



BLUEFIN API LIBRARY

Processing API's for PayConex, Bluefin Reporting Services,
Bluefin Transaction Status and the Bluefin Scheduling Layer

QSAPI 3.8, SLAPI 3.8, TSAPI 3.8, RSAPI 3.8

Bluefin has made efforts to ensure the accuracy and completeness of the information in this document. However, Bluefin disclaims all representations, warranties and conditions, whether express or implied, arising by statute, operation of law, usage of trade, course of dealing or otherwise, with respect to the information contained herein. Bluefin assumes no liability to any party for any loss or damage, whether direct, indirect, incidental, consequential, special or exemplary, with respect to (a) the information; and/or (b) the evaluation, application or use of any product or service described herein. Bluefin disclaims any and all representation that its products or services do not infringe upon any existing or future intellectual property rights. Bluefin owns and retains all right, title and interest in and to the Bluefin intellectual property, including without limitation, its patents, marks, copyrights and technology associated with the Bluefin services. No title or ownership of any of the foregoing is granted or otherwise transferred hereunder. Bluefin reserves the right to make changes to any information herein without further notice.

Related Bluefin Trademarks
Bluefin Payment Systems®
Bluefin®
PayConex®
QuickSwipe®
Decryptx®

©2019 All Rights Reserved.

8200 Roberts Drive, Suite 150
Atlanta, GA 30350
Toll Free (800) 675-6573
Fax: (858) 836-5692

UPDATED: January 24, 2019
Document Version 19.1

Contents

- Introduction – Bluefin API Library 6**
 - PayConex6
 - Reporting Services6
 - Transaction Status6
 - Bluefin Scheduling Layer.....6
 - API Requirements7
 - Benefits of the Bluefin Payment Gateway.....8
- Getting Started 9**
- HASH for Security..... 9**
- PayConex Transaction Interface 10**
 - Orientation10
 - API Functions11
 - QSAPI Request Format Variables Index14
 - QSAPI Response Format Variables Index.....24
 - Account Balance Inquiry (Elavon Only).....26
- Transaction Status Interface 28**
 - Orientation28
 - TSAPI Functions30
 - TSAPI Request Format Variables Index.....30
 - TSAPI Response Format Variables Index.....31
- Scheduling Layer Interface 32**
 - Orientation32
 - API Functions32
 - SLAPI Request Format Variables Index34
 - SLAPI Response Format Variables Index.....36
 - SLAPI Appendix A: List of Recurring Schedules.....37
 - Start Date versus Reference Date.....38
- Reporting Service..... 39**
 - Orientation39
 - Special Note on ACH reporting:39
- RSAPI Functions 40**
 - RSAPI Request Format Variables Index.....40

RSAPI Response Format Variables Index	42
RSAPI HTTP Response Format Variables Index	45
Appendix: Using HASH for Authenticated Transactions	46
Variables Index	47
Appendix: Transparent Redirect.....	48
Overview.....	48
Configuration	48
Configuration (PayConex)	50
Variables Index	51
Appendix: AJAX / CORS Support.....	52
Overview.....	52
Configuration	52
Browser Support.....	52
Sample Code (PHP and jQuery):	53
Appendix: POSTback.....	55
Orientation	55
Implementation steps:.....	55
Custom Process Flow	56
POSTback Response Payload	56
POSTback Example Response	57
Appendix: Transaction Flow Diagrams	58
Appendix: Point to Point Encryption	61
Overview:.....	61
Orientation:	61
Variables Index	63
Appendix: Best Practices.....	64
Utilizing responses.....	64
Utilizing reporting.....	64
Card capture	64
Error messages:	65
Zero Dollar (\$0) Authorizations	65
URL Encoding	65
Appendix: Request and Response Codes	66
Overview.....	66
Code Samples	68

QSAPI	70
TSAPI	82
SLAPI	84
RSAPI	87
Appendix: Transaction Responses and Messages	90
Transaction Origins:	90
Transaction Authorization Response Codes:	90
Authorization Response Codes:	90
AVS/CVV2 Authorization Responses:	90
FraudFirewall Authorization Responses:	91
Traffic Limit Authorization Responses:	91
Bank Authorization Responses:	91
Transaction Status Codes:	94
EMV Data Overview	95
Request	95
Response	96
Example Request (Credit Card, EMV Contact)	96
Example Response (EMV related data)	97
EMV Tag Info:	97
Key exchange request	100
Key Exchange Response	100
Example Request	100
Example Response	100
Key Exchange Attributes	101

Introduction – Bluefin API Library

This document reviews Bluefin’s family of Application Programming Interface (API) services. These services provide our clients with unparalleled access, flexibility, and security for processing and managing electronic payment transactions.

PayConex

PayConex is Bluefin’s flagship transaction processing solution. The PayConex API (QSAPI) allows our customers to programmatically submit transactions through the PayConex Gateway. QSAPI’s flexible solutions allow our customers several options for submitting transactions while maintaining PCI compliance through the entire process. QSAPI supports any application or device that can connect through the Internet-based API and also offers PCI compliance scope reduction through technologies such as end-to-end encryption (E2E) and tokenization. When used in conjunction with our Transparent Redirection product, a merchant can greatly reduce PCI compliance scope by bypassing any permanent or temporary storage of cardholder data (CHD) on servers, networks, or computing devices.

Reporting Services

The Bluefin Reporting Services API (RSAPI) provides our customers with a level of access to reporting data rarely found in the industry. Using RSAPI, our customers can request formatted exports of transaction data from previous days. RSAPI’s reports contain no sensitive cardholder data, such as card numbers, meaning the data provided by RSAPI is 100% PCI compliant.

Transaction Status

The Transaction Status API (TSAPI) ensures processing and communication integrity. Transactions submitted through QSAPI are transmitted over the Internet. From time to time, an Internet Service Provider (ISP) or upstream Internet network (the backbone of the Internet) may lose a packet or timeout on a communication during the response from QSAPI to your system. TSAPI allows you to pre-fetch token IDs and then submit the token ID with a new transaction. If there is ever an Internet timeout, you can query TSAPI to give you the status of the transaction and whether it was received, approved, or declined. This reduces duplicate charges and enhances the overall integrity of the communication process.

Bluefin Scheduling Layer

The Bluefin scheduling layer allows our clients to create recurring payments without having to build a client-side recurring payment solution. The Scheduling Layer API (SLAPI) allows our clients to create a wide range of recurring transaction scenarios to manage the unique transaction processing needs of their business. The scheduling layer also allows our clients to access existing recurring payment records using our secure PCI compliant token system to modify, cancel, or view recurring payment schedules and details.

API Requirements

Use of the Bluefin Gateway and its API's has certain minimum requirements that must be met. In addition, there are various security configurations that are enforced, which are explained below.

- A merchant account that can accept transactions through First Data, Paymentech (PNS-Tampa) Netconnect, Elavon, Vital/TSYS, or ACHWorks/TSS is required. Other processors are being added, so please inquire with your sales representative for a current list of processor interfaces in progress.
- The merchant account must be properly underwritten and configured to support the intended payment channel: Ecommerce, Card Not Present (CNP), Card Present (swipe), etc.
- The merchant account must have the appropriate entitlements configured to support the appropriate bankcard or charge-card type: Visa, MasterCard, Discover, American Express, Diners Club, JCB, ACH and EBT.
- An appropriate PCI PED/PTS-compliant injected keypad or swipe device in order to accept PIN numbers, swiped card tracks, or to implement P2PE (point-to-point encryption) or E2E (end-to-end encryption) is required.
- The application must be capable of performing a CGI FORM POST over TLS1.1 or greater (HTTPS) via port 443 and storing access credentials securely.
- The software application, any service provider or host that is transmitting, storing, or processing cardholder data, and the merchant must be in compliance with the appropriate PCI SSC (Payment Card Industry Security Standards Council) security initiative, PCI-DSS (Data Security Standard) for merchants and service providers, or PA-DSS (Payment Application) for software vendors. PCI compliance for the application and merchant are the responsibility of the merchant and its application partners. For customers who want to ensure their PCI compliance, Bluefin provides an array of compliance services as part of their added service lines. Please contact your sales representative for more information.

Did You Know?

Implemented correctly, the Bluefin transaction process is PCI compliant. If you have any concerns about the PCI compliance of your existing corporate infrastructure, your legacy applications, or how to properly implement QSAPI for compliancy, please contact us at pciassist@bluefin.com to have a representative discuss our PCI compliance consulting services.

When you secure your merchant account with Bluefin, you have access to a very unique resource. Bluefin is one of a handful of merchant account providers who maintain their own Merchant Compliance Assistance Department. This means that your questions about PCI are answered expeditiously, you have access to our convenient online tools, and if you have need of external scans or need assistance with preparing for an on-site audit by a QSA (Qualified Security Assessor), Bluefin can serve as your expert advocate.

For PCI compliance and security reasons, merchants should not store cardholder data for any reason. Tokenization functionality is described herein that will allow merchants to perform reissues, refunds, returns, voids, and recurring billing without the need to store the card number.

Benefits of the Bluefin Payment Gateway

Feature	Details
Tokenization	The payment gateway stores the card number so the client does not have to, significantly minimizing the vendor/merchant's PCI footprint.
Point-to-Point Encryption (P2PE)	Bluefin encrypts the magnetic stripe (track data) and card data at the point of entry using a secure device rather than the computer keyboard. Merchants implementing Bluefin's PCI-validated P2PE solution can consider their encrypted cardholder data to be completely out of PCI scope.
Transparent Redirect	The transparent redirect feature is an elegant token-based method to securely and transparently collect card data directly from the cardholder while allowing the merchant to still manage the authorization process. Removes the vendor/merchant from PCI transmission scope.
Store & Convert	Bluefin supports a "STORE" transaction type that allows a vendor/merchant to convert all of their stored credit card and ACH numbers to tokens, via the API.
Transaction Status	The Transaction Status provides the vendor/merchant the ability to determine the status of a previously submitted authorization.
Recurring Payments	Recurring enables predefined, scheduled transactions, for credit card and ACH transactions. Vendor/merchants can manage, create, update, and delete via the API.
Reporting API	The Reporting API lets the vendor/merchant receive their previous day's transaction activity programmatically.

Getting Started

The first step in developing a connection from your application to any of the PayConex API's is getting set up with a certification account. You should have been assigned certification credentials during your setup process, but if you have not been assigned an account yet, please use the form at the following address to request an account:

<http://www.bluefin.com/software-integration/developer-resources/>

The certification environment (also referred to as CERT) allows our customers to develop their solution in an environment that is identical to our live production environment. Developing on CERT allows you the ability to test transactions without the fear of cards being submitted for actual processing, and removes any concerns over having to remove test data from production.

CERT accounts are provided with every integration project, and can be made available again at a later date should you wish to make coding changes on your end that you would like to test. The CERT environment does support all API functions, including support for M100/M130 encrypted devices as well as P2PE key injected devices.

There are two URLs that are important for the API and must be followed. When developing and/or testing, you need to use the CERT URL. For production (live) processing, you need to just change the first portion of the URL. Both URLs are shown below:

CERT (For Existing Implementations Only)

<https://cert.payconex.net/api/qsapi/3.8/>

PRODUCTION

<https://secure.payconex.net/api/qsapi/3.8/>

Note: The URLs shown above are the current standards. The CERT URL is for future and current use in your **testing**. All Merchants should be using TLS 1.2.

You will be issued different API credentials for certification and production. After the CERT process, when you are ready to process cards, you will need to update your API authentication credentials with the production credentials that were provided to you for your transactions to process correctly.

If you have any technical issues during the certification process that you need help with, please email us at support@bluefin.com. Members of our support staff can help answer your technical questions

HASH for Security

Bluefin highly recommends using a HASH for security reasons. A hash function is an algorithm that transforms (hashes) an arbitrary set of data elements, such as a text string or file, into a single fixed length value (the hash). The computed hash value is a means of protecting sensitive data. Bluefin has included this functionality into its API.

See "[Appendix: Using HASH for Authenticated Transactions](#)" (Pg. 43) for more information and its use.

PayConex Transaction Interface

PayConex's flexible, secure and PCI compliant API allows our customers the flexibility to run a wide variety of transaction types.

Post a Sale (Payment)

This allows a merchant to post a sale transaction to a variety of tender types (Card, ACH, etc.).

Post an Authorization (Authorization ONLY)

This allows a merchant to verify the ability of a card to accept a specific transaction amount without actually charging the card. Funds are reserved on the card for future capture.

Post a Capture

This allows a merchant to capture a previously authorized card, thus converting an Authorization (pre-authorization) into a Sale.

Post a Reissue

This allows a merchant to create a new transaction, using a previously tokenized card. This important function allows merchants to create new transactions based on card numbers stored at Bluefin, without storing cardholder data on their own systems.

Post a Refund

The most common use of this transaction type allows a merchant to refund an existing transaction, but will also reverse a recent payment transaction. You can refund the entire amount or a partial amount. If a transaction is refunded the same day it is run, then it results in voiding the sale back to a pre-authorization state. If the transaction is refunded after it has already been captured/settled, then it results in crediting the funds back to the card. Bluefin flexibly manages which is the appropriate action to take, so all you need to do is submit the refund.

Post a Credit

This allows a merchant the ability to credit money back onto a card. With this transaction type, there is no correlation to an original payment.

Store Only

This allows a merchant to store the cardholder data (card number, expiration, name, etc.) for a card, for later use, without running any actions against the card immediately.

Orientation

Bluefin has created this guide to describe the programming required to connect applications to Bluefin's PayConex Gateway in order to transmit payment transactions and related operations to various payment processors and payment gateways using QSAPI (PayConex Application Programming Interface).

The QSAPI interface is both a secure and PCI compliant processing solution.

TIP:

To post an automatically recurring transaction, please visit the Scheduling Layer API (SLAPI) portion of the document.

QSAPI is a language-agnostic, hosted payment acceptance system that supports the HTTP protocol over TLS (HTTPS) encrypted connections. QSAPI utilizes the HTTPS Request/Response model and will only support the HTTP POST method. The HTTP GET method is not accepted for security reasons.

API Functions

The content contained in this portion of the document outlines some of the key functions that can be performed using QSAPI. The examples listed below are not a holistic guide to all functions that can be accomplished with QSAPI; rather they are examples of common actions and are meant to help developers get acclimated with the QSAPI functions. Please visit the variables index at the end of the QSAPI portion of the documentation to review all of the posts that can be made through the QSAPI interface as well as the responses that the Bluefin payment gateway will make in response to those posts. The *Appendix: Code Requests and Responses* at the end of this document contain coding examples for QSAPI Requests and Responses.

TIP:

To see the responses that list the reasons for failed transactions, please review authorization messages listed in the index of QSAPI.

A. Post a sale

Our merchants can post a sale transaction to QSAPI for a variety of tender types. These types include credit cards, debit cards, check cards, ACH, EFT, electronic check, EBT, and gift card / stored value card. Using our secure API access key, our merchants can post the transactions and receive back immediate responses that will indicate the successful posting of a sale.

Merchants who post transactions can use our invisible redirect which passes the variables to our API, which in turn will pass a token ID for the transaction. That token ID then allows the merchant to retrieve reference data related to the customer and the transaction without ever actually having the cardholder data touch our merchant's web servers. The token ID can be stored on the merchant's servers and used again when the merchant wants to perform new actions for the customer.

TIP:

All the tender types from a coding standpoint function the same. Tender types based on credit card or debit and check cards resolve with the financial institutions in real time. ACH and electronic checks require longer resolution times to resolve the transfer of funds with the issuing financial institution responses.

B. Post an authorization

Authorization (or pre-authorization) allows for a card to be authorized to approve a dollar amount for a potential future purchase.

Authorizations are mainly used for two reasons. The first is by restaurants or other businesses that need to approve a card and account for an additional amount of money (usually tips) being added to the merchant-generated bill before the card is actually charged. The second most common use for an authorization is when a merchant wishes to charge a card that it has been previously tokenized with Bluefin, and wants to validate that the card can take a future charge.

For customer data that is securely held on the Bluefin servers, our merchants can use the token ID previously given to them for a consumer's card during a previous transaction via QSAPI, provided that the merchant stored the token ID and properly associated it with a previous sales record. In this case, it is typical to pre-authorize for \$0. This will validate that the card number, expiration date, address, and

CVV are correct and that the card account is actually open, while simultaneously tokenizing the card for future charges by the merchant. Examples for both scenarios are listed in the Appendix.

C. Post a capture

Using the capture feature of the API, our merchants can convert a previously pre-authorized card into a sale transaction via QSAPI.

There are limited scenarios in which a merchant may get verification for a card outside of Bluefin. These scenarios are most common when a merchant contacts the card issuer over the phone, verifies the ability of that card to process the desired transaction, and the merchant is given a verification ID number. For those cases, please use the Force transaction type listed in this document.

You may also capture for more than the originally authorized amount for merchants that are allowed to accept tips. Similarly, you can capture less than the original authorization amount.

D. Post a reissue

A reissue allows our merchants the ability to post new charges against cards previously used by their customers. Merchants who have previously posted a transaction and who have the Bluefin issued token ID associated with that card stored on their system can post that token ID with new sales transaction requests in lieu of providing the original card number. This function is an important part of the tokenization process. By using Reissues and Refunds with token IDs instead of providing the full cardholder information, a merchant can effectively eliminate the need to store cardholder data on their own servers.

E. Post a refund

By posting a refund, a merchant can directly reverse a sale charge that was applied to a card. The refund action is predicated on there being a previous transaction that was sent through QSAPI. The refund function is an intelligent function that can verify if the previous transaction (identified by its token ID) has been fully settled with the issuing bank. Based on whether it has been fully settled or not, it will perform either a credit action or void the sale, converting back to an authorization-only status.

F. Post a credit

Credits allow merchants to return funds to a card or ACH account. There is no actual tie to a pre-existing transaction. The only limit imposed would be on the merchant account by their issuing bank. The card and/or ACH account will be verified however.

G. Post a force

There are rare instances in which a merchant may not be able to or wish to get proper authorization on a transaction. In these cases, the merchant is provided with a message to call their merchant processor for an approval code. Once received, this method can be used to submit the new approval account along with a transaction to force the transaction to settle.

FYI:

Due to the complexity of proper card charging policies and the risk of merchant losses due to improper implementation, in most cases, a refund is the most appropriate action instead of a credit.

The use of the force allows the merchant to post the transaction to the cardholder's account without approval, and then using the force, post pass an approval code to complete the transaction. These charges, however, will not be applied until settlements are run during the batch process.

H. Store a card

Most cardholder data that we store is the result of an initial transaction. However, merchants often need to store cards for future transactions with their customers, without ever needing to apply a charge at the time of the record creation with Bluefin. Common usages for this are businesses that require delayed potential transactions such as hotels that require a card for potential incidentals, or online merchants who put a card on file when a user creates a shopping account.

It is highly recommended to pre-authorize a card for \$0.00 instead of doing a “Store Only” function because you will then know in real-time whether the card number is active and good, has funds, and that details such as expiration date, address, and card verification codes match with what the cardholder’s issuing bank has on file. All too often, a merchant stores a card and then when going to charge the card at a later point finds out that invalid details were provided by the cardholder. If they had authorized a card for \$0.00, they would have known in real-time and could have prompted the user to correct the details or provide a new payment method. However, some merchants will still choose to perform a store only operation.

TIP:

If you would like to store a card to be used in a recurring payment, please visit the SLAPI portion of this document to learn how to setup the recurring schedule after you store or authorize the card.

Using QSAPI, the cardholder data is sent to Bluefin and not passed onto the issuing bank in real-time for authorization or verification. QSAPI then acknowledges receipt of that information by passing back a token ID that our merchant can then use at a later date to use the card number when they wish to apply actions such as a sale.

Note: With the STORE transaction type, no actual transaction occurs. As such, the “transaction approved” response will be false. To confirm whether the store request was submitted correctly, simply look for the “transaction id” for the token that was returned. To ensure no errors occurred in the STORE request, you may look for failed responses in the error, error code, or error message parameters as well.

I. Handling a partial authorization

If you are processing cards in a retail or e-commerce environment there are unique instances, most notably with stored value cards, in which a card will not have sufficient value to cover the entire purchase. In such instances, you can perform what is called a partial auth. Rather than failing as insufficient funds, the merchant will be presented with the opportunity to capture the sale with two separate transactions.

If partial authorization is enabled in your account settings, the stored card will be authorized for its maximum value, and that value will be sent back to the merchant’s software application. Based on how the merchant chooses to handle the situation programmatically, they can accept the initial authorization and add a second new transaction for the remaining value, or they can void the initial authorization and ask the customer to run the transaction with a different card that can handle the full value.

Here is an example scenario:

The Customer wishes to purchase an item for \$100. They present the retailer with a pre-paid Visa card, which says \$100 on it, but in reality, only has \$75 left. The merchant sends Bluefin the card data and transaction amount, and QSAPI responds to the merchant with the following information:

auth_amount = 75

requested_amount = 100

This indicates the card was just authorized for \$75, and the real transaction amount was \$100. At this point the \$75 authorization will either need to be captured as a sale, or cancelled. If canceled the merchant would have their software handle the cancel like any other cancelled transaction.

