



[evopayments.com](https://evopayments.com)



*Simplifying Payments* AROUND THE GLOBE  
150+ CURRENCIES ACROSS 50 MARKETS WORLDWIDE

# DiamondCloud

## API Integration Guide

Updated: 7/20/2022

EVO PROPERTY

# Table of Contents

<b>DIAMONDCLOUD .....</b>	<b>1</b>	CONVERT CHECKS TO JSON .....	15
<b>WHAT IS DIAMONDCLOUD? .....</b>	<b>2</b>	Required Field Types .....	15
<b>ISV CONSIDERATIONS .....</b>	<b>2</b>	POST CHECK TO CLOUD .....	16
PROCESSING INDUSTRY TYPES .....	2	POST.....	16
ANDROID PAYMENT TERMINAL TYPES .....	2	Response.....	16
DIFFERENCES BASED ON TERMINAL/APP .....	3	GET STATUS OF CHECKS .....	17
Sending Tip Amounts.....	3	GET .....	17
Sending Cashback Amounts .....	3	Response.....	17
TRANSACTION RESULTS HANDLING .....	3	PARTIALLY APPROVED TRANSACTION HANDLING .....	19
<b>RETAIL INTEGRATION .....</b>	<b>4</b>	OTHER REST API CALLS .....	19
REQUEST FORMATTING .....	4	Remove/Cancel Check.....	19
List of Transaction Request Commands .....	4	Search for Unpaid Checks .....	20
Request Responses .....	5	Response.....	20
Request Transaction Status - Details .....	5	ERROR HANDLING .....	21
TRANSACTION EXAMPLES .....	7	Transaction is Pending Processing.....	21
Sale – Credit/Debit.....	7	Transaction Failed Offline.....	21
Reverse/Void – Credit.....	7	Transaction Failed Online.....	21
Return – Credit.....	8	Transaction Cancelled At Terminal .....	21
BatchClose – Close Payment Terminal .....	8	EXAMPLE - CHECK CONVERTED FROM XML TO JSON .....	22
Batch.....	8	TESTING ENDPOINT .....	23
Optional Transaction Types .....	9	PRODUCTION ENDPOINT .....	23
PARTIALLY APPROVED TRANSACTION HANDLING .....	14		
<b>HOSPITALITY INTEGRATION.....</b>	<b>15</b>		

# What is DiamondCloud?

In short, the DiamondCloud provides a simple, easy, and painless integration to the latest certified Android payment terminals for ISV developers. The cloud offers POS applications to initiate transaction processing on the Android Payment Terminals with a simplified and hardware manufacturer agnostic RESTful API. The POS can trigger actions directly on the payment terminal via the cloud or upload bills to the cloud that will appear as selectable items on the Android Payment Terminal which creates the easiest Pay At Table processing experience on the market.

## ISV Considerations

To get started, consider the different integration options and tailor the integration to your solutions needs and target customers.

### Processing Types

- **Instant/Active** (*retail/counter pay*): In the Instant configuration your POS will send one transaction request to the cloud and this will immediately activate the Android payment to perform the transaction. For example, sending a \$5 Sale request will immediately prompt for card entry on the Android Payment Terminal.
  - **When to use Instant processing**
    - Customer facing payment experiences such as payments at the counter and multilane checkouts. The Android payment terminal can optionally prompt for Cash back or Tip entry.
    - Back office non-customer facing payment experiences can also be performed send a transaction request directly to the Android payment terminal and key a card or perform other transactions without a customer interaction.
- **Queued Bills** (*PayAtTable*): In the queued configuration, your POS may send several checks to the cloud and the Android payment terminal will display all the open checks for a location on the device screen so that the staff can select any of the open checks and perform the payment whenever the check needs to be closed directly on the Android payment device.
  - **When to use Queued Bills Processing**
    - Customer facing payment experiences away from the POS workstation where orders are entered. Table Service Restaurants, Salons, and Spas are all ideal for this type of interaction and payment experience.

### Android Payment Terminal Types

Once integrated, your solution will be able to offer merchants any of the below payment devices that suits their needs.

- Portable – NEXGO N5, PAX A920, or PAX A77
- Countertop – PAX A80, A35
- Multilane – PAX Aries8

## Differences based on Terminal/App

### Sending Tip Amounts

- NEXGO – when an ISV sends a sale command with a tip included, the tip amount will prompt on the device and can be changed on the NEXGO device at this prompt.
- PAX - when an ISV sends a sale command with a tip included, the tip amount will NOT prompt on the device and CANNOT be changed on the PAX device.

