Beanstream Internet Commerce

Process Transaction API

Document Version 4.8
Last updated January 17, 2008

For further information please contact Beanstream customer support at (250) 472-2326 or support@beanstream.com.

Document Revision History

Date	Revision Description	Author	Revision Number
5/13/05	Updated section 1.7.2. Updated information about the rbFirstBilling Field.	Marcus Doty	2.0
06/01/05	Updated termURL	Jane Waite	2.1 and 2.2
06/03/05	Added reference to Transaction Reporting	Jane Waite	2.3
06/08/05	Updated section reference in 2.1.	Jane Waite	2.4
06/14/05	Added rbEndMonth	Jane Waite	2.5
07/28/05	Updated 1.5.7 Recurring Billing variables	Jane Waite	2.6
08/04/05	Updated province / state codes	Jane Waite	2.7
08/17/05	Updated test card numbers	Jane Waite	2.8
10/06/05	Updated errorPage request parameter description and sample code to indicate that the field is not required for Server to Server implementations	Shawn Gerty	3.0
10/11/05	Updated Ref variables for DBA.	Jane Waite	3.1
10/19/05	Added max size to ref variables.	Jane Waite	3.2
12/05	Updated Rec Billing API	Jane Waite	3.3
01/12/06	Updated section 1.8. - and removed extra Server to Server numbers	Marcus Doty / JW	3.4
03/03/06	Updated response codes and added rb response codes.	Jane Waite	3.5
03/13/06	Added trnType to list of response variables in section 1.5.7	Marcus Doty	3.52
03/29/06	Updated MessageId to 1-3N section 2.3.3	Jane Waite	3.6
03/31/06	Updated country list	Marcus Doty	3.7
05/10/06	Changed secure webspace reference to /secure/	Jane Waite	3.8
08/01/06	Added cvdId response parameter to indicate Credit Card CVD verification result. Added missing trnOrderNumber and responseType parameters to Server to Server response.	Shawn Gerty	3.9
08/30/06	Removed reference to enabling AVS in merchant account in section 1.5.7	Marcus Doty	4.0
09/18/06	Added references to XML server to server option in section 2.3.2 and section 2.3.3	Marcus Doty	4.1
12/7/06	Updated Server to Server response codes to include AVS and CAV.	Shawn Gerty	4.2
3/12/07	Remove section involving common declines from section 1.8	Marcus Doty	4.4
4/24/07	Added payment method to list of response variables in section 1.5.7	Marcus Doty	4.5
6/21/07	Updated section 1.3.3 reducing duplicate checking from 24 hrs to 1 hr.	Marcus Doty	4.6
8/29/07	Updated section 1.3.3 removing card expiry date from the criteria used for duplicate checking.	Marcus Doty	4.7
1/17/08	Update section 2.3.3, added additional server to server response variables	Marcus Doty	4.8

Overview

Transaction processing API allows you to link any e-commerce order processing system directly to the Beanstream transaction server. You may choose to integrate anything as simple as a one page order form, or as complex as a full shopping cart.

While other processing methods provide you with a web interface that may be built into your site, *Transaction Processing API* allows you to build your own web interface, thereby offering you the highest level of flexibility. With this interface you may process not only **purchase** transaction types (purchase, pre-auth), but also **adjustment** transaction types (return, void purchase, void return, pre-auth completion).

For additional security, programmers may wish to connect using Server to Server API. This more complex method of integration will allow all transaction information to be passed using a separate, secure browser. *Server to Server* reduces the chances for hackers to intercept data in transition and therefore protects your website against fraudulent transactions.

This document provides you with the data needed for both methods of integration.

Getting Started

The Process Transaction API is secured via 128-bit SSL encryption to prevent third parties from seeing the details of your customer's transaction request during transit to the Beanstream server. In order to access the Process Transaction API, you must have a merchant account with Beanstream and be able to communicate with our web server via 40-bit or 128-bit SSL.

1 Process Transaction API

1.1 The Customer Transaction

Transactions are submitted to the Beanstream transaction server via a standard HTTP POST or GET request. From start to finish, the transaction process takes the following form:

- ✓ A shopper browses your web site for products or services they would like to order.
- ✓ He or she completes the payment form located on either your website or the Beanstream server.
- ✓ All order information is submitted to be processed by Beanstream.
- ✓ ***The Beanstream server returns a web page to your customer's browser, where they must enter their VBV password.The password is verified and the results are sent to the Beanstream server.***
- ✓ The results of the transaction are returned to a web page that you specify.

* This step only occurs if Verified by Visa (VBV) is enabled for your account. In this case, customer credit cards are protected by a password which must be entered each time a new card is used. As VBV steps are transparent to the merchant, the integration steps outlined in this document are exactly the same for both VBV and non-VBV implementations.

1.2 POST vs GET

All information passed over an SSL connection is encrypted to prevent viewing by a third party. However, we recommend using POST rather than GET as the most secure method of connecting to our server. Data passed using GET is visible in the browser's address bar meaning it can be viewed by the user or the browser submitting data.

Another element to take into consideration is that POST requests have no limit to the amount of data that can be submitted. On the other hand, GET requests are limited by the browser to an average of 1 k of data. If a large transaction is submitted using GET (including shipping, billing and product information) some of the data may be truncated and the transaction will fail.

1.3 Submitting the Transaction

All transaction fields are populated and submitted to the transaction server at the following URL:
https://www.beanstream.com/scripts/process_transaction.asp

Transaction details are sent to this page as a set of field name/value pairs and submitted through either a form post or a query string. The call to process_transaction.asp will contain all of the information required to complete the transaction.

The order information is then posted to our processing script to complete the transaction. There are a number of required fields that must be entered in order to complete a transaction. The remaining optional fields are used for informational purposes and product tracking.

1.3.1 System Generated Errors

To help debug your *Process Transaction* online order form or shopping cart integration at the development stage, you may wish to set automatic transaction error messages. If a field contains invalid data, or if a transaction conflicts with your account settings, you will be notified immediately with a system generated message outlining any missing or conflicting information. In a properly working system, these messages will never be displayed to a shopper. Messages are displayed on the transaction processing error page and are not redirected back to the address specified by the **errorPage** variable.

Possible error messages include the following:

- ✓ Connection is not secure
- ✓ Invalid merchant ID
- ✓ Authorization Failed
- ✓ Missing transaction data
- ✓ Missing errorPage address

1.3.2 User Generated Errors

If any customer credit or billing information is invalid or missing, Process Transaction will redirect back to the address specified by the **errorPage** variable. All of the submitted information will be returned along with two error parameters:

errorMessage	Contains a descriptive message indicating all of the errors found on the order. Use this message to prompt customers to correct information on their order form.
errorFields	Contains a comma-separated list of all the field names that contain invalid data. Use this to customize your own error handling.

To detect if a system generated or user generated error has occurred refer to the errorType response field. The errorType field will contain one of the following values:

Value	**Description**
N	No System Generated or User Generated errors detected in the transaction request. The transaction has been passed to the bank for authorization.
S	A System Generated error has been detected in the transaction request.
U	A User Generated error has been detected in the transaction request.

If no errors are detected, the transaction will be processed and the messageText parameter will contain the response received from the bank.

Example

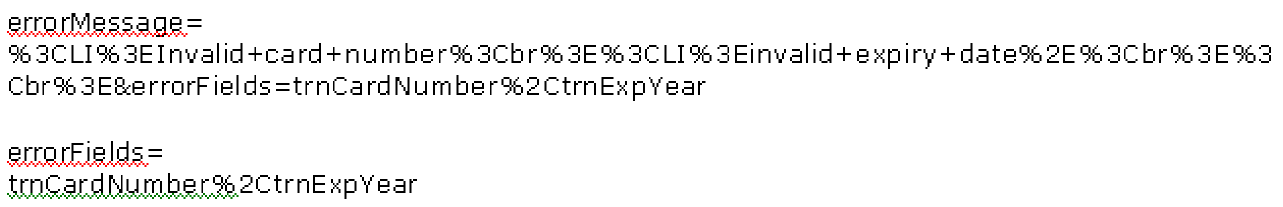
If:

- ✓ A process_transaction.asp is called with the errorPage variable https://www.beanstream.com/secure/sample/order_form.asp.

- ✓ A customer passes an invalid credit card number and expiry date.

Then:

The process_transaction.asp will redirect to https://www.beanstream.com/secure/sample/order_form.asp with the following additional two URL encoded variables:



```
errorMessage=
%3CLI%3EInvalid+card+number%3Cbr%3E%3CLI%3Einvalid+expiry+date%2E%3Cbr%3E%3
Cbr%3E&errorFields=trnCardNumber%2CtrnExpYear

errorFields=
trnCardNumber%2CtrnExpYear
```

Please note that in this example only the credit card number and expiry are invalid. As a result, these are the only two field names passed back ***in the* errorFields *parameter***.

Transaction Request:

https://www.beanstream.com/scripts/process_transaction.asp?errorPage=%2Fsamples%2Forder_form.asp&merchant_id=109040000&trnCardOwner=Paul+Randal&trnCardNumber=510012341234&trnExpMonth=01&trnExpYear=08&trnOrderNumber=2232&trnAmount=10.00&ordEmailAddress=prandal@mydomain.net&ordName=Paul+Randal&ordPhoneNumber=9999999&ordAddress1=1045+Main+Street&ordAddress2=&ordCity=Vancouver&ordProvince=BC&ordPostalCode=V8R+1J6&ordCountry=CA

Transaction Response:

https://www.beanstream.com/samples/order_form.asp?**errorMessage=%3CLI%3EInvalid+card+number%3Cbr%3E%3CLI%3EInvalid+expiry+date%2E%3Cbr%3E%3Cbr%3E&errorFields=trnCardNumber%2CtrnExpYear**&errorPage=%2Fsamples%2Forder_form.asp&merchant_id=109040000&trnCardOwner=Paul+Randal&trnCardNumber=510012341234&trnExpMonth=01&trnExpYear=08&trnOrderNumber=2232&trnAmount=10.00&ordEmailAddress=prandal@mydomain.net&ordName=Paul+Randal&ordPhoneNumber=9999999&ordAddress1=1045+Main+Street&ordAddress2=&ordCity=Vancouver&ordProvince=BC&ordPostalCode=V8R+1J6&ordCountry=CA

Addendum as of January 1 2002. In the case where a *User Generated Error* occurs, the system will not return the value of the trnCardNumber and trnCardExpiry fields to the errorPage URL. All other parameters will be returned as normal.

1.3.3 Checking for Duplicate Transactions

To minimize the possibility for error, Beanstream has created an automatic check for duplicate transactions. Beanstream will mark an item as a duplicate if the transaction amount, the transaction type, and the card number are identical to another transaction recorded within the hour.

If your system passes an *Order Number*, this will also be used in the duplicate check. If your system does not pass an order number, the unique transaction IDs will be mapped into the order number fields but will not be used in the duplicate check.

1.4 Transaction Completion

After order information has been validated, the transaction is passed to the bank for authorization. Depending on the status of the transaction, the customer is then directed to either the *Transaction Approved* or *Transaction Declined* URL which you have chosen. To set the approved/declined pages, go to *Administration → Account Setting → Order Settings.*

You may also wish to specify a Response Notification URL using the *Order Settings* module. This feature will send you an immediate transaction response. When enabled, the Beanstream system will post the transaction response to this URL. Use the same variables as for the approved/declined pages.

1.5 Data Variables

Use the variables listed in the following tables for your Process Transaction API.

Each table has five columns:

Field Name	The name of the form element used to store data. All required fields must be passed to the Transaction Processing script in order to complete the purchase.
Max Size	The maximum number of characters that may be contained in a field.
Data Type	The kind of characters to be used in a field. ***N***= numbers only ***A***= letters or a combination of numbers and letters ***$***= a currency in decimal form
Required	Identifies if a field is required. ***R*** = A required field for all transaction types ***O*** = An optional field for all Purchase Transaction Types ***P*** = A required field for Purchase (P) and Pre-Authorization (PA) transactions ***A*** = A required field for Adjustment Transaction only. Adjustment Transactions include all R, VP, VR, and PAC transactions which are processed against a previously completed transaction or authorization.
Description	Detailed description of the information which should be contained in each field.

Transaction type abbreviations:

P = Purchase	***R*** = Return	***VP*** = Void Purchase
VR = Void Return	***PA*** = Pre Authorization	***PAC*** = Pre Authorization Completion

1.5.1 Basic Input Variables

Please note that all input variables are case specific.

Field Name	**Max Size**	**Data Type**	**Required**	**Description**
merchant_id	9	N	R	The nine-digit Beanstream assigned identification

Field Name	**Max Size**	**Data Type**	**Required**	**Description**
				number.
trnOrderNumber	30	A	R	The order number of the shopper's purchase.

paymentMethod	2	A	O	This parameter is required for INTERAC Online transactions.
				IO = INTERAC Online
				CC = Credit Card (default value)

trnType	3	A	A	Indicates the type of transaction to perform (i.e. P,R,VP,VR, PA, PAC). This is an optional field. If omitted, this field will default to P. Please note that VP and VR are not valid transaction types for INTERAC Online.
trnLanguage	3	A	O	Three digit ISO language code. This value is passed back to the approval/decline page untouched.
errorPage	128	A	R	If any billing or credit card information is invalid or missing the customer will be redirected to this address along with the error message. This field is not required for Server to Server implementations.
username	16	A	A	These fields are mandatory if you have activated the username/password validation option located in the Beanstream *Order Settings* module. This option must be enabled in order to process R, VP, VR, and PAC transactions. Once enabled, these parameters are required for all transaction types including P and PA. Please note that VP and VR are not valid transactions types for INTERAC Online
password	16	A	A	
adjId	8	N	A	This field is required when the transaction type is an R, VP, VR, PAC. These all modify previous transactions that are identified by this adjustment ID. This number must be the trnId of the original transaction. Please note that VP and VR are not valid transaction types for INTERAC Online.
approvedPage	-	A	O	If a transaction request is approved, the customer's browser will be redirected to this URL. This parameter overrides the URL set within the *Order Settings* module of the Membership Area. There is no length restriction on this field.
declinedPage	-	A	O	If a transaction request is declined, the customer's browser will be redirected to this URL. This parameter overrides the URL set within the *Order Settings* module of the membership area. There is no length restriction on this field.
termURL	-	A/N	O	An extra parameter passed with VBV and INTERAC Online transactions. For INTERAC Online: This value must be https%3A%2F%2Fpayments.beanstream.com%2Fscripts%2Fprocess_transaction_auth.asp. For VBV: The customer will be redirected to the URL indicated in

Field Name	Max Size	Data Type	Required	Description
				this parameter after completing VBV authentication. The termURL variable must be encoded.
trnComments	8000	A	O	An optional comment that can be stored with the transaction.
cavEnabled	1	N	O	Pass this parameter to specify that you want the transaction verified by CAV before it is processed. This is applicable ***only*** if you have requested the CAV address verification service and you have the *Require CAV on all transactions* option turned ***off*** in Beanstream's membership area. Pass cavEnabled=1 to enable CAV, or cavEnabled=0 to disable this service.
cavPassCode	32	A/N	0	Pass this parameter if you have enabled CAV. This field is mandatory if you have entered a value for the access passcode option located in the Beanstream *CAV* module.
vbvEnabled	1	N	O	If you have requested Verified by Visa authentication service, pass this parameter to enable or disable VbV for the transaction. You must enable the username/password validation option in Beanstream's *Order Settings* module. Pass vbvEnabled=1 to enable VbV, or vbvEnabled=0 to disable this service.
scEnabled	1	N	O	If you have requested MasterCard SecureCode authentication service, pass this parameter to enable or disable SecureCode for the transaction. You must enable the username/password validation option in Beanstream's *Order Settings* module. Pass scEnabled=1 to enable SecureCode, or scEnabled=0 to disable this service.
cavServiceVersion	3	N	O	This variable may be passed if you have requested Canadian Address Verification only. If CAV has been enabled and no service version is passed, the default value will be 1.0. Please see the Beanstream Canadian Address Verification Guide for more information.