If the merchant elects to keep the \$75 authorization, then they would need to have their software take the value of the \$100 requested amount and subtract the authorized amount of \$75 to come up with a remaining \$25. The merchant would now run a new and separate \$25 transaction for the customer.

*Please note that partial authorizations are only supported on the First Data North, First Data Omaha, and Elavon networks.

J. Level 2 Transactions / Purchasing Cards

If you are supporting purchasing cards and level 2 transactions you will be required to capture and pass additional data fields in order to comply with the requirements issued by the major card brands. Without the use of the additional fields, your transactions will not qualify as level 2 transactions.

The additional fields to be included in the post are:

level2_tax (required, Visa must be greater than 0.00, MasterCard allows 0.00)

level2_merchant_reference (required)

level2_zip (recommended)

level2_orderid (recommended)

*see QSAPI Request Format Variables Index for more details on these fields.

K. Restaurant Transaction

To pass qualifying restaurant transactions the following additional values will need to be passed in with your post.

restaurant_server_id (must be a numeric value)

restaurant_gratuity

Please note that the value of “restaurant_gratuity” passed in a transaction is NOT added to the total value of your sale, it is INCLUDED in the total sent. For example, if you authorize for \$10, and your customer adds a \$2 tip, you must pass in a sale value of \$12, and state that \$2 of that total value is intended for restaurant gratuity.

QSAPI Request Format Variables Index

Below are all of the variables that can be posted to QSAPI along with a brief description of their function.

Variable Name	Max	Type	Req'd	Description
account_id	12	Numeric	Yes	The Payconex account identification number that you are issued after your account has been setup.
api_accesskey	32	Alphanumeric	Yes	The secret key that you will be provided when your Payconex account is set up and when you have requested access to QSAPI.

timestamp	19	YYYY-MM-DD HH:MM:SS	No	If used, MUST be included in a hash for authenticated transactions. See "Appendix: Using HASH for Authenticated Transactions"
tender_type	n/a	Enumerated	Yes	The payment tender type that you are submitting. The following enumerated values are allowed: CARD: credit, debit, and check cards ACH: ACH , EFT, or electronic check EBT: Electronic Benefits Transfer (Elavon only) DEBIT: PIN Debit card only (Elavon/Omaha/North)
transaction_type	n/a	Enumerated	Yes	The type of transaction you are requesting, with these enumerated values allowed: AUTHORIZATION: authorizes (holds) the funds on the card but does not transfer them. Most banks support a \$0.00 authorization in order to validate the card number, expiration date, and account status. This does, however, incur an authorization charge on the merchant account and a transaction charge in Payconex. SALE: authorizes the funds on the card and flags the transaction to be captured for settlement at the next settlement time. REFUND: refunds a previous sale. If the transaction has not yet been settled, then this results in a void. Otherwise, for Cards only, it results in a credit back on the card. ACH transactions can't be refunded once they are submitted for settlement (NOTE: For actual transaction settlement times, contact Bluefin support). You can specify an amount less than the original sale amount. Requires token_id. CREDIT: puts money onto a card or into a bank with no offsetting sale. Most operations can be managed via REFUND. Only allowed if account is configured to allow Credits. CAPTURE: flags a previous authorization to be captured for settlement at the next settlement time. Requires token_id. SETTLEBATCH: settles all un-settled

				<p>sales, refunds, credits, or authorizations that have been captured.</p> <p>STORE: stores credit card/ach account info for later use.</p> <p>FORCE: forces through a transaction. A 6 digit authorization_code must also be provided.</p> <p>REVERSAL: removes an authorization request on a credit card or debit/ebt. Requires token_id (Elavon, Paymentech, RapidConnect only)</p> <p>BALANCE: request account balance on an EBT/Debit card (Elavon only).</p> <p>CANCEL: removes a refund on a credit card or debit/ebt. Requires the refund token_id (RapidConnect only)</p>
transaction_amount	9	Numeric with decimal	Yes	This is the dollar (or other currency) amount of the transaction. Only numbers and a single decimal are allowed. Commas are not allowed. The maximum amount is 999999.99. That is 1 cent less than 1 million. This is because the decimal is counted in the max size. Values with no decimal and no cents are allowed. Values with only a single number after the decimal are allowed and will be assumed to have a trailing 0.
transaction_description	65K	Character	No	A description of the payment. This is an open field. If emails are sent to the customer or merchant, this will show in the "Description:" field. You may use this to send any information that you wish. It can store up to 65,000 characters.

card_number	16	Numeric	Yes/No	The card number with no spaces, dashes, or hyphens. Required for card transactions using KEYED entry (see page 55-56). It is not required for ACH/electronic checks.
card_expiration	4	Numeric	Yes	The card expiration date in the format of MMY. This does not include hyphens, dashes, spaces, or slashes. It is required if submitting a card. It is not required for ACH/electronic checks.
card_verification	4	Numeric	No	The CVV/CVC/CID value which is the 3 digits from the signature panel on the back of a Visa/MasterCard/Discover or the 4 digits from the front of an American Express.
card_tracks	?	Character	Yes/No	The characters from the full, unmodified payload from the magnetic stripe on a card. This is now the preferred method to send card track data and replaces both "card_track1" and "card_track2" parameters, so do not send "card_tracks" in combination with those. Track data may not be stored for any reason. Required for card transactions using SWIPED entries (see page 55-56).
card_track1	?	Character	No	Should use "card_tracks" parameter instead. The characters from track one of the magnetic stripe on a card. Track 1 begins with "%" and ends with "?" and includes the cardholder name. If you do NOT use "card_tracks", this is the primary choice in choosing which track to send. Track 1, Track 2, or both can be sent, but NOT with "card_tracks". Track data may not be stored for any reason.
card_track2	?	Character	No	Should use "card_tracks" parameter instead. The characters from track 2 of the magnetic stripe on a card. Track 2 begins with ";" and ends with "?" and does not include the cardholder name. Track 1, Track 2, or both can be sent, but NOT with "card_tracks". Track data may not be stored for any reason.

cashier	100	Character	No	The name or id of the cashier that is submitting the transaction. This is shown with PayConex transaction details as the originator of the transaction. You may use any designation you wish. Good choices are the user name of the POS clerk, email address, or the name of the application that is connecting.
bank_account_number	26	Numeric	No	The bank account (DDA) number that is required for an ACH/electronic check.
bank_routing_number	9	Numeric	No	The bank routing (ABA) number that is required for an ACH/electronic check.
check_number	15	Numeric	No	The number of the check used for electronic checks that are processed via ACH. It is optional.
ach_account_type	n/a	Enumerated	No	This is the type of bank account for ACH/electronic check transactions. It will default to checking if none is specified. The allowed values are: CHECKING : checking account SAVINGS : savings account
token_id	12	Numeric	No	12 digit transaction_id of a previous transaction. The token_id is used for reissues, refunds, and recurring transaction creation. Please see the SLAPI documentation for more information.
group	12	Alphanumeric	No	Groups are pre-configured flexible groups that can be used for various reasons, including: a) to direct transactions to separate back-end merchant accounts or depository accounts. Please work with your Bluefin Representative to configure any of these options. b) to assign transactions to a specific grouping that you wish.
custom_id	50	Character	No	This is a custom identifier that can be used for any purpose you wish. Often times this is a customer number or some other foreign key used to match up reports and transactions lists with customer information on your database. For some processors, such as Paymentech, this ID is passed through to the processor and available in their reporting. This can ease syncing up reporting.

custom_data	65K	Character	No	An open variable. Many developers use this variable to transmit structured data formats or serialized variables. You can send through many variable=value pairs through this single variable.
input_group	10	Character	No	An open variable. Used to identify or group transactions together in some fashion.
first_name	50	Character	Yes/No	Card: The first name of the cardholder as it appears on the front of the card. It is not required for cards. ACH: The first name of the account holder as it appears on the front of the check or bank statement. It is required by NACHA to provide first and last name.
last_name	50	Character	Yes/No	Card: The last name of the cardholder as it appears on the front of the card. It is not required for cards. ACH: The last name of the account holder as it appears on the front of the check or bank statement. It is required by NACHA to provide first and last name.
street_address1	100	Character	No	Street address of the cardholder or bank account holder.
street_address2	-	Character	No	Suite number or other qualifying part of the address. NOT sent to the processor. Total of address1 and 2 are 100 maximum and will be truncated
city	100	Character	No	The city portion of the cardholder or account holder address. This is not sent to the processor. It is only stored for your reporting purposes.
state	2	Alphabetic	No	The two digit state code of the cardholder or account holder address. This is not sent to the processor. It is only stored for your reporting purposes.
zip	10	Numeric with hyphen	No	The 5 digit format or 5+4 formatted zip code of the cardholder or account holder. For example, 12345 or 12345-1234. Only numbers and a hyphen are allowed. INTERNATIONAL: Can contain any combination of letters or numbers, and either a space or a hyphen.

country	3	Alphabetic	No	This is a 2 or 3 character country code value for the card/account holder: http://www.iso.org/iso/country_codes Note: We do NOT validate the value.
authorization_code	6	Alphanumeric	No	For credit/debit cards, this is the 6 digit authorization code from a previously authorized transaction that is required to be provided with a Force transaction. It may also be obtained by calling the merchant account processor for a force code. For EBT cards, this is the authorization code that is required along with the voucher number to capture a prior authorized transaction.
pin	?	Alphanumeric	No	The encrypted PIN Block portion for PIN debit or PIN EBT transactions. It must be obtained from a PCI PTS/PED Certified device that is injected by Bluefin's Encryption Service Organization or Key Injection Facility (KIF). This PIN Block may never be stored for any reason.
ksn	?	Alphanumeric	No	The Key Sequence Number (DUKPT) portion for PIN debit, PIN EBT, or EMV transactions. It must be obtained from a PCI PTS/PED Certified device that is injected by Bluefin's Encryption Service Organization or Key Injection Facility (KIF). The KSN may never be stored for any reason.
ebt_type	n/a	Enumerated	Yes/No	For EBT (Electronic Benefits Transfer), the type can be as follows: FOOD : this is for a food sale CASH : this is for a cash sale VOUCHER : this is for a voucher Required if tender_type=EBT
ebt_voucher	15	Numeric	No	The EBT Voucher number for prior (phone) authorized EBT transactions. Required if "ebt_type" is VOUCHER
disable_avs	1	Boolean	No	Disable Address Verification.
disable_cvv	1	Boolean	No	Disable Card Verification such as CVV, CVC, CID.
disable_fraudfirewall	1	Boolean	No	Disable any Fraud Firewall controls.

ach_sec_code	n/a	Enumerated	No	<p>The Standard Entry Class (SEC) code that is required for ACH/echeck transactions. If no SEC code is provided, the default that is set up for the account is used. Please note that if you provide an SEC Code here that the account is not underwritten for, the bank will decline the transaction.</p> <p>Acceptable types are:</p> <p>CCD: Cash Concentration or Disbursement. This is the default type for corporations. Requires a signature.</p> <p>PPD: Prearranged Payment & Deposit. Requires a signature.</p> <p>WEB: Web-originated, ecommerce transactions.</p> <p>TEL: Telephone-initiated transactions. Voice recording required.</p> <p>POP: Point-of-Purchase, in-person transaction.</p> <p>ARC: Accounts Receivable. This is for converting a check into an electronic ACH transaction.</p> <p>RCK: Re-presented Check. This is used to present a declined check an additional time.</p> <p>DEF: This can be sent to tell PayConex to use the default ACH SEC code. Sending nothing will result in the same action.</p>
ach_opcode	n/a	Enumerated	No	For processor-specific ACH features. 01, 02, 03, S, R
phone	20	Alphanumeric	No	The phone number of the cardholder/account holder. It does not expect any specific format, is not sent to the processor, and is stored only for your reporting use.
email	100	Character	No	The email address of the cardholder/account holder. It does not expect any specific format, is not sent to the processor, and is stored only for your reporting use or sending email receipts.
send_customer_receipt	1	Boolean	No	Use this to override the default setting to send or not send email receipts to the customer.
send_merchant_receipt	1	Boolean	No	Use this to override the default setting to send or not send email receipts to the merchant.
ip_address	15	NNN.NNN.NNN.NNN	No	IP address of the client which initiated the transaction.

transaction_id	12	Numeric	No	The transaction_id returned from a TSAPI "GET_TRANSACTION_ID" request. It is used to create a new transaction with the specified transaction_id. Please see the TSAPI documentation.
response_format	5	Enumerated	No	Desired response format. FORM: www-form-urlencoded string (default format) JSON: JavaScript Object Notation DEBUG: human readable array output
allow_partial	1	Numeric	No	Allows for support of partial auths. By default QSAPI assumes all transactions do not support partial auth unless specified. Values for partial auth are: 0: declares no partial auths allowed. Unneeded declaration however since by default partial auths are not allowed. 1: partial auths allowed. *only supported in QSAPI 3.7 or higher
level2_tax	???	Numeric with decimal	No	Used exclusively for level 2 transactions. Required for level 2 transactions. For Visa cards, value must be expressed as greater than 0.00, for MasterCard value may be expressed as 0.00 if desired. *only supported in QSAPI 3.7 or higher
level2_zip	10	Numeric with hyphen	No	Used exclusively for level 2 transactions. REQUIRED for level 2 transactions and it is probably the same value as the "zip" variable value. This value is not validated like the "zip" variable value, so any alphanumeric value up to 10 digits is valid. *only supported in QSAPI 3.7 or higher
level2_order_id	17	Alphanumeric	No	Used exclusively for level 2 transactions. Merchant determines any custom alphanumeric value. *only supported in QSAPI 3.7 or higher
level2_merchant_reference	25	Alphanumeric	No	Used exclusively for level 2 transactions. Required for level 2 transactions. Merchant determines any custom alphanumeric value. *only supported in QSAPI 3.7 or higher
restaurant_server_id	3	Numeric	No	Used exclusively for restaurant transactions. Value can be null, 0 ~999 *only supported in QSAPI 3.7 or higher

restaurant_gratuity	9	Numeric	No	Used exclusively for restaurant transactions. Value can be 0 ~ 999999.99 Value must contain decimal point with two decimal point values. *only supported in QSAPI 3.7 or higher
payment_type	11	Enumerated	No	The type of payment for this transaction. Valid values can be: ECOMMERCE, INSTALLMENT, RECURRING, or MOTO (Mail & Telephone Order)
installment_number	???	Numeric	No	Used exclusively for installment transactions, and is the current installment number. *only supported in QSAPI 3.7 or higher
installment_count	???	Numeric	No	Used exclusively for installment transactions. Total number of installments. Please note that the installment count value never decreases, only the installment number should increase. *only supported in QSAPI 3.7 or higher.
reissue	1	Boolean	No	Reissue a transaction. If included (set to 1), must use "token_id" with known existing value. Amount, name, description, expiration can be changed.
disable_redirect	1	Boolean	No	Disabling redirect will override the success_url and decline_url settings, and force the return of transaction response in the format specified by response_format. This setting is required for using AJAX API calls.
merchant_reference_num	11	Numeric	No	An optional merchant reference number that can be passed to processor for reconciliation purposes.
cashback_amt	9	Numeric	No	Amount requested as cash back (cash returned to the customer)
surcharge_amt	9	Numeric	No	Amount charged for fees, etc.
etoken	???	Alphanumeric	No	eToken value of a previously ran Bluefin iFrame transaction. Typically used in a Sale transaction, with the eToken variable and value sent. No card or PAN needed when using eToken.

QSAPI Response Format Variables Index

Below are all the variables that can be received into response to posts made to QSAPI along with a brief description of their function.

Variable Name	Max	Type	Description
transaction_id	12	Numeric	The transaction id for the new transaction. When using tokenization, this is the transaction_id that you submit as the token_id.
original_transaction_id	12	Numeric	Will be returned for refunds. This is the original transaction_id of the Sale transaction that was refunded. It is provided for reference purposes.
tender_type	n/a	Enumerated	This will be the same as the tender_type provided in the request. It is provided for reference purposes. See Response Format for list.
transaction_timestamp	19	YYYY-MM-DD HH:MM:SS	This is the timestamp of the newly created transaction.
card_brand	n/a	Enumerated	VISA, MASTERCARD, AMERICAN EXPRESS, DISCOVER, ACH, EBT
transaction_type	n/a	Enumerated	This is the transaction_type of the original transaction.
last4	4	Numeric	The last four digits of the card number or Primary Account Number (PAN). For ACH, it is the last four digits of the account number.
card_expiration	4	Numeric	The month and year of the card expiration date in the format MMY. For example, 0120 for January 2020.
authorization_code	6	Alphanumeric	This is the auth code returned by the processor.
authorization_message	50	Alphanumeric	APPROVED or the auth message from the processor (e.g. AUTH DECLINED 200).
transaction_amount	9	Numeric with decimal	The same variable submitted in the request. If a partial auth is valid, this represents the actual amount authorized.

avs_response	1	Enumerated	A single letter address verification response: DFJMQVXY - Address and ZIP code match LWZ - ZIP code match, address is wrong ABOP - Address match, ZIP code is wrong KN - No match, address and ZIP is wrong U - No data from issuer/banknet switch R - AVS System unable to process S - Issuing bank does not support AVS E - Error, AVS not supported for your business C - Invalid address and ZIP format (International) I - Address not verifiable (International) G - Global non-verifiable address (International) ? - Unrecognized codes (none of the above) (empty) - No AVS data (blank)
cvv2_response	1	Enumerated	A single letter card verification value response: M - CVV match N - CVV does not match P - CVV not processed S - Card has CVV, customer says it doesn't U - No CVV data from issuer ? - Unrecognized codes (none of the above) (empty) - No CVV data (blank)
custom_id	50	Character	The same variable submitted in the request.
keyed	1	Boolean	True for keyed transaction.
swiped	1	Boolean	True for swiped transaction.
transaction_approved	1	Boolean	True for approved.
custom_data	65K	Character	The same variable submitted in the request.
transaction_description	65K	Character	The same variable submitted in the request.
ip_address	15	NNN.NNN.NNN.NNN	IP address of the client which initiated the transaction.
first_name	50	Character	The same variable submitted in the request. *May be returned with last_name concatenated
last_name	50	Character	The same variable submitted in the request. *May be returned concatenated in the first_name field.
request_amount	9	Numeric	Used only in partial auth. Indicates the amount of money that the merchant attempted to authorize the card for. *only supported in QSAPI 3.7 and higher
error	1	Boolean	True for error conditions or decline.
error_code	5	Numeric	0 for no error, > 0 for error number.

error_message	?	Character	DECLINED or textual description of other API errors (e.g., "Must send card_number" or "Invalid bank_account_number").
error_msg	?	Character	Deprecated (Legacy), same as error_message
account_balance_1	9	Numeric	See Balance Chart below for more information ONLY returned when transaction_type=BALANCE
account_balance_2	9	Numeric	See Balance Chart below for more information ONLY returned when transaction_type=BALANCE
account_balance_3	9	Numeric	See Balance Chart below for more information ONLY returned when transaction_type=BALANCE
entry_mode	9	Enumerated	Indicates the mode of entry of the transaction. Can be used instead of keyed and swipe variables with responses of: Keyed Swiped EMV Contactless Fallback Swiped NFC
NOTE: For Boolean responses, if the "response_format" variable was FORM, responses will return a 1 for True and NULL for False. Other response formats will return the native Boolean response of True or False.			

Account Balance Inquiry (Elavon Only)

In the QSAPI request, if you use the transaction_type with a value of BALANCE, then the response is based on the table below.

Variable Name	tender_type=CARD	tender_type=DEBIT	tender_type=EBT
account_balance_1	Pre-paid	Pre-paid	Food Stamp
account_balance_2	Gift Card	Gift Card	Cash Benefit
account_balance_3	Loyalty	Loyalty	N/A

Depending on the tender_type value (also in the QSAPI request), the variable that could be returned and the description of the balance is shown.

Example if QSAPI request has:
transaction_type=BALANCE
tender_type = DEBIT

If there is account_balance_1 returned, the value is the actual balance of the pre-paid portion remaining on the debit card used, according to the end processor (Elavon).

Transaction Status Interface

The Transaction Status is a unique service that Bluefin provides for merchants to allow their systems to reserve token IDs prior to a transaction so that our merchants can have an added level of control, assurance and reporting for their internal applications.

The Transaction Status Interface API (TSAPI) allows our merchants to request a token ID from the Bluefin Gateway prior to a transaction. Using the standard QSAPI process, a merchant would post a transaction and during the response, Bluefin would provide a unique transaction ID for the merchant to use to reference that transaction. With TSAPI, the merchant first obtains a transaction ID from TSAPI and then provides the transaction ID along with the transaction data to QSAPI. The previously obtained transaction ID is now the token of record for that transaction with Bluefin.