### Sending Cashback Amounts

- NEXGO – when an ISV sends a sale command with a cashback amount included, the cashback amount will prompt on the device and can be changed on the NEXGO device at this prompt.
- PAX - when an ISV sends a sale command with a cashback included, the cashback amount will NOT prompt on the device and CANNOT be changed on the PAX device.

## Transaction Results Handling

After a transaction is performed by the Android payment terminal, the response is sent to the DiamondCloud. The DiamondCloud solution offers 2 ways of handling transaction results/responses getting back to the integration solution.

- **REST API** – the POS can perform GET API calls to 'pull' the transaction status results. The POS would do this every 3-5 seconds until the transaction results are available.
- **Callback URL** – the DiamondCloud service can write all the transaction responses for merchants to a callback URL (a webpage the integrator hosts that the DiamondCloud log's responses can be written to).
  - NOTE: In order for an integration to match transaction responses to requests from many different POS endpoints and merchants, the cloud will add source request data to the responses sent to the Callback URL.



```
{ "IsvId": "8FB7B0CB2AE242B3BC8F5ED15DDEFAC0",
  "InvoiceId": "",
  "CloudTxnId": "85DNQ3XKZBK",
  "Device": "87654321000401_1",
  "Response":{ "Normalized Response fields" }}
```

# Instant Integration



**Note:** Uploading a JSON of a check is not expected in the instant processing environment. The transactions will use a simple REST API URL string to trigger a transaction on the payment terminal immediately. Open checks will not be queued or contain line item detail.

## Request Formatting

URL	//Cloud URL	/tomcat	/command	/POS_ID	/transaction
<b>What it is</b>	The URL of the Cloud environment being called	Web services host – always tomcat	Type of request being sent. /command	Assigned ID of the POS and Payment Terminal pair- 16 digits	Type of transaction command being sent. sale, return, void, etc.
<b>Example</b>	//qr.simplatabcloud.com	/tomcat	/command	/1234567890123456	/sale

## List of Transaction Request Commands

Command	Details
sale	Trigger a sale transaction to occur on the payment terminal. Send amount in the JSON and optionally include, tip amount, payment token, and cash back.
void	Trigger a void of a sale transaction on the payment terminal. Send the 32 character transaction id from the sale in the JSON data.
return	Trigger a return transaction on the payment terminal. Send amount in the JSON and optionally include the payment token.
batchClose	Trigger a batch close to occur on the payment terminal.
tip	Trigger a tip adjustment for a previous sale transaction. Send original amount, the tip amount, and the 32 character transaction id from the sale in the JSON data.
auth	Trigger an auth (pre-auth) transaction on the payment terminal. Send amount in the JSON and optionally include the payment token.
capture	Trigger a capture (post-auth) transaction on the payment terminal. Send the amount and the 32 character transaction id from the auth in the JSON data and optionally include a tip amount
giftActivate	Trigger a gift card activation transaction on the payment terminal. Send the amount in the JSON data.
giftReload	Trigger a gift card reload (add value) on the payment terminal. Send the amount in the JSON data.
giftRedeem	Trigger a gift card redeem (sale/purchase) on the payment terminal. Send the amount in the JSON data.
giftBalance	Trigger a gift card balance inquiry on the payment terminal. No added fields needed.
ebtFood	Trigger an EBT Food Stamp sale on the payment terminal. Send the amount in the JSON data.
ebtCash	Trigger an EBT Cash Benefit sale on the payment terminal. Send the amount in the JSON data.
ebtBalance	Trigger an EBT balance inquiry on the payment terminal. No added fields needed.
ebtReturn	Trigger an EBT return on the payment terminal. Send the amount in the JSON data.
test	Trigger a test message to appear on the payment terminal.

## Request Responses

Upon successful request submission, the Cloud will respond with an OK message plus an ID value. This plaintext ID value is the cloud ID which is a unique transaction identifier on the cloud for the request made.

Example:

11ABC22DEF3



The POS should store this ID and use it to query the cloud for the transaction results. The instant Cloud ID is 11 characters, the queued Cloud ID is 32 characters.

## Request Transaction Status - Details

To retrieve the results of transaction requests sent to the payment terminal, the POS will send the command details api call.