1.5.2 Credit Information

Field Name	Max Size	Data Type	Required	Description
trnCardOwner	64	A	P	Name of the card owner as seen on their credit card.
trnCardNumber	20	N	P	Customer credit card number
trnExpMonth	2	N	P	Customer credit card expiry month. ***Please note*** that expiry dates must be on or later than the current month and year. The month must be entered numerically (January = 01)
trnExpYear	2	N	P	Customer credit card expiry year. The year must be entered as a number less than 50. ***Please note***, if the year is the current year, then the month must be

Field Name	Max Size	Data Type	Required	Description
				greater than the current month.
trnCardCvd	4	N	O	Customer credit card CVD. This will be the last 3 digits (4 for Amex) on the back of the customer's credit card. This is for security purposes and is optional unless *Require Credit Card CVD* is enabled in the Beanstream *Order Settings* module.

1.5.3 Billing Information

Field Name	Max Size	Data Type	Required	Description
ordName	64	A	P	The primary contact name for the person/company listed in the billing information.
ordEmailAddress	64	A	P	The email address for the person/company listed as the primary contact in the billing information. This must be in a valid email address format: e.g. a@b.com.
ordPhoneNumber	32	A	P	The phone number for the primary contact as listed in the billing information.
ordAddress1	64	A	P	The billing address.
ordAddress2	64	A	O	An additional billing address field for long addresses
ordCity	32	A	P	The name of the city for billing information.
ordProvince	2	A	P	The billing province/state ID. See section 1.8.2 for a list of province/state codes.
ordPostalCode	16	A	P	Billing address postal/zip code
ordCountry	2	A	P	Billing address country ID. See section 1.8.1 for a list of valid country codes.

1.5.4 Shipping Information

Field Name	Max Size	Data Type	Required	Description
shipName	64	A	O	The order will be shipped to this person/company .
shipEmailAddress	64	A	O	Shipping contact's email address. This must be in a valid email format, for example: a@b.com
shipPhoneNumber	32	A	O	Shipping contact's phone number.
shipAddress1	64	A	O	Shipping address.
shipAddress2	64	A	O	Additional shipping address field for long addresses
shipCity	32	A	O	Shipping address city.
shipProvince	2	A	O	Shipping address province/state ID. See section 1.8.2 for a list of province and state codes.
shipPostalCode	16	A	O	Shipping address postal/Zip code
shipCountry	2	A	O	Shipping address country ID. See section 1.8.1 for a list of country codes.
shippingMethod	64	A	O	Description of the shipping method used for this Order.

Field Name	Max Size	Data Type	Required	Description
deliveryEstimate	9	N	O	Estimated delivery time in days.

1.5.5 Product and Pricing Information

Use these parameters if you wish to log product information with transaction details and include order specific information in customer email receipts. With the exception of trnAmount, these parameters are used only for reporting purposes and have no affect on the dollar amount charged to the card holder.

All parameters marked with an asterix are required if you have selected to "Validate orders against inventory" using the Beanstream Order Settings module (under Account Setting in the left menu of the Beanstream membership area). In this case, all product information must match the information stored in the Beanstream Inventory.

If you pass the parameters listed in this table:

- The Beanstream system will add the product to the merchant's Beanstream inventory if it does not already exist.
- Product information will be included on the Beanstream email receipt if the %productInfo% is included in the email template.
- Product information will be logged with the transaction and be viewable in the Beanstream Transaction Details screen.

Field Name	Max Size	Alpha/ Numeric / Currency	Required	Description
prod_id_n *	32	A	O	The product ID or SKU number used to uniquely identify a product.
prod_name_n	64	A	O	Description of the ordered product.
prod_quantity_n *	9	N	O	Quantity of the ordered product.
prod_shipping_n	9	$	O	Shipping cost of the product.
prod_cost_n	9	$	O	Purchase cost of the product.
ordItemPrice *	9	$	O	The total price of all items on the order, taking into account product quantities.
ordShippingPrice *	9	$	O	Order shipping charges. This includes the individual product shipping charges, if any.
ordTax1Price *	9	$	O	Total amount of tax1 charged to the order.
ordTax2Price *	9	$	O	Total amount of tax2 charged to the order.
trnAmount	9	$	R	Total order amount. This should be equal to the sum of the ordItemPrice, ordServicePrice, ordShippingPrice, ordTax1Price, and ordTax2Price. Must be a positive numeric value.

There is no limit to the number of product informational fields that may be included with your order. All field names must be numbered from 1-n (ie. prod_id_1, prod_quantity_1, prod_id_2, prod_quantity_2, etc.).

1.5.6 Custom Information

Field Name	Max Size	Alpha/ Numeric/ Currency	Required	Description
refn	256	A	O	Up to five reference fields may be passed along with the order to contain any site-specific information you need to maintain during the order process. When this information is sent out to the processing script, it is stored in Beanstream's database along with the details of the transaction and returned back to your site unmodified. These fields may be used to maintain information such as shopper IDs, account numbers, order numbers, or any other tracking information you may require. These field names must be numbered from 1 to 5 in the format ref1, ref2, ref3, ref4 and ref5. *Please note that only ref1, ref2, ref3 and ref4 may be used for Dynamic DBA.

1.5.7 Response Variables

The following fields are returned upon completion of a transaction:

Name	Max Size	Alpha/ Numeric/ Currency	Description
trnId	8	N	Unique ID number used to identify an individual transaction
messageId	3	N	The ID number of the transaction response message.
messageText	128	A	The text message associated with the messageID
authCode	6	A	If the transaction has been approved, this parameter will contain the authorization code returned from the bank. If the transaction has been declined, the parameter will contain no value. The authorization code is the unique transaction identifier assigned by the bank. The authorization code must be displayed to the card holder when a transaction is complete.
responseType	1	A	This variable will always return a responseType of 'T' to indicate that a transaction has been completed.
trnAmount	9	$	The amount of the transaction
trnDate	20	A	The date and time that the transaction was processed
trnOrderNumber	10	N	The order number associated with the transaction
trnLanguage	3	A	Contains the value of the trnLanguage field submitted in the transaction request.

Name	Max Size	Alpha/ Numeric/ Currency	Description
trnCustomerName	32	A	Contains the value of the trnCardOwner field submitted in the transaction request.
trnEmailAddress	64	A	Contains the value of the ordEmailAddress field submitted in the transaction request.
trnPhoneNumber	32	A	Contains the value of the ordPhoneNumber field submitted in the transaction request.
rspCodeCav rspCavResult rspCodeCredit1 rspCodeCredit2 rspCodeCredit3 rspCodeCredit4 rspCodeAddr1 rspCodeAddr2 rspCodeAddr3 rspCodeAddr4 rspCodeDob rspCodeSafeScan rspCodeSafeScanId	-	-	These variables will be returned if you have applied for Canadian Address Verification. If cavServiceVersion = 1.0 then only one credit card and one address response code will be returned as variables rspCodeCredit and rspCodeAddr, rather than four each. If cavServiceVersion = 1.1 then four numbered rspCodeCredit and rspCodeAddr codes will be returned. rspCodeDob will only be returned for cavServicVersion 1.2 or greater. For more information, please consult our CAV documentation.
avsProcessed	1	N	Set to a value of 1 if the issuing bank has successfully processed an AVS check on the transaction. Set to a value of 0 if no AVS check has been performed.
avsId	1	A	The id number of the AVS response message. See section 1.6.3 for a listing of possible values.
avsResult	1	N	Set to a value of 1 if AVS has been validated with both a match against address and a match against postal/ZIP code.
avsAddrMatch	1	N	Set to a value of 1 if the ordAddress1 parameter matches the consumers address records at the issuing bank. Set to a value of 0 if the ordAddress1 parameter does not match the customer’s address records or if AVS was not processed for the transaction.
avsPostalMatch	1	N	Set to a value of 1 if the ordPostalCode parameter matches the consumers address records at the issuing bank. Set to a value of 0 if the ordPostalCode parameter does not match the customer’s address records or if AVS was not processed for the transaction.
avsMessage	128	A	Set to the value of the text message associated with the avsId response code.
cvdId	1	N	The id number of the CVD response message. This parameter will be returned if the Credit Card CVD value has been submitted in the transaction request. See section 1.6.4 for a listing of possible values.
trnType	3	A	The original value sent to indicate the type of transaction to perform (i.e. P,R,VP,VR, PA, PAC).
ref1	256	A	The value of the ref1 field submitted in the transaction request.
ref2	256	A	The value of the ref2 field submitted in the transaction request.
ref3	256	A	The value of the ref3 field submitted in the transaction request.
ref4	256	A	The value of the ref4 field submitted in the transaction request.

Name	Max Size	Alpha/ Numeric/ Currency	Description
ref5	256	A	The value of the ref5 field submitted in the transaction request (not to be used for Dynamic DBA).
paymentMethod	2	A	The payment method used by the customer to complete the transaction. (i.e. IO, CC …).
rbAccountId	4	N	The identification number of the recurring billing profile. This value is only returned upon creation of the account.
ioConfCode	15	A	In an INTERAC Online transaction this confirmation number will be returned by the customer's financial institution if the transaction has been processed successfully. This value must be displayed to the customer on a transaction confirmation page.
ioInstName	30	A	The name of the customer's financial institution. This value is retuned for INTERAC Online transactions only.

1.6 Variable Codes

1.6.1 Country Codes

ID	Name	ID	Name	ID	Name
AF	Afghanistan	GE	Georgia	MP	Northern Mariana Islands
AR	Argentina	DE	Germany	NO	Norway
AX	Åland Islands	GH	Ghana	OM	Oman
AL	Albania	GI	Gibraltar	PK	Pakistan
DZ	Algeria	GB	Great Britain	PW	Palau
AS	American Samoa	GR	Greece	PS	Palestinian Territory, Occupied
AD	Andorra	GL	Greenland	PA	Panama
AO	Angola	GD	Grenada	PG	Papua New Guinea
AI	Anguilla	GP	Guadeloupe	PY	Paraguay
AQ	Antarctica	GU	Guam	PE	Peru
AG	Antigua and Barbuda	GT	Guatemala	PH	Philippines
AM	Armenia	GN	Guinea	PN	Pitcairn
AW	Aruba	GW	Guinea Bissau	PL	Poland
AU	Australia	GY	Guyana	PT	Portugal
AT	Austria	HT	Haiti	PR	Puerto Rico
AZ	Azerbaijan	HM	Heard and McDonald Islands	QA	Qatar
BS	Bahamas	HN	Honduras	RE	Reunion
BH	Bahrain	HK	Hong Kong	RO	Romania
BD	Bangladesh	HU	Hungary	RU	Russian Federation
BB	Barbados	IS	Iceland	RW	Rwanda
BY	Belarus	IN	India	KN	Saint Kitts and Nevis
BE	Belgium	ID	Indonesia	LC	Saint Lucia
BZ	Belize	IR	Iran, Islamic Republic of	VC	Saint Vincent and the Grenadines
BJ	Benin	IQ	Iraq	WS	Samoa
BM	Bermuda	IE	Ireland	SM	San Marino
BT	Bhutan	IL	Israel	ST	Sao Tome and Principe
BO	Bolivia	IT	Italy	SA	Saudi Arabia

ID	Name	ID	Name	ID	Name
BA	Bosnia and Herzegovina	JM	Jamaica	SN	Senegal
BW	Botswana	JP	Japan	CS	Serbia and Montenegro
BV	Bouvet Island	JO	Jordan	SC	Seychelles
BR	Brazil	KZ	Kazakhstan	SL	Sierra Leone
IO	British Indian Ocean Territory	KE	Kenya	SG	Singapore
BN	Brunei Darussalam	KI	Kiribati	SK	Slovakia
BG	Bulgaria	KP	Korea, Democratic People's Republic	SI	Slovenia
BF	Burkina Faso	KR	Korea, Republic of	SB	Solomon Islands
BI	Burundi	KW	Kuwait	SO	Somalia
KH	Cambodia	KG	Kyrgyzstan	ZA	South Africa
CM	Cameroon	LA	Lao People's Democratic Republic	GS	South Georgia – South Sandwich Islands
CA	Canada	LV	Latvia	ES	Spain
CV	Cape Verde	LB	Lebanon	LK	Sri Lanka
KY	Cayman Islands	LI	Liechtenstein	SH	St. Helena
CF	Central African Republic	LS	Lesotho	PM	St. Pierre and Miquelon
TD	Chad	LR	Liberia	SD	Sudan
CL	Chile	LY	Libyan Arab Jamahiriya	SR	Suriname
CN	China	LT	Lithuania	SJ	Svalbard and Jan Mayen
CX	Christmas Island	LU	Luxembourg	SZ	Swaziland
CC	Cocos (Keeling) Islands	MO	Macau	SE	Sweden
CO	Columbia	MK	Macedonia, Former Yugoslav Republic of	CH	Switzerland
KM	Comoros	MG	Madagascar	SY	Syrian Arab Republic
CG	Congo	MW	Malawi	TW	Taiwan
CD	Congo, The Democratic Republic of the	MY	Malaysia	TJ	Tajikistan
CK	Cook Islands	MV	Maldives	TZ	Tanzania, United Republic of
CR	Costa Rica	ML	Mali	TH	Thailand
CI	Cote d'Ivoire – Really Ivory Coast	MT	Malta	TG	Togo
HR	Croatia	MH	Marshall Islands	TK	Tokelau
CU	Cuba	MQ	Martinique	TO	Tonga
CY	Cyprus	MR	Mauritania	TT	Trinidad and Tobago
CZ	Czech Republic	MU	Mauritius	TN	Tunisia
DK	Denmark	YT	Mayotte	TR	Turkey
DJ	Djibouti	MX	Mexico	TM	Turkmenistan
DM	Dominica	FM	Micronesia, Federated States of	TC	Turks and Caicos Islands
DO	Dominican Republic	MD	Moldova, Republic of	TV	Tuvalu
TL	East Timor	MC	Monaco	UG	Uganda
EC	Ecuador	MN	Mongolia	UA	Ukraine
EG	Egypt	MS	Montserrat	AE	United Arab Emirates
SV	El Salvador	MA	Morocco	US	United States