After the transaction is run, the status of the transaction can be queried. This way, if a timeout occurs, the merchant can retry the query against TSAPI to receive the transaction success or decline details.

The added layer of assurance comes into play because since the merchant is in effect acting as the “originating” entity for the transaction ID, the merchant can then know the applied token ID for that transaction regardless of any timeouts in responses that may occur due to Internet connectivity issues or other data transfer issues.

Orientation

This section describes how to obtain tokens to create a new transaction, and to check the status of transactions after they’ve been submitted.

When programmatically submitting transactions to a payment gateway, it is possible that an Internet interruption or network glitch occurs and either the transmission from your software or the response back to your software is missed due to a host of reasons to do with the Internet, your Internet Service Provider (ISP) or even a temporary problem with your software or server. While this is rare, it can occur in high volume situations.

In such a case, your software may submit a request to QSAPI and timeout waiting to receive a response. This leaves you with the option of retrying the transaction or not. If you retry the transaction and it was already processed, it can result in a duplicate transaction. If you do not retry the transaction, then it results in a lost sale, no authorization, or manual work to research and manage the exception. Thus, it is important to have a means to ensure that each request receives a response, and if not, to retry with confidence.

Did You Know?

It is not required to submit a token when you submit a transaction into QSAPI. When you submit a transaction to QSAPI, you receive back the `transaction_id` in the response payload which serves as the token for future tokenized transaction activities. You would pass this `transaction_id` through the `token_id` field where appropriate per the API.

Please note that PayConex uses the transaction ID as the token for the transaction, thus the terms transaction ID and token ID are interchangeable. We use the term “token” to reference the use of a transaction id for other purposes such as tokenization, reissuing, refunding, etc.

PayConex provides this transaction transmission assurance through the Transaction Status Application Programming Interface (TSAPI).

Prior to submitting a transaction, your software would connect to TSAPI to obtain a “transaction_id”. Your software would then pass this “transaction_id” in with the new transaction to QSAPI as a “transaction_id”.

If your system does not receive a response back, it has two options:

- 1) It can automatically retry the transaction and if the transaction already exists, you will receive a duplicate transaction ID error message back. This means the original transaction went through just fine and the transaction was not duplicated.
- 2) It can automatically query TSAPI as to the status of the “token_id”. If the transaction was successfully run, then the transaction results will be returned as with a normal transaction so your software can collect this data as normal and move on. If the transaction has never been received, then TSAPI will return with transaction ID not found

To make a production request to TSAPI, make a URL-encoded HTTP 1.1 POST using SSL 3.0 or TLS 1.1 or greater to <https://secure.payconex.net/api/tsapi/3.8/> through Port 443.

TSAPI Functions

The TSAPI API is based on a single post action and API response. However, to help illustrate the entire lifecycle of how the TSAPI API is used, we will be showing both the originating TSAPI API call that requests the token ID, and then the separate QSAPI call that utilizes the token ID returned in the initial TSAPI call.

Please visit the variables index below to review all of the posts that can be made through the TSAPI interface as well as the responses that the Bluefin payment gateway will make in response to those posts. The *Appendix: Code Requests and Responses* at the end of this document contain coding examples for TSAPI Requests and Responses.

TSAPI Request Format Variables Index

Below are all the variables that can be posted to TSAPI along with a brief description of their function.

Variable Name	Max	Type	Req'd	Description
account_id	12	Numeric	Yes	This is the Payconex account identification number that you are issued after your account has been setup.
api_accesskey	32	Alphanumeric	Yes	This is a secret key that you will be provided when your Payconex account is set up and when you have requested access to QSAPI.
action	n/a	Enumerated	Yes	GET_TRANSACTION_ID: Returns a transaction_id for use with a new SALE, AUTHORIZATION, CREDIT transaction to be passed through the transaction_id variable. GET_TRANSACTION_STATUS: Returns the status of a transaction.
transaction_id	12	Numeric	Yes/No	Required for GET_TRANSACTION_STATUS action.
response_format	5	Enumerated	No	Desired response format. FORM: www-form-urlencoded string (default format) JSON: JavaScript Object Notation JSONP: JSON w/ Padding DEBUG: human readable array output

TSAPI Response Format Variables Index

Below are all the variables that can be returned via TSAPI along with a brief description of their function.

Variable Name	Max	Type	Description
transaction_id	12	Numeric	The only variable returned from the GET_TRANSACTION_ID action.
found	1	Boolean	Returned from GET_TRANSACTION_STATUS action as True if transaction is found.
transaction_id	12	Numeric	Only returned from GET_TRANSACTION_STATUS if transaction is found. Same as the response from a QSAPI SALE transaction_type.
transaction_approved	1	Boolean	Only returned from GET_TRANSACTION_STATUS if transaction is found. Same as the response from a QSAPI SALE transaction_type.
transaction_timestamp	19	YYYY-MM-DD HH:MM:SS	Only returned from GET_TRANSACTION_STATUS if transaction is found. Same as the response from a QSAPI SALE transaction_type.
authorization_message	50	Alphanumeric	Only returned from GET_TRANSACTION_STATUS if transaction is found. Same as the response from a QSAPI SALE transaction_type.
transaction_amount	9	Numeric with decimal	Only returned from GET_TRANSACTION_STATUS if transaction is found. Same as the response from a QSAPI SALE transaction_type.
cashier	100	Alphanumeric	Only returned from GET_TRANSACTION_STATUS if transaction is found. Same as the response from a QSAPI SALE transaction_type.

Please note: For your security, transactions older than 18 months may be purged from our system.

Scheduling Layer Interface

This section delineates the requests and responses for the Scheduling Layer API (SLAPI). The scheduling layer enables a merchant to create and modify recurring schedules on behalf of a customer. The actions that can be performed with SLAPI are as follows:

- Create a recurring schedule
- Modify the schedule of an existing recurring payment
- Modify the amount of a recurring payment
- Cancel the schedule of an existing recurring payment

TIP

You can combine any of the following post actions to SLAPI into a single post. The actions are separated in the document to provide quick access to developers who are looking for quick references on performing specific actions.

Orientation

The SLAPI provides programmatic access to schedule and update recurring transactions as well as other automatic payment gateway tasks. To request the report in production, make a URL-encoded HTTP 1.1 POST using TLS 1.1 or greater to <https://secure.payconex.net/api/slapi/3.8/> through Port 443.

API Functions

The following pages detail the types of actions that you can perform via SLAPI. Each entry briefly lists the business functions you can perform, and code samples are located at the end of this document under *Appendix: Code Requests and Responses*. A list of all protocol references can be found in two indexes located in the last two pages of this document.

A. Create a recurring payment schedule

SLAPI allows you to create a recurring payment schedule. You can set the value of the charge, the schedule for when the charges should occur, and the number of times you would like the recurring transaction to occur.

SLAPI does not handle credit card numbers and bank account information. Instead, you first tokenize the payment details using QSAPI. You then use the token from QSAPI to setup a recurring payment schedule.

The key components of a recurring payment are:

- **Recurring schedule** - use the list of acceptable values from Appendix A
- **Recurring payment amount** - the dollar amount to be paid each period
- **Payments remaining** - limits the number of times the recurring payment will happen
- **Start date** - identifies when the initial recurring transaction runs

TIP

If you are making a change to a recurring schedule on the day of a transaction, please contact Bluefin support at support@bluefin.com to confirm when the recurring transactions are scheduled to run. Depending on when the change is made, the recurring payments for that day may have already been processed.

- **Reference Date** - date recurring schedule can be calculated from. If not specified start_date will be used.
- **Status** - information that tells the Bluefin scheduler whether to run the recurring transaction.

B. Modify the schedule of an existing recurring payment

You can modify the schedule of an existing recurring payment to change when the next transaction will be processed. As previously mentioned, scheduled payments can occur on a wide variety of options, but can be modified. You can't specify that a payment should be made on the 29th, 30th, or 31st of each month, since not all months have those days.

C. Modify the value of a recurring payment

You can modify the amount that a customer is charged on all their future recurring charges. Once you modify the amount of a transaction, you will not be able to review the historical amount associated with the recurring transaction. If you run reporting over a date range for the transaction charges you will be able to see the different amounts charged but there will be no way to retrieve/lookup an individual record via the API.

D. Disable the schedule of an existing recurring payment

You can modify a recurring transaction so that it will make no additional charges. You can change the status of a recurring payment at any time to DISABLED to stop future transactions. Conversely you can also change the status of a disabled recurring transaction to ENABLED to make it run again.

E. Cancel an existing recurring payment permanently

When you cancel a recurring payment schedule, it is basically removed from the system (any transaction made via that schedule are not removed). You cannot re-enable a canceled recurring schedule, so use with caution. It must be setup as a new recurring payment.

F. Restart a recurring record marked as on-hold

When a recurring transaction cannot be processed (usually for an invalid card number), it will get marked as "on-hold". This stops the system from continually trying to process it. Once the issue has been resolved (card number or expiration updated, etc.) the transaction can be "restarted".

TIP

You can cancel a recurring transaction using the CANCEL action. This will remove the recurring transaction from being displayed. You can change the status of the recurring transaction to DISABLED to stop future recurring transactions. The SLAPI backend will change the status of the recurring transaction to FINISHED when the posted number of remaining transactions reaches zero.

SLAPI Request Format Variables Index

The following table includes an index of all API request for SLAPI.

Variable Name	Max	Type	Req'd	Description
account_id	12	numeric	Yes	This is the Payconex account identification number that you are issued after your account has been set up.
api_accesskey	32	text	Yes	This is a secret key that you will be provided when your Payconex account is set up and when you've requested access to QSAPI.
action	11	enumerated	Yes	SETUP, EDIT, CANCEL, GET_DETAILS
status	8	enumerated	No	Either ENABLED or DISABLED Default for SETUP is ENABLED Optional: EDIT Not allowed: CANCEL, GET_DETAILS
token_id	12	numeric	Yes/No	This is the token ID from a QSAPI STOREd transaction. Required: SETUP Optional: EDIT Not allowed: GET_DETAILS, CANCEL
recurring_id	12	numeric	Yes/No	This is the recurring ID, from an existing recurring schedule. Required: EDIT, CANCEL, GET_DETAILS Not allowed: SETUP
recurring_payment_amount	9	numeric with decimal	Yes/No	Amount for this payment. Required: SETUP Optional: EDIT Not allowed: GET_DETAILS, CANCEL
recurring_payments_remaining	5	numeric	No	Number of recurring payments remaining. 0 means no scheduled payments remain. NULL (empty) means payment recurs forever. Optional: EDIT, SETUP Not allowed: GET_DETAILS, CANCEL
recurring_schedule		enumerated	Yes/No	See list of acceptable values in Appendix A. Required: SETUP Optional: EDIT Not allowed: GET_DETAILS, CANCEL
start_date	10	date (YYYY-MM-DD)	Yes/No	Date of first recurring payment. Required: SETUP Optional: EDIT Not allowed: GET_DETAILS, CANCEL

reference_date	10	date (YYYY-MM-DD)	Varies	Date recurring schedule is calculated from. If not specified start_date will be used.
label	50	text	No	If not sent, label will be generated automatically (first name + last name + unique ID, space permitting). Used for display in Payconex. Optional: EDIT Not allowed: SETUP, GET_DETAILS, CANCEL
response_format		enumerated	No	Desired response format. FORM: www-form-urlencoded (default) JSON: JavaScript Object Notation DEBUG: Human readable array output
description	65K	Character	No	A description of the payment. This is an open field. If emails are sent to the customer or merchant, this will show in the "Description:" field. You may use this to send any information that you wish.
restart_billing	1	boolean	No	Allows a restart of an "on hold" recurring transaction record. Value of 1 can be used in EDIT with recurring_id to restart schedule. Optional: EDIT Not allowed: SETUP, GET_DETAILS, CANCEL

SLAPI Response Format Variables Index

The following table includes an index of all API responses for SLAPI.

Variable Name	Max	Type	Description
recurring_id	12	numeric	Use this to EDIT or CANCEL a recurring entry.
status	8	enumerated	<p>ENABLED: Payment will recur as scheduled.</p> <p>DISABLED: Payment has been disabled and will not recur.</p> <p>CANCELED: Recurring payment has been stopped and will not recur.</p> <p>RETRYING: Payment failed, and will retry up to the number of times in your account's settings.</p> <p>FAILED: Payment failed, and will not be retried.</p> <p>FINISHED: No more recurring payments remain.</p>
token_id	12	numeric	This is the token from a QSAPI STOREd transaction.
recurring_payment_amount	9	numeric with decimal	Amount charged on next recurring payment date
recurring_payments_remaining	3	numeric	Number of recurring payments remaining. 0 means no scheduled payments remain. NULL (or empty) means payment will recur forever.
recurring_schedule		enumerated	See list of possible values in Appendix A.
recurring_schedule_description		text	Human-readable form of recurring schedule. E.g., "First Monday of every month" or "Every Friday"
start_date	10	date (YYYY-MM-DD)	Date of first recurring payment.
label	50	text	Used for display in Payconex. Originates from the QSAPI variable "custom_id"
next_recurring_payment_date	10	date (YYYY-MM-DD)	Date of the next recurring payment.
description	65K	Character	The same variable submitted in the request.

SLAPI Appendix A: List of Recurring Schedules

Recurring Schedule	Description	End Of Life	Uses Period Date?
PERIOD_1W	Once a Week		Yes
PERIOD_2W	Every 2 Weeks (Bi-Weekly)		Yes
PERIOD_1M	Once a Month		Yes
MONTHLY_1_15	1st and 15th of every month		No
MONTHLY_5_20	5th and 20th of every month		No
MONTHLY_LAST	Last day of every month		No
PERIOD_2M	Every 2 Months		Yes
PERIOD_3M	Every 3 Months		Yes
QUARTERLY_1	First day of every Quarter		No
QUARTERLY_LAST	Last day of every Quarter		No
PERIOD_6M	Every 6 Months		Yes
PERIOD_1Y	Once a Year		Yes
YEARLY_Q1_1	Annually, every quarter 1 on the 1st		No
YEARLY_Q2_1	Annually, every quarter 2 on the 1st		No
YEARLY_Q3_1	Annually, every quarter 3 on the 1st		No
YEARLY_Q4_1	Annually, every quarter 4 on the 1st		No
MONTHLY_1	1st of every month	Yes - use PERIOD_1M	No
MONTHLY_2	2nd of every month	Yes - use PERIOD_1M	No
MONTHLY_3	3rd of every month	Yes - use PERIOD_1M	No
MONTHLY_4	4th of every month	Yes - use PERIOD_1M	No
MONTHLY_5	5th of every month	Yes - use PERIOD_1M	No
MONTHLY_6	6th of every month	Yes - use PERIOD_1M	No
MONTHLY_7	7th of every month	Yes - use PERIOD_1M	No
MONTHLY_8	8th of every month	Yes - use PERIOD_1M	No
MONTHLY_9	9th of every month	Yes - use PERIOD_1M	No
MONTHLY_10	10th of every month	Yes - use PERIOD_1M	No
MONTHLY_11	11th of every month	Yes - use PERIOD_1M	No
MONTHLY_12	12th of every month	Yes - use PERIOD_1M	No
MONTHLY_13	13th of every month	Yes - use PERIOD_1M	No
MONTHLY_14	14th of every month	Yes - use PERIOD_1M	No
MONTHLY_15	15th of every month	Yes - use PERIOD_1M	No
MONTHLY_16	16th of every month	Yes - use PERIOD_1M	No
MONTHLY_17	17th of every month	Yes - use PERIOD_1M	No
MONTHLY_18	18th of every month	Yes - use PERIOD_1M	No
MONTHLY_19	19th of every month	Yes - use PERIOD_1M	No
MONTHLY_20	20th of every month	Yes - use PERIOD_1M	No
MONTHLY_21	21st of every month	Yes - use PERIOD_1M	No

MONTHLY_22	22nd of every month	Yes - use PERIOD_1M	No
MONTHLY_23	23rd of every month	Yes - use PERIOD_1M	No
MONTHLY_24	24th of every month	Yes - use PERIOD_1M	No
MONTHLY_25	25th of every month	Yes - use PERIOD_1M	No
MONTHLY_26	26th of every month	Yes - use PERIOD_1M	No
MONTHLY_27	27th of every month	Yes - use PERIOD_1M	No
MONTHLY_28	28th of every month	Yes - use PERIOD_1M	No

Start Date versus Reference Date

As mentioned earlier, the Start Date (start_date) is when the recurring transaction could start, or the date of the first recurring entry. The Reference Date (reference_date) is used on some recurring schedules to determine the actual day or date. Period Date is optional on those recurring schedules and if omitted, the Start Date will be used. Here are some examples to help determine how Reference Date can be used.

Schedule Example #1:

Today is January 5th, 2015 (2015-01-05)
recurring_schedule is set to PERIOD_1M, which is once a month.
start_date is set to today (2015-01-05)

If “reference_date” is not used, the recurring transaction will run every month, on the 5th, starting today.

If “reference_date” is entered as “2015-01-10”, then the recurring transaction will run every month, on the 10th, starting on the 10th (5 days from now)

Schedule Example #2:

Today is January 5th, 2015 (2015-01-05), a Monday
recurring_schedule is set to PERIOD_1W, which is once a week.
start_date is set to today (2015-01-05)

If “reference_date” is not used, the recurring transaction will run every week, every Monday, starting today.

If “reference_date” is entered as “2015-01-10” (a Saturday), then the recurring transaction will run every week, every Saturday, starting on the 10th (5 days from now)

Reporting Service

The Bluefin Reporting Service API (RSAPI) allows our merchants to securely export the transaction data for any given day. These reports can be run to verify merchant records of transactions.

Reporting via an API call is one of the many unique and differentiating features of Bluefin's service offerings. All data is passed securely from Bluefin through the API, and because no unique cardholder data is ever passed, the reports are PCI compliant.

Each report from Bluefin can be generated with comma delimited, JSON, or XML output. The reporting data contains no card numbers, PIN blocks, or card verification values, thus it is PCI compliant.

Orientation

The RSAPI provides access to transaction reporting data by accepting a formatted request and responding with a formatted file. To request the report in production, make a URL-encoded HTTP 1.1 POST using TLS 1.1 or greater to <https://secure.payconex.net/api/rsapi/3.8/> through Port 443.

RSAPI responds with header information in order to provide information to your program regarding the response format and MIME/Type.

For example, for CSV files, the HTTP Header will include:

Content-type: text/csv

A custom response code is also provided within the HTTP header:

QS-response-code: NNN Message

For error conditions (Response Codes other than 100), a human-readable error message will be returned as the body of the response:

QS-response-code: 601 Authentication failed

Account number and/or API access key is missing or invalid.

Special Note on ACH reporting:

It is important to note that ACH operates under a different business model than credit card payments. ACH transactions utilize a different national infrastructure than credit cards. While credit cards can give you near real time statuses on actions, ACH transactions are done in daily batches, and the financial institutions can often take days to respond with a status to an ACH transaction.

If you are running an RSAPI report to capture your previous day's transactions, ACH transactions will show "BATCHED" in the "transaction_date" records. This indicates that the ACH transaction was successfully sent at the end of the business day to the processor.

Since the reply time for an ACH transaction can vary so greatly a merchant should run a separate daily RSAPI report to see what updates were sent back on the ACH transactions that had been previously sent days ago. You will want to make sure to target "action_date" in your RSAPI report, and you will be able to see any new ACH responses that were received on the previous day.

RSAPI Functions

RSAPI responds to report requests by generating a report of all transactions for a given day, or a range of dates. RSAPI contains no unique cardholder data and as such ensures that our host organizations stay in PCI compliance. Note that because some data is reconciled with financial institutions nightly, the earliest a report can be run is from the previous business day. If you plan on programmatically running automated reports for multiple single dates, the suggested time to run these reports is between 4:00 AM – 6:00 AM eastern. This time range should give you optimal server response time and should allow enough time for the previous days records to reconcile from any west coast entities.

RSAPI Request Format Variables Index

The following table includes an index of all API request posts for RSAPI.