The POS can parse, store and use this info later:

- Print customer receipts.
- Perform transactions like adjustments and void transactions using the original cloud transaction id value.
- Store the 'paymentToken' for use later on the customer's account online and subscriptions.

GET /command/{POS-ID}/details/{Cloud Transaction ID}

Response:

```
{
  "merchantId" : "123456789012",
  "merchantName" : "Cloud Pizza",
  "transactionStatus" : "",
  "transactionType" : "AUTH",
  "maskedCard" : "411111xxxxxx4321",
  "cardBrand" : "Visa",
  "entryMethod" : "Chip",
  "dateTime" : "2021-11-24T15:22:33.756",
  "batchNumber" : "0123",
  "transactionId" : "95f1959dd7c040e8ba0afe59d6fb725d",
  "approvalCode" : "A12345",
  "requestAmount" : "15.00",
  "approvedAmount" : "15.00",
  "partialApproval" : "false",
  "tip" : "0.00",
  "cashback" : "0.00",
  "tax" : "",
  "surchargeFee" : "0.00",
  "ebtCashBalance" : "0.00",
  "ebtFoodBalance" : "0.00",
  "giftCardBalance" : "",
  "avsResult" : "",
  "cvdResult" : "",
  "aid" : "A000000025010801",
  "tvr" : "08008000",
  "tsi" : "E800",
  "paymentToken" : "95f1959d-d7c0-40e8-ba0a-fe59d6fb725d1759d7af-306c-43cb-823b-b4391adb54f9"
}
```



If the 'transactionStatus' returns a value of 'pending', the payment terminal has not completed processing the customers card transaction. The POS should repeat the command Details request every 3-5 seconds until a completion is returned or the transaction is cancelled.



**Note:** if Payment Tokens are being supported, be aware the token is a 72 character value.

Request Field	Description	Values	Field Type
CLOUD_URL	DiamondCloud URL	URL	Static URL
POS_ID	EVO assigned number indicating the POS Merchant & Lane integration.	Specific to each ISV, Merchant and Lane	16 Char ASCII
id	Cloud ID returned after push function is performed.	Variable	11-32 Char ASCII
Response Field	Description	Values	Field Type
merchantId	Merchants processing ID	Varied by merchant	Up to 16 digits
merchantName	Merchant Name	Varied by merchant	Variable ASCII
transactionStatus	Transaction response status	APPROVED or DECLINED	Variable ASCII
transactionType	Transaction type performed	AUTH, CAPTURE, RETURN, VOID	Variable alphanumeric
maskedCard	Masked card PAN with last 4 digits	X's followed by last 4 of card.	Variable alphanumeric
cardBrand	Card brand used in the transaction	Visa, MasterCard, AmericanExpress, Discover, Debit, EBT	Variable alphanumeric
entryMethod	Card account entry method	Keyed, swipe, chip, contactless	Variable alphanumeric
dateTime	Date and Time of transaction (UTC)	YYYY-MM-DDTHH:MM:SS.SSS	YYYY-MM-DDTHH:MM:SS.SSS
batchNumber	Current batch number	4 digit value	Numeric
transactionId	Transaction ID generated during authorization	32 character value	32 alphanumeric char
approvalCode	Transaction approval code from issuer	Variable	Variable alphanumeric
requestAmount	Amount requested for payment.	00.01 – 999999.99	Variable numeric w/decimal
approvedAmount	Amount approved	00.01 – 999999.99	Variable numeric w/decimal
partialApproval	Indicator of partial approval	true or false	Alphanumeric
tip	Tip amount added by cardholder	00.01 – 999999.99	Variable numeric w/decimal
cashback	Cashback amount added by cardholder	00.01 – 999999.99	Variable numeric w/decimal
tax	Tax amount processed	00.01 – 999999.99	Variable numeric w/decimal
surchargeFee	Surcharge fee processed	00.01 – 999999.99	Variable numeric w/decimal
ebtCashBalance	EBT cash benefit balance	00.01 – 999999.99	Variable numeric w/decimal
ebtFoodBalance	EBT food stamp balance	00.01 – 999999.99	Variable numeric w/decimal
giftCardBalance	Gift card remaining balance	00.01 – 999999.99	Variable numeric w/decimal
avsResult	AVS result value if entered	Result code and text	Variable alphanumeric
cvdResult	CVD/CVV result value if entered	Result code and text	Variable alphanumeric
aid	Chip card Application AID	Card application	Variable alphanumeric
tvr	Chip card TVR	TVR value	10 digit numeric
tsi	Chip card TSI	TSI value	4 digit numeric
CardholderVerificationMethod	Not in use - RFU	Not in use	NULL
paymentToken	Card account payment token	Token value	72 character variable alphanumeric

# Transaction Examples

## Sale – Credit/Debit

To trigger a Credit or Debit sale on a payment terminal, the POS should send a sale request with an amount. The DiamondCloud will return a transaction invoice id, the POS will need to store this ID and use it for status updates and reconciliation.

