	number	1	3	
price#	number	1	6	
tax#	number	1	6	
shippinghandling#	number	1	6	
wallet	y/n	1	1	TC Wallet
walletsize	string	3	8	TC Wallet
walletexposure	string	1	3	TC Wallet
start	string	2	10	Citadel
cycle	string	2	4	Citadel
payments	number	1	4	Citadel
authnow	y/n	1	1	Citadel
lastpaymentunstore	y/n	1	1	Citadel
fraudthreshold	number	1	2	CrediGuard (Scoring)
ip	number	7	15	CrediGuard

				(Blacklist)
demo	y/n	1	1	
offlineauthcode	string	6	6	