Variable Name	Max	Type	Req'd	Description
account_id	12	Numeric	Yes	This is the Payconex account identification number that you are issued after your account has been set up.
api_accesskey	32	Alphanumeric	Yes	This is a secret key that you will be provided when your Payconex account is set up and when you've requested access to QSAPI.
transaction_date *	19	YYYY-MM-DD HH:MM:SS	No	Download transactions ran for this date (default: yesterday).
action_date *	10	YYYY-MM-DD	No	Download transactions with this action date (for ACH only; default: yesterday). This is useful for viewing ACH updated transactions on a specific date.
tender_type	4	Enumerated	No	Payment type to include in the report: ALL, CARD, ACH, EBT, or GIFT (default: ALL).
response_format	4	Enumerated	No	How the resulting data is returned. Valid options are: CSV : Comma-separated value formatted file (default) JSON : JavaScript Object Notation formatted file. XML : Extended markup language formatted file.
group *	25	Alphanumeric	No	Groups are flexible groups that can be used for various reasons, including: a) assign transactions to a specific group. b) direct transactions to separate back-end or depository accounts.
start_date *	10	YYYY-MM-DD	No	Used for getting transactions in a date range. Use with end_date.
end_date *	10	YYYY-MM-DD	No	Used for getting transactions in a date range. Use with start_date.
transaction_id	12	Numeric	No	The "front-end" transaction id for a transaction.
cashier *	100	Alphanumeric	No	The cashier that created the original transaction.
amount	10	Decimal	No	The amount of the original transaction. Must be zero or positive decimal numbers.

amount_min	10	Decimal	No	The minimum amount of a transaction amount range. Use with amount_max. Must be zero or positive decimal numbers.
amount_max	10	Decimal	No	The maximum amount of a transaction amount range. Use with amount_min. Must be zero or positive decimal numbers.
custom_id *	50	Alphanumeric	No	Any custom text value that may have been used.
name	100	Alphanumeric	No	The name used for the customer on the original transaction.
status	8	Enumerated	No	Value can be either "APPROVED" or "DECLINED" which will return matching transactions. If omitted, BOTH types will be returned.
batch_detail	1	Boolean	No	If sent as "1", batch number and batch date/time will be returned.
reportable_fields	1	Boolean	No	If sent as "1", any reportable field values from HPF (Hosted Payment Forms) will be added to the end of the RSAPI response, with column names matching account setup.

Note: In the RSAPI requests, wildcard searches are possible for the “transaction_date”, “action_date”, “group”, “cashier”, “custom_id”, and “name” variables. By default, all parameters are treated as exact matches. A question mark (?) is used for a SINGLE missing character, and an asterisk () specifies zero or more unknown characters.

Timestamp searching is also possible for the “transaction_date” variable. When using an asterisk (*), it is used for any value, similar to these:

transaction_date=2018-08-??

transaction_date=2018-08-22 14:*:*

If using a timestamp, you MUST include hours, minutes, and seconds or use a wildcard for their values.

Using, or including, the “start_date” and “end_date” variables will override a “transaction_date” value, even if “start_date” is blank. The “start_date” and “end_date” variable combination should be used for getting data from a date range, and “transaction_date” should be used for getting data for a particular date.

If you specify a “start_date”, but include no “end_date” variable or value, it is assumed that you want all transactions from the “start_date” to today (now).

If you specify an “end_date”, but include no value for “start_date”, it is assumed that you want all transactions from your account start date up to the “end_date”.

If “batch_detail” is set to “0” or omitted completely, no batch ID or batch date/time will be returned. Additional batch data is only returned if “batch_detail” is set to “1”.

RSAPI Response Format Variables Index

The following table includes an index of all API responses for RSAPI.

Variable	Max	Type	Description
transaction_id	12	Numeric	The transaction id for the new transaction. When using tokenization, this is the transaction_id that you submit as the token_id.
account_id	12	Numeric	The account ID for the transactions
authorization_date	19	YYYY-MM-DD HH:MM:SS	The date and time when the transaction was initiated (YYYY-MM-DD HH:MM:SS)
tender_type	4	Enumerated	The method of transaction that was made: CARD, ACH, EFT, EBT, USDA, FNS, GIFT.
transaction_type	14	Enumerated	Type of transaction: AUTHORIZATION, SALE, REFUND, CREDIT, CAPTURE, SETTLE-BATCH, STORE, FORCE.
keyed	1	Enumerated	1 indicates key entry of card information, 0 indicates not a keyed entry.
swiped	1	Enumerated	1 indicates swiped entry of card information, 0 indicates not a swiped entry.
transaction_amount	9	Numeric w/ decimal	Amount of funds involved in the transaction.
name	100	Character	Customer name
card_brand	n/a	Enumerated	Customers brand of card used. VISA, MASTERCARD, AMERICAN EXPRESS, DISCOVER, ACH, EBT.
last4	4	Numeric	The last four digits of the card number or Primary Account Number (PAN). For ACH, it is the last four digits of the account number.
card_expiration	4	Numeric	The month and year of the card expiration date in the format MMY. For example, 0115 for January 2015.
description	65K	Character	The description entered in the transaction
user_data	65K	Character	User data entered in the transaction. Originates from the QSAPI variable "custom_data".
authorization_msg	50	Alphanumeric	APPROVED or the auth message from the processor (e.g. AUTH DECLINED 200).
authorization_code	6	Alphanumeric	The authorization code returned by the processor.

avs_response	1	Enumerated	A single letter address verification response: DFJMQVXY Address and ZIP code match LWZ ZIP code match, address is wrong ABOP Address match, ZIP code is wrong KN No match, address and ZIP is wrong U No data from issuer/banknet switch R AVS System unable to process S Issuing bank does not support AVS E Error, AVS not supported for your business C Invalid address and ZIP format (International) I Address not verifiable (International) G Global non-verifiable address (International) ? Unrecognized codes (none of the above) _ No AVS data (blank)
cvv2_response	1	Enumerated	Single letter card verification value response: M CVV match N CVV does not match P CVV not processed S Card has CVV, customer says it doesn't U No CVV data from issuer ? Unrecognized codes (none of the above) _ No CVV data (blank)
ip_address	15	NNN.NNN.NNN.NNN	IP address of the client which initiated the transaction.
cashier	100	Character	The cashier value from the transaction
street_address1	50	Character	Customer's street address
city	100	Character	Customer's city
state	2	Alphanumeric	Customer's state
zip	10	Numeric with hyphen	Customer's zip
country	3	Alphanumeric	Customer's country
phone	20	Alphanumeric	Customer's phone
email	100	Character	Customer's email
group	12	Alphanumeric	Group value from the transaction
refund_id	12 *	Numeric	Refund ID from the transaction (length may be longer if multiple refunds were ran, all refund transaction id's separated by a pipe indicator)
refund_balance	9	Decimal	Refund balance (if applicable) from the transaction
custom_id	50	Character	Custom ID value from the transaction.

action_date	10	Date (YYYY-MM-DD)	The action date indicates that an ACH update has been applied to a previous ACH transaction. A status response to an ACH settlement takes days to receive. By using action_date and the date you wish to check you can see what ACH transactions have been updated on a specific day. Responses are: DECLINED\ERRORS DECLINED\RETURNED SETTLED\FUNDED
noc_data	50	Character	The NOC data from the transaction
recurring_id	12	Numeric	This unique identifier is provided in addition to the transaction ID to allow for identifying and segmenting recurring transactions. This variable is in version 3.6.1 and higher.
input_group	10	Character	Input group from the transaction
invoice_entry	65k	Character	Concatenated list of invoice line description and amounts, each separated by &, when invoice feature is used for transactions.
trace_num	12	Integer	back-end transaction number that might be used by some end processors for reporting or communication.
company	50	String	Company name field from the transaction
entry_mode	17	Enumerated	The value of the entry mode from the transaction: KEYED SWIPED EMV CONTACTLESS FALLBACK SWIPED
batch_id	4	Numeric	The batch number (ID) of the transaction. May be empty if not included in a batch yet. ONLY returned if batch_detail=1 was submitted.
batch_date	16	Character	The batch date and time of the transaction. May be empty if not included in a batch yet. ONLY returned if batch_detail=1 was submitted.
Reportable Fields **	?	Character	If "reportable_fields" was set to "1" and there are reportable fields on the HPF (Hosted Payment Form), those columns will be included at the end of the response.
ach_return_code	4	alphanumeric	The ACH Return Code for ACH transactions. Will only be included for ACH transactions that have cleared and batched (could be up to 5 days depending on the banks).
time_zone	3	Character	This is a timezone indicator for the RSAPI fields that contain actual times. Would be similar to EST, CST, etc (batch_date will ALWAYS be in CST)

Please note: For your security, transactions older than 18 months may be purged from our system.

RSAPI HTTP Response Format Variables Index

In addition to the RSAPI responses shown above, in the HTTP header of the response, you will find an additional piece of data that you may wish to capture, called “QS-response-code”. An example of an HTTP header for a RSAPI request is similar to the following:

```
HTTP/1.1 200 OK
Date: Tue, 03 Aug 2013 12:23:05 GMT
Server: Apache
QS-response-code: 601 Authentication failed
Connection: close
Vary: Accept-Encoding,User-Agent
Content-Encoding: gzip
X-Frame-Options: SAMEORIGIN
Cache-Control: private, no-cache, no-store, proxy-revalidate, no-transform
Pragma: no-cache
Content-Length: 63
Content-Type: text/html; charset=UTF-8
```

Values for the “QS-response-code” variable (Code and Message concatenated) are as follows:

Response Code	Response Message	Description
100	OK	Success. Report file follows.
601	Authentication Failed	Account number and/or API access key is incorrect.
607	Invalid Date	Date format is incorrect (YYYY-MM-DD) or invalid date (2009-02-31).
608	No Data	No data or transactions matched the search criteria.
611	Data Unavailable	A temporary system error has prevented access to the data requested. Please try again later or contact administrator.
621	Invalid Parameter	One or more parameters do not match the RSAPI Request Format specification.
622	Missing Parameter	One or more required parameters are missing.
623	Conflicting Parameters	Combination of parameters is invalid (e.g., specifying action_date for card transactions).

If “QS-response-code” is NOT equal to “100 OK”, then you will receive the Description from above in the actual response from your request.

Appendix: Using HASH for Authenticated Transactions

A hash function is an algorithm that transforms (hashes) an arbitrary set of data elements, such as a text string or file, into a single fixed length value (the hash). The computed hash value is a means of protecting sensitive data. Bluefin has included this functionality in to its API's, and it uses a Secure Hash Algorithm (SHA-256) type of hash.

There are two pieces that are tied to sending a hash; the hash value and the "hash_key" value (string), although the "hash_key" is not always required. A description of each REQUIRED parameter in any HASH is included below.

Parameter	Req'd	Description
account_id	Yes	This is the merchant account ID. For any hash, this MUST be included and be the first parameter.
api_accesskey	Yes	This is the merchants unique "key". It should NEVER be given out to anyone. For any hash, this MUST be included and be the second parameter. If it is ever sent in the hash, and outside the hash (it is sent twice), you will receive a "Security Violation" error.
timestamp	Yes	This is a 10-digit UNIX timestamp representing the number of seconds since Epoch (00:00:00 on January 1, 1970) and should represent the time this transaction occurs. For any hash, this MUST be included and be the third parameter.

It is possible in any transaction type to use the HASH method for more than just the required fields (listed above). This is strictly up to the vendor/merchant, but it is recommended by Bluefin to do so. If the merchant wishes to include other data values in the hash, they can be appended to the hash value, but another element must be sent, called the "hash_key", which is explained below.

Parameter	Req'd	Description
hash_key	No	A string containing a comma delimited list of parameters used to build the hash. This list should NOT include the three parameters above, or other parameters that are used in transparent redirect (see the Transparent Redirect section for more information).

NOTE: All HASH parameters are case sensitive.

NOTE: When using the transparent redirect method, the HASH method is REQUIRED.

Example #1:

If the merchant wishes to send the minimum values of a hash, given the following values:

account_id :: 123456789012

api_accesskey :: e6f157d2-66cf-43d5-8a56-c4c57d5760d7

timestamp :: 1360870400

This would be the hash string:

123456789012,e6f157d2-66cf-43d5-8a56-c4c57d5760d7,1360870400

With a resulting hash value that is sent as something like:
b48171ba3c4ffbc1345093087d661d52a109d836462455d208f52bf7392cbf95

Example #2:

Given the same minimum values, and a transaction amount of \$123.00, the hash string would look like:
123456789012,e6f157d2-66cf-43d5-8a56-c4c57d5760d7,1360870400,123.00

And the resulting hash value that is sent is something like:
c602825bed7fdc9b256ec6ce074b88e6befc18bd0eb295a9acb7af024708aedef

Note that in the above example #2 that the merchant would also have to send the following, which would indicate the optional "hash_key" parameter is included in the hash:
hash_key = "transaction_amount"

Variables Index

When using a hash, the table below shows the variables that would be sent to QSAPI, along with the request format variables for the given transaction type.

Parameter	Req'd	Description
account_id	Yes	This is the merchant account ID.
timestamp	Yes	This is a 10-digit UNIX timestamp representing the number of seconds since Epoch (00:00:00 on January 1, 1970) and should represent the time this transaction occurs.
hash	Yes	The hash value (NOT the hash string)
hash_key	No	Only required if additional (optional) parameters were included in the hash string
NOTE: The parameter "api_accesskey" is NOT to be sent as a separate variable when using transparent redirect. Doing so compromises the integrity of the data, PCI compliancy, and will result in a "security violation" error.		

NOTE: All HASH parameters are case sensitive.

NOTE: The HASH function is new in version 3.8 of the API. Prior versions do not contain this feature.

Appendix: Transparent Redirect

Overview

Transparent redirect is built on the principle of keeping sensitive PCI data off of a merchant’s web server. Payment forms utilizing transparent redirect submit their data directly to Bluefin from the customer’s browser so that the sensitive PCI data never touches the merchant’s servers. Bluefin captures the transaction details, and then relays the response back to the merchant’s website where it can be recorded, and gives control of the post-transaction user experience back to the merchant.

In addition to transparent redirect, Bluefin provides a support function that allows the merchant to receive a post back to their servers with the transaction details. This additional feature ensures that the merchant gets a copy of the transaction response details. The post back happens independently from the response sent through the customer’s browser.

Configuration

This version of the API is built around allowing the merchant to have the greatest level of control over the transparent redirect process by passing the URLs in each transaction. It also allows the merchant the option to selectively not send URLs if they don’t wish to. So transparent redirect can be used, or not used, at any time, with any transaction.

Transparent redirect **REQUIRES** the use of a hash, and the parameters required to create the hash are indicated below:

Parameter	Req'd	Description
account_id	Yes	This is the merchant account ID. This MUST be the first parameter.
api_accesskey	Yes	This is the merchant’s unique “key”. It should NEVER be given out to anyone. This MUST be the second parameter. If it is ever sent in the hash, and outside the hash (it is sent twice), you will receive a “Security Violation” error.
timestamp	Yes	This is a 10-digit UNIX timestamp representing the number of seconds since Epoch (00:00:00 on January 1, 1970) and should represent the time this transaction occurs. For any hash, this MUST be included and be the third parameter.
success_url	Yes	A string containing a valid URL that the merchant wishes to be used for successful transactions. * It needs to be the 4th parameter in the hash.
decline_url	No	A string containing a valid URL that the merchant wishes to be used for declined transactions. * If used, it needs to be the 5th parameter in the hash.

Example #1:

If the merchant wishes to use and send transparent redirect, given the following values:

```
account_id :: 123456789012
api_accesskey :: e6f157d2-66cf-43d5-8a56-c4c57d5760d7
timestamp :: 1360870400
success_url:: https://www.example.com/success.aspx
decline_url:: https://www.example.com/decline.aspx
```

This would be the hash string:

```
123456789012,e6f157d2-66cf-43d5-8a56-
c4c57d5760d7,1360870400,https://www.example.com/success.aspx,
https://www.example.com/decline.aspx
```

With a resulting hash value that is sent is something like:

```
b6814a1818a0f7b3fdd0e58cd601be17be2dc5495be81279704079b6079a1ecc
```

Example #2:

Given the same minimum values, and a transaction amount of \$123.00, the hash string would look like:

```
123456789012,e6f157d2-66cf-43d5-8a56-
c4c57d5760d7,1360870400,https://www.example.com/success.aspx,
https://www.example.com/decline.aspx,123.00
```

And the resulting hash value that is sent is something like:

```
3e19dacbc92fa9b1a88ce8d57c7493f374a44de35ea0b4405b7f111fb20d26d9
```

Note that in the above example #2 that the merchant would also have to send the following, which would indicate the optional “hash_key” parameter is included in the hash:

```
hash_key = “transaction_amount”
```

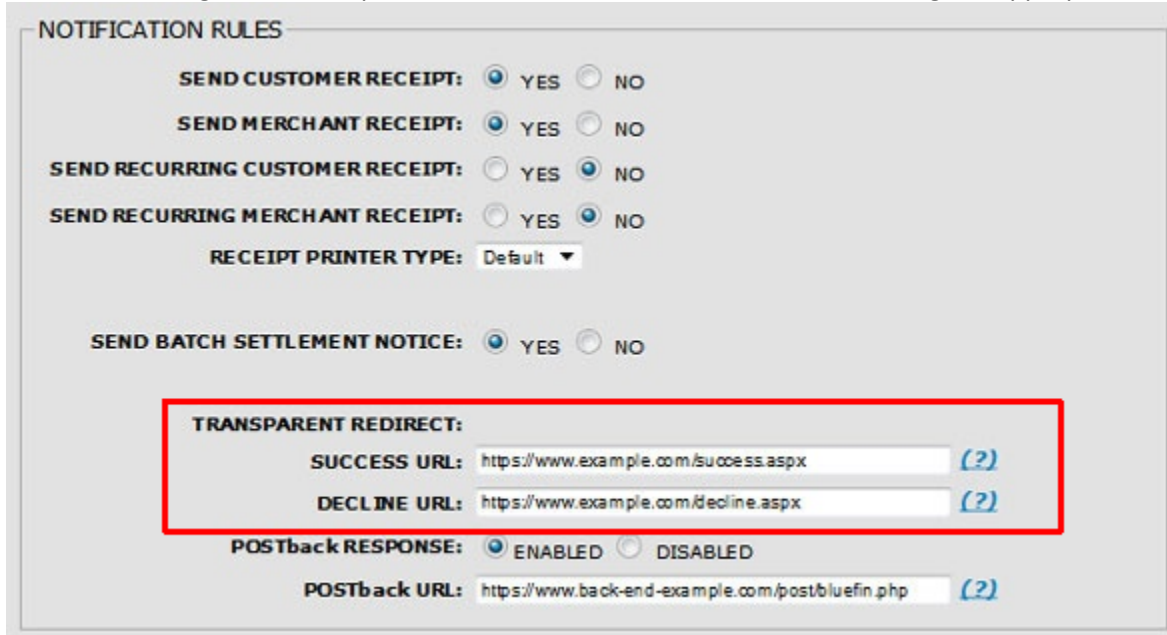
When QSAPI receives a request it will:

- Detect the hash parameter in the POST.
- Validate timestamp has not expired (currently 30 minute life).
- Look for “success_url” in the POST data.
- Look for “declines_url” in the POST data.
- If found, and the URLs validate, the response data will be sent back to the client with an **HTTP/1.1 303** response header, causing the client’s browser to redirect to the success or decline URL, as applicable.

Configuration (PayConex)

Transparent Redirect can also be configured inside of PayConex (Settings – Manage Settings) if desired, but it isn't as flexible as the above configuration method. Using this method allows the Merchant to enter URLs for success and decline, at a global level. This allows the Merchant to leave the URLs out of each transaction because the global setting applies to all transactions.

See the following for an example of the “Notification Rules” section containing the appropriate values:



NOTIFICATION RULES

SEND CUSTOMER RECEIPT: YES NO

SEND MERCHANT RECEIPT: YES NO

SEND RECURRING CUSTOMER RECEIPT: YES NO

SEND RECURRING MERCHANT RECEIPT: YES NO

RECEIPT PRINTER TYPE: Default ▾

SEND BATCH SETTLEMENT NOTICE: YES NO

TRANSPARENT REDIRECT:

SUCCESS URL: (?)

DECLINE URL: (?)

POSTback RESPONSE: ENABLED DISABLED

POSTback URL: (?)

NOTE: If you have URLs entered in the PayConex configuration screens for “global” success and decline, sending URLs with a transaction will override the configuration settings. It is best if the URLs for transparent redirect are sent in each transaction request, as an errant or unauthorized change to the globally configured URLs could have a negative impact on your transaction responses.

Variables Index

When using transparent redirect, the table below shows the variables that would be sent to QSAPI, along with the request format variables for the given transaction type.

Parameter	Req'd	Description
account_id	Yes	This is the merchant account ID.
timestamp	Yes	This is a 10-digit UNIX timestamp representing the number of seconds since Epoch (00:00:00 on January 1, 1970) and should represent the time this transaction occurs.
success_url	Yes	A string containing a valid URL that the merchant wishes to be used for successful transactions.
decline_url	No	A string containing a valid URL that the merchant wishes to be used for declined transactions.
hash	Yes	The hash value (NOT the hash string)
hash_key	No	Only required if additional (optional) parameters were included in the hash string
NOTE: The parameter "api_accesskey" is NOT to be sent as a separate variable when using transparent redirect. Doing so compromises the integrity of the data, PCI compliancy, and will result in a "security violation" error.		

Appendix: AJAX / CORS Support

Overview

The PayConex API is designed to support asynchronous methodologies, commonly referred to as AJAX. Similar to Transparent Redirect, using AJAX will ensure that the cardholder data is never transmitted back to your web servers or network, and thus may reduce the scope of your PCI compliance requirements by removing transmitted cardholder data from your environment.