### POST

```
POST [https://{Cloud URL}/tomcat/command/{POS_ID}/sale] HTTP/1.1
Accept: application/json
Host: [https://{Cloud URL}/]
Content-Type: application/json
{
  "amount": "1.00",
  "panDatatoken": "",
  "tip_amount": "",
  "cash_back": ""
}
```



If payment tokens are being stored and used, place the token in the panDatatoken field to process the sale using a payment token instead of requiring card entry.

### Response

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2020 21:10:53 GMT
Server: Apache/2.4.18 (Ubuntu)
Status: 200 OK
Content-Length: 0
Keep-Alive: timeout=5, max=100
Content-Type: application/json
cloudID
```

## Reverse/Void – Credit

To trigger a Credit reversal on a payment terminal, the POS should send a void request with the transaction ID from the original sale. The DiamondCloud will return an OK response and trigger the payment terminal to perform a reversal.



**Note:** The reversal must be sent to the payment terminal used for the original sale transaction. If that cannot be done, the POS must send a refund request.

### POST

```
POST [https://{Cloud URL}/tomcat/command/{POS_ID}/void] HTTP/1.1
Accept: application/json
Host: [https://{Cloud URL}/]
Content-Type: application/json
{
  "transaction_id": "[cloud ID]"
}
```



## Response


```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2020 21:10:53 GMT
Server: Apache/2.4.18 (Ubuntu)
Status: 200 OK
Content-Length: 0
Keep-Alive: timeout=5, max=100
Content-Type: application/json
cloudID
```

## Return – Credit

To trigger a Credit Return on a payment terminal, the POS should send a return request with the amount.

## POST

```
POST [https://{Cloud URL}/tomcat/command/{POS_ID}/return] HTTP/1.1
Accept: application/json
Host: [https://{Cloud URL}/]
{
  "amount": "1.00",
  "panDatatoken": ""
}
```

 If payment tokens are being stored and used, place the token in the panDatatoken field to process the return using a payment token instead of requiring card entry.

## Response

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2020 21:10:53 GMT
Server: Apache/2.4.18 (Ubuntu)
Status: 200 OK
Content-Length: 0
Keep-Alive: timeout=5, max=100
Content-Type: application/json
cloudID
```

## BatchClose – Close Payment Terminal Batch

To trigger a batch close on a payment terminal, the POS should send a batchClose request to the DiamondCloud API. This will trigger the Android payment terminal to send a batch close request to the processing system and print a batch report.

## POST

```
POST [https://{Cloud URL}/tomcat/command/{POS_ID}/batchClose] HTTP/1.1
Accept: application/x-www-form-urlencoded
Host: [https://{Cloud URL}/]
```

## Response

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2020 21:10:53 GMT
Server: Apache/2.4.18 (Ubuntu)
Status: 200 OK
Content-Length: 0
Keep-Alive: timeout=5, max=100
Content-Type: application/json
cloudID
```

## Optional Transaction Types

### Tip Adjust – Credit

To trigger a Tip Adjust on a payment terminal, the POS should send a tip request with transaction id, and the amount.

#### POST

```
POST [https://{Cloud URL}/tomcat/command/{POS_ID}/tip] HTTP/1.1
Accept: application/x-www-form-urlencoded
Host: [https://{Cloud URL}/]
{
  "tip_amount": "1.00",
  "amount": "10.00",
  "transaction_id": "[cloud ID]"
}
```

#### Response

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2020 21:10:53 GMT
Server: Apache/2.4.18 (Ubuntu)
Status: 200 OK
Content-Length: 0
Keep-Alive: timeout=5, max=100
Content-Type: application/json
cloudID
```

### Auth – Credit

To trigger a Credit Auth on a payment terminal, the POS should send an auth request with an amount. The DiamondCloud will return a transaction invoice id, the POS will need to store this ID and use it for the capture request.