ID	Name	ID	Name	ID	Name
GQ	Equatorial Guinea	MZ	Mozambique	UM	United States Minor Outlying Islands
ER	Eritrea	MM	Myanmar	UY	Uruguay
EE	Estonia	NA	Namibia	UZ	Uzbekistan
ET	Ethiopia	NR	Nauru	VU	Vanuatu
FK	Falkland Islands (Malvinas)	NP	Nepal	VA	Vatican City state
FO	Faroe Islands	NL	Netherlands	VE	Venezuela
FJ	Fiji	AN	Netherlands Antilles	VN	Viet Nam
FI	Finland	NC	New Caledonia	VG	Virgin Islands (British)
FR	France	NZ	New Zealand	VI	Virgin Islands (US)
GF	French Guiana	NI	Nicaragua	WF	Wallis and Futuna
PF	French Polynesia	NE	Niger	EH	Western Sahara
TF	French Southern Territories	NG	Nigeria	YE	Yemen
GA	Gabon	NU	Niue	ZM	Zambia
GM	Gambia	NF	Norfolk Island	ZW	Zimbabwe

1.6.2 Province/State Codes

ID	Name	ID	Name	ID	Name
AB	Alberta	ME	Maine	PR	Puerto Rico
AK	Alaska	MI	Michigan	QC	Quebec
AL	Alabama	FM	Micronesia	RI	Rhode Island
AS	American Samoa	MN	Minnesota	SC	South Carolina
AR	Arkansas	MO	Missouri	SD	South Dakota
AZ	Arizona	MS	Mississippi	SK	Saskatchewan
BC	British Columbia	MT	Montana	TN	Tennessee
CA	California	NB	New Brunswick	TX	Texas
CO	Colorado	NC	North Carolina	UT	Utah
CT	Connecticut	ND	North Dakota	VA	Virginia
DC	District of Columbia	NE	Nebraska	VI	Virgin Islands
DE	Delaware	NL	Newfoundland/Labrador	VT	Vermont
FL	Florida	NH	New Hampshire	WA	Washington
GA	Georgia	NJ	New Jersey		
GU	Guam	NM	New Mexico	WI	Wisconsin
HI	Hawaii	NS	Nova Scotia	WV	West Virginia
IA	Iowa	NT	Northwest Territories	WY	Wyoming
ID	Idaho	NU	Nunavut	YT	Yukon
IL	Illinois	NV	Nevada	--	Outside U.S./Canada
IN	Indiana	NY	New York		
KS	Kansas	OH	Ohio		
KY	Kentucky	OK	Oklahoma		
LA	Louisiana	ON	Ontario		
MA	Massachusetts	OR	Oregon		
MB	Manitoba	PA	Pennsylvania		
MD	Maryland	PE	Prince Edward Island		

1.6.3 AVS Response Codes

ID	Result	Processed	Address Match	Postal/ZIP Match	Message
0	0	0	0	0	Address Verification not performed for this transaction.
5	0	0	0	0	Invalid AVS Response.
9	0	0	0	0	Address Verification Data contains edit error.
A	0	1	1	0	Street address matches, Postal/ZIP does not match.
B	0	1	1	0	Street address matches, Postal/ZIP not verified.
C	0	1	0	0	Street address and Postal/ZIP not verified.
D	1	1	1	1	Street address and Postal/ZIP match.
E	0	0	0	0	Transaction ineligible.
G	0	0	0	0	Non AVS participant. Information not verified.
I	0	0	0	0	Address information not verified for international transaction.
M	1	1	1	1	Street address and Postal/ZIP match.
N	0	1	0	0	Street address and Postal/ZIP do not match.
P	0	1	0	1	Postal/ZIP matches. Street address not verified.
R	0	0	0	0	System unavailable or timeout.
S	0	0	0	0	AVS not supported at this time.
U	0	0	0	0	Address information is unavailable.
W	0	1	0	1	Postal/ZIP matches, street address does not match.
X	1	1	1	1	Street address and Postal/ZIP match.
Y	1	1	1	1	Street address and Postal/ZIP match.
Z	0	1	0	1	Postal/ZIP matches, street address does not match.

1.6.4 CVD Response Codes

ID	Message
1	CVD Match
2	CVD Mismatch
3	CVD Not Verified
4	CVD Should have been present
5	CVD Issuer unable to process request
6	CVD Not Provided

1.7 Recurring Transactions

In certain cases you may wish to schedule regular billing for a customer. Companies that charge monthly service fees (for example internet service providers or magazine distributors) are among those who could benefit from this option. A recurring billing account can be created for any successful transaction that is processed and approved by the issuing bank.

To create a recurring billing account for a card holder, you must pass additional parameters indicating billing frequency, pricing and other options. Please note that you must have the recurring billing option enabled by Beanstream or these fields will be ignored.

1.7.1 Billing Frequency

Set a schedule for billing a client using the rbBillingPeriod and rbBillingIncrement fields. The billing period determines how you will calculate pay periods. Set this variable to days, weeks, months, or years. The billing increment determines how often you will charge an account.

For example IF:

- ✓ The billing period is set to weeks
- ✓ The billing increment is set to 2

Your customer will be billed every two weeks.

1.7.2 Billing Dates

Specify exact billing dates using the rbFirstBilling and rbSecondBilling parameters.

rbFirstBilling :

Use this parameter to indicate the date that a recurring billing account first becomes active. The first billing date can be set to any date in the future and to any date in the past up to one billing frequency minus a day. For example, if you have chosen to bill every week you cannot set the first billing date more than six days prior to the current system date. If you do not pass a value for this parameter, the first billing date will default to the current date in the merchant's time zone. The time zone is determined by the *Company Time Zone* setting in the merchant's *Company Information* module.

rbSecondBilling :

The parameter rbSecondBilling is the date that the second billing period begins. This second billing date is used when an account is created partway through a billing period. As the merchant, you will need to charge the customer only a partial bill for the first billing period. Regular payments will begin with the second billing period.

For example, a merchant decides to create a recurring billing account set with a $30 monthly fee charged on the first of the month. For any account created part way through the month, the merchant will want to pro-rate the customer's first payment. This means that he will want to charge a partial fee from the time the account is created to the end of the month. For an account created with a first billing date of April 20 and a second billing date of May 1, the system will calculate the pro-rated amount and charge $10.00 for the remaining 10 days of April and then the full $30 on May 1st. The customer will continue to be charged the full $30 on the 1st of each subsequent month.

If you do not pass a value for this parameter, the default setting for the second billing period will be the first billing date plus the billing frequency you have chosen. For example, if the first billing date is set at April 20 and you have chosen to bill once every month, then default settings for the second billing date will be May 20.

1.7.3 Recurring Billing Data Variables

The following data variables are used for flagging a transaction request as a *Recurring Transaction* and for setting the billing frequency. ***Please note*** that these variables must be entered in addition to those listed in section 1.5.

Any field formatting errors that are detected in the recurring billing parameters will be handled as *User Generated Errors* (see section 1.3.2).

Field Name	Max Size	Alpha/ Numeric/ Currency	Required	Description
trnRecurring	1	N	O	Flags a transaction as a recurring item. Enter 1 for a recurring transaction. The default value of this field is 0, indicating that the request is not a recurring transaction. All items with a * must be included if this field is set to 1.
rbBillingPeriod	1	A	O*	To bill clients according to periods of : Days enter D Weeks enter W Months enter M Years enter Y.
rbBillingIncrement	-	N	O*	Choose how often you will charge a client (every two weeks, every nine days).
rbFirstBilling	8	N	O	Set to the first recurring billing collection date. See section 1.7.1 for details. This value must be formatted as MMDDYYYY. November 26, 2003 would be passed as 11262003.
rbSecondBilling	8	N	O	Set to the second recurring billing collection date. See Section 1.7.1. This value must be formatted as MMDDYYYY. If your first billing was September 1, 2004 and you have chosen to bill once a month, this value would be 10012004 by default. If your first billing date did not fall within the regular schedule that you wish to set (for a customer signed on mid-month rather than at the beginning) you may set the second billing date to one other than the default settings. All subsequent intervals will be counted after this date.
rbCharge	1	N	O	Set rbCharge=0 to create a recurring billing account without processing a Purchase or Pre-auth transaction. The customer will be charged on the date entered for rbFirstBilling.
rbExpiry	8	N	O	Set to the date that the recurring billing account will expire. If no value is passed the account will be flagged to never expire.
rbEndMonth	1	N	O	Set this variable to 1 to charge a customer on the last day of the month for the chosen recurring billing cycle. For example, if you bill every three months, this setting will allow you to bill Jan 31, April 30, July 31 etc. If set as 1, the value of your rbBillingPeriod must be set as "M" for monthly billing.
rbApplyTax1		N	O	If set to the value 1, tax1 will be applied to all recurring billing payments. If set to a value of 0, no tax1 charges will be applied. Default value of this field is 0.

Field Name	Max Size	Alpha/ Numeric/ Currency	Required	Description
rbApplyTax2		N	O	If set to the value 1, tax2 will be applied to all recurring billing payments. If set to a value of 0, no tax2 charges will be applied. Default value of this field is 0.

* Field required if trnRecurring is passed with a value of 1.

1.7.4 Recurring Billing Response Variables

The following variables will be returned to the URL specified in the Recurring Billing Response Notification field in the Beanstream Order Settings module. To add or make changes to the Recurring Billing Response URL, log into the Beanstream membership area at www.beanstream.com and go to *Administration → Account Settings → Order Settings* in the left menu.

Field Name	Max Size	Alpha/ Numeric/ Currency	Description
billingId	4	N	The identification number for the recurring billing account. This number references rbAccountId. ***Note:** this is the variable name returned to the Recurring Billing Response Notification URL. At the time of creation of a new recurring billing account, the variable name returned in the transaction response is **rbAccountId** (see 1.5.7)
trnApproved	4	N	Indicates if the transaction was approved or declined. 1=Approved, 0=Declined.
trnId	8	N	Unique ID number used to identify an individual transaction
messageId	3	N	The ID number of the transaction response message.
messageText	128	A	The text message associated with the messageID
authCode	6	A	If the transaction has been approved, this parameter will contain the authorization code returned from the bank. If the transaction has been declined, the parameter will contain no value. The authorization code is the unique transaction identifier assigned by the bank. The authorization code must be displayed to the card holder when a transaction is complete.
accountName	32	A	The billing account name.
emailAddress	64	A/N	The billing email address.
billingAmount	13	N	The current billing amount of the recurring transaction.

Field Name	Max Size	Alpha/ Numeric/ Currency	Description
billingDate	10	N	The date in the recurring period on which the transaction is billed.
billingPeriod	1	N	Indicates how the billing increment is calculated. D = periods of days, M = periods of months, Y = periods of years.
billingIncrement	-	N	The amount by which the billing period is incremented. If the billing period is D and the billing increment is 7, the customer is billed every seven days.
periodFrom	10	N	The start date of the recurring transaction period.
periodTo	10	N	The end date of the recurring transaction period.
ref1	256	A	The value of the ref1 field submitted in the transaction request.
ref2	256	A	The value of the ref2 field submitted in the transaction request.
ref3	256	A	The value of the ref3 field submitted in the transaction request.
ref4	256	A	The value of the ref4 field submitted in the transaction request.
ref5	256	A	The value of the ref5 field submitted in the transaction request (not to be used for Dynamic DBA).

1.7.5 Modifying / Cancelling Recurring Billing Files

To modify or cancel recurring billing files, using a custom interface or automated process, an HTTPS POST must be sent to https://www.beanstream.com/scripts/recurring_billing.asp. The API will verify all input parameters and return a response message. Review the sample string shown below:

https://www.beanstream.com/scripts/recurring_billing.asp?serviceVersion=1.0&operationType=M&merchantId=merchantId&passCode=passCode&rbAccountId =1244&amount=12.00

Field Name	Max Size	Alpha/ Numeric/ Currency	Required	Description
serviceVersion	4	A/N	R	The recurring billing service version being used. The current version is 1.0.

operationType	1	A	R	The type of operation to be performed on the recurring billing file.
				
				M = Modify
				C = Cancel

merchantId	9	N	R	The merchant's unique Beanstream identification number.

passCode	32	A/N	R	A unique password assigned allowing the user to update and modify recurring billing items. Passcodes can be generated through the Beanstream Order Settings module.
rbAccountID	4		R	The recurring billing ID number. This variable is passed to indicate an item that is to be updated. The value can be found in the recurring billing profile in the Beanstream Membership area.
amount		C	O	The new amount for the recurring billing item.
ref(1-5)	256	A/N	O	Five reference variables may be modified according to individual needs

The response returned will be an xml response containing a numeric response code and message related to that code.

Code	Message
1	"Request successful"
2	"Secure connection required."
3	"Service version not supported"
4	"Invalid login credentials"
5	"Invalid operation type"
6	"Invalid amount value
7	"Invalid recurring billing account id"
8	"Merchant account is closed or disabled"
9	"Invalid XML message"
10	"Unexpected error"

1.7.6 Modifying Recurring Billing Files Using SOAP

The *Web Services Description Language* (WSDL) and *Web Services Meta Language* (WSML) files are located at the following URLs:

http://www.beanstream.com/soap/recurring_billing.wsdl
http://www.beanstream.com/soap/recurring_billing.wsml

Transaction requests are made with a single XML message passed to the *SendRequest* method. Sample XML messages for requests and responses are shown below.

Sample Request XML Message

```
<recurring_billing>
        <merchantId>109040000</merchantId>
        <passCode>testing123</passCode>
        <serviceVersion>1.0</serviceVersion>
        <rbAccountId>7291</rbAccountId>
        <operationType>M</operationType>
        <ref1>test 1</ref1>
        <amount>2.25</amount>
</recurring_billing>
```

Sample Response XML Message - Approved

```
<response>
   <code>1</code>
   <message>Request successful</message>
</response>
```

Sample Response XML Message - Declined

```
<response>
   <code>7</code>
   <message>Invalid recurring billing account id</message>
</response>
```

1.8 Test Card Numbers

Several test credit card numbers are available to test your integration. Use these test card numbers to emulate the full transaction process, from submitting your payment request to receiving your payment response.