In order to support AJAX requests, Bluefin's API has been extended to support CORS (Cross-Origin Resource Sharing). This function allows pages rendered by your server to interact with the API that is hosted by Bluefin's servers via JavaScript.

Configuration

CORS support is enabled by default on all POST requests that do not result in a redirect (see Appendix: Transparent Redirect). To ensure that your POST results in a formatted response and not a redirect, and to ensure the response is returned in a format that can be parsed by JavaScript, set the following parameters in your HTTPS POST:

Variable Name	Max	Type	Req'd	Description
disable_redirect	1	Boolean	Yes	Disabling redirect will override the success_url and decline_url settings, and force the return of transaction response in the format specified by response_format. This setting is required for performing API calls via AJAX.
response_format	5	Enumerated	Yes	In order to interpret the response using JavaScript Object Notation, you must set the response to the value of: JSON

Browser Support

Bluefin does not provide a client-side library, as there are numerous client-side libraries that facilitate the passage of POST data to a form handler, and parsing of the response. By far, the most popular such library is jQuery. Bluefin's AJAX support has been tested to work with modern browsers that support CORS. These browsers include: IE 10+, Firefox 3.5+, Chrome 4+, Safari 4+, iOS Safari 3.2+, Opera 12.1+, Android 2.1+, Chrome for Android 42+, Firefox for Android 37+, IE Mobile 10+, Opera Mobile 12+, and UC Browser for Android 9.9+

Sample Code (PHP and jQuery):

```
<?php
// Calculate the hash before rendering page.
// Hashed authentication is required for AJAX requests.
// API access key should never be visible in source code.
// See Appendix: Using HASH for Authenticated Transactions
$account_id = "220000000000";
$api_accesskey = "87f519105543418daea2cb6cb9945c7f";
$timestamp = time();
$hash_string = $account_id." ".$api_accesskey." ".$timestamp;
$hash_value = hash("sha256",$hash_string);
?>
<!-- jQuery -->
<!-- Source: http://jquery.com/download/ -->
<script type="text/javascript" src="jquery-1.11.1.min.js"></script>
<script type="text/javascript">
$( function(){
// Listen for the form to be submitted
$('#cc_form').on( 'submit', function( e ){
// Preventing default should stop browser from submitting the form.
e.preventDefault();
// Collect all the data that is going off to Bluefin
var request_data = {
account_id:    "<?php echo $account_id; ?>",
timestamp:    "<?php echo $timestamp; ?>",
hash:         "<?php echo $hash_value; ?>",
tender_type:  "CARD",
transaction_type: "STORE",
transaction_amount: 0.00,
response_format: "JSON",
disable_redirect: 1,
card_number:   $('#cc_num').val(),
card_expiration: $('#expiration').val(),
card_verification: $('#cvv').val(),
zip:          $('#zip').val()
}
// Create an AJAX HTTPS POST request to Bluefin
$.ajax({
url: 'https://cert-tls12.payconex.net/api/qsapi/3.8/',
data: request_data,
type: 'POST',
dataType: 'json',
success: function( response_data ){
// If successful, parse and act based on API response
for ( key in response_data )
$('#result').append( '<li>' + key + ': ' + response_data[key] + '</li>' );
},
},
```

```
error: function( error ){  
  // Log any error.  
  console.log( "ERROR:", "Request did not work." );  
  }  
  });  
  // Return false to once again tell the  
  // browser not to submit the form.  
  return false;  
  });  
  });  
</script>
```

Appendix: POSTback

Orientation

POSTback is similar in function to Transparent Redirect, but instead of routing response data back through the web clients browser, it sends the data directly to the merchant's back-end system, securely, via HTTPS POST calls. POSTback can be done in addition to transparent redirect, or independently of transparent redirect.

Implementation steps:

You first need to log into your account and modify the Merchant Account Setup/Configuration.

From the PayConex account settings (Settings – Manage Settings) page, you can view and modify the following (in the Notification Rules section):

Enable/Disable POSTback radio buttons. Click Enable to reveal the URL field
POSTback URL – enter a valid HTTPS URL to receive POSTback data.

NOTIFICATION RULES

SEND CUSTOMER RECEIPT: YES NO

SEND MERCHANT RECEIPT: YES NO

SEND RECURRING CUSTOMER RECEIPT: YES NO

SEND RECURRING MERCHANT RECEIPT: YES NO

RECEIPT PRINTER TYPE: Default ▾

SEND BATCH SETTLEMENT NOTICE: YES NO

TRANSPARENT REDIRECT:

SUCCESS URL: (?)

DECLINE URL: (?)

POSTback RESPONSE: ENABLED DISABLED

POSTback URL: (?)

No change to the QSAPI implementation code is required.

Custom Process Flow

QSAPI requests are handled normally from the receipt of a request through the normal submission process.

When the QSAPI reply is ready, if “**POSTback RESPONSE**” is enabled in the merchant’s account, and a valid “**POSTback URL**” has been entered, QSAPI will do the following:

- Store the transaction Postback data into a Messaging Queue.
- Trigger an asynchronous posting of the transaction to the merchant’s Postback URL.
- Include a validation hash to send with the Postback data, comprised of the following:
 - account_id
 - api_accesskey
 - timestamp

If a connection cannot be made to the merchant’s “*Postback URL*”, the Postback messaging sub-system will continue to retry the Postback periodically (interval TBD).

Upon successful connection and transmission, the messaging sub-system will record the response from the merchant’s system and tag the Postback as complete.

Because the Postback actually happens asynchronously, there is no delay in the response to the customer/client.

POSTback Response Payload

The POSTback response payload is a structured data object. It is comprised of five (5) main components:

- **account_id** The merchant Payconex account ID
- **timestamp** The UNIX timestamp of the POSTback
- **count** The number of transaction responses being sent
- **hash** The authentication hash (see above for description)
- **responses** An array of transaction responses. For the typical merchant this array will contain only one transaction. Merchants utilizing Enhanced Payment Pages, or performing Split Transactions via QSAPI, would receive multiple responses bundled together.

The data format of the POSTback response can be controlled by passing a “*response_format*” parameter in the original transaction request. The following payload formats are supported:

- **FORM** HTTP query string key=value pairs (default)
- **JSON** (recommended)
- **XML** (pending)

POSTback responses will contain the same variables as the QSAPI response, just a different data format; however, it will also contain the account_id, timestamp, count, and a hash in case you need it. The hash is included so you can verify that the data sent back actually came from Bluefin. See example below.

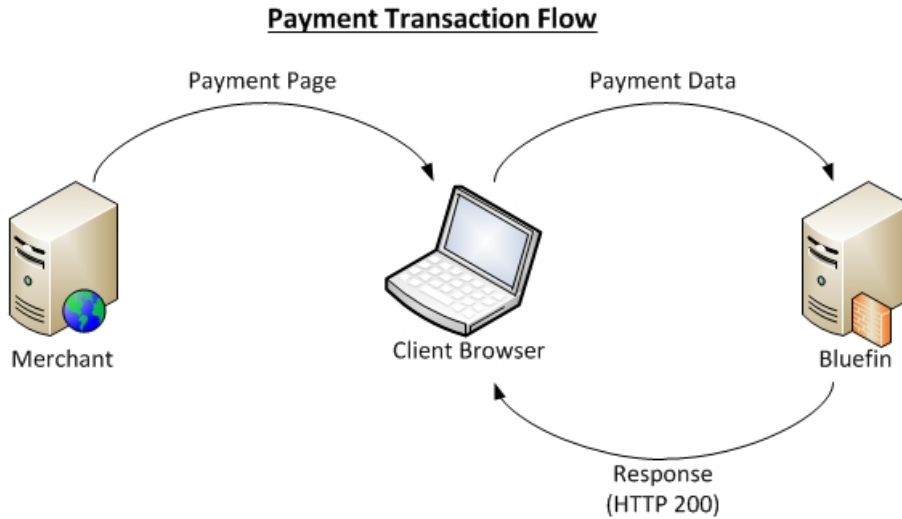
POSTback Example Response

A typical POSTback payload could look like this (JSON format used):

```
{
  "account_id":"120908675309",
  "timestamp":1374346390,
  "count":1,
  "hash":"d3a0b8cddacab6b761fd61d9176013e9287384848b052362b6b4bc090a257c6a",
  "responses":[
    {
      "transaction_id":"000282870523",
      "tender_type":"CARD",
      "transaction_timestamp":"2018-07-20 13:53:09",
      "card_brand":"VISA",
      "transaction_type":"SALE",
      "last4":"4321",
      "card_expiration":"1218",
      "authorization_code":"096932",
      "authorization_message":"APPROVED",
      "request_amount":"345.98",
      "transaction_amount":"345.98",
      "first_name":"Robert",
      "last_name":"Smith",
      "keyed":"1",
      "swiped":"",
      "transaction_approved":"1",
      "cvv2_response":"N",
      "error":"",
      "error_code":"0",
      "error_message":"",
      "error_msg":"",
      "customer_feedback":"",
      "description":"Widgets: P/N BA-0523-C"
      "custom_id":"Customer 1234567890"
      "custom_data":"Widget BA-0523-C – Customer 1234567890"
    }
  ]
}
```

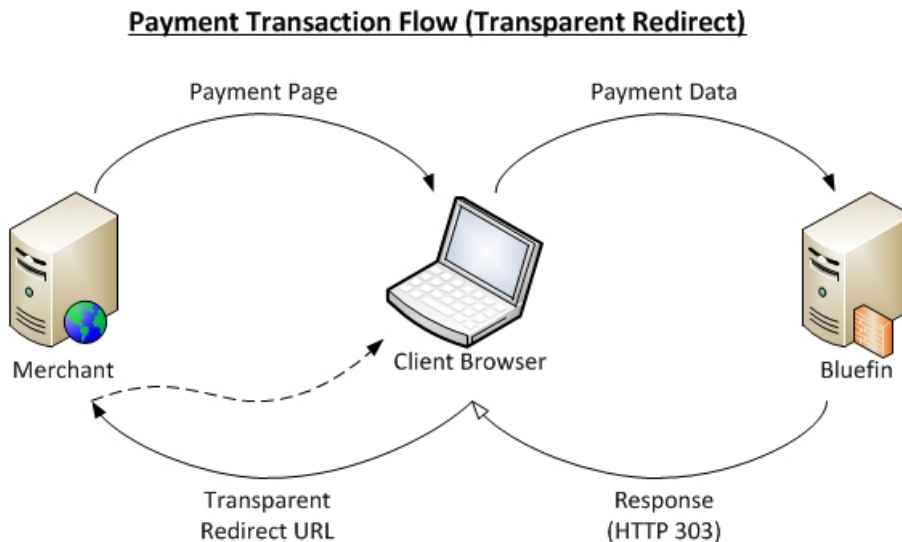
Appendix: Transaction Flow Diagrams

The images that follow are various ways that transactions can flow from a Merchant to the client, to Bluefin, and back, depending on the configuration.



Notes:

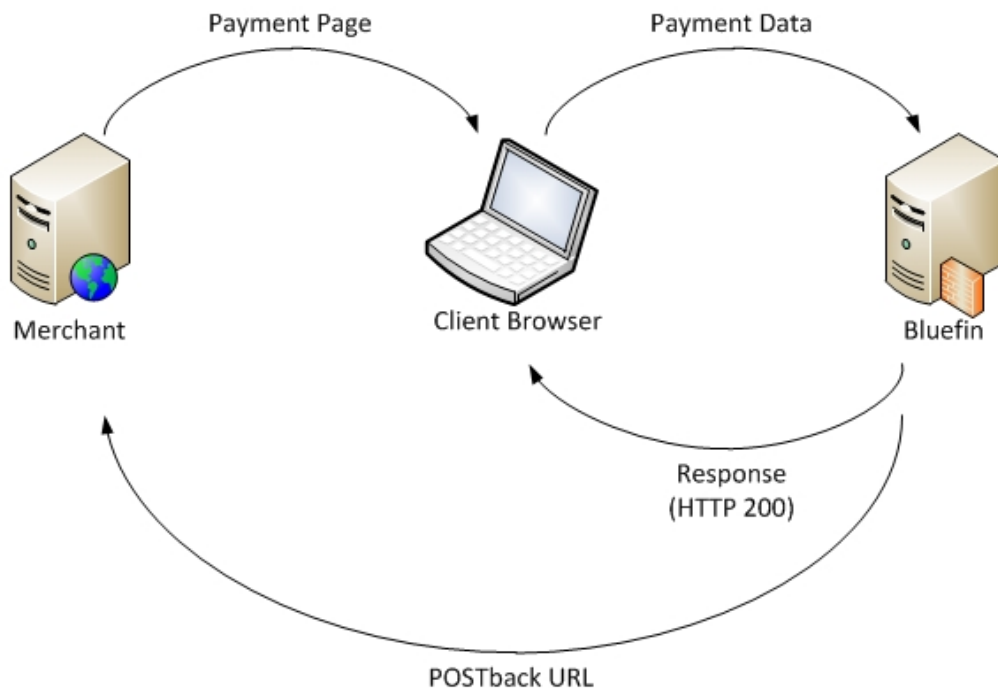
1. Client requests Merchant payment page, which is sent to the client browser from the Merchants website.
2. Client enters payment data which is sent to Bluefin for processing.
3. Bluefin replies (with HTTP 200) with a success/fail response directly to the client browser.



Notes:

1. Client requests Merchant payment page, which is sent to the client browser from the Merchants website.
2. Client enters payment data which is sent to Bluefin for processing.
3. Bluefin replies (with HTTP 303) with a success/fail response to the client browser, which is redirected to the specified transparent redirect URL, which actually sends a merchant specified webpage to the client browser.

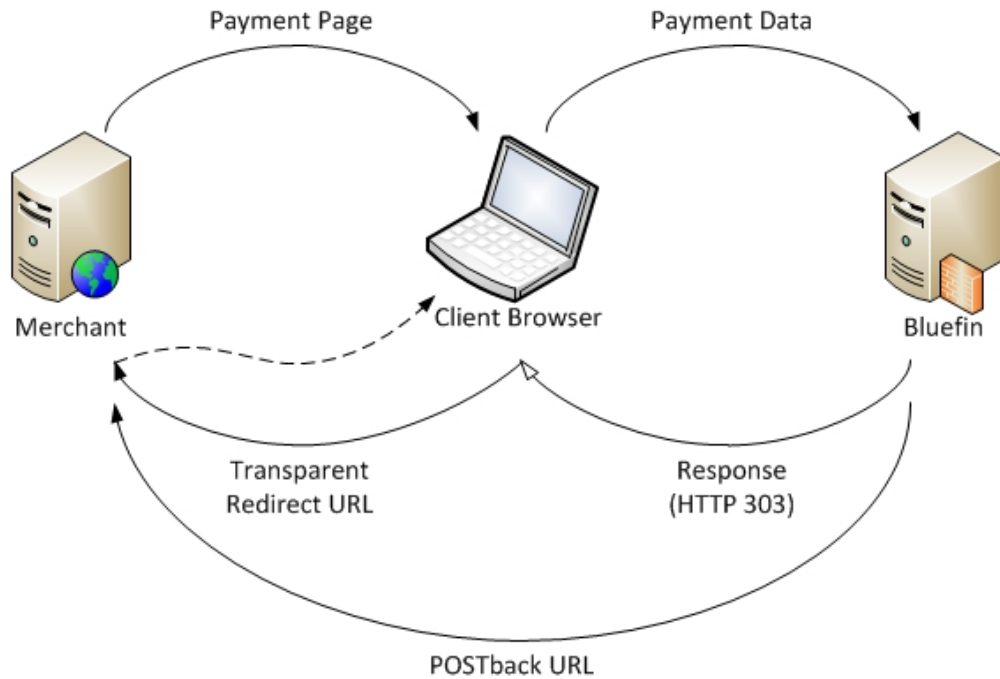
Payment Transaction Flow (POSTback)



Notes:

1. Client requests Merchant payment page, which is sent to the client browser from the Merchants website.
2. Client enters payment data which is sent to Bluefin for processing.
3. Bluefin replies (with HTTP 200) with a success/fail response directly to the client browser.
4. Asynchronously, the transaction data is also sent directly back to the merchants POSTback URL for their use.

Payment Transaction Flow (POSTback and Transparent Redirect)



Notes:

1. Client requests Merchant payment page, which is sent to the client browser from the Merchants website.
2. Client enters payment data which is sent to Bluefin for processing.
3. Bluefin replies (with HTTP 303) with a success/fail response to the client browser, which is redirected to the specified transparent redirect URL, which actually sends a merchant specified webpage to the client browser.
4. Asynchronously, the transaction data is also sent directly back to the merchants POSTback URL for their use.

Appendix: Point to Point Encryption

Overview:

Point to Point Encryption (P2PE) allows the merchant the ability to safeguard their customer's data from the point of capture all the way through its delivery to Bluefin.

Bluefin approved encryption devices such as the SecuRED and M130 encrypt the data at the reader. This ensures that only encrypted data leaves the capture device and enters the merchant's computer via the USB connector. The encrypted data cannot be decrypted locally. It can only be decrypted once safely on the Bluefin servers.

The encryption device does however allow for PCI approved data such as the card holders name, first four of the card, last four of the card, and expiration date to be shown for you to capture if needed.

Orientation:

Bluefin supports several devices for card swipe or keyed entry. Popular choices among P2PE devices are the SecuRED and SHRED devices, and for E2E encryption, the M100 and M130. For mobile applications, Bluefin supports the Shuttle device for E2E encryption and the forthcoming Prima M device for P2P encryption means form a mobile platform.

They all function as keyboard emulators. Their output can be targeted in the exact same way the output of a keyboard can be targeted. The M100 is a key entry only device. The SHRED and the M130 are both a keyed entry and a card swipe device. The SecuRED is a card swipe only device. Both of the mobile devices are swipe only, audio jack, devices.

When accepting card swipe or keyed input from an encrypted POI terminal such as the M100 or M130, the data is captured directly as keyboard input, as if someone had typed it into your application. In order to ensure that these "keystrokes" are recorded into the correct field, the merchant application must have keyboard focus on the field that will receive the input.

For a web form, you can use HTML code such as the following to place the keyboard focus on the appropriate form field as soon as the page loads:

```
<body onLoad="document.getElementById('encryptedpayload').focus();">
```

Elsewhere, in the form itself, there should be an input field using the same name (in this example, we have used "encryptedpayload", although the name of the field could be anything). We recommend using a "password" input type because the encrypted output does not need to be visible and may create confusion to the user. Here is a simple example:

```
<input type="password" name="encryptedpayload">
```

In order to successfully submit a **swiped transaction**, you will need to pass the encrypted output in its entirety. You can pass it using the variable "card_tracks". For swipe transactions, you do **not** need to

supply the card_number, card_expiration, first_name, last_name or card_verification fields, as these are included in the encrypted data.

In order to successfully submit a **keyed transaction**, you will need to pass the encrypted output in its entirety. You can pass it using the variable "card_number". For keyed transactions, you do **not** need to supply the card_number, card_expiration, or card_verification fields, as these are included in the encrypted data. The first_name, last_name fields are not generally required unless they are explicitly required by the processor.

*Please note with keyed entries, the M100 and M130 will prompt you for additional card fields.

Note: If you are using Transparent Redirect, we recommend creating two separate pages or payment sections related to swipe or keyed transactions and allow the cashier to select the appropriate field before swiping or keying the credit card information.

The M100 & M130 outputs their data in an encrypted format. Please note that other than the "in the clear" data elements listed below, no other components of the output should be stored. The remaining output will not be able to be decrypted, and any submissions over the API of a partial output will cause a failed submission.

For **swiped transactions**, the output data is comprised of encrypted card data and the following "in the clear" elements.

Card holders name = CHolder

First four and last four card digits = MskPAN

Card expiration = Exp

For **keyed entries**, there are several options available on the M100 and M130 device to choose from. The M100 and M130, by default, are set to **Configuration #1**, but can be adjusted to support up to 5 different formats if you wish to use a different one. All possible configurations are listed below, but Bluefin recommends configuration #1 (default) or #4.

Configuration #1 (Recommended Configuration)

First four and last four card digits = MskPAN

Card expiration = Exp

Configuration #2

First four and last four card digits = MskPAN

Card expiration = Exp

Zip code = AVSZip

Configuration #3

First four and last four card digits = MskPAN

Card expiration = Exp

Street number of the Address = AVSAddr (not utilized by Bluefin for processing, or stored)

Zip code = AVSZip

Configuration #4 (Recommended Configuration)

First four and last four card digits = MskPAN

Card expiration = Exp

Zip code = AVSZip (not utilized by Bluefin for processing or stored)

Security code = ***

*This configuration asks for the CVV2 security code, but it is encrypted and not included in the output.