#### POST

```
POST [https://{Cloud URL}/tomcat/command/{POS_ID}/auth] HTTP/1.1
Accept: application/json
Host: [https://{Cloud URL}/]
{
  "amount": "1.00",
  "panDatatoken": ""
}
```



If payment tokens are being stored and used, place the token in the panDatatoken field to process the auth using a payment token instead of requiring card entry.

#### Response

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2020 21:10:53 GMT
Server: Apache/2.4.18 (Ubuntu)
Status: 200 OK
Content-Length: 0
Keep-Alive: timeout=5, max=100
Content-Type: application/json
cloudID
```

## Capture – Credit

To trigger a Credit Capture on a payment terminal, the POS should send a capture request with transaction invoice id, and the amount.

### POST

```
POST [https://{Cloud URL}/tomcat/command/{POS_ID}/capture] HTTP/1.1
Accept: application/json
Host: [https://{Cloud URL}/]
{
  "amount": "1.00",
  "transaction_id": "[cloud ID]",
  "tip_amount": ""
}
```

### Response

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2020 21:10:53 GMT
Server: Apache/2.4.18 (Ubuntu)
Status: 200 OK
Content-Length: 0
Keep-Alive: timeout=5, max=100
Content-Type: application/json
cloudID
```

## Gift Card Activation

To trigger a Gift Card activation on a payment terminal, the POS should send a giftActivate request with the amount.

### POST

```
POST [https://{Cloud URL}/tomcat/command/{POS_ID}/giftActivate] HTTP/1.1
Accept: application/json
Host: [https://{Cloud URL}/]
{
  "amount": "1.00"
}
```

### Response

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2020 21:10:53 GMT
Server: Apache/2.4.18 (Ubuntu)
Status: 200 OK
Content-Length: 0
Keep-Alive: timeout=5, max=100
Content-Type: application/json
cloudID
```

## Gift Card Reload

To trigger a Gift Card reload on a payment terminal, the POS should send a giftReload request with the amount.

### POST

```
POST [https://{Cloud URL}/tomcat/command/{POS_ID}/giftReload] HTTP/1.1
Accept: application/json
Host: [https://{Cloud URL}/]
{
  "amount": "1.00"
}
```

### Response

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2020 21:10:53 GMT
Server: Apache/2.4.18 (Ubuntu)
Status: 200 OK
Content-Length: 0
Keep-Alive: timeout=5, max=100
Content-Type: application/json
cloudID
```

## Gift Card Redemption

To trigger a Gift Card redemption (sale) on a payment terminal, the POS should send a giftRedeem request with the amount.

### POST

```
POST [https://{Cloud URL}/tomcat/command/{POS_ID}/giftRedeem] HTTP/1.1
Accept: application/json
Host: [https://{Cloud URL}/]
{
  "amount": "1.00"
}
```

### Response

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2020 21:10:53 GMT
Server: Apache/2.4.18 (Ubuntu)
Status: 200 OK
Content-Length: 0
Keep-Alive: timeout=5, max=100
Content-Type: application/json
cloudID
```

## Gift Card Balance Inquiry

To trigger a Gift Card Balance Inquiry on a payment terminal, the POS should send a giftBalance request.

### POST

```
POST [https://{Cloud URL}/tomcat/command/{POS_ID}/giftBalance] HTTP/1.1
Accept: application/x-www-form-urlencoded
Host: [https://{Cloud URL}/]
```

### Response

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2020 21:10:53 GMT
Server: Apache/2.4.18 (Ubuntu)
Status: 200 OK
Content-Length: 0
Keep-Alive: timeout=5, max=100
Content-Type: application/json
cloudID
```

## EBT Food Stamp & Cash Benefit Sale

To trigger an EBT Food Stamp Sale or Cash Benefit Sale on a payment terminal, the POS should send ebtFood or ebtCash request with the amount.

### POST

```
POST [https://{Cloud URL}/tomcat/command/{POS_ID}/ebtFood] HTTP/1.1
Accept: application/json
Host: [https://{Cloud URL}/]
{
  "amount": "1.00"
}
```

### Response

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2020 21:10:53 GMT
Server: Apache/2.4.18 (Ubuntu)
Status: 200 OK
Content-Length: 0
Keep-Alive: timeout=5, max=100
Content-Type: application/json
cloudID
```

## EBT Balance Inquiry

To trigger an EBT Balance Inquiry on a payment terminal, the POS should send an ebtBalance request.

### POST

```
POST [https://{Cloud URL}/tomcat/command/{POS_ID}/ebtBalance] HTTP/1.1
Accept: application/x-www-form-urlencoded
Host: [https://{Cloud URL}/]
```

### Response

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2020 21:10:53 GMT
Server: Apache/2.4.18 (Ubuntu)
Status: 200 OK
Content-Length: 0
Keep-Alive: timeout=5, max=100
Content-Type: application/json
cloudID
```

## EBT Food Stamp Return

To trigger an EBT Food Stamp Return on a payment terminal, the POS should send an ebtReturn request with the amount.