These test card numbers are only active while your merchant account is in test mode. Test card numbers are listed below, along with their expected responses. Any expiry date that is equal to or later than the current month and year or later is valid.

Visa		
Approved	4030000010001234	**Expiry dates:** Please use any expiry date in the future. **CVV/CVV2:** Please use any CVV/CVV2 number.
Approved $100 Limit	4504481742333	
Approved VBV	4123450131003312 passcode 12345	
Declined	4003050500040005	
MasterCard		
Approved	5100000010001004	
Approved	5194930004875020	
Approved	5123450000002889	
Approved 3D Secure	5123450000000000 passcode 12345	
Declined	5100000020002000	
Amex		
Approved	371100001000131	
Declined	342400001000180	
Discover		
Approved	6011500080009080	
Declined	6011000900901111	

In order to test VbV, Beanstream must enable a special service on your account. Please email support@beanstream.com requesting that VbV testing be enabled on your account prior to testing. We can only enable this service for merchants who have already signed and submitted the VbV agreement

1.9 URL Encoding Chart

The following chart shows the URL-encoded equivalent for every character in the ASCII set. Items in **bold** must be represented with their encoded equivalents when used in a URL - all others are optional.

Code		Code		Code		Code		Code		Code	
%00		%30	0	%60	`	%90		%C0	À	%F0	ð
%01		%31	1	%61	a	%91	‘	%C1	Á	%F1	ñ
%02		%32	2	%62	b	%92	’	%C2	Â	%F2	ò
%03		%33	3	%63	c	%93	“	%C3	Ã	%F3	ó
%04		%34	4	%64	d	%94	”	%C4	Ä	%F4	ô
%05		%35	5	%65	e	%95	•	%C5	Å	%F5	õ
%06		%36	6	%66	f	%96	–	%C6	Æ	%F6	ö
%07		%37	7	%67	g	%97	—	%C7	Ç	%F7	÷
%08		%38	8	%68	h	%98	˜	%C8	È	%F8	ø
%09	**Tab**	%39	9	%69	i	%99	™	%C9	É	%F9	ù
%0A	**LF**	%3A	:	%6A	j	%9A	š	%CA	Ê	%FA	ú
%0B		%3B	;	%6B	k	%9B	›	%CB	Ë	%FB	û
%0C		%3C	<	%6C	l	%9C	œ	%CC	Ì	%FC	ü
%0D	**CR**	%3D	=	%6D	m	%9D		%CD	Í	%FD	ý
%0E		%3E	>	%6E	n	%9E		%CE	Î	%FE	þ
%0F		**%3F**	**?**	%6F	o	%9F	Ÿ	%CF	Ï	%FF	ÿ
%10		%40	@	%70	p	%A0		%D0	Ð		
%11		%41	A	%71	q	%A1	¡	%D1	Ñ		
%12		%42	B	%72	r	%A2	¢	%D2	Ò		
%13		%43	C	%73	s	%A3	£	%D3	Ó		
%14		%44	D	%74	t	%A4	¤	%D4	Ô		
%15		%45	E	%75	u	%A5	¥	%D5	Õ		
%16		%46	F	%76	v	%A6	¦	%D6	Ö		
%17		%47	G	%77	w	%A7	§	%D7	×		
%18		%48	H	%78	x	%A8	¨	%D8	Ø		
%19		%49	I	%79	y	%A9	©	%D9	Ù		
%1A		%4A	J	%7A	z	%AA	ª	%DA	Ú		
%1B		%4B	K	%7B	{	%AB	«	%DB	Û		
%1C		%4C	L	%7C	|	%AC	¬	%DC	Ü		
%1D		%4D	M	%7D	}	%AD		%DD	Ý		
%1E		%4E	N	%7E	~	%AE	®	%DE	Þ		
%1F		%4F	O	%7F		%AF	¯	%DF	ß		
%20	**space**	%50	P	%80		%B0	°	%E0	à		
%21	!	%51	Q	%81		%B1	±	%E1	á		
%22"	**"**	%52	R	%82	‚	%B2	²	%E2	â		
%23	#	%53	S	%83	ƒ	%B3	³	%E3	ã		
%24	$	%54	T	%84	„	%B4	´	%E4	ä		
%25	**%**	%55	U	%85	…	%B5	µ	%E5	å		
%26	**&**	%56	V	%86	†	%B6	¶	%E6	æ		
%27	'	%57	W	%87	‡	%B7	·	%E7	ç		
%28	(	%58	X	%88	ˆ	%B8	¸	%E8	è		
%29	)	%59	Y	%89	‰	%B9	¹	%E9	é		
%2A	*	%5A	Z	%8A	Š	%BA	º	%EA	ê		
%2B	**+**	%5B	[	%8B	‹	%BB	»	%EB	ë		
%2C	,	%5C	\	%8C	Œ	%BC	¼	%EC	ì		
%2D	-	%5D	]	%8D		%BD	½	%ED	í		
%2E	.	%5E	^	%8E		%BE	¾	%EE	î		
%2F	/	%5F _	_	%8F		%BF	¿	%EF	ï		

1.10 Examples

The following is an example of how to call the Process Transaction API using the transaction parameters listed below:

Merchant Information:

Beanstream Merchant ID	=	123456789
Your Order Number	=	288
Transaction Username	=	ab12345678901234
Transaction Password	=	cd56789012345678
Transaction Type	=	P
Error Page to Handle Errors	=	https://www.beanstream.com/secure/sample/order_form.asp

Customer Information:

Name	=	Bob Buyer
Email Address	=	bob@buyer.com
Phone Number	=	555-1234
Address1	=	1234 Main St
City	=	Victoria
Province	=	BC
Postal Code	=	V8T5J1
Country	=	CA
Card Owner	=	Bob D Buyer
Card Number	=	5100000010001004
Card Expiry Month	=	10
Card Expiry Year	=	03

Optional product information where two products are sold is:

First Product Name	=	ACER AcerPower SE Minitower
First Product ID or SKU	=	SYS-ACER
First Product Quantity	=	3
First Product Unit Price	=	$799.95
First Product Extended Price	=	$2399.85
First Product Tax 1	=	$168.00
First Product Tax 2	=	$168.00
First Product Shipping Cost	=	$45.60
Second Product Name	=	12/4/32x CD Rocket Mach 12 ReWriter
Second Product ID or SKU	=	STR-WRITER
Second Product Quantity	=	2
Second Product Unit Price	=	$389.95
Second Product Extended Price	=	$779.90
Second Product Tax 1	=	$54.60
Second Product Tax 2	=	$54.60
Second Product Shipping Cost	=	$12.25

Total Order Information:

Total Shipping Charge	=	$156.80	
Total Tax 1	=	$177.57	
Total Tax 2	=	$177.57	
Total Product Cost	=	$3237.60	(Including Shipping, Less Taxes)
Grand Total Cost	=	$3691.69	(Including Taxes and Shipping)

The call to the Process Transaction API would look like:

https://www.beanstream.com/scripts/process_transaction.asp?merchant_Id=123456789&username=ab12345678901234&password=cd56789012345678&trnOrderNumber=288&errorPage=https://www.beanstream.com/samples/order_form.asp&prod_id_1=SYS-ACER&prod_upc_1=SYS-CER&prod_name_1=ACER+AcerPower+SE+Minitower&prod_quantity_1=3&prod_cost_1=799.95&prod_id_2=STR-WRITER&prod_upc_2=STR-WRITER&prod_name_2=12/4/32x+CD+Rocket+Mach+12+ReWriter&prod_quantity_2=2&prod_cost_2=389.95&ordItemPrice=3179.75&ordShippingPrice=156.80&ordTax1Price=177.57&ordTax2Price=177.57&trnType=P&trnAmount=3691.69&trnCardOwner=Bob%20D%20Buyer&trnCardNumber=5100000010001004&trnExpMonth=10&trnExpYear=08&ordName=Bob%20Buyer&ordEmailAddress=bob@buyer.com&ordPhoneNumber=555-1234&ordAddress1=1234%20Main%20St&ordCity=Victoria&ordProvince=BC&ordPostalCode=V8T5J1&ordCountry=CA

Note that this URL has been URL-encoded. Refer to section 1.11 for a full listing of characters and their URL-encoded equivalents.

1.11 Transaction Processing with PGP

PGP is a key security feature available for use with Process Transaction.

PGP signing will take your public key and transaction data and generate a hash value. This hash value is wrapped around your transaction request and is ready to be submitted to Beanstream for processing. If either the transaction data or hashed signature is modified, Beanstream will not be able to identity the signature and will decline the request.

PGP encryption creates an added level of security so that sensitive information cannot be viewed. It must be used in combination with PGP signing.

1.11.1 To activate PGP

✓ Generate a private key that will be used for transaction processing with Beanstream.

✓ Configure your Beanstream account to require PGP signing and (optionally) PGP Encryption. Go to *Administration* → *Account Settings* → *Order Settings* on the Beanstream membership page. Under *Transaction validation options*, select the box called *Require PGP signing of all transactions*. Enter you public key information in the field provided and click *Update*.

All transaction requests submitted through your account will now be rejected unless they have been correctly signed with a matching PGP Public key and PGP Key Id.

Please note that Beanstream will also sign all transaction responses using the Beanstream public key. You must validate the signature of all responses against the Beanstream public key in order to ensure that an approval/decline response has originated from the Beanstream transaction server. You may download the Beanstream public key from the following URL:
https://www.beanstream.com/support/pgp/beanstream.asc

There are two sample scripts available to handle the PGP signing of the transaction request and PGP signature verification of the transaction response. Both scripts are written in ASP Visual Basic Script. A copy of the "NSDPGP.DLL v3.20" package containing the PGP functions used in the sample scripts can be downloaded from: http://community.wow.net/grt/nsdpgp.html

1.11.2 Sample Encrypt Transaction Function

This script is written in ASP Visual Basic Script. A copy of the "NSDPGP.DLL v3.20" package containing the PGP functions used in the sample scripts can be downloaded from: http://community.wow.net/grt/nsdpgp.html

```
<%
function EncryptTransactionString(trnString, passPhrase, signKeyId)

dim merchantId
dim decryptedFile
dim encryptedFile
dim beanstreamKeyId
dim appPath
dim trnString
dim objPgp
dim fs
dim f

beanstreamKeyId = "0x38180389"

'Collect the merchant id from the passed form data
merchantId = request("merchant_id")

'Create an instance of NSDPGP.DLL(v3.20) COM Interface to PGP 6.5.2
set objPgp = CreateObject("NSDPGP")

Set fs = CreateObject("Scripting.FileSystemObject")

'Get the tempory folder specified by the server's TMP environment variable
appPath = fs.GetSpecialFolder(2)

'Build tempory file names
decryptedFile = appPath & "\" & fs.GetTempName
encryptedFile = appPath & "\" & fs.GetTempName

'If decryptedFile already exists, delete it first to avoid errors when creating the file.
if fs.FileExists(decryptedFile) = true then fs.deleteFile decryptedFile

'Write the passed order form transaction string to the decrypted file
set f = fs.OpenTextFile(decryptedFile, 8, true, -2)
        f.writeline trnString
f.close

objPgp.EncryptFileEx beanstreamKeyId, signKeyId, decryptedFile, encryptedFile, passPhrase

'Read in the encrypted transaction string
set f = fs.OpenTextFile(encryptedFile)
        trnString = f.readall
f.close

'Remove the tempory files
objPgp.WipeFile(decryptedFile)
objPgp.WipeFile(encryptedFile)

set fs = nothing

EncryptTransactionString = trnString

end function
```

```
%>
```

1.11.3 Sign Transaction Function

This script is written in ASP Visual Basic Script. A copy of the "NSDPGP.DLL v3.20" package containing the PGP functions used in the sample scripts can be downloaded from:
http://community.wow.net/grt/nsdpgp.html

```
<%
function SignTransactionString(trnString, passPhrase, signKeyId)
        'Purpose: To sign a transaction with the merchants PGP key for submission to
'       the Beanstream Transaction Server.
'Pre:    PGP Security Suite has been installed on the web server that this
'       script is executing on. A PGP Key has been generated for use in
'       submitting transactions to Benastream. The NSDPGP COM object has
'       been installed and registered on the Web Server that this script
'       is executing on. The TMP Environment variable has been declaired
'       on the web server and points to a folder with write permissions.
        '       trnString contains all required transaction parameters to be
'       passed to the Beanstream Transaction Server.
        'Post:   None.
        'Returns: The signed trnString is returned to the calling application.

        dim unsignedFile
        dim signedFile
        dim appPath
        dim objPgp
        dim fs
        dim f

        'Create an instance of NSDPGP.DLL(v3.20) COM Interface to PGP 6.5.2
        set objPgp = CreateObject("NSDPGP")

        Set fs = CreateObject("Scripting.FileSystemObject")

        'Get the tempory folder specified by the server's TMP environment variable
        appPath = fs.GetSpecialFolder(2)

        'Build tempory file names
        unsignedFile = appPath & "\" & fs.GetTempName
        signedFile = appPath & "\" & fs.GetTempName

        'If unsignedFile already exists, delete it first to avoid errors when creating.
        if fs.FileExists(unsignedFile) = true then fs.deleteFile unsignedFile

        'Write the passed order form transaction string to the decrypted file
        set f = fs.OpenTextFile(unsignedFile, 8, true, -2)
                f.writeline trnString
        f.close

        'Sign the order form transaction string
        call objPgp.SignFile (2, signKeyId, unsignedFile, signedFile, passPhrase)

        'Read in the signed transaction string
        set f = fs.OpenTextFile(signedFile)
                SignTransactionString = f.readall
        f.close

        'Remove the tempory files
        objPgp.WipeFile(unsignedFile)
        objPgp.WipeFile(signedFile)
        set fs = nothing

end function
%>
```

1.12 Additional Security Features

You can ensure optimal security throughout the stages of a customer transaction by enabling the following features:

- ✓ Require a username/password validation
- ✓ Validate orders against inventory
- ✓ Validate that the referring host (the web site that the customer comes from) is yours

Please note that to validate orders against your inventory, you must upload your inventory to the Beanstream system. For more information on these features, please refer to our *General Administration Guide*.

2 Server to Server with Process Transaction API

2.1 Server to Server Basics

Server to Server communication is an integration technique used in conjunction with Process Transaction API in order to ensure maximum security for each transaction that is processed. When processing through the API without Server to Server, the customer is actually transferred from the merchant's server to Beanstream's, then redirected back to the merchant's approved/declined page. While generally secure, this method does present the possibility that confidential information may be read in transit.

Server to Server communication resolves this problem by eliminating much of the jumping back and forth. Merchant servers are configured to open a separate, secure session when sending Beanstream any customer transaction details. Beanstream in turn, sends the transaction details and an approved/declined message back via the secure session. The customer is informed through this secure session rather than being redirected to the approve/decline pages specified in the Beanstream Order Settings module.