Configuration #5

First four and last four card digits = MskPAN

Card expiration = Exp

Street number of the Address= AVSAddr (not utilized by Bluefin for processing or stored)

Zip code = AVSZip (not utilized by Bluefin for processing or stored)

Security code = ***

*This configuration asks for the CVV2 security code, but it is encrypted and not included in the output.

To modify the configurations on your M100 and M130 unit, press the “Admin” key. The screen will display “Select Config 1-5.” Simply hit the number on the keypad indicating the configuration you want, and then press Enter. Again, Bluefin recommends Configuration #1, or optionally #4.

Variables Index

The following table includes an index of the API request variables for point to point encryption (P2PE) or end to end encryption (E2EE):

Variable Name	Max	Type	Description
card_tracks		Alphanumeric	This allows for the submission of the entire data payload from a swipe produced on an encrypted device such as the M100 or M130.
card_number		Alphanumeric	This allows for the submission of the entire data payload from a keyed entry produced on an encrypted device such as the M100 or M130.

There are no unique responses for submissions sent using encrypted data.

Appendix: Best Practices

Utilizing responses

Response checking /error handling:

Merchants should make sure that all responses messages from Bluefin are captured and utilized to confirm a successful transaction.

Utilizing reporting

Daily Reporting:

RSAPI reports return all transactions from a 24-hour period. RSAPI reports should not be run for the current day because too many transaction statuses are subject to change until all batches for that day are settled at the end of the day.

ACH Reporting:

As mentioned previously in this document, ACH transactions can take several days and sometimes even weeks to update their statuses. As a matter of practice merchants should setup a daily report to check for any ACH statuses that changed, so they can update their own records. To do so, run an RSAPI report using “action_date”. The report will show you the data for all transaction confirmations that were received from the previous day.

Recurring reporting:

Starting with version 3.6.1 of QSAPI, it returns a recurring ID (“recurring_id”) that merchants can use to segment out their recurring transactions from their standard transactions. This extra ID is in addition to the standard transaction ID. For many of our older API users who wish to have recurring IDs shown in an RSAPI report the switch to a newer version can usually be easily accommodated. Contact support@bluefin.com to find out what would need to be modified for an upgrade.

Parsing the data:

The data should be parsed using field names. It should not be based on the incremental position of data rows. Future versions of RSAPI may include new field names or exclude previous field names, and parsing based on incremental values could lead to internal migration complexities for merchants.

Card capture

Identifying a card brand:

Issuer identification numbers are used to identify which card issuing company a particular card belongs to. You can determine this by checking the first few numbers of a credit card.

Visa – Starts with 4

Mastercard – Starts with numbers 51-55

American Express – Starts with 34, 37

Discover – Starts with 6011, 622126-622925, 644-649, 65

JCB – Starts with 3528-3589

Verifying if a credit card is a valid number prior to submitting a transaction:

All credit card numbers can be proven as being a valid credit card number by running a “Mod 10” or “Luhn algorithm” against a card number. Some merchants choose to verify a card prior to submission, to reduce the number of declined transactions that they receive. This practice is typically handled by a java script. Numerous examples of scripts that handle “Mod 10” can be found on the internet. Bluefin does not require merchants to perform a “Mod 10” check prior to submitting a card.

Error messages:

When coding your applications, please note that the wording on error handling responses are subject to change.

Zero Dollar (\$0) Authorizations

When the Merchant issues a \$0 authorization thru the API, depending on the end processor, Bluefin may register the \$0 authorization but in reality, is sending a \$1 authorization to the processor, receiving approval from the processor, passing the approval to the Merchant and at the same time, automatically issuing a reversal of the \$1 authorization. So, it is possible that the actual cardholder could see a \$1 authorization on their card, even though the authorization was reversed and in effect, cancelled.

URL Encoding

If you do not use the Bluefin PHP class in interacting with the API’s, the data element values that are sent must meet the “URL Encoding” standards.

URL Encoding is the process of converting string(s) into a valid URL format. Valid URL format means that the URL contains only what is termed as "alpha, digit, safe, extra, escape" characters, replacing “unsafe ASCII” characters with a “%” followed by two hexadecimal digits.

URL encoding is normally performed to convert data passed via html forms, because such data may contain special character, such as "/", ".", "#", and so on, which could either: a) have special meanings; or b) is not a valid character for an URL; or c) could be altered during transfer. For instance, the "#" character needs to be encoded because it has a special meaning of that of an html anchor. The <space> character also needs to be encoded because is not allowed on a valid URL format. Also, some characters, such as "~" might not transport properly across the internet.

Appendix: Request and Response Codes

Overview

There are several ways to access Bluefin's APIs. You are free to choose whichever method works best for your development environment. The following examples are intended to show you the basic structure of posts and responses. Since many of our merchants use PHP, the following examples were formatted for use with PHP.

If you are using PHP, we recommend you use our simple PHP classes to call the APIs. Please see the sample PHP script below:

```
<?php
header('Content-type: text/plain');

require_once('includes/qsapi-post-3.8.class.php');
$qqs = new QuickSwipePost();

$qqs->setParams(array (
    'account_id' => '000000000001',
    'api_accesskey' => 'abcdef123456',
    'tender_type' => 'CARD',
    'transaction_type' => 'SALE',
    'transaction_amount' => 176.58,
    'transaction_description' => 'Test transaction',
    'card_number' => '4444333322221111',
    'card_expiration' => '0315',
    'card_verification' => '315',
    'first_name' => 'John',
    'last_name' => 'Smith',
    'street_address1' => '123 Main St',
    'city' => 'Anytown',
    'state' => 'NY',
    'zip' => '10101',
    'phone' => '212-555-1212',
));

$qqs->process();
$response = $qqs->getResponse();

if ($response['error']) {
    echo "There was an error processing your payment.\n";
    echo "The error message was $response[error_message].\n";
    if (!empty($response['authorization_message'])) {
        echo "The authorization message was $response[authorization_message].\n";
    }
} else {
    echo "Your transaction was approved!\n";
}
```

```
echo "Here is your receipt.\n";
echo "Transaction ID $response[transaction_id]\n";
echo "Tender type $response[tender_type]\n";
echo "Date/time $response[transaction_timestamp]\n";
echo "Transaction type $response[transaction_type]\n";
echo "Transaction amount " . sprintf('%.2f', $response[transaction_amount]) . "\n";
echo "Last 4 $response[last4]\n";
if ($response['tender_type'] === 'CARD') {
    echo "Card type $response[card_brand]\n";
    echo "Card expiration $response[card_expiration]\n";
}
if (!empty($response['transaction_description'])) echo "\nTransaction
description:\n$response[transaction_description]\n";
}
?>
```

NOTE: Use of the PHP class does not support transparent redirect.

Code Samples

To demonstrate another way to call the APIs, here are other examples, using curl, demonstrating the different response formats available.

Note: The URLs shown in the post examples below are the current standards. Previous URLs that were accepted in earlier API versions (like: "cert.quickswipe.com") will still function as they always have and could be changed to the current standards over time.

Post a sale, using the default www-form-urlencoded format

```
curl -s -d account_id=000000000001 -d api_accesskey=abcdef123456 -d tender_type=CARD -d transaction_type=SALE -d transaction_amount=176.58 -d transaction_description='Test transaction' -d card_number=4444333322221111 -d card_expiration=0315 -d card_verification=315 -d first_name=John -d last_name=Smith -d street_address1='123 Main St' -d city=Anytown -d state=NY -d zip=10101 -d phone=212-555-1212 https://secure.payconex.net/api/qsapi/3.8/
```

Response

```
transaction_id=000000000057&tender_type=CARD&transaction_timestamp=2012-06-21%2015%3A46%3A57&card_brand=VISA&transaction_type=SALE&last4=1111&card_expiration=0315&authorization_code=094292&authorization_message=APPROVED&transaction_amount=176.58&first_name=John&last_name=Smith&keyed=1&swiped=&transaction_approved=1&avs_response=Y&transaction_description=Test%20transaction&error=&error_code=0&error_message=&error_msg=
```

Post a sale, using the JSON format

```
curl -s -d account_id=000000000001 -d api_accesskey=abcdef123456 -d tender_type=CARD -d transaction_type=SALE -d transaction_amount=176.58 -d transaction_description='Test transaction' -d card_number=4444333322221111 -d card_expiration=0315 -d card_verification=315 -d first_name=John -d last_name=Smith -d street_address1='123 Main St' -d city=Anytown -d state=NY -d zip=10101 -d phone=212-555-1212 -d response_format=JSON https://secure.payconex.net/api/qsapi/3.8/
```

Response

```
{"transaction_id":"000000000059","tender_type":"CARD","transaction_timestamp":"2012-06-21 15:49:48","card_brand":"VISA","transaction_type":"SALE","last4":"1111","card_expiration":"0315","authorization_code":"094355","authorization_message":"APPROVED","transaction_amount":176.58,"first_name":"John","last_name":"Smith","keyed":true,"swiped":false,"transaction_approved":true,"avs_response":"Z","transaction_description":"Test transaction","error":false,"error_code":0,"error_message":null,"error_msg":null}
```

Post a sale, using the JSONP format

```
curl -s -d account_id=000000000001 -d api_accesskey=abcdef123456 -d tender_type=CARD -d transaction_type=SALE -d transaction_amount=176.58 -d transaction_description='Test transaction' -d card_number=4444333322221111 -d card_expiration=0315 -d card_verification=315 -d first_name=John -d last_name=Smith -d street_address1='123 Main St' -d city=Anytown -d state=NY -d zip=10101 -d phone=212-555-1212 -d response_format=JSONP -d jsonp=c_func https://secure.payconex.net/api/qsapi/3.8/
```

Response

```
c_func({"transaction_id":"000000000059","tender_type":"CARD","transaction_timestamp":"2012-06-21 15:49:48","card_brand":"VISA","transaction_type":"SALE","last4":"1111","card_expiration":"0315","authorization_code":"094355","authorization_message":"APPROVED","transaction_amount":176.58,"first_name":"John","last_name":"Smith","keyed":true,"swiped":false,"transaction_approved":true,"avs_response":"Z","transaction_description":"Test transaction","error":false,"error_code":0,"error_message":null,"error_msg":null})
```

Post a sale, using the DEBUG output (which is used in all samples below)

```
curl -s -d account_id=000000000001 -d api_accesskey=abcdef123456 -d tender_type=CARD -d transaction_type=SALE -d transaction_amount=176.58 -d transaction_description='Test transaction' -d card_number=4444333322221111 -d card_expiration=0315 -d card_verification=315 -d first_name=John -d last_name=Smith -d street_address1='123 Main St' -d city=Anytown -d state=NY -d zip=10101 -d phone=212-555-1212 -d response_format=DEBUG https://secure.payconex.net/api/qsapi/3.8/
```

Response

```
array (
  'transaction_id' => '000000000060',
  'tender_type' => 'CARD',
  'transaction_timestamp' => '2012-06-21 15:50:49',
  'card_brand' => 'VISA',
  'transaction_type' => 'SALE',
  'last4' => '1111',
  'card_expiration' => '0315',
  'authorization_code' => '094387',
  'authorization_message' => 'APPROVED',
  'transaction_amount' => 176.58,
  'first_name' => 'John',
  'last_name' => 'Smith',
  'keyed' => true,
  'swiped' => false,
  'transaction_approved' => true,
  'avs_response' => 'A',
  'transaction_description' => 'Test transaction',
  'error' => false,
  'error_code' => 0,
  'error_message' => NULL,
  'error_msg' => NULL,
)
```

QSAPI

Example request for post a sale (Keyed)

```
$qs->setParams(array (  
    'account_id' => '000000000001',  
    'api_accesskey' => 'abcdef123456',  
    'tender_type' => 'CARD',  
    'transaction_type' => 'SALE',  
    'transaction_amount' => 176.58,  
    'transaction_description' => 'Test transaction',  
    'card_number' => '4444333322221111',  
    'card_expiration' => '0315',  
    'card_verification' => '315',  
    'first_name' => 'John',  
    'last_name' => 'Smith',  
    'street_address1' => '123 Main St',  
    'city' => 'Anytown',  
    'state' => 'NY',  
    'zip' => '10101',  
    'phone' => '212-555-1212',  
));
```

Example response for post a sale (Keyed)

```
array (  
    'transaction_id' => '0000000000027',  
    'tender_type' => 'CARD',  
    'transaction_timestamp' => '2012-06-20 11:09:54',  
    'card_brand' => 'VISA',  
    'transaction_type' => 'SALE',  
    'last4' => '1111',  
    'card_expiration' => '0315',  
    'authorization_code' => '090047',  
    'authorization_message' => 'APPROVED',  
    'transaction_amount' => 176.58,  
    'first_name' => 'John',  
    'last_name' => 'Smith',  
    'keyed' => true,  
    'swiped' => false,  
    'entry_mode' => 'keyed',  
    'transaction_approved' => true,  
    'avs_response' => 'U',  
    'transaction_description' => 'Test transaction',  
    'error' => false,  
    'error_code' => 0,  
    'error_message' => NULL,  
    'error_msg' => NULL,  
)
```

Example request for post a sale (Swiped)

```
$qs->setParams(array (
  'account_id' => '000000000001',
  'api_accesskey' => 'abcdef123456',
  'tender_type' => 'CARD',
  'transaction_type' => 'SALE',
  'transaction_amount' => 176.58,
  'transaction_description' => 'Test transaction',
  'card_tracks' => '<DvcMsg Ver="1.1"><Dvc App="SecureKey Software" AppVer="1.0" DvcType="M130-
IDTECH" DvcSN="54122505007" Entry="SWIPE"></Dvc><Card CEncode="0"
ETrk1="CD42D06BF708D44F847E5B561E68B3C920EAF5943F63131D2CA09D9E69EE522CD8E69FA9394
385CE5A214F466F6315CA0F9DC12330B88E301DC
7BAE2EC752B109D97B5152F425A6639CB2AB4BB981DE"
ETrk2="07A7782E3BDBD50FE7A6F03E1E294AEBCD8C222AD5CDBA725FFFFDFE71E9BACB04FE4284B32
12132" CDataKSN="629949123B00018001F0" Exp="1604" MskPAN="4012*****1111"
CHolder="ABCD TEST CARD /VISA" EFormat="4"></Card><Addr></Addr><Tran
TranType="CREDIT"></Tran></DvcMsg>'
));
```

Example response for post a sale (Swiped)

```
array (
  'transaction_id' => '0000000000023',
  'tender_type' => 'CARD',
  'transaction_timestamp' => '2018-05-23 11:23:23',
  'card_brand' => 'VISA',
  'transaction_type' => 'SALE',
  'last4' => '1111',
  'card_expiration' => '0416',
  'authorization_code' => '090047',
  'authorization_message' => 'APPROVED',
  'transaction_amount' => 176.58,
  'first_name' => 'John',
  'last_name' => 'Smith',
  'keyed' => false,
  'swiped' => true,
  'entry_mode' => 'swiped',
  'transaction_approved' => true,
  'avs_response' => 'U',
  'transaction_description' => 'Test transaction',
  'error' => false,
  'error_code' => 0,
  'error_message' => NULL,
  'error_msg' => NULL,
)
```

Example request for post a sale with level 2 data

```
$qs->setParams(array (
  'account_id' => '000000000001',
  'api_accesskey' => 'abcdef123456',
  'tender_type' => 'CARD',
  'transaction_type' => 'SALE',
  'transaction_amount' => 176.58,
  'transaction_description' => 'Test transaction',
  'card_number' => '4444333322221111',
  'card_expiration' => '0315',
  'card_verification' => '315',
  'first_name' => 'John',
  'last_name' => 'Smith',
  'street_address1' => '123 Main St',
  'city' => 'Anytown',
  'state' => 'NY',
  'zip' => '10101',
  'phone' => '212-555-1212',
  'level2_tax' => '21.18',
  'level2_merchant_reference' => 'Just because',
  'level2_zip' => '12345',
  'level2_orderid' => '123456789012345',
));
```

Example response for post a sale with level 2 data

```
array (
  'transaction_id' => '000000000027',
  'tender_type' => 'CARD',
  'transaction_timestamp' => '2018-06-20 11:09:54',
  'card_brand' => 'VISA',
  'transaction_type' => 'SALE',
  'last4' => '1111',
  'card_expiration' => '0315',
  'authorization_code' => '090047',
  'authorization_message' => 'APPROVED',
  'transaction_amount' => 176.58,
  'first_name' => 'John',
  'last_name' => 'Smith',
  'keyed' => true,
  'swiped' => false,
  'entry_mode' => 'keyed',
  'transaction_approved' => true,
  'avs_response' => 'U',
  'transaction_description' => 'Test transaction',
  'error' => false,
  'error_code' => 0,
  'error_message' => NULL,
```



```
'error_msg' => NULL,  
)
```

Example request for pre-authorization for a card not on file and without a token

```
$qs->setParams(array (  
  'account_id' => '000000000001',  
  'api_accesskey' => 'abcdef123456',  
  'tender_type' => 'CARD',  
  'transaction_type' => 'AUTHORIZATION',  
  'transaction_amount' => 176.58,  
  'card_number' => '4444333322221111',  
  'card_expiration' => '0315',  
  'card_verification' => '315',  
));
```

Example response request for pre-authorization for a card not on file and without a token

```
array (  
  'transaction_id' => '000000000039',  
  'tender_type' => 'CARD',  
  'transaction_timestamp' => '2018-06-20 15:33:50',  
  'card_brand' => 'VISA',  
  'transaction_type' => 'AUTHORIZATION',  
  'last4' => '1111',  
  'card_expiration' => '0315',  
  'authorization_code' => '099105',  
  'authorization_message' => 'APPROVED',  
  'transaction_amount' => 176.58,  
  'keyed' => true,  
  'swiped' => false,  
  'entry_mode' => 'keyed',  
  'transaction_approved' => true,  
  'error' => false,  
  'error_code' => 0,  
  'error_message' => NULL,  
  'error_msg' => NULL,  
)
```

Example request for pre-authorization for a card on file with a token

```
$qs->setParams(array (  
  'account_id' => '000000000001',  
  'api_accesskey' => 'abcdef123456',  
  'tender_type' => 'CARD',  
  'transaction_type' => 'AUTHORIZATION',  
  'transaction_amount' => 18,  
  'token_id' => '000000000027',  
  'reissue' => true,  
));
```

Example response request for pre-authorization for a card on file with a token

```
array (  
  'transaction_id' => '000000000040',  
  'tender_type' => 'CARD',  
  'transaction_timestamp' => '2018-06-20 15:35:13',  
  'card_brand' => 'VISA',  
  'transaction_type' => 'AUTHORIZATION',  
  'last4' => '1111',  
  'card_expiration' => '0315',  
  'authorization_code' => '099182',  
  'authorization_message' => 'APPROVED',  
  'transaction_amount' => 18,  
  'first_name' => 'John Smith',  
  'keyed' => true,  
  'swiped' => false,  
  'entry_mode' => 'keyed',  
  'transaction_approved' => true,  
  'avs_response' => 'U',  
  'transaction_description' => 'Test transaction',  
  'error' => false,  
  'error_code' => 0,  
  'error_message' => NULL,  
  'error_msg' => NULL,  
)
```

Example request for post a capture

```
$qs->setParams(array (  
  'account_id' => '000000000001',  
  'api_accesskey' => 'abcdef123456',  
  'transaction_type' => 'CAPTURE',  
  'token_id' => '000000000040',  
));
```

Example response for post a capture

```
array (  
  'transaction_id' => '000000000027',  
  'tender_type' => 'CARD',  
  'transaction_timestamp' => '2018-06-20 11:09:54',  
  'card_brand' => 'VISA',  
  'transaction_type' => 'SALE',  
  'last4' => '1111',  
  'card_expiration' => '0315',  
  'authorization_code' => '090047',  
  'authorization_message' => 'APPROVED',  
  'transaction_amount' => 176.58,  
  'first_name' => 'John',  
  'last_name' => 'Smith',  
  'keyed' => true,
```

```
'swiped' => false,  
'entry_mode' => 'keyed',  
'transaction_approved' => true,  
'avs_response' => 'U',  
'transaction_description' => 'Test transaction',  
'error' => false,  
'error_code' => 0,  
'error_message' => NULL,  
'error_msg' => NULL,  
)
```

Example request for posting a reissue of a sale

```
$qs->setParams(array (  
  'account_id' => '00000000001',  
  'api_accesskey' => 'abcdef123456',  
  'tender_type' => 'CARD',  
  'transaction_type' => 'SALE',  
  'transaction_amount' => 176.58,  
  'transaction_description' => 'Test transaction',  
  'first_name' => 'John',  
  'last_name' => 'Smith',  
  'street_address1' => '123 Main St',  
  'city' => 'Anytown',  
  'state' => 'NY',  
  'zip' => '10101',  
  'phone' => '212-555-1212',  
  'token_id' => '000000000027',  
  'reissue' => 1,  
));
```