### POST

```
POST [https://{Cloud URL}/tomcat/command/{POS_ID}/ebtReturn] HTTP/1.1
Accept: application/json
Host: [https://{Cloud URL}/]
{
  "amount": "1.00"
}
```

### Response

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2020 21:10:53 GMT
Server: Apache/2.4.18 (Ubuntu)
Status: 200 OK
Content-Length: 0
Keep-Alive: timeout=5, max=100
Content-Type: application/json
cloudID
```

## Pairing/Connection Test

To test the connection to the payment terminal, the POS can send a test transaction to the payment terminal. The Payment terminal will display 'Connect Test Successful' to confirm functionality.

### POST

```
POST [https://{Cloud URL}/tomcat/command/{POS_ID}/test] HTTP/1.1
Accept: application/json
Host: [https://{Cloud URL}/]
{}
```

### Response

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2020 21:10:53 GMT
Server: Apache/2.4.18 (Ubuntu)
Status: 200 OK
Content-Length: 0
Keep-Alive: timeout=5, max=100
Content-Type: text/plain
cloud ID
```

## Partially Approved Transaction Handling

Many Retail and Restaurant merchant category codes are required to accept partially approved transactions. A partially approved transaction will be indicated by the transaction response details in the "partialApproval" field. When this occurs the user at the terminal can opt to continue the transaction and pay the balance with another card. When the remaining balance is paid, the cloud will return the second transaction results using the same Cloud ID. This overwrites the first response where the partialApproval occurred.

- Integrations using the Callback URL for responses will see 2 responses with the same cloud ID returned.
- Integrations using the API to query the Cloud, will see the first response and should continue to query the same Cloud ID for the second transaction to complete.
  - Note if the user/customer opts to reverse the partially approved transaction or pay using another method that does not happen on the payment terminal (ie Cash), the Integration should mark the transaction completed and cease performing a query on the API.

EVO PROPERTY

# Queued Bills Integration

## Convert Checks to JSON

The first step for the POS developer is to convert and provide a receipt, check, or bill in the JSON format. For anyone unfamiliar with JSON, there are free online developer resources where data in XML, HTML, or even CSV can be converted into JSON (see an example in this document).

Every POS system is different, but our machine learning development method will perform all the integration mapping of the receipt fields.

## Required Field Types

Minimally a receipt must have a check number (can also be called order number, ticket number, or receipt number) that is unique to the bill or order as well as a merchant identifier (EVO Merchant number), a subtotal and a tax amount.



**Note:** When designing the point of sale integration, keep in mind that once the check is paid and closed be sure to include a simple way for the server staff to confirm the payment was completed.

EVO PROPERLY



## POST Check to Cloud

Integrate your POS to POST the JSON formatted checks to the DiamondCloud server and receive the id and a payment URL in the response.

Note: the JSON of the check or bill must have the EVO assigned merchant MID.

### POST

```
POST [https://qr.simpletabcloud.com/tomcat/command/{POS_ID}/push] HTTP/1.1
Accept: application/json
Content-Length: xxx
Content-Type: application/json
Host: [CLOUD URL]
{ [JSON OF CHECK] }
```

### Response

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2020 21:10:53 GMT
Server: Apache/2.4.18 (Ubuntu)
Status: 200 OK
Content-Length: 32
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/json
{
  "status": "success",
  "code": "",
  "message": "",
  "timestamp": "2022-01-13 12:04:10",
  "data": {
    "qr": "https://qr.simpletabqr.com/dashboard/#/nfc?tag=XXXXXXXX;3450",
    "id": "[cloud id]"
  }
}
```

Request Field	Description	Values	Field Type
CLOUD_URL	DiamondCloud URL	URL	Static URL
POS_ID	EVO assigned number indicating the POS Merchant & Lane integration.	Specific to each ISV, Merchant and Lane	16 Char ASCII
Response Field	Description	Values	Field Type
id	Cloud ID returned after push function is performed.	Variable	32 Char ASCII
qr	URL value for the POS system to provide to the cardholder as a link or QR code.	Web URL	ASCII

## GET Status of Checks

After receiving presenting the check QR code to the cardholder, the POS will need to send a GET pull from DiamondCloud server to confirm payment status. The POS would need to automatically perform the GET pull of open checks every 15-20 seconds until the check is Paid or Cancelled.

### GET