There are many methods of construction, all of which follow standard methods used in Internet applications such as POST. The method chosen will depend on the platform and programming language being used. Code examples for various programming languages are provided in section 2.7 of this document.

2.2 Transaction Process Flow

When submitting transactions to Beanstream using Server to Server communication, the transaction will proceed as follows:

- ✓ The customer enters their credit card and payment information on the merchant server.
- ✓ The customer submits their payment information to a processing script on the merchant server.
- ✓ The merchant processing script will create a browser object to POST the transaction request to the Beanstream Process Transaction API.
- ✓ Beanstream will process the request and return a response back to the browser object on the merchant site.
- ✓ The merchant processing script will interpret the response from Beanstream and display the transaction status to the customer.

Using this method, the customer’s browser no longer has to be directed to the Beanstream site for processing. A request to redirect back to the merchant’s approved/declined page(s) is also unnecessary.

If there are any transaction errors or problems communicating with Beanstream, these exceptions can be handled in the merchant processing script and communicated appropriately to the customer.

2.3 Basic Integration

2.3.1 Creating the Browser Object

The method employed to create the browser object will depend on the platform and programming language being used. Code examples for various programming languages are provided in section 2.7 of this document.

2.3.2 Submitting Transactions

All of the transaction data sent to Beanstream through an HTML form POST will be sent, without modification, through the browser object. To comply with data length restrictions, ensure that the browser object uses POST and not the GET method.

An additional parameter must be passed with your request to notify the system that the transaction request type is Server-to-Server; the name of this parameter must be *requestType* and assigned a value of *BACKEND*. The system will then respond with either querystring or XML formatted response parameters rather than redirecting to your approval, decline, or error page dependant upon the posted string type.

2.3.3 Response Parameters

The Beanstream Transaction Server will respond to a transaction request with a list of either querystring or XML formatted response parameters dependant upon the posted string type. This parameter list will be returned for all approved and declined transactions and user generated errors. A text description will be returned in the case of a system generated error.

The following parameters will be returned in response to a transaction request. In the *Type* column, numbers represent field size, ***N*** indicates a numeric field, while ***A*** indicates an alphanumeric field.

Field Name	Type	Description
trnApproved	1N	0 = Transaction refused, 1 = Transaction approved
trnId	8N	Unique id number used to identify an individual transaction.
messageId	1-3N	The id number of the transaction response message.
messageText	A	The text message associated with the message ID.
trnOrderNumber	1-30A	The value of trnOrderNumber submitted in the transaction request.
authCode	0-32A	If the transaction is approved this parameter will contain a unique code.
errorType	1A	This field will return the value N, S, or U. See section 1.3.2 for details.
errorFields	A	If the case of a user generated error this field will contain a comma separated listing of all transaction request parameters detected as invalid.
responseType	1A	Set to the value of 'T' to indicate a transaction completion response. If VBV is enabled on the merchant account a value of 'R' may be returned to indicate a VBV redirection response.
trnAmount	9N	The amount of the transaction.
trnDate	20A	The date and time that the transaction was processed.
rbAccountId	4N	The identification number of the recurring billing profile. This value is only returned upon creation of the account.
avsProcessed	1N	Set to a value of 1 if the issuing bank has successfully processed an AVS check on the transaction. Set to a value of 0 if no AVS check has been performed.
avsId	1A	The id number of the AVS response message. See section 1.6.3 for a listing of possible values.
avsResult	1N	Set to a value of 1 if AVS has been validated with both a match against address and a match against postal/ZIP code.
avsAddrMatch	1N	Set to a value of 1 if the ordAddress1 parameter matches the consumers address records at the issuing bank. Set to a value of 0 if the ordAddress1 parameter does not match the customer's address records or if AVS was not processed for the transaction.
avsPostalMatch	1N	Set to a value of 1 if the ordPostalCode parameter matches the

		consumers address records at the issuing bank. Set to a value of 0 if the ordPostalCode parameter does not match the customer's address records or if AVS was not processed for the transaction.
avsMessage	128A	Set to the value of the text message associated with the avsId response code.
rspCodeCav rspCavResult rspCodeCredit1 rspCodeCredit2 rspCodeCredit3 rspCodeCredit4 rspCodeAddr1 rspCodeAddr2 rspCodeAddr3 rspCodeAddr4 rspCodeDob rspCodeSafeScan rspCodeSafeScanId rspCustomerDec	-	These variables will be returned if you have applied for Canadian Address Verification. If cavServiceVersion = 1.0 then only one credit card and one address response code will be returned as variables rspCodeCredit and rspCodeAddr, rather than four each. If cavServiceVersion = 1.1 then four numbered rspCodeCredit and rspCodeAddr codes will be returned. rspCodeDob will only be returned for cavServicVersion 1.2 or greater. For more information, please consult our CAV documentation. If cavServiceVersion 1.3 or greater is passed, then customer declaration data will be returned
cvdId	1N	The id number of the CVD response message. This parameter will be returned if the Credit Card CVD value has been submitted in the transaction request. See section 1.6.4 for a listing of possible values.
trnType	3A	The original value sent to indicate the type of transaction to perform (i.e. P,R,VP,VR, PA, PAC).
paymentMethod	2A	The payment method used by the customer to complete the transaction. (i.e. IO, CC …).
ioConfCode	15A	In an INTERAC Online transaction this confirmation number will be returned by the customer's financial institution if the transaction has been processed successfully. This value must be displayed to the customer on a transaction confirmation page.
ioInstName	30A	The name of the customer's financial institution. This value is retuned for INTERAC Online transactions only.
ref1	256A	The value of the ref1 field submitted in the transaction request.
ref2	256A	The value of the ref2 field submitted in the transaction request.
ref3	256A	The value of the ref3 field submitted in the transaction request.
ref4	256A	The value of the ref4 field submitted in the transaction request.
ref5	256A	The value of the ref5 field submitted in the transaction request.

Sample Approval Post Response:

```
trnApproved=1&trnId=10003067&messageId=1&messageText=Approved&trnOrderNumber=E40089&aut
hCode=TEST&errorType=N&errorFields=&responseType=T&trnAmount=10%2E00&trnDate=1%2F17%2F
2008+11%3A36%3A34+AM&avsProcessed=0&avsId=0&avsResult=0&avsAddrMatch=0&avsPostalMatch=
0&avsMessage=Address+Verification+not+performed+for+this+transaction%2E&rspCodeCav=0&rspCavR
esult=0&rspCodeCredit1=0&rspCodeCredit2=0&rspCodeCredit3=0&rspCodeCredit4=0&rspCodeAddr1=0&r
spCodeAddr2=0&rspCodeAddr3=0&rspCodeAddr4=0&rspCodeDob=0&rspCustomerDec=&trnType=P&pay
mentMethod=CC&ref1=&ref2=&ref3=&ref4=&ref5=
```

Sample Decline Post Response:

```
trnApproved=0&trnId=10003068&messageId=16&messageText=Duplicate+Transaction+%2D+This+tran
saction+has+already+been+approved&trnOrderNumber=E40089&authCode=&errorType=N&errorFields=
&responseType=T&trnAmount=10%2E00&trnDate=1%2F17%2F2008+11%3A38%3A10+AM&avsProcesse
d=0&avsId=0&avsResult=0&avsAddrMatch=0&avsPostalMatch=0&avsMessage=Address+Verification+not
+performed+for+this+transaction%2E&rspCodeCav=0&rspCavResult=0&rspCodeCredit1=0&rspCodeCredi
t2=0&rspCodeCredit3=0&rspCodeCredit4=0&rspCodeAddr1=0&rspCodeAddr2=0&rspCodeAddr3=0&rspCo
deAddr4=0&rspCodeDob=0&rspCustomerDec=&trnType=P&paymentMethod=CC&ref1=&ref2=&ref3=&ref
4=&ref5=
```

Sample Approval XML Response:

```
   <?xml version="1.0" encoding="ISO-8859-1" ?>
- <response>
  <trnApproved>1</trnApproved>
  <trnId>10048766</trnId>
  <messageId>1</messageId>
  <messageText>Approved</messageText>
  <trnOrderNumber>E40089</trnOrderNumber>
  <authCode>TEST</authCode>
  <errorType>N</errorType>
  <errorFields />
  <responseType>T</responseType>
  <cvdId>1</cvdId>
   </response>
```

Sample Decline XML Response:

```
   <?xml version="1.0" encoding="ISO-8859-1" ?>
- <response>
  <trnApproved>0</trnApproved>
  <trnId>10048769</trnId>
  <messageId>16</messageId>
  <messageText>Duplicate Transaction - This transaction has already been
    approved</messageText>
  <trnOrderNumber>E40089</trnOrderNumber>
  <authCode />
  <errorType>N</errorType>
  <errorFields />
  <responseType>T</responseType>
  <cvdId>6</cvdId>
   </response>
```

2.3.4 Error Handling

There are two types of error messages that the Beanstream transaction server may return when processing a transaction: system generated errors and user generated errors. A complete description of these error messages can be found in section 1.3.2 of this document.

2.4 Integrating with Verified by Visa

Verified by Visa (VBV) is a security feature that requires customers to enter a password every time they use their Visa card to complete a transaction.

VBV Sever to Server integration requires ***two*** transaction requests. In the first request, the customer will enter his or her purchase information including a credit card number. The request will be forwarded to Beanstream. If the card is VBV enabled, the merchant must forward the customer to their issuing bank where they will enter their Visa password. When the password verification is completed, the merchant must generate a second transaction request to forward authentication results to Beanstream and complete the transaction. A sample integration flow of the VBV process can be found in point 2.4.1.

Because of this process, two different types of transaction response messages can be returned to sites that are integrated with VBV. For all transaction requests, the Beanstream system will respond with a responseType parameter. To determine the status of the transaction refer to the Server to Server response parameters.

responseType=R	This ***redirection response message*** will be returned from the initial transction request if the VISA card used was VBV enabled. The cardholder must be redirected to their issuing bank for password verification.
responseType=T	The second request in a VBV transaction will always return a responseType value of 'T' to indicate if a transaction has been completed as "approved" or "declined."

2.4.1 Transaction Processing Flow

The following five steps outline how typical request and response messages are passed from the merchant to Beanstream in the VBV process:

1. The customer enters their address and payment information into the order form presented in their browser and submits their payment to the merchant system.

2. The merchant system receives the customer's payment request, generates a transaction request and forwards it to Beanstream according to the standard method. An extra parameter called termUrl must be passed with the transaction. The customer will be redirected to the URL indicated in this parameter after completing VBV authentication (see step 5).

   The following code shows a sample request POSTed to process_transaction.asp where the termUrl is https://www.merchantserver.com/auth_script.asp.

   ```
   requestType=BACKEND&
   merchant_id=109040000&trnCardOwner=Paul+Randal&trnCardNumber=4030000010001234&trnExpMonth=01
   &trnExpYear=05&trnOrderNumber=2232&trnAmount=10.00&ordEmailAddress=prandal@mydomain.net&ordNa
   me=Paul+Randal&ordPhoneNumber=9999999&ordAddress1=1045+Main+Street&ordAddress2=&ordCity=Vanco
   uver&ordProvince=BC&ordPostalCode=V8R+1J6&ordCountry=CA&termUrl=https%3A%2F%2Fwww%2Em
   erchantserver%2Ecom%2Fauth_script.asp
   ```

   Notice that the "=" in this example is not URL encoded. The termURL variable itself (https%3A%2F%2Fwww%2Emerchantserver%2Ecom%2Fauth_script.asp) must be URL coded in order to function properly.

3. Beanstream sees that it's a Visa transaction, and checks to see if the card is enrolled in VBV. If not, then Beanstream performs the Server to Server transaction according to normal procedures and returns an approved or declined *transaction response message* back to the merchant server. If the card ***is*** enrolled in VBV, Beanstream will return a *redirection response message.*

4. The merchant system checks to see if a *redirection response message* or responseType=R was returned. If this is the case, the Java Script redirect (contained in URL-encoded format in the pageContents variable) is displayed to the customer's web browser. The redirect forwards the customer's browser to the credit card issuing bank, where a VBV password can be entered. After the customers has entered their Visa password, the issuing bank will redirect the customer back to the merchant server terminal URL page along with the authentication results.

   A sample redirection response message can be seen below.

```
responseType=R&pageContents=%3CHTML%3E%3CHEAD%3E%3C%2FHEAD%3E%3CBODY%3E%3CFORM+ac
tion%3D%22https%3A%2F%2F203%2E42%2E45%2E62%2F0%2F9uaBIZYbdsFSN7FADKaAsH3N6O80%22+m
ethod%3DPOST+id%3Dform1+name%3Dform1%3E%3CINPUT+type%3Dhidden+name%3DPaReq+value%3D
%22eJxtUl1vgjAUfd%2BvILyPVihf5lqDI0azzJnJHra3DholkYIFhu7Xr0WYW7IHknvObc89PReYn4uj8cllnZdiZk4sbBp
cpGWWi%2F3MfE2W94E5p3eQHCTn8Y6nreQUnnhdsz038kxdwS7BIcbYd1078MPQdnFIbIeEvuMT4pkUttELP1EYpl
A1xLIBjVCpyfTAREOBpafFekPJRF13AQ0QCi7XMe1Jzw9CPLSvNAhWcJrwujFGIUA9B2nZikZeKHY8QCOAVh7poWm
qKUJd11n7il0KLpraSssCkO4Culnatrqqldo5z%2BhTHHX%2FfF%2BbJJoB0icgYw2nNsYOVnEY2J4Sb2r7gHoeWKFt0
AmxMFYPvCKo9JBobOnObwZU6FItZXzIiICfq1Io61Sl%2BVMDull%2BWOlM00bFFLZssX5%2F%2B8jq5W7jL6P4kU
X1ytl4z4E2cj2kFXMdGMFBL6kBIC2DhiWiYfeq%2BvNPfAM1YLvf%22%3E%3CINPUT+type%3Dhidden+name%3D
MD+value%3D%22requestType%3DBACKEND%2526trnCardNumber%3D4123450131003312%2526trnExpMon
th%3D08%2526trnExpYear%3D04%2526trnType%3DPA%2526trnAmount%3D14%2E00%2526merchant%5Fid
%3D107380000%2526errorPage%3Dhttps%253A%252F%252Fwww%252Ebeanstream%252Ecom%252Fsampl
es%252Forder%5Fform%2Easp%2526trnCardOwner%3DPaul%2BRandal%2526trnOrderNumber%3D2232%25
26ordEmailAddress%3Dalau%40beanstream%2Ecom%2526ordName%3DPaul%2BRandal%2526ordPhoneNumb
er%3D9999999%2526ordAddress1%3D1045%2BMain%2BStreet%2526ordAddress2%3D%2526ordCity%3DVa
ncouver%2526ordProvince%3DBC%2526ordPostalCode%3DV8R%2B1J6%2526ordCountry%3DCA%2526termU
rl%3Dhttps%253A%252F%252Fwww%2Ebeanstream%2Ecom%252Fsamples%252Fsample%5Fs2s%5Fvbv%5F
auth%2Easp%2526xid%3D350%22%3E%3CINPUT+type%3Dhidden+name%3DtermUrl+value%3D%22https%
3A%2F%2Fwww%2Ebeanstream%2Ecom%2Fsamples%2Fsample%5Fs2s%5Fvbv%5Fauth%2Easp%22%3E%3
C%2FFORM%3E%3CSCRIPT+language%3D%22JavaScript%22%3Edocument%2Eform1%2Esubmit%28%29%3
B%3C%2FSCRIPT%3E%3C%2FBODY%3E%3C%2FHTML%3E
```