Example response for posting a reissue of a sale

```
array (  
  'transaction_id' => '000000000027',  
  'tender_type' => 'CARD',  
  'transaction_timestamp' => '2018-06-20 11:09:54',  
  'card_brand' => 'VISA',  
  'transaction_type' => 'SALE',  
  'last4' => '1111',  
  'card_expiration' => '0315',  
  'authorization_code' => '090047',  
  'authorization_message' => 'APPROVED',  
  'transaction_amount' => 176.58,  
  'first_name' => 'John',  
  'last_name' => 'Smith',  
  'keyed' => true,  
  'swiped' => false,  
  'entry_mode' => 'keyed',
```

```
'transaction_approved' => true,
'avs_response' => 'U',
'transaction_description' => 'Test transaction',
'error' => false,
'error_code' => 0,
'error_message' => NULL,
'error_msg' => NULL,
)
```

Example request for posting a refund

```
$qs->setParams(array (
    'account_id' => '000000000001',
    'api_accesskey' => 'abcdef123456',
    'transaction_type' => 'REFUND',
    'token_id' => '000000000041',
));
```

Example response for posting a refund

```
array (
    'original_transaction_id' => '000000000041',
    'transaction_id' => '000000000042',
    'tender_type' => 'CARD',
    'transaction_timestamp' => '2018-06-20 15:39:26',
    'card_brand' => 'VISA',
    'transaction_type' => 'REFUND',
    'last4' => '1111',
    'card_expiration' => '0315',
    'authorization_message' => 'VOID',
    'transaction_amount' => 25,
    'first_name' => 'John Smith',
    'keyed' => true,
    'swiped' => false,
    'entry_mode' => 'keyed',
    'transaction_approved' => true,
    'transaction_description' => 'Test transaction',
    'error' => false,
    'error_code' => 0,
    'error_message' => NULL,
    'error_msg' => NULL,
)
```

Example request for posting a credit

```
$qs->setParams(array (
    'account_id' => '000000000001',
    'api_accesskey' => 'abcdef123456',
    'tender_type' => 'CARD',
    'transaction_type' => 'CREDIT',
    'transaction_amount' => 100,
```

```
'card_number' => '4444333322221111',
'card_expiration' => '0315',
'card_verification' => '315',
));
```

Example response for posting a credit

```
array (
  'transaction_id' => '000000000043',
  'tender_type' => 'CARD',
  'transaction_timestamp' => '2018-06-20 15:42:37',
  'card_brand' => 'VISA',
  'transaction_type' => 'CREDIT',
  'last4' => '1111',
  'card_expiration' => '0315',
  'authorization_message' => 'CREDIT',
  'transaction_amount' => 100,
  'keyed' => true,
  'swiped' => false,
  'entry_mode' => 'keyed',
  'transaction_approved' => true,
  'error' => false,
  'error_code' => 0,
  'error_message' => NULL,
  'error_msg' => NULL,
)
```

Example request for posting a force

```
$qs->setParams(array (
  'account_id' => '000000000001',
  'api_accesskey' => 'abcdef123456',
  'tender_type' => 'CARD',
  'transaction_type' => 'FORCE',
  'authorization_code' => '123456',
  'transaction_amount' => 12.5,
  'card_number' => '4444333322221111',
  'card_expiration' => '0315',
  'card_verification' => '315',
));
```

Example response for posting a force

```
array (
  'transaction_id' => '000000000044',
  'tender_type' => 'CARD',
  'transaction_timestamp' => '2018-06-20 16:14:04',
  'card_brand' => 'VISA',
  'transaction_type' => 'FORCE',
  'last4' => '1111',
  'card_expiration' => '0315',
```

```
'authorization_code' => '123456',
'authorization_message' => 'FORCE',
'transaction_amount' => 12.5,
'keyed' => true,
'swiped' => false,
'entry_mode' => 'keyed',
'transaction_approved' => true,
'error' => false,
'error_code' => 0,
'error_message' => NULL,
'error_msg' => NULL,
)
```

Example request for storing a card

```
$qs->setParams(array (
  'account_id' => '000000000001',
  'api_accesskey' => 'abcdef123456',
  'tender_type' => 'CARD',
  'transaction_type' => 'STORE',
  'card_number' => '4444333322221111',
  'card_expiration' => '0315',
));
```

Example response for storing a card

```
array (
  'transaction_id' => '000000000045',
  'tender_type' => 'CARD',
  'transaction_timestamp' => '2018-06-20 16:15:13',
  'card_brand' => 'VISA',
  'last4' => '1111',
  'card_expiration' => '0315',
  'transaction_amount' => 0,
  'keyed' => true,
  'swiped' => false,
  'entry_mode' => 'keyed',
  'transaction_approved' => false,
  'error' => false,
  'error_code' => 0,
  'error_message' => NULL,
  'error_msg' => NULL,
)
```

Example request for post a sale (with partial authorization)

```
$qs->setParams(array (
  'account_id' => '000000000001',
  'api_accesskey' => 'abcdef123456',
  'tender_type' => 'CARD',
  'transaction_type' => 'SALE',
```

```
'transaction_amount' => 176.58,  
'transaction_description' => 'Test transaction',  
'card_number' => '4444333322221111',  
'card_expiration' => '0315',  
'card_verification' => '315',  
'first_name' => 'John',  
'last_name' => 'Smith',  
'street_address1' => '123 Main St',  
'city' => 'Anytown',  
'state' => 'NY',  
'zip' => '10101',  
'phone' => '212-555-1212',  
));
```

Example response for post a sale (with partial authorization)

```
array (  
  'transaction_id' => '000000000027',  
  'tender_type' => 'CARD',  
  'transaction_timestamp' => '2018-06-20 11:09:54',  
  'card_brand' => 'VISA',  
  'transaction_type' => 'SALE',  
  'last4' => '1111',  
  'card_expiration' => '0315',  
  'authorization_code' => '090047',  
  'authorization_message' => 'APPROVED FOR LESSER AMOUNT',  
  'request_amount' => 176.58,  
  'transaction_amount' => 100.00,  
  'first_name' => 'John',  
  'last_name' => 'Smith',  
  'keyed' => true,  
  'swiped' => false,  
  'entry_mode' => 'keyed',  
  'transaction_approved' => true,  
  'transaction_description' => 'Test transaction',  
  'error' => false,  
  'error_code' => 0,  
  'error_message' => NULL,  
  'error_msg' => NULL,  
)
```

Example request for reversal of an authorization

```
$qs->setParams(array (  
  'account_id' => '000000000001',  
  'api_accesskey' => 'abcdef123456',  
  'transaction_type' => 'REVERSAL',  
  'token_id' => '000000000051',  
));
```

Example response for reversal of an authorization

```
array ( 'original_transaction_id' => '000000000051',
  'transaction_id' => '000000000052',
  'tender_type' => 'CARD',
  'transaction_timestamp' => '2018-08-06 09:42:03',
  'card_brand' => 'VISA',
  'transaction_type' => 'REVERSAL',
  'last4' => '1111',
  'card_expiration' => '0320',
  'authorization_code' => '111111',
  'authorization_message' => 'APPROVED',
  'transaction_amount' => 1.0,
  'first_name' => 'John Smith',
  'keyed' => true,
  'swiped' => false,
  'entry_mode' => 'keyed',
  'transaction_approved' => true,
  'transaction_description' => 'Test transaction',
  'error' => false,
  'error_code' => 0,
  'error_message' => NULL,
  'error_msg' => NULL,
)
```

Example request for cancel of a refund

```
$qs->setParams(array (
  'account_id' => '000000000001',
  'api_accesskey' => 'abcdef123456',
  'transaction_type' => 'CANCEL',
  'token_id' => '000000000061',
));
```

Example response for cancel of a refund

```
array ( 'original_transaction_id' => '000000000061',
  'transaction_id' => '000000000062',
  'tender_type' => 'CARD',
  'transaction_timestamp' => '2018-08-06 09:44:03',
  'card_brand' => 'VISA',
  'transaction_type' => 'CANCEL',
  'last4' => '1111',
  'card_expiration' => '0320',
  'authorization_code' => '111111',
  'authorization_message' => 'APPROVED',
  'transaction_amount' => 1.0,
  'first_name' => 'John Smith',
  'keyed' => true,
  'swiped' => false,
  'entry_mode' => 'keyed',
```



```

'transaction_approved' => true,
'transaction_description' => 'Test transaction',
'error' => false,
'error_code' => 0,
'error_message' => NULL,
'error_msg' => NULL,
)

```

Example request for a Sale using an eToken

```

$qs->setParams(array (
  'account_id' => '000000000001',
  'api_accesskey' => 'abcdef123456',
  'tender_type' => 'CARD',
  'transaction_type' => 'SALE',
  'transaction_amount' => 176.58,
  'transaction_description' => 'Test transaction',
  'etoken' => 'RzjHUvuvbOIfZLaDvNm-
2Aq6_rQ0vzqqa3_0hRNu_NvptjsuyWAPV5yrB2RZthgfHorU_6Grj8qqIqQf_FgH0shvS3FWwSKioSCx2uDaG
gSTTZD18ZvdDCdjpFAaKzUznvGvSmo20z4cLh-fGZO32c4Yrm-er-
RnfteB0pjqWfi39ggox65AQ03BWikCvFPOPe7VEYIfHMGF4Vt3cGDEw4YAp13ANjf_'
)

```

Example response for a Sale using an eToken

```

array (
  'transaction_id' => '0000000000023',
  'tender_type' => 'CARD',
  'transaction_timestamp' => '2018-05-23 11:23:23',
  'card_brand' => 'VISA',
  'transaction_type' => 'SALE',
  'last4' => '1111',
  'card_expiration' => '0416',
  'authorization_code' => '090047',
  'authorization_message' => 'APPROVED',
  'transaction_amount' => 176.58,
  'first_name' => '',
  'last_name' => '',
  'keyed' => true,
  'swiped' => false,
  'entry_mode' => 'keyed',
  'transaction_approved' => true,
  'avs_response' => '',
  'transaction_description' => 'Test transaction',
  'error' => false,
  'error_code' => 0,
  'error_message' => NULL,
  'error_msg' => NULL,
)

```

TSAPI

Example request

```
$ts->setParams(array (  
    'account_id' => '000000000001',  
    'api_accesskey' => 'abcdef123456',  
    'action' => 'GET_TRANSACTION_ID',  
));
```

Example response

```
array (  
    'transaction_id' => '000000000050',  
    'error' => false,  
    'error_code' => 0,  
    'error_message' => NULL,  
    'error_msg' => NULL,  
)
```

Example QSAPI post, using fetched transaction ID

```
$qs->setParams(array (  
    'account_id' => '000000000001',  
    'api_accesskey' => 'abcdef123456',  
    'transaction_id' => '000000000050',  
    'tender_type' => 'CARD',  
    'transaction_type' => 'SALE',  
    'transaction_amount' => 29.5,  
    'card_number' => '4444333322221111',  
    'card_expiration' => '0315',  
    'card_verification' => '315',  
));
```

Example QSAPI response, using fetched transaction ID

```
array (  
    'transaction_id' => '000000000050',  
    'tender_type' => 'CARD',  
    'transaction_timestamp' => '2018-06-20 17:29:13',  
    'card_brand' => 'VISA',  
    'transaction_type' => 'SALE',  
    'last4' => '1111',  
    'card_expiration' => '0315',  
    'authorization_code' => '092113',  
    'authorization_message' => 'APPROVED',  
    'transaction_amount' => 29.5,  
    'keyed' => true,  
    'swiped' => false,  
    'transaction_approved' => true,  
    'error' => false,  
    'error_code' => 0,
```

```
'error_message' => NULL,  
'error_msg' => NULL,  
)
```

Example request to get transaction status

```
$ts->setParams(array (  
  'account_id' => '000000000001',  
  'api_accesskey' => 'abcdef123456',  
  'action' => 'GET_TRANSACTION_STATUS',  
  'transaction_id' => '000000000050',  
));
```

Example response to get transaction status

```
array (  
  'transaction_id' => '000000000050',  
  'transaction_timestamp' => '2018-06-20 17:29:13',  
  'authorization_message' => 'APPROVED',  
  'transaction_amount' => 29.5,  
  'cashier' => 'QSAPI 3.8',  
  'transaction_approved' => true,  
  'found' => true,  
  'error' => false,  
  'error_code' => 0,  
  'error_message' => NULL,  
  'error_msg' => NULL,  
)
```

SLAPI

Example request for new recurring payments

Note: For documentation purposes, these examples use the DEBUG response format, since it is more easily readable. In actual usage, you will generally use one of the machine-readable formats, such as FORM or JSON.

```
$sl->setParams(array (  
  'account_id' => '000000000001',  
  'api_accesskey' => 'abcdef123456',  
  'action' => 'SETUP',  
  'token_id' => '0000000000054',  
  'recurring_payment_amount' => 20,  
  'recurring_payments_remaining' => NULL,  
  'recurring_schedule' => 'WEEKLY_MON',  
  'start_date' => '2018-08-01',  
));
```

Example response for recurring payments

```
$sl->setParams(array (  
  'account_id' => '000000000001',  
  'api_accesskey' => 'abcdef123456',  
  'action' => 'SETUP',  
  'token_id' => '0000000000054',  
  'recurring_payment_amount' => 20,  
  'recurring_payments_remaining' => NULL,  
  'recurring_schedule' => 'WEEKLY_MON',  
  'start_date' => '2018-08-01',  
));
```

Example request for modifying the schedule of an existing recurring payment

Note: The token_id is the value that was returned when you setup the recurring transaction initially. There is no need to pass values that do not need to be changed (e.g., recurring_payment_amount).

```
$sl->setParams(array (  
  'account_id' => '000000000001',  
  'api_accesskey' => 'abcdef123456',  
  'action' => 'EDIT',  
  'recurring_id' => 7,  
  'recurring_payment_amount' => 25,  
  'recurring_schedule' => 'WEEKLY_TUE',  
));
```

Example response for modifying the schedule of an existing recurring payment

```
array (  
  'recurring_id' => 7,  
  'token_id' => '0000000000054',  
  'recurring_schedule' => 'WEEKLY_TUE',  
  'recurring_payment_amount' => 25,
```

```
'status' => 'ENABLED',
'label' => 'Recurring entry - 1340295974',
'recurring_payments_remaining' => NULL,
'start_date' => '2018-08-01',
'recurring_schedule_description' => 'Every week on Tuesday',
'next_recurring_payment_date' => '2018-08-01',
'error' => false,
'error_code' => 0,
'error_msg' => NULL,
'error_message' => NULL,
)
```

Example post for modifying the value of a recurring payment

```
$sl->setParams(array (
    'recurring_id' => 1,
    'token_id' => '000000000010',
    'recurring_schedule' => 'WEEKLY_MON',
    'recurring_payment_amount' => 10,
    'status' => 'ENABLED',
    'label' => 'Recurring entry - 1340137652',
    'recurring_payments_remaining' => NULL,
    'start_date' => '2012-07-19',
    'recurring_schedule_description' => 'Every week on Monday',
    'next_recurring_payment_date' => '2018-06-25',
    'error' => false,
    'error_code' => 0,
    'error_msg' => NULL,
    'error_message' => NULL,
));
```

Example response for modifying the value of a recurring payment

```
array (
    'recurring_id' => 1,
    'token_id' => '000000000010',
    'recurring_schedule' => 'WEEKLY_MON',
    'recurring_payment_amount' => 10,
    'status' => 'ENABLED',
    'label' => 'Recurring entry - 1340137652',
    'recurring_payments_remaining' => NULL,
    'start_date' => '2012-07-19',
    'recurring_schedule_description' => 'Every week on Monday',
    'next_recurring_payment_date' => '2018-06-25',
    'error' => false,
    'error_code' => 0,
    'error_msg' => NULL,
    'error_message' => NULL,
)
```

Example post for disabling the schedule of an existing recurring payment

```
$sl->setParams(array (  
    'account_id' => '000000000001',  
    'api_accesskey' => 'abcdef123456',  
    'action' => 'EDIT',  
    'recurring_id' => '000000000001',  
    'status' => 'DISABLED',  
));
```

Example response for disabling the schedule of an existing recurring payment

```
array (  
    'recurring_id' => 1,  
    'token_id' => '000000000010',  
    'recurring_schedule' => 'WEEKLY_MON',  
    'recurring_payment_amount' => 10,  
    'status' => 'DISABLED',  
    'label' => 'Recurring entry - 1340137652',  
    'recurring_payments_remaining' => NULL,  
    'start_date' => '2018-07-19',  
    'recurring_schedule_description' => 'Every week on Monday',  
    'next_recurring_payment_date' => '2018-06-25',  
    'error' => false,  
    'error_code' => 0,  
    'error_msg' => NULL,  
    'error_message' => NULL,  
)
```

RSAPI

Example request

```
$rs->setParams(array (  
    'account_id' => '000000000001',  
    'api_accesskey' => 'abcdef123456',  
    'transaction_date' => '2018-06-20',  
    'tender_type' => 'CARD',  
));
```

Example response

```
"transaction_id","account_id","authorization_date","tender_type","transaction_type","keyed","swiped"  
,"transaction_amount","name","card_brand","last4","card_expiration","description","user_data","auth  
orization_msg","authorization_code","avs_response","cvv2_response","ip_address","cashier","street_a  
ddress1","city","state","zip","country","phone","email","group","refund_id","custom_id","action_date",  
"noc_data"  
"0000000000050","0000000000001","2018-06-20  
17:29:13","CARD","SALE","1","0","29.50","","VISA","1111","0315","","","APPROVED","092113","","N","  
,"QSAPI 3.8",,,,,,  
"0000000000044","0000000000001","2018-06-20  
16:14:04","CARD",,"1","0","12.50","","VISA","1111","0315","","","FORCE","123456",,"","QSAPI  
3.8",,,,,,  
"0000000000043","0000000000001","2018-06-20 15:42:37","CARD","CREDIT","1","0",-  
100.00",,"VISA","1111","0315",,"","CREDIT",,"","QSAPI 3.8",,,,,,  
"0000000000042","0000000000001","2018-06-20 15:39:26","CARD","REFUNDED  
SALE","1","0","25.00","John Smith","VISA","1111","0315","Test transaction",,"VOID",,"","QSAPI  
3.8","123 Main St","Anytown","NY","10101",,"212-555-1212",,"","0000000000041",,"",""  
"0000000000041","0000000000001","2018-06-20 15:38:26","CARD","REFUND","1","0",-25.00,"John  
Smith","VISA","1111","0315","Test transaction",,"VOID","099373","U",,"","QSAPI 3.8","123 Main  
St","Anytown","NY","10101",,"212-555-1212",,"","0000000000042",,"",""  
"0000000000040","0000000000001","2018-06-20 15:35:13","CARD","SALE","1","0","18.00","John  
Smith","VISA","1111","0315","Test transaction",,"APPROVED","099182","U",,"","QSAPI 3.8","123  
Main St","Anytown","NY","10101",,"212-555-1212",,"",""  
"0000000000039","0000000000001","2018-06-20  
15:33:50","CARD","AUTHORIZATION","1","0","176.58","","VISA","1111","0315","","","APPROVED","099  
105",,"N",,"QSAPI 3.8",,,,,,  
"0000000000027","0000000000001","2018-06-20 11:09:54","CARD","SALE","1","0","176.58","John  
Smith","VISA","1111","0315","Test transaction",,"APPROVED","090047","U","N",,"QSAPI 3.8","123  
Main St","Anytown","NY","10101",,"212-555-1212",,"","",,"",,""
```

JSON formatted example response

To receive the report in JSON simply add "response_format = JSON" to your request parameters. The output will look similar to this:

```
{
  "api": "RSAPI",
  "version": "3.8",
  "count": 1,
  "transactions": [
    {
      "transaction_id": "000000924339",
      "account_id": "000000000001",
      "authorization_date": "2018-07-30 10:38:12",
      "tender_type": "CARD",
      "transaction_type": "SALE",
      "keyed": "1",
      "swiped": "0",
      "entry_mode": "keyed",
      "transaction_amount": "176.58",
      "name": "John Smith",
      "card_brand": "VISA",
      "last4": "1111",
      "card_expiration": "0315",
      "description": "Test transaction",
      "user_data": "",
      "authorization_msg": "APPROVED",
      "authorization_code": "097468",
      "avs_response": "A",
      "cvv2_response": "N",
      "ip_address": "",
      "cashier": "QSAPI 3.8",
      "street_address1": "123 Main St",
      "city": "Anytown",
      "state": "NY",
      "zip": "10101",
      "country": "",
      "phone": "212-555-1212",
      "email": "",
      "group": "",
      "refund_id": "",
      "custom_id": "",
      "action_date": "",
      "noc_data": "",
      "recurring_id": 0
    }
  ]
}
```


XML formatted example response

To receive the report in XML simply add “response_format = XML” to your request parameters. The output will look similar to this:

```
<transaction>
  <transaction_id>000000924339</transaction_id>
  <account_id>000000000001</account_id>
  <authorization_date>2018-07-30 10:38:12</authorization_date>
  <tender_type>CARD</tender_type>
  <transaction_type>SALE</transaction_type>
  <keyed>1</keyed>
  <swiped>0</swiped>
  <entry_mode>keyed</entry_mode>
  <transaction_amount>176.58</transaction_amount>
  <name>John Smith</name>
  <card_brand>VISA</card_brand>
  <last4>1111</last4>
  <card_expiration>0315</card_expiration>
  <description>'Test transaction'</description>
  <user_data/>
  <authorization_msg>APPROVED</authorization_msg>
  <authorization_code>097468</authorization_code>
  <avs_response>A</avs_response>
  <cvv2_response>N</cvv2_response>
  <ip_address/>
  <cashier>QSAPI 3.8</cashier>
  <street_address1>'123 Main St'</street_address1>
  <city>Anytown</city>
  <state>NY</state>
  <zip>10101</zip>
  <country/>
  <phone>212-555-1212</phone>
  <email/>
  <group/>
  <refund_id/>
  <custom_id/>
  <action_date/>
  <noc_data/>
  <recurring_id>0</recurring_id>
</transaction>
```

Appendix: Transaction Responses and Messages

The following information is for use by Bluefin merchants as well as programmers to help understand transaction related terms you may find within the Bluefin gateway and processing tunnels.