```
GET [https://qr.simpletabcloud.com/tomcat/command/{POS_ID}/pull/{Cloud_id}] HTTP/1.1
Accept: application/x-www-form-urlencoded
Host: [CLOUD URL]
```

### Response

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2020 21:10:53 GMT
Server: Apache/2.4.18 (Ubuntu)
Status: 200 OK
Content-Length: 4
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/json
{ "merchantId": "87654321000401",
  "merchantName": "PAX Terminal Merchant",
  "transactionStatus": "APPROVED",
  "transactionType": "AUTH",
  "maskedCard": "372848XXXXX3000",
  "cardBrand": "AmericanExpress",
  "entryMethod": "",
  "dateTime": "2021-12-28T13:54:28.137",
  "batchNumber": "0210",
  "transactionId": "99db05f411304bcc850aa8913d9ceb01",
  "approvalCode": "016827",
  "requestAmount": "4.71",
  "approvedAmount": "4.71",
  "partialApproval": "false",
  "tip": "0.00",
  "cashback": "0.00",
  "tax": "",
  "surchargeFee": "0.00",
  "ebtCashBalance": "0.00",
  "ebtFoodBalance": "0.00",
  "giftCardBalance": "",
  "avsResult": "",
  "cvdResult": "",
  "aid": "A000000025010801",
  "tvr": "0000008000",
  "tsi": "null",
  "paymentToken": "99db05f4-1130-4bcc-850a-a8913d9ceb018ef4c58c-e102-430d-b0be-371686ff9d95" }
```

Request Field	Description	Values	Field Type
CLOUD_URL	DiamondCloud URL	URL	Static URL
POS_ID	EVO assigned number indicating the POS Merchant & Lane integration.	Specific to each ISV, Merchant and Lane	16 Char ASCII
id	Cloud ID returned after push function is performed.	Variable	11 Char ASCII
Response Field	Description	Values	Field Type
merchantId	Merchants processing ID	Varied by merchant	Up to 16 digits
merchantName	Merchant Name	Varied by merchant	Variable ASCII
transactionStatus	Transaction response status	APPROVED or DECLINED	Variable ASCII
transactionType	Transaction type performed	AUTH, CAPTURE, RETURN, VOID	Variable alphanumeric
maskedCard	Masked card PAN with last 4 digits	X's followed by last 4 of card.	Variable alphanumeric
cardBrand	Card brand used in the transaction	Visa, MasterCard, AmericanExpress, Discover, Debit, EBT	Variable alphanumeric
entryMethod	Card account entry method	Keyed, swipe, chip, contactless	Variable alphanumeric
dateTime	Date and Time of transaction (UTC)	YYYY-MM-DDTHH:MM:SS.SSS	YYYY-MM-DDTHH:MM:SS.SSS
batchNumber	Current batch number	4 digit value	Numeric
transactionId	Transaction ID generated during authorization	32 character value	32 alphanumeric char
approvalCode	Transaction approval code from issuer	Variable	Variable alphanumeric
requestAmount	Amount requested for payment.	00.01 – 999999.99	Variable numeric w/decimal
approvedAmount	Amount approved	00.01 – 999999.99	Variable numeric w/decimal
partialApproval	Indicator of partial approval	true or false	Alphanumeric
tip	Tip amount added by cardholder	00.01 – 999999.99	Variable numeric w/decimal
cashback	Cashback amount added by cardholder	00.01 – 999999.99	Variable numeric w/decimal
tax	Tax amount processed	00.01 – 999999.99	Variable numeric w/decimal
surchargeFee	Surcharge fee processed	00.01 – 999999.99	Variable numeric w/decimal
ebtCashBalance	EBT cash benefit balance	00.01 – 999999.99	Variable numeric w/decimal
ebtFoodBalance	EBT food stamp balance	00.01 – 999999.99	Variable numeric w/decimal
giftCardBalance	Gift card remaining balance	00.01 – 999999.99	Variable numeric w/decimal
avsResult	AVS result value if entered	Result code and text	Variable alphanumeric
cvdResult	CVD/CVV result value if entered	Result code and text	Variable alphanumeric
aid	Chip card Application AID	Card application	Variable alphanumeric
tvr	Chip card TVR	TVR value	10 digit numeric
tsi	Chip card TSI	TSI value	4 digit numeric
CardholderVerificationMethod	Not in use - RFU	Not in use	NULL
paymentToken	Card account payment token	Token value	72 character variable alphanumeric