5. The merchant server receives the following VBV authentication parameters.

Variable	Definition
PaRes	VBV Authentication Code
MD	Unique payment id

   The merchant's Terminal URL Page must POST these two values to Beanstream's process_transaction_auth.asp. The following example shows a sample ASP POST. Please note that this script may vary depending on your implementation method.

```
<%
'This is a sample Terminal URL page that the merchant must have on their web
'server.  The Issuer Access Control Server (ACS) will redirect to this page
'during the Authentication stage (after the customer enters his/her password).

set objXMLHTTP = Server.CreateObject("MSXML2.ServerXMLHTTP.4.0")
```

```
objXMLHTTP.Open "POST", "https://www.beanstream.com/scripts/process_transaction_auth.asp", false
objXMLHTTP.setRequestHeader "Content-Type", "application/x-www-form-urlencoded"
objXMLHTTP.Send("PaRes=" & request("PaRes") & "&MD=" & request("MD"))
response.write objXMLHTTP.ResponseText
set objXMLHTTP = nothing
%>
```

Process_transaction_auth.asp will validate the results of the password authentication. If accepted, the Visa transaction will proceed as usual passing the transaction to the bank for payment authentication. If the transaction does not pass VBV, it will immediately be declined with a message id of 311 indicating that the 3D Secure verification failed. In either case the Beanstream system will respond with a *transaction response message* indicating the approved or declined status of the request.

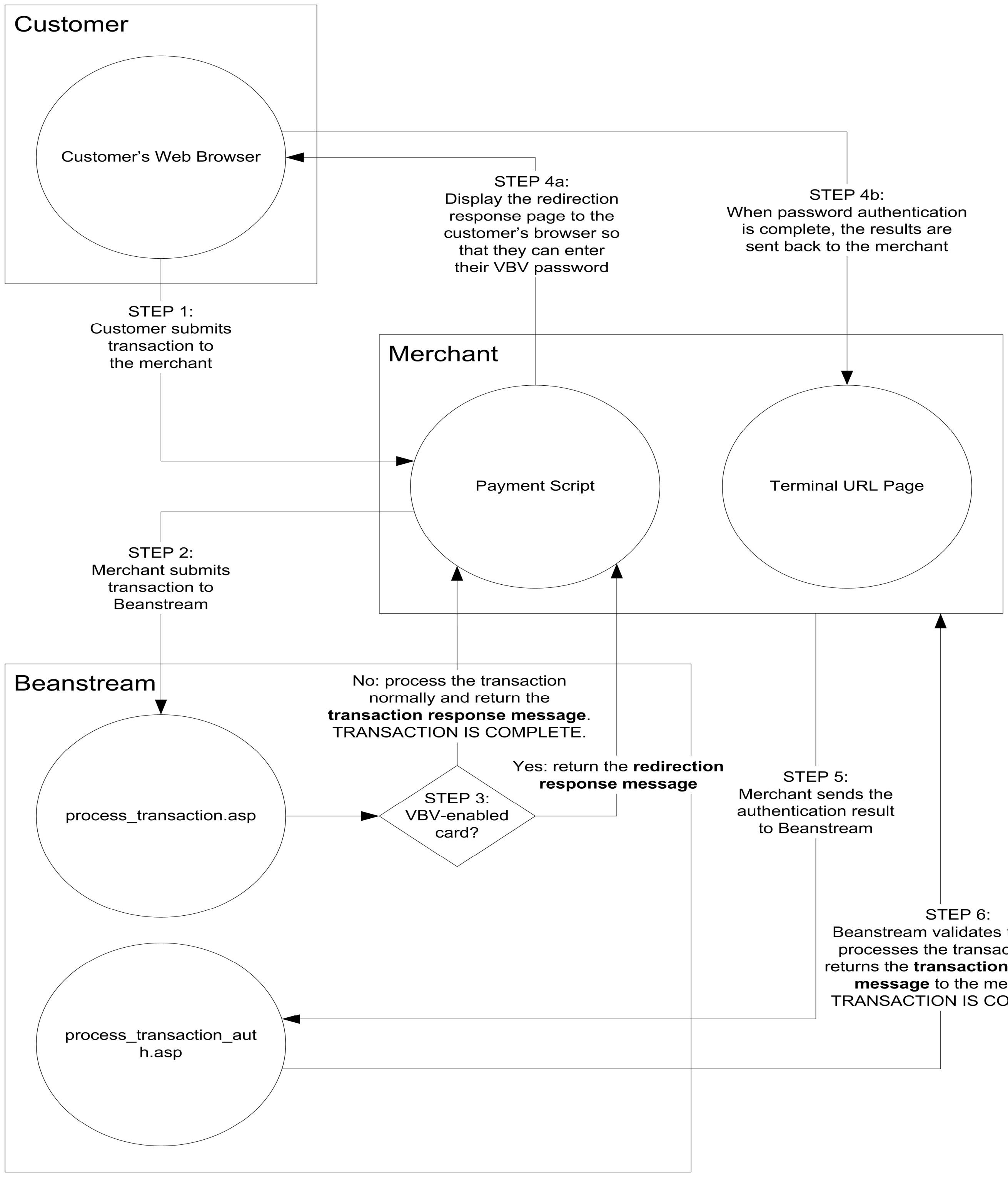

Customer
Customer's Web Browser
STEP 4a: Display the redirection response page to the customer's browser so that they can enter their VBV password
STEP 4b: When password authentication is complete, the results are sent back to the merchant
STEP 1: Customer submits transaction to the merchant
Merchant
Payment Script
Terminal URL Page
STEP 2: Merchant submits transaction to Beanstream
Beanstream
No: process the transaction normally and return the **transaction response message**. TRANSACTION IS COMPLETE.
Yes: return the **redirection response message**
STEP 5: Merchant sends the authentication result to Beanstream
process_transaction.asp
STEP 3: VBV-enabled card?
STEP 6: Beanstream validates the result, processes the transaction, and returns the **transaction response message** to the merchant. TRANSACTION IS COMPLETE.
process_transaction_auth.asp

Figure 1: Verified by Visa Server-to-Server Process Flow

2.4.2 VBV Testing

To test your VBV setup, you should process sample transaction against both a VBV enabled credit card and a standard credit card. This will ensure that you have integrated correctly for both types of cards.

A full list of test card numbers can be found in section 1.8. You will know that your implementation is successful if you are presented with a VBV password page. The transaction will proceed as normal but will not be processed.

2.5 INTERAC Online

Beanstream's INTERAC Online service allows merchants to offer a new payment option to online shoppers. Using this alternative to online credit card processing, customers can pay for purchases directly from their bank account as they would when using a debit card at a traditional bricks and mortar store. While INTERAC Online processing is available to all merchants, consumers purchasing through the merchant's website will only be able to use this payment type if they already have access to online banking and if their bank is a participating financial institution.

2.5.1 Transaction Processing Flow

INTERAC Online request and response messages are passed from the merchant to Beanstream in the following manner:

1. The customer initiates a purchase on the Merchant's website and elects to pay via INTERAC Online.

2. The merchant system receives the customer's payment request, generates a transaction request and forwards it to Beanstream. The sample code provided below shows a typical request POSTed to process_transaction.asp. The termURL will always be https%3A%2F%2Fpayments.beanstream.com%2Fscripts%2Fprocess_transaction_auth.asp. The merchant username is "test" and the merchant password is "testing123". User name and password authentication variables are only passed if *User Name and Password Authentication* has been activated in the Beanstream membership area under *Administration → Account Admin → Order Settings*.

   requestType=BACKEND&errorPage=%2Fsamples%2Forder_form.asp&merchant_id148280000&trnCardOwner=Paul+Test&trnOrderNumber=1a&trnAmount=5.00&ordEmailAddress=mdoty@beanstream.com &ordName=Paul+Randal&ordPhoneNumber=9999999&ordAddress1=1045+Main+Street&ordAddress2=&ordCity=Vancouver&ordProvince=BC&ordPostalCode=V8R+1J6&ordCountry=CA&paymentMethod=IO&**termUrl=https%3A%2F%2Fpayments.beanstream.com%2Fscripts%2Fprocess_transaction_auth.asp**&username=test&password=testing123

   Notice that the "=" in this example is not URL encoded. The termURL variable itself must be URL encoded in order to function properly.

3. Beanstream sees that this is an Interac Online (IO) transaction and returns a *redirection response message*.

4. The merchant system sees that the redirection response message or response Type=R was returned. The Java Script redirect (contained in URL-encoded format in the pageContents variable) is displayed

to the customer’s web browser. The redirect forwards the customer’s browser to the INTERAC Online Gateway page.

A sample redirection response message can be seen below.

```
responseType=R&pageContents=%3CHTML%3E%3CHEAD%3E%3C%2FHEAD%3E%3CBODY%3E%3
CFORM%20action%3D%22https%3A%2F%2Fmerchant%2Einteracidebit%2Eca%2Fgateway%2Fmer
chant%5Fprocessor%2Edo%22%20method%3DPOST%20id%3DfrmIOnline%20name%3DfrmIOnline
%3E%3Cinput%20type%3D%22hidden%22%20name%3D%22IDEBIT%5FMERCHNUM%22%20%20
value%3D%2210010152000001%22%3E%3Cinput%20type%3D%22hidden%22%20name%3D%22
IDEBIT%5FAMOUNT%22%20%20value%3D%2210100%22%3E%3Cinput%20type%3D%22hidden
%22%20name%3D%22IDEBIT%5FTERMINID%22%20value%3D%22%22%3E%3Cinput%20type%
3D%22hidden%22%20name%3D%22IDEBIT%5FCURRENCY%22%20value%3D%22CAD%22%3E%
3Cinput%20type%3D%22hidden%22%20name%3D%22IDEBIT%5FINVOICE%22%20value%3D%2
21a%22%3E%3Cinput%20type%3D%22hidden%22%20name%3D%22IDEBIT%5FMERCHDATA%22
%20value%3D%2281D692EC%2D75EB%2D4AD8%2DA600D5A60F1CF4%22%3E%3Cinput%20type
%3D%22hidden%22%20name%3D%22IDEBIT%5FFUNDEDURL%22%20value%3D%22https%3A%
2F%2Fpayments%2Ebeanstream%2Ecom%2Fscripts%2Fprocess%5Ftransaction%5Fauth%2Easp%3
Ffunded%3D1%22%3E%3Cinput%20type%3D%22hidden%22%20name%3D%22IDEBIT%5FNOTFU
NDEDURL%22%20value%3D%22https%3A%2F%2Fpayments%2Ebeanstream%2Ecom%2Fscripts%
2Fprocess%5Ftransaction%5Fauth%2Easp%3Ffunded%3D0%22%3E%3Cinput%20type%3D%22hid
den%22%20name%3D%22IDEBIT%5FMERCHLANG%22%20value%3D%22en%22%3E%3Cinput%2
0type%3D%22hidden%22%20name%3D%22IDEBIT%5FVERSION%22%20value%3D%221%22%3
E%3C%2FFORM%3E%3CSCRIPT%20language%3D%22JavaScript%22%3Edocument%2EfrmIOnline
%2Esubmit%28%29%3B%3C%2FSCRIPT%3E%3C%2FBODY%3E%3C%2FHTML%3E
```

On the INTERAC Gateway page, the customer is prompted to select a financial institution. From there they will be redirected to their online banking login page. At the online banking website, the customer will enter their login information. This page will also include a link to go back to the Merchant’s website and may include a link to register for or activate online banking. The customer is then offered the choice to accept or decline payment and chose an account to pay from. If they chose to approve, they are then presented with a confirmation page. Once the transaction information is confirmed the user is redirected back to the merchant’s site.

5. Process_transaction_auth.asp will validate the results of the transaction and return a response to the merchant’s Approved/Declined response pages.
   For a complete description of the INTERAC Online process flow and website requirements, please see the Interac Online User Guide.

2.5.2 Testing Interac Online

2.6 Nextwave Card Authorization

Nextwave Card Authorization is a security feature that requires customers to enter a password every time they use their Nextwave card to complete a transaction.

Nextwave Card Authorization with Sever to Server integration requires ***two*** transaction requests. In the first request, the customer will enter his or her purchase information including a credit card number. The request will be forwarded to Beanstream. If the card is Nextwave enabled, the merchant must forward the customer to their issuing bank to enter their Visa password. Once the password has been verified, the merchant must generate a second transaction request to forward authentication results to Beanstream and complete the transaction. A sample integration flow for the Nextwave Card Authorization process can be found in point 2.6.1.

Because of this process, two different types of transaction response messages can be returned to sites that are integrated with Nextwave. For all transaction requests, the Beanstream system will respond with a responseType parameter. To determine the status of the transaction refer to the Server to Server response parameters.

responseType=R	This redirection response message will be returned from the initial transaction request if the card used was Nextwave enabled. The cardholder must be redirected to their issuing bank for password verification.
responseType=T	The second request in a Nextwave transaction will always return a responseType value of 'T" to indicate if a transaction has been completed as "approved" or "declined."

2.6.1 Sample Integration Flow

The following five steps outline how typical request and response messages are passed from the merchant to Beanstream in the Nextwave Card Authorization process:

1. The customer enters their address and payment information into the order forms presented in their browser and submits their payment to the merchant system.

2. The merchant system receives the customer's payment request, generates a transaction request and forwards it to Beanstream according to the standard method. An extra parameter called termURL must be passed with the transaction. The customer will be redirected to the URL indicated in this parameter after completing Nextwave authentication (see step 5).