The list is comprised of three sections: Transaction Origins, Transaction Authorization Response Codes, and Transaction Status Codes, all listed in detail below. While we try to keep this information up to date and accurate, some information may change from time to time and may not be immediately updated here. If you receive an error response and are unclear of the error response, or believe them to be in error, contact Bluefin support at support@bluefin.com

Transaction Origins:

Error Code	Description
AUTO-REFUND	System initiated refund
V-TERM	Virtual Terminal, Manual action by a logged in user
N2.SIGNUP	Hosted payment form, membership signup (Interactive Mode 2.2)
N2.PURCHASE	Hosted payment form, one time purchase (Interactive Mode 2.2)
ND2.TRANS	Direct Mode v2.1 (signup or one time purchase)
ND3.TRANS	Direct Mode v3.1 (one time purchase)
ND3.SIGNUP	Direct Mode v3.1 (membership signup transaction)
RAD.TEST	System Generated Test (Only on test/demo accounts)
BA.TRANS	Batch Upload Transactions
RETRY	Rare, manual internal retry
RECURRING	A re-bill initiated by Bluefin automatically

Transaction Authorization Response Codes:

Authorization Response Codes:

Error Code	Description
DECLINED	This is the most common, generic decline message. It is an actual decline from the issuing bank and details are not provided.

AVS/CVV2 Authorization Responses:

Error Code	Description
BAD ADDRESS	The AVS response received was not in the list of acceptable AVS codes. The transaction has been aborted.
CVV2 MISMATCH	The CVV2/CID response received was not in the list of acceptable CVV2 codes. The transaction has been aborted.

FraudFirewall Authorization Responses:

When FraudFirewall is enabled, transactions that fail a FraudFirewall checkpoint will be rejected prior to an attempt to authorize the transaction.

Error Code	Description
A/DECLINED	The transaction exceeded the traffic limits.
B/DECLINED	The transaction contains information found in the blacklist or high-risk country list.
C/DECLINED	The country of origin found in the high risk country list.
E/DECLINED	The email address in the transaction is not valid.
I/DECLINED	The country of origin did not match the country specified by card holder (IP, Card, Address)
J/DECLINED	The transaction contains profane or otherwise suspicious information. This check is disabled in the virtual terminal.
L/DECLINED	The transaction failed to pass US Location Verification.
R/DECLINED	The transaction contained a high-risk country or came from an anonymous domain.

Traffic Limit Authorization Responses:

Transactions that exceed a traffic limit set by FraudFirewall settings or your preset quotas will be rejected prior to an attempt to authorize the transaction.

Error Code	Description
A/QUOTA EXCEEDED	The transaction exceeded the maximum dollar volume limit per card.
C/QUOTA EXCEEDED	The transaction exceeded the maximum number of credits/refunds allowed within a time period (contact Bluefin Customer Service).
M/QUOTA EXCEEDED	The transaction exceeded the maximum per transaction amount limit.
R/QUOTA EXCEEDED	The transaction exceeded the maximum dollar volume of credits/refunds allowed within a time period (contact Bluefin Customer Service).
S/QUOTA EXCEEDED	The transaction exceeded the maximum number of sales allowed within a time period (contact Bluefin Customer Service).

Bank Authorization Responses:

Some banks may return authorization responses which do not clearly state the nature of the response. Below are the obvious, and not so obvious, responses we have on file:

Error Code	Description
ACCT FROZEN	Account frozen, cannot transfer funds
APPROVED	Transaction authorized
CALL 01	Refer to issuer
CALL 02	Refer to issuer, special condition
NO REPLY 28	File is temporarily unavailable
NO REPLY 91	Issuer switch is unavailable
HOLD-CALL 04	Pick up card
HOLD-CALL 07	Pick up card, special condition

HOLD-CALL 41	Pick up card, lost
HOLD-CALL 43	Pick up card, stolen
ACCT LENGTH ERR	EA Verification error
ALREADY REVERSED 79	Already reversed at switch
AMOUNT ERROR 13	Invalid Amount
CANT VERIFY PIN 83	Cannot verify PIN
CANT VERIFY PIN 86	Cannot verify PIN
CARD NO. ERROR 14	Invalid card number
CASHBACK NOT APP 82	Cashback limit exceeded
CASHBACK NOT AVL N3	Cashback service not available
CHECK DIGIT ERR	EB Verification error
CID FORMAT ERROR	EC Verification error
CVV2 MISMATCH N7	CVV2 data does not match
DATE ERROR 80	Invalid Date
DECLINE 02	Force Transaction with Voice Authorization
DECLINE 05	Do not honor
DECLINE 51	Insufficient funds
DECLINE N4	Exceeds issuer withdrawal limit
DECLINE 61	Exceeds withdrawal limit
DECLINE 62	Invalid service code, restricted
DECLINE 65	Activity limit exceeded
DECLINE 93	Violation, cannot complete
DECLINE AVS 06	Denied for AVS
ENCRYPTION ERROR 81	Cryptographic error
ERROR CODE 98	Unknown Error
EXPIRED CARD 54	Expired card
FAILURE CV	Card Type Verification Error (Card Type not accepted)
Failure HV	Configuration error when your account was created
GENERAL ERROR 98	Potentially NULL value passed, experienced when CVV2 left blank
INV MERCH NUM 19	Invalid Merchant ID
Invalid Account Number	Invalid account number length or format
INVALID BANK ACT	Invalid bank account
INVALID ROUTING 92	Destination not found
INVALID TRANS 12	Invalid transaction (may be invalid account #)
NO ACCOUNT 78	No account
No Acct/Cannot Locate	No account exists
NO ACTION TAKEN 21	Unable to back out transaction
NO ACTION TAKEN 76	Unable to locate, no match
NO ACTION TAKEN 77	Inconsistent data, reversal or repeat
NO CHECK ACCOUNT 52	No checking account
NO CREDIT ACCT 39	No credit account

NO SAVE ACCOUNT 53	No savings account
NO SUCH ISSUER 15	No such issuer
OFFLINE RETURN	Refund of check or credit transaction
PIN EXCEEDED 75	PIN tries exceeded
RE ENTER 19	Re-enter transaction
SEC VIOLATION 63	Security violation
SERV NOT ALLOWED 57	Transaction not permitted - card
SERV NOT ALLOWED 58	Transaction not permitted - term
SYSTEM ERROR 96	System malfunction
TERM ID ERROR 03	Invalid Merchant ID
WRONG PIN 55	Incorrect PIN
XXXXXXXXXXXXXXXXXX XX	Undefined response

Transaction Status Codes:

Error Code	Description
ACH.CB/ISSUED	Check chargeback has been issued
ACH/FAILED	Check did not pass fraud scrubbing or was invalid and not submitted to banking network.
ACH/RETURN	Check returned after being submitted to banking network
ACH/ISSUED	Check issued but not yet submitted to banking network, can be cancelled
ACH/PENDING	Check issued, submitted to banking network, and awaiting approval
ACH/OK	Check issued, submitted to banking network, approved and funds transferred
ACH/REFUNDED	Check transaction has been refunded, refund transaction is separate
AUTH/FAILED	Auth only attempt declined
AUTH/OK	Auth only attempt approved
CREDIT/OPEN	Credit to card which was not originally charged, not yet settled, can NOT be voided
CREDIT/OK	Credit to card which was not originally charged, settled, funds transferred
REFUND/FAILED	Refund declined, may have invalid information
REFUND/OK	Refund approved, funds transferred
REFUND/OPEN	Refund [Credit Card] issued, not yet settled, can NOT be voided
REFUND/ISSUED	Refund [Check] issued, not yet submitted to banking network
REFUND/PENDING	Refund [Check] issued, submitted to banking network, and awaiting approval
S/REFUNDED	Credit Card transaction has been refunded; this is the original sale, not the actual refund transaction
SALE/FAILED	Sale declined
SALE/NOP	Status set manually for problematic transaction (extremely rare)
SALE/ERROR	Status set manually for problematic transaction (extremely rare)
SALE/OK	Sale approved, settled, and funds transferred
SALE/OPEN	Sale approved, not yet settled, can be voided
SALE/CB	Previously approved transaction that has been charged back by the customer and imported into our system. Previously transaction would have been a SALE/OK but now has changed status.

EMV Data Overview

Request

The Point Of Sale (POS) device posts to QSAPI the following data:

Variable Name	Max	Type	Req'd	Description
account_id	12	Numeric	Yes	The Payconex account identification number that you are issued after your account has been setup.
api_accesskey	32	Alphanumeric	Yes	The secret key that you will be provided when your Payconex account is set up and when you have requested access to QSAPI.
timestamp	19	YYYY-MM-DD HH:MM:SS	No	If used, MUST be included in a hash for authenticated transactions. See "Appendix: Using HASH for Authenticated Transactions"
tender_type	n/a	Enumerated	Yes	The payment tender type that you are submitting. The following enumerated values are allowed: CARD: credit, debit, and check cards EBT: Electronic Benefits Transfer (Elavon only) DEBIT: PIN Debit card only (Elavon/Omaha/North)
transaction_type	n/a	Enumerated	Yes	The type of transaction you are requesting, with these enumerated values allowed: SALE: authorizes the funds on the card and flags the transaction to be captured for settlement at the next settlement time. REFUND: refunds a previous sale. If the transaction has not yet been settled, then this results in a void. Otherwise, for Cards only, it results in a credit back on the card. ACH transactions can't be refunded once they are submitted for settlement (NOTE: For actual transaction settlement times, contact Bluefin support). You can specify an amount less than the original sale amount. Requires token_id. BALANCE: request account balance on a EBT/Debit card (Elavon only).
transaction_amount	9	Numeric with decimal	Yes	This is the dollar (or other currency) amount of the transaction. Only numbers and a single decimal are allowed. Commas are not allowed. The maximum amount is 999999.99. That is 1 cent less than 1 million. This is because the decimal is counted in the max size. Values with no decimal and no cents are allowed. Values with only a single number after the decimal are allowed and will be assumed to have a trailing 0.
transaction_description	65K	Character	No	A description of the payment. This is an open field. If emails are sent to the customer or merchant, this will show in the "Description:" field. You may use this to send any information

				that you wish. It can store up to 65,000 characters.
card_tracks	?	Character	Yes/No	The characters from the full, un-modified payload. Must be well formed XML string, see “ EMV Tag Info ” section below (request column)
pin	?	Alphanumeric	No	The encrypted PIN Block portion for PIN debit or PIN EBT transactions. It must be obtained from a PCI PTS/PED Certified device that is injected by Bluefin’s Encryption Service Organization or Key Injection Facility (KIF). This PIN Block may never be stored for any reason.
ksn	?	Alphanumeric	No	The Key Sequence Number (DUKPT) portion for PIN debit, PIN EBT, or EMV transactions. It must be obtained from a PCI PTS/PED Certified device that is injected by Bluefin’s Encryption Service Organization or Key Injection Facility (KIF). The KSN may never be stored for any reason.
ebt_type	n/a	Enumerated	Yes/No	For EBT (Electronic Benefits Transfer), the type can be as follows: FOOD : this is for a food sale CASH : this is for a cash sale Required if tender_type=EBT
customer_data	65K	Character	No	Any customer data relevant to the transaction (for semi-integrated only)
cashier *	100	Alphanumeric	No	The cashier that created the original transaction.
token_id	12	Numeric	No	12 digit transaction_id of a previous transaction, when doing a refund. Please see the SLAPI documentation for more information.

Response

The response will be the same as a QSAPI response (see above section on QSAPI Response) with a few differences.

EMV transaction response additionally includes processor’ EMV related data (emv_tags). This data is to communicate to the card’s chip for update.

emv – contains an XML string of chip related processor response – see “**EMV Tag Info**” section below (request column, ones marked with an ‘X’)

Note: AppVer parameter of Dvc tag will indicate the following methods:

AppVer="1.0" (Swipe)

AppVer="2.0" (Contactless)

AppVer="2.1" (EMV and Contactless)

Example Request (Credit Card, EMV Contact)

```
card_tracks=' <PAX Ver="2.0"><Dvc App="Bluefin" AppVer="2.1" DvcType="S500"
DvcSN="65000111" Entry="EMV"></Dvc><ICC _9f26="000000002000" _9f27="80"
_9f37="549F8B9C" _9f36="0001" _95="0480000000" _9c="00" _82="5D80"
```



```

_9f33="E0F8C8" _9f34="5E0300" _9a="160519" _5f2a="840" _9f02="000000002000"
_9f03="000000000000" _9f35="22" _5f34="00" _5f24="161130"
_9f10="06020103A00000" _9f1a="840"></ICC><Card CEncode="0"
ETrk1="670219E5024E11D06EF137DC8F35C89FFAD0FBCC82BE30A29FE152B9F5183A49AF8A5B
DF048757C32147F771F099410BDC66035DC7CC0DD3C2E73EFE4CFE6D90"
ETrk2="90A52091B09589BE7195CF3787E51DCB1314C7B11BC99028BE4905149DC674F295C687
F7D06A3D93" CDataKSN="62994912410000800051" CHolder="FDCS/TEST CHECK CARD"
EFormat="0"></Card><Addr></Addr><Tran TranType="CREDIT"></Tran></PAX>'

```

Example Response (EMV related data)

```

...
emv_tags=<icc _9f36="000B" _5f34="01" EMV_Key_Date="06152016" />
...

```

EMV Tag Info:

A list of EMV tags with requirements, lengths and sample data to allow one to build or understand the EMV payloads. Also note if the tag value is included in the response.

TagID	Req'd	Length	Type	Sample Value	Notes	Request	Response
9F26	Yes	16	Hex	B3CD6686EFC20095	Value returned by the chip (ICC) in response to a "Generate AC" command.	X	
9F27		2	Hex	80	The Cryptogram Information Data	X	
9F37	Yes	8	Hex	E180229B	Value is used to provide variability and uniqueness to the generation of a cryptogram.	X	
9F36		4	Hex	000B	The Application Transaction Counter (ATC)	X	X
95	Yes	10	Hex	8000008000	Status of the different functions as seen from the terminal.	X	
9C		2	Numeric	00	Type of financial transaction, represented by the first two digits of ot the ISO 8583	X	
82		4	Hex	1C00	Application interchange profile -- capabilities of the card to support specific function within the app.	X	

9F33	Yes	6	Hex	E0B8C8	Value that indicates the card data input, CVM, and security capabilities of the terminal.	X	
9F34	Yes	6	Hex	410302	Results of the last Card Verification Method (CVM) performed.	X	
9A		6	Numeric	100812	Local date that the transaction was authorized (YYMMDD).	X	
5F2A		3	Numeric	124	Currency code of the transaction according to the ISO 4217 standard.	X	
9F02		12	Numeric	000000001000	Authorized amount of the transaction (excluding adjustments).	X	
9F03		12	Numeric	000000000000	Value is used to indicate a secondary "Cashback" amount associated with the transaction.	X	
9F35	Yes	2	Numeric	22	Terminal Type	X	
5F34		2	Numeric	01	The Application PAN Sequence Number (CSN)	X	X
5F24	Yes	6	Numeric	151231	The Application Expiration Date (YYMMDD)	X	
9F10		64	Hex	06010A03A48800	The Issuer Application Data	X	
9F1A		3	Numeric	840	The Terminal Country Code	X	
8A		2	Alpha		The Authorization Response Code	X	
9F6E		8	Hex		Form Factor Indicator	X	
9F7C		64	Hex		The Customer Exclusive Data	X	
9F6E		64	Hex		Third Party Data	X	
84		32	Hex		Dedicated File Name (Application Identifier)	X	

71		256	Hex		The Card Issuer Script Command to be submitted to the Chip card		X
72		256	Hex		The Card Issuer Script Command to be submitted to the Chip card		X
91		32	Hex		The Issuer Authentication Data		X
		8	Numeric	03092016	The date of the last Host EMV Key Update in MMDDYYYY format		X

Key exchange request

When the POS (Point Of Sale) gets the EMV_Key_Date field back from Elavon in a response message, the POS needs to compare that date to the date the POS has stored. If the host date is newer, the POS needs to send to Bluefin an "EMV Key Exchange Request".

In order to do this, the POS posts to QSAPI the following:

Variable Name	Max	Type	Req'd	Description
account_id	12	Numeric	Yes	The Payconex account identification number that you are issued after your account has been setup.
api_accesskey	32	Alphanumeric	Yes	The secret key that you will be provided when your Payconex account is set up and when you have requested access to QSAPI.
transaction_type	n/a	Enumerated	Yes	The type of transaction you are requesting, with enumerated value of: KEYEXCHANGE
custom_id			Yes	The value returned in the last response for the " record_number " attribute. For very 1st Key Exchange Request this value should be 0.

Key Exchange Response

Key exchange response includes

transaction_timestamp

error

error_code

error_message

emv_tags

Note: emv_tags value is XML string formatted as follow:

<icc attributes />

See section below entitled "Key Exchange Attributes" for more detail on returned attributes.

Example Request

account_id=180000000045&api_accesskey=5f6964d6babeb1d927470931dc68d83c
&transaction_type=KEYEXCHANGE&custom_id=0

Example Response

transaction_timestamp=2016-07-18+12%3A15%3A01
&error=0&error_code=0&error_message=&emv_tags= <icc record_number="1"
additional_key_available="1" data_type="01" rid="A000000003" pki="94" hash_id="01"
digital_signature_id="01"
public_key="ACD2B12302EE644F3F835ABD1FC7A6F62CCE48FFEC622AA8EF062BEF6FB8BA8BC68BBF6A
B5870EED579BC3973E121303D34841A796D6DCBC41DBF9E52C4609795C0CCF7EE86FA1D5CB041071E
D2C51D2202F63F1156C58A92D38BC60BDF424E1776E2BC9648078A03B36FB554375FC53D57C73F5160
EA59F3AFC5398EC7B67758D65C9BFF7828B6B82D4BE124A416AB7301914311EA462C19F771F31B3B57
336000DFF732D3B83DE07052D730354D297BEC72871DCCF0E193F171ABA27EE464C6A97690943D59B

DABB2A27EB71CEEBAFA1176046478FD62FEC452D5CA393296530AA3F41927ADFE434A2DF2AE3054F8840657A26E0FC617" exponent="03" check_sum="C4A3C43CCF87327D136B804160E47D43B60E6E0F" ca_public_key_length="F8" ca_public_key_exp_date="221231" emv_key_date="06152016"/>

Key Exchange Attributes

A list of the Key Exchange Attributes that can be returned in a response as follows:

AttributeID	Req'd	Length	Type	Sample Value	Notes
Record_Number	Optional	4	Numeric	0	Next Request Record Number.
Additional_Key_Available	Yes	1	Numeric	1	0 = No more keys to download; 1 = there are more keys available for download.
Data_Type		2	Numeric	01	Type of EMV key. 01 = CA Public Key
RID	Yes	10	Alpha	A000000003	Registered Application Provider Identifier.
PKI	Yes	2	Hex	94	CA Public Key Index
Hash_ID		2	Alpha	01	Hash Algorithm Identifier (01 = default).
Digital_Signature_ID		2	Alpha	01	Digital Signature Algorithm Identifier (01 = default)
Public_Key	Yes	496	Hex	ACD2B12302...	Value of the Public Key
Exponent	Yes	6	Alpha	03	Public Key Exponent Value
Check_Sum	Yes	40	Hex	C4A3C43CCF...	CA Public Key Check Sum
CA_Public_Key_Length	Yes	2	Hex	F8	Length of the Public Key in hex
CA_Public_Key_Exp_Date	Yes	6	Numeric	221231	CA Public Key Expiration Date in YYMMDD format
EMV_Key_Date		8	Numeric	06152016	The date of the last Host EMV Key Update in MMDDYYYY format