   The following code shows a sample request POSTed to process_transaction.asp where the termUrl is https://www.merchantserver.com/auth_script.asp:

   ```
   requestType=BACKEND&
   merchant_id=109040000&trnCardOwner=Paul+Randal&trnCardNumber=4030000010001234&trnExpMonth=01&t
   rnExpYear=05&trnOrderNumber=2232&trnAmount=10.00&ordEmailAddress=prandal@mydomain.net&ordName=
   Paul+Randal&ordPhoneNumber=9999999&ordAddress1=1045+Main+Street&ordAddress2=&ordCity=Vancouver&
   ordProvince=BC&ordPostalCode=V8R+1J6&ordCountry=CA&termUrl=https%3A%2F%2Fwww%2Emerchant
   server%2Ecom%2Fauth_script.asp
   ```

3. Beanstream checks to see if the transaction request used a Nextwave card. If not, then Beanstream performs the Server to Server transaction according to normal procedures and returns an approved or declined *transaction response message* back to the merchant server. If the card ***is*** a Nextwave card, Beanstream will return a *redirection response message.*

4. The merchant system checks to see if *redirection response message* or responseType=R was returned. If this is the case, the Java Script redirect (contained in URL-encoded format in the pageContents

variable) is displayed to the customer's web browser. The redirect forwards the customer's browser to the credit card issuing bank, where a Nextwave password can be entered. After the password has been entered, the issuing bank will redirect the customer back to the merchant server terminal URL page along with the authentication results.

A sample redirection response message can be seen below.

```
responseType=R&pageContents=%3CHTML%3E%3CHEAD%3E%3C%2FHEAD%3E%3CBODY%3E%3CFORM%20actio
n%3D%22http%3A%2F%2Fmm%2Dcardweb%2Dcert%2Edatawave%2Ecom%2Fauth%2Findex%2Ecfm%3Faction%3
Dentrance%26display%3Dpopup%22%20method%3DPOST%20id%3DfrmNextWave%20name%3DfrmNextWave%3E
%3CINPUT%20type%3Dhidden%20name%3DsessionID%20value%3D%2231232132l%22%3E%3CINPUT%20type%
3Dhidden%20name%3DpaymentID%20value%3D%225A72C487%2D8E61%2D47D7%2D9CFBF5E8CAFF28D%22%3
E%3CINPUT%20type%3Dhidden%20name%3DTermURL%20value%3D%22https%3A%2F%2F10%2E10%2E10%2E1
63%2Fsamples%2Fsample%5Fdatawave%5FtermUrl%2Easp%22%3E%3CINPUT%20type%3Dhidden%20name%3Dtr
nCardNumber%20value%3D%2262220982130610767746%22%3E%3C%2FFORM%3E%3CSCRIPT%20language%3D
%22JavaScript%22%3Edocument%2EfrmNextWave%2Esubmit%28%29%3B%3C%2FSCRIPT%3E%3C%2FBODY%3E
%3C%2FHTML%3E
```

5. The merchant server receives the following Nextwave Authentication parameters.

Variable	**Definition**
PAC	Payment authorization code generated by the Nextwave system.
paymentID	Beanstream payment id. Identifier generated by the Beanstream system used to identify the payment submitted in step 3 of the process.
sessionID	Optional merchant assigned parameter to identify the consumer or order.
responseCode	This variable will be passed back if there is an error with the transaction request or if a customer returns to the merchant's page before their password is validated

The merchant's Terminal URL Page must then POST these values to Beanstream's process_transaction_auth.asp. The following example shows a sample ASP post. Please note that script may vary depending on your implementation method.

```
<%
'This is a sample Terminal URL page that the merchant must have on their web
'server.  The Issuer NextWave will redirect to this page
'during the Authentication stage (after the customer enters their password).

dim postData   'Data passed back form NextWave
postData= request.form

'Possible responseCode values
'000 - Success
'001 - Failure -- any other reason other than 000 and 002
'002 - User Abort

select case request("responseCode")
case "002":
        'If additional data is required for your 'Back to Merchant' link
        'append it to the postData variables data using the code:
        'postData= postData & additionalData

        'User abort, user redirected to merchant site's payment form
        call SendServerXMLObject("https://www.merchantServer.com/payment_form.asp",postData)
case else:
        'Transaction is to be processed, redirect to beanstream for processing
        call SendServerXMLObject("https://www.beanstream.com/scripts/process_transaction_auth.asp",postData)
end select

sub SendServerXMLObject(stringURL,postData)
'Function to send the data to the specified URL
dim objXMLHTTP

        'Create the ServerXMLHTTP object
        set objXMLHTTP = Server.CreateObject( "MSXML2.ServerXMLHTTP.4.0" )

        objXMLHTTP.setOption(2) = 4096
        objXMLHTTP.setOption(3) = ""

        'This is the location of the stringURL
        objXMLHTTP.Open "POST", stringURL, false

        'Set the HTTP header's content type
        objXMLHTTP.setRequestHeader "Content-Type", "application/x-www-form-urlencoded"

        objXMLHTTP.Send( postData )
        response.write objXMLHTTP.ResponseText
        set objXMLHTTP = nothing
end sub
%>
```

6. Process_transaction_auth.asp will validate the results of the password authentication. If accepted, the Nextwave card transaction will proceed as usual passing the transaction to the bank for payment authentication. If the customer's password is rejected more than the number of times allowed by Nextwave, he or she will be returned to the merchant's website where they will be shown a "transaction declined" message.

2.6.2 Test Nextwave Card Numbers

The following test Nextwave card number may be used to test your Server to Server integration.

Card Type	Card Number	Response	Password
Nextwave	6220982130610767746	Approved	Yes (Passcode: 5502)

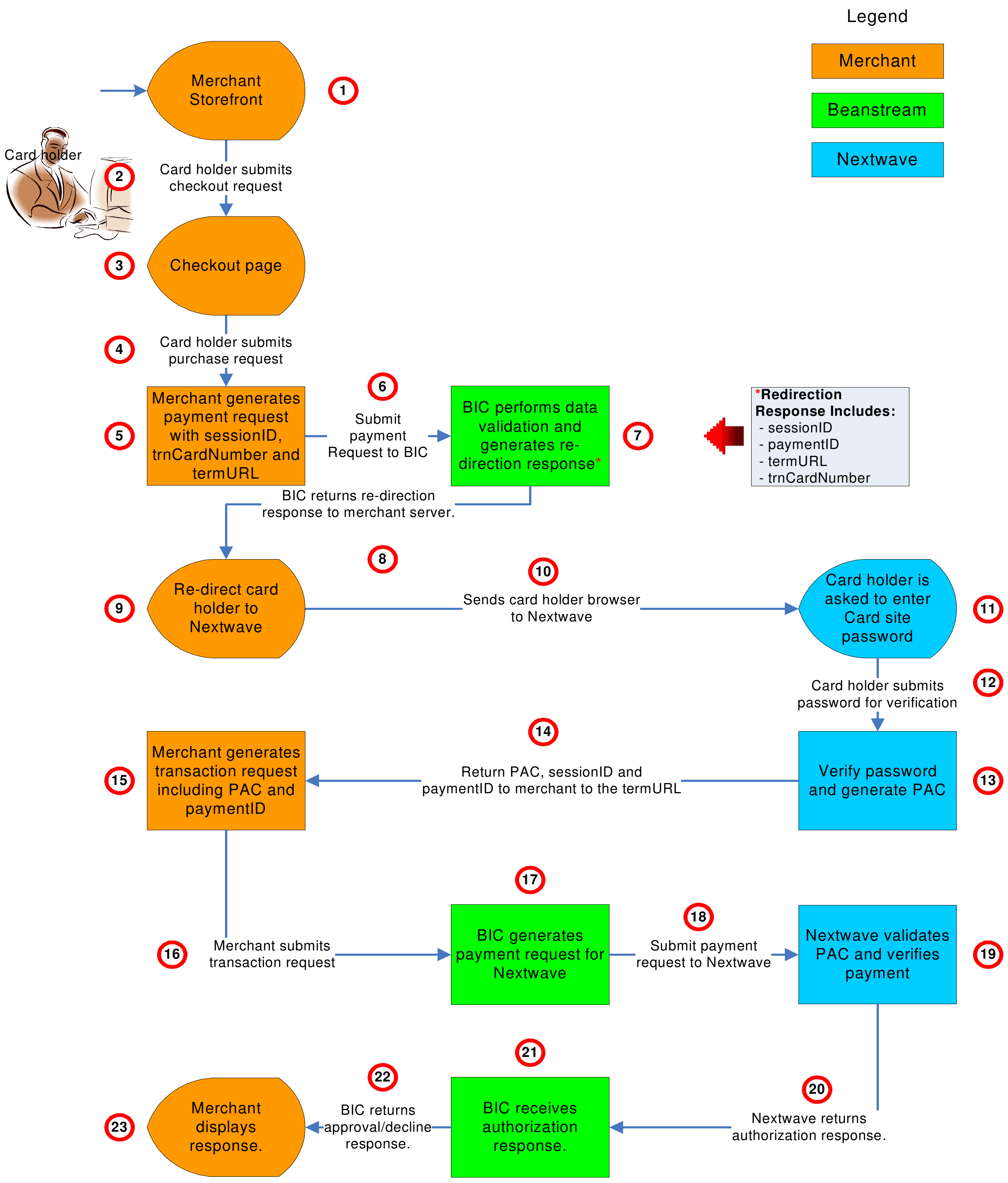

Legend
Merchant
Beanstream
Nextwave
Card holder
1 Merchant Storefront
2 Card holder submits checkout request
3 Checkout page
4 Card holder submits purchase request
5 Merchant generates payment request with sessionID, trnCardNumber and termURL
6 Submit payment Request to BIC
BIC performs data validation and generates re-direction response*
7
*Redirection Response Includes:
- sessionID
- paymentID
- termURL
- trnCardNumber
BIC returns re-direction response to merchant server.
8
9 Re-direct card holder to Nextwave
10 Sends card holder browser to Nextwave
Card holder is asked to enter Card site password 11
Card holder submits password for verification 12
Verify password and generate PAC 13
14 Return PAC, sessionID and paymentID to merchant to the termURL
15 Merchant generates transaction request including PAC and paymentID
16 Merchant submits transaction request
17 BIC generates payment request for Nextwave
18 Submit payment request to Nextwave
Nextwave validates PAC and verifies payment 19
20 Nextwave returns authorization response.
21 BIC receives authorization response.
22 BIC returns approval/decline response.
23 Merchant displays response.

Figure 2: Nextwave Card Authorization Server-to-Server Process Flow

2.7 Sample Integration Code

The following examples demonstrate how to submit a transaction to the Beanstream server via the Server-To-Server method using various programming languages. In each of these examples, the following sample parameters will be submitted via HTTPS POST:

```
requestType=BACKEND&merchant_id=109040000&trnCardOwner=Paul+Randal&trnCardNumber=62209821306107
67738&trnOrderNumber=2232&trnAmount=10.00&ordEmailAddress=prandal@mydomain.net&ordName=Paul+Randa
l&ordPhoneNumber=9999999&ordAddress1=1045+Main+Street&ordAddress2=&ordCity=Vancouver&ordProvince=BC
&ordPostalCode=V8R+1J6&ordCountry=CA
```

These parameters will be submitted to the Beanstream payment gateway, which is located at https://www.beanstream.com/scripts/process_transaction.asp.

2.7.1 Sample ASP Code

The following is an example of how to POST a transaction to the Beanstream server using ASP and the Microsoft XML Core Services (MSXML) version 4.0. (MSXML is also known as the Microsoft XML Parser).

We do not recommend using WinInet to do the POST because WinInet is not thread safe, and hence is not suitable for use in server applications.

To use this example, you must have MSXML 3.0 or 4.0 installed on your server. For more information on how to download and install MSXML, see the MSDN documentation at http://msdn.microsoft.com/downloads/default.asp?url=/downloads/sample.asp?url=/msdn-files/027/001/766/msdncompositedoc.xml

Sample Code

```
<%
option explicit

'Set to the address of the Beanstream server.
const BEANSTREAM_SERVER = "www.beanstream.com"
const MERCHANT_ID       = 109040000
const TERM_URL          = "https://www.merchantserver.com/auth_script.asp"

dim objXMLHTTP
dim beanstreamResponse
dim postData

'Send transaction request string to be posted to the Beanstream system
postData=
"requestType=BACKEND&trnType=P&trnCardNumber=6220982130610517737&trnAmount=1%2e00&merchant_id="
& MERCHANT_ID &
"&trnCardOwner=Paul+Randal&trnOrderNumber=1a&ordEmailAddress=prandal@mydomain.net&ordName=Paul+Ran
dal&ordPhoneNumber=60411234567&ordAddress1=1045+Main+Street&ordAddress2=&ordCity=Vancouver&ordProvi
nce=BC&ordPostalCode=V8R+1J6&ordCountry=CA&termUrl=" & server.urlEncode(TERM_URL) & "&sessionId=" &
request("sessionId")

'Create the ServerXMLHTTP object
```

```
set objXMLHTTP = Server.CreateObject( "MSXML2.ServerXMLHTTP.4.0" )
objXMLHTTP.setOption(2) = 4096
objXMLHTTP.setOption(3) = ""

'This is the location of the Beanstream payment gateway
objXMLHTTP.Open "POST", "https://" & BEANSTREAM_SERVER & "/scripts/process_transaction.asp", false

'Set the HTTP header's content type
objXMLHTTP.setRequestHeader "Content-Type", "application/x-www-form-urlencoded"

'Submit the transaction request to the Beanstream server
objXMLHTTP.Send( postData )

'Read the transaction response returned from the Beanstream system
beanstreamResponse = objXMLHTTP.ResponseText

'We have now received a response from Beanstream.  Now check if this response is a Redirection
'Response Page by checking the value of the responseType parameter.  If the responseType paramter
'is set to "R" it is a redirection repsonse.  If the response type parameter is a "T" it is a
'transaction approved/delined response.  For datawave cards the system should allways return a
'redirection response.

'response.write beanstreamResponse : response.end
if GetQueryValue(beanstreamResponse, "responseType" ) = "R" then
        'We have a Redirection Response Page, so show it to the browser to redirec the user to datawave for
verification
        response.write GetQueryValue(beanstreamResponse, "pageContents")
else
         'This is a normal transaction, so beanstreamResponse contains the results of the transaction.
         if GetQueryValue(beanstreamResponse, "trnApproved" ) = "1" then
                 response.write "Transaction Approved"
         else
                 response.write "Transaction Declined: " & beanstreamResponse
         end if
end if


Function GetQueryValue(queryString, paramName)
 'Purpose: To return the value of a parameter in an HTTP query string.
 'Pre:     queryString is set to the full query string of url encoded name value pairs. ex:
"value1=one&value2=two&value3=3"
 '    paramName is set to the name of one of the parameters in the queryString.  ex: "value2"
 'Post:    None
 'Returns: The function returns the query string value assigned to the paramName parameter.  ex: "two"

   Dim pos1
   dim pos2
   Dim qString

   qString = "&" & queryString & "&"
   pos1 = InStr(1, qString, paramName & "=")
   If pos1 > 0 Then
      pos1 = pos1 + Len(paramName) + 1
      pos2 = InStr(pos1, qString, "&")
      If pos2 > 0 Then
         GetQueryValue = DecodeQueryValue(Mid(qString, pos1, pos2 - pos1))
      End If
   End If
End Function

Function DecodeQueryValue(qValue)
 'Purpose: To URL decode a string
 'Pre:     qValue is set to a url encoded value of a query string parameter.  ex: "one+two"
 'Post:    none
 'Returns: Returns the url decoded value of qValue.  ex: "one two"

   Dim i
   Dim qChar
   dim newString

          if IsNull(qValue) = false then
              For i = 1 To Len(qValue)
                  qChar = Mid(qValue, i, 1)
                  If qChar = "%" Then
                              on error resume next
                     newString = newString & Chr("&H" & Mid(qValue, i + 1, 2))
                              on error goto 0
                     i = i + 2
                  ElseIf qChar = "+" Then
                      newString = newString & " "
                  Else
                      newString = newString & qChar
                  End If
              Next
                     DecodeQueryValue = newString
          else
                     DecodeQueryValue = ""
          end if

End Function
%>
```

2.7.2 Sample PHP Code

The following is an example of how to POST a transaction to the Beanstream server using PHP and the libcurl CURL library.

To use this example, you must install the CURL package. CURL allows you to connect to servers using a variety of protocols, and in this example, it uses it to communicate with Beanstream via HTTPS POST. For information on how to install CURL, see the PHP manual at http://www.php.net/manual/en/ref.curl.php.

Sample Code

```
<?php
// Initialize curl
$ch = curl_init();

// Get curl to POST
curl_setopt( $ch, CURLOPT_POST, 1 );
curl_setopt($ch, CURLOPT_SSL_VERIFYHOST,0);
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, 0);


// Instruct curl to suppress the output from Beanstream, and to directly
// return the transfer instead.  (Output will be stored in $txResult.)
```

```
curl_setopt( $ch, CURLOPT_RETURNTRANSFER, 1 );

// This is the location of the Beanstream payment gateway
curl_setopt( $ch, CURLOPT_URL, "https://www.beanstream.com/scripts/process_transaction.asp" );

// These are the transaction parameters that we will POST
curl_setopt( $ch, CURLOPT_POSTFIELDS,
"requestType=BACKEND&merchant_id=109040000&trnCardOwner=Paul+Randal&trnCardNumber=51000000
10001004&trnExpMonth=01&trnExpYear=05&trnOrderNumber=2232&trnAmount=10.00&ordEmailAddress=pr
andal@mydomain.net&ordName=Paul+Randal&ordPhoneNumber=9999999&ordAddress1=1045+Main+Street
&ordAddress2=&ordCity=Vancouver&ordProvince=BC&ordPostalCode=V8R+1J6&ordCountry=CA" );

// Now POST the transaction.  $txResult will contain Beanstream's response
$txResult = curl_exec( $ch );

echo "Result:<BR>";
echo $txResult;

curl_close( $ch );
?>
```

2.7.3 Sample Java Code

The section contains an example of how to POST a transaction to the Beanstream server using Java. It has been tested with JDK 1.3 and 1.4.

Sample Code

```
import java.io.*;
import java.net.*;
import javax.net.ssl.*;

public class HttpsPost
{
        public static void main( String[] args ) throws Exception
        {
                int ch;

                // These are the transaction parameters that we will POST
                String messageString =
"requestType=BACKEND&merchant_id=109040000&trnCardOwner=Paul+Randal&trnCardNumber=51000000
10001004&trnExpMonth=01&trnExpYear=05&trnOrderNumber=2232&trnAmount=10.00&ordEmailAddress=pr
andal@mydomain.net&ordName=Paul+Randal&ordPhoneNumber=9999999&ordAddress1=1045+Main+Street
&ordAddress2=&ordCity=Vancouver&ordProvince=BC&ordPostalCode=V8R+1J6&ordCountry=CA";

                // Set the location of the Beanstream payment gateway
                URL url = new URL( "https://www.beanstream.com/scripts/process_transaction.asp" );

                // Open the connection
                URLConnection conn = url.openConnection();

                // Set the DoOutput flag to true because we intend
                // to use the URL connection for output
                conn.setDoOutput( true );

                // Send the transaction via HTTPS POST
                OutputStream ostream = conn.getOutputStream();
                ostream.write( messageString.getBytes() );
```

```
                    ostream.close();

                    // Get the response from Beanstream
                    InputStream istream = conn.getInputStream();
                    while( ( ch = istream.read() ) != -1 )
                    {
                            System.out.print( ( char )ch );
                    }
                    istream.close();
            }
    }
```

To Use This Example:

The following is a checklist of things you will need to do in order use the sample code:

- ✓ Install the Java Secure Socket Extension (JSSE) if you are using a version of the JDK earlier than 1.4
- ✓ Ensure that jsse.jar, jnet.jar and jcert.jar are in your classpath if using a version of the JDK earlier than 1.4
- ✓ Ensure that the java.security file is complete
- ✓ Import the Equifax certificate to the client's (your computer's) trusted certificate keystore

Installing JSSE

If you are using a version of the JDK that is earlier than version 1.4, you will need to download and install the Java Secure Socket Extension. This will implement a Java version of Secure Sockets Layer (SSL), which is required to securely communicate with the Beanstream server. You can download it from the Sun website at http://java.sun.com/products/jsse/.

Setting the Classpath

If you are using a version of the JDK that is earlier than version 1.4, you will need to ensure that jsse.jar, jnet.jar and jcert.jar are in your classpath. In Windows, this is done by modifying the CLASSPATH environment variable in Control Panel → System → Advanced tab. Under the *Advanced* tab, click the *Environment Variables* button to bring up the *Environment Variables* dialog. In the *System Variables* section of this dialog, make sure there is a variable called CLASSPATH and that it contains paths to jsse.jar, jnet.jar and jcert.jar.

In UNIX/Linux, there are two ways set the CLASSPATH environment variable, depending on your shell. In csh, the CLASSPATH is modified with the setenv command. For example:

```
setenv CLASSPATH=/usr/java/jdk1.3.1_01/jre/lib/jsse.jar
```

In sh, the CLASSPATH is modified with these commands:

```
CLASSPATH=/usr/java/jdk1.3.1_01/jre/lib/jsse.jar export CLASSPATH
```

Modify java.security

Your java.security file should contain the following lines. If not, you will need to add them.

```
security.provider.1=sun.security.provider.Sun
security.provider.2=com.sun.net.ssl.internal.ssl.Provider
security.provider.3=com.sun.rsajca.Provider
```

Adding the Equifax Certificate to the Keystore

Beanstream uses a certificate provided by Equifax, which Java does not recognize. Because of this, you will need to add the Equifax certificate (provided by Beanstream) to your computer's trusted certificate keystore, which is a file called cacerts. To do this, use the keytool utility provided by the JDK. For example: `keytool -import -alias equifax -keystore cacerts -file ESCA.cer`

The above example will work if you are in the directory where the cacerts file is located and have copied the ESCA.cer certificate to the same directory. If this is not the case, you will need to specify the correct pathnames to these files.

In UNIX/Linux, the cacerts file is located in your JDK directory under ./jre/lib/security/. In Windows, there may be two copies of the cacerts file—one in the JDK directory under .\jre\lib\security, and one in the Program Files directory under .\java\j2re1.4.0_01\lib\security (JDK 1.3) or .\java\j2re1.4.0_01\lib\security (JDK 1.4). Usually, the cacerts file in the Program Files directory is the one that is used, but if that doesn't work for you, try the one in the JDK directory.

If you do not have the ESCA.cer file, you can download it from Beanstream via the following URL: https://www.beanstream.com/admin/support/ESCA.cer

2.7.4 Troubleshooting

Issue I've imported the Equifax certificate into my cacerts file, but I still get the error: "Exception in thread "main" javax.net.ssl.SSLHandshakeException: Could not find trusted certificate".

Resolution You may not have added the certificate to the existing cacerts file. If you run the keytool utility to install the certificate and keystore cannot find the cacerts file, it will create a new one in the current directory. Make sure that you have added the certificate to the existing cacerts file by specifying the correct path to the cacerts file when running the keytool utility, or by running the keytool utility while in the directory where cacerts is located.

Also, if you are using Windows, there may be more than one cacerts file. It is commonly located in both the JDK directory and in Program Files\Javasoft (JDK 1.3) or Program Files\Java (JDK 1.4). This may be the reason that the Java runtime reports that the certificate has not been imported into the cacerts file.

Issue I get the following error: "java.net.MalformedURLException: unknown protocol: https".

Resolution You need to install the Java Secure Socket Extension (JSSE). You can download it from the Sun website at http://java.sun.com/products/jsse/.

2.7.5 ASP Example with Verified by Visa

The following script is an example of how to integrate a Verified by Visa-capable solution using ASP and the Microsoft XML Core Services (MSXML) version 4.0. (MSXML is also known as the Microsoft XML Parser).

This piece of code will perform the initial transaction request, and if a redirection response page is found in the response, will show this page to the client's web browser. The Terminal URL page used here is https://www.beanstream.com/samples/sample_s2s_vbv_auth.asp; you will have to change this to whatever location your actual Terminal URL page is located for this example to work. (The line containing the location of the Terminal URL page has been bolded for your convenience.)

To use this example, you must have MSXML 3.0 or 4.0 installed on your server. For more information on how to download and install MSXML, see the MSDN documentation at http://msdn.microsoft.com/downloads/default.asp?url=/downloads/sample.asp?url=/msdn-files/027/001/766/msdncompositedoc.xml

Sample Code

```
<%
option explicit

'Set to the address of the Beanstream server.
const BEANSTREAM_SERVER = "www.beanstream.com"
const MERCHANT_ID       = 107380000
const TERM_URL          = "https://www.beanstream.com/samples/sample_s2s_vbv_auth.asp"

dim objXMLHTTP
dim beanstreamResponse
dim postData

'Send transaction request string to be posted to the Beanstream system
postData=
"requestType=BACKEND&trnType=P&trnCardNumber=4030000010001234&trnExpMonth=12&trnExpYear=22
&trnAmount=1%2e00&merchant_id=" & MERCHANT_ID &
"&trnCardOwner=Paul+Randal&trnOrderNumber=1a&ordEmailAddress=prandal@mydomain.net&ordName=Pa
ul+Randal&ordPhoneNumber=60411234567&ordAddress1=1045+Main+Street&ordAddress2=&ordCity=Vanco
uver&ordProvince=BC&ordPostalCode=V8R+1J6&ordCountry=CA&termUrl=" & server.urlEncode(TERM_URL)

'Create the ServerXMLHTTP object
set objXMLHTTP = Server.CreateObject( "MSXML2.ServerXMLHTTP.4.0" )
objXMLHTTP.setOption(2) = 4096
objXMLHTTP.setOption(3) = ""

'This is the location of the Beanstream payment gateway
objXMLHTTP.Open "POST", "https://" & BEANSTREAM_SERVER & "/scripts/process_transaction.asp", false

'Set the HTTP header's content type
objXMLHTTP.setRequestHeader "Content-Type", "application/x-www-form-urlencoded"

'Submit the transaction request to the Beanstream server
objXMLHTTP.Send( postData )

'Read the transaction response returned from the Beanstream system
beanstreamResponse = objXMLHTTP.ResponseText

'We have now received a response from Beanstream.  Now check if this response is a Redirection
'Response Page by checking the value of the responseType parameter.  If the responseType paramter
'is set to "R" it is a redirection repsonse.  If the response type parameter is a "T" it is a
'transaction approved/delined response.

'response.write beanstreamResponse : response.end
if GetQueryValue(beanstreamResponse, "responseType" ) = "R" then
	'We have a Redirection Response Page, so show it to the browser
	response.write GetQueryValue(beanstreamResponse, "pageContents")
else
```

```
            'This is a normal transaction, so beanstreamResponse contains the results of the transaction.
            if GetQueryValue(beanstreamResponse, "trnApproved" ) = "1" then
                    response.write "Transaction Approved"
            else
                    response.write "Transaction Declined: " & beanstreamResponse
            end if
end if


Function GetQueryValue(queryString, paramName)
 'Purpose: To return the value of a parameter in an HTTP query string.
 'Pre:     queryString is set to the full query string of url encoded name value pairs. ex:
"value1=one&value2=two&value3=3"
 '    paramName is set to the name of one of the parameters in the queryString.  ex: "value2"
 'Post:    None
 'Returns: The function returns the query string value assigned to the paramName parameter.  ex: "two"

    Dim pos1
    dim pos2
    Dim qString

    qString = "&" & queryString & "&"
    pos1 = InStr(1, qString, paramName & "=")
    If pos1 > 0 Then
        pos1 = pos1 + Len(paramName) + 1
        pos2 = InStr(pos1, qString, "&")
        If pos2 > 0 Then
            GetQueryValue = DecodeQueryValue(Mid(qString, pos1, pos2 - pos1))
        End If
    End If

End Function

Function DecodeQueryValue(qValue)
 'Purpose: To URL decode a string
 'Pre:     qValue is set to a url encoded value of a query string parameter.  ex: "one+two"
 'Post:    none
 'Returns: Returns the url decoded value of qValue.  ex: "one two"

    Dim i
    Dim qChar
    dim newString

            if IsNull(qValue) = false then
                For i = 1 To Len(qValue)
                    qChar = Mid(qValue, i, 1)
                    If qChar = "%" Then
                                on error resume next
                        newString = newString & Chr("&H" & Mid(qValue, i + 1, 2))
                                on error goto 0
                        i = i + 2
                    ElseIf qChar = "+" Then
                        newString = newString & " "
                    Else
                        newString = newString & qChar
                    End If
                Next
                        DecodeQueryValue = newString
            else
```

```
                    DecodeQueryValue = ""
            end if

End Function
%>
```

Terminal URL Page Sample Code:

```
<%
'This is a sample Terminal URL page that the merchant must have on their web
'server.  The Issuer Access Control Server (ACS) will redirect to this page
'during the Authentication stage (after the customer enters his password).

set objXMLHTTP = Server.CreateObject("MSXML2.ServerXMLHTTP.4.0")
objXMLHTTP.Open "POST", "https://www.beanstream.com/scripts/process_transaction_auth.asp", false
objXMLHTTP.setRequestHeader "Content-Type", "application/x-www-form-urlencoded"
objXMLHTTP.Send("PaRes=" & request("PaRes") & "&MD=" & request("MD"))
response.write objXMLHTTP.ResponseText
set objXMLHTTP = nothing
%>
```

3 Transaction Reporting

For information on transaction reporting, please review the Beanstream Reporting Guide. You can find this document in you membership area under the left menu heading “documentation.”

The Reporting Guide contains information on:

- ✓ Response messages (trn_Message)
- ✓ Online reporting interfaces
- ✓ Report downloads