## Partially Approved Transaction Handling

Many Retail and Restaurant merchant category codes are required to accept partially approved transactions. A partially approved transaction will be indicated by the transaction response details in the "partialApproval" field. When this occurs the user at the terminal can opt to continue the transaction and pay the balance with another card. When the remaining balance is paid, the cloud will return the second transaction results using the same Cloud ID. This overwrites the first response where the partialApproval occurred.

- Integrations using the Callback URL for responses will see 2 responses with the same cloud ID returned.
- Integrations using the API to query the Cloud, will see the first response and should continue to query the same Cloud ID for the second transaction to complete.
  - Note if the user/customer opts to reverse the partially approved transaction or pay using another method that does not happen on the payment terminal (ie Cash), the Integration should mark the transaction completed and cease performing a query on the API.

## Other REST API Calls

When working with checks in the cloud, a POS will likely need to either remove and cancel checks or reverse a check for one reason or another.

### Remove/Cancel Check

If a check is paid through another method or is no longer valid, the POS should send a remove. The response will be a success/fail. After performing a remove the check will return a snap:cancelled upon performing a /pull status.

#### PUT

```
PUT [https://qr.simpletabcloud.com/tomcat/command/{POS_ID}/remove/{Cloud_id}] HTTP/1.1
Accept: application/x-www-form-urlencoded
Host: [CLOUD URL]
```

#### Response

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2020 21:10:53 GMT
Server: Apache/2.4.18 (Ubuntu)
Status: 200 OK
Content-Length: 0
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
```



**Best practice** - After getting the 200 OK response, perform a GET status to verify that the check is cancelled

## Search for Unpaid Checks

A POS can query the cloud to confirm which checks are open for a particular device. This requires sending the searchbyunpaid request. This request uses the A360 ID of the location (without a lane id or POS ID) to return all the locations open checks.

### PUT

```
PUT [https://qr.simpletabcloud.com/tomcat/searchbyunpaid/{A360 ID}] HTTP/1.1
Accept: application/x-www-form-urlencoded
Host: [CLOUD URL]
```

### Response

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2020 21:10:53 GMT
Server: Apache/2.4.18 (Ubuntu)
Status: 200 OK
Content-Length: 4
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/json
{
  "[cloud ID]": [
    "[cloud ID]|{Check#}",
    "[cloud ID]|{Check#}"
  ]
}
```

## Error Handling

Transactions errors can occur, typical authorization declines will fit the same general format and data provided with successful transactions. This is because a decline occurs at the issuing bank and EVO and the DiamondCloud store a complete record of the transaction attempt. The challenge is when the transaction fails at an earlier step in the process.

Here are some examples of these errors. Always review the transactionStatus field value to determine what the issue may be.

### Transaction is Pending Processing

"transactionStatus": "ERROR:Transaction is not found"

Action: Wait for customer and payment terminal to complete transaction.

### Transaction Failed Offline

*(ie Timeout on Payment Terminal - NEXGO)*

"transactionStatus": "-504 Transaction Failed"

Action: Make sure the customer is ready to present payment and resend the transaction.

### Transaction Failed Online

*(ie Processing at EVO - NEXGO)*

"transactionStatus": "-507 Transaction Failed"

Action: Payment Terminal or Card type is not configured on EVO for this merchant account, contact EVO Support desk.

### Transaction Cancelled At Terminal

*(ie User canceled transaction - NEXGO)*

"transactionStatus": "-501 Transaction Failed",

Action: Confirm cancellation reason, resend the transaction.

## Example - Check Converted from XML to JSON

### Sample Printed Check

Date: 2/7/2020      Time: 7:00 pm  
Check: 1469      Server: 2006

Seat 1	
Sandwich	\$5.99
Soda	\$2.49
Seat 2	
Salad	\$4.99
Subtotal:	
	\$13.47
Tax:	\$1.08
Total:	\$14.55

### XML Data

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
  <localtime>2020-02-07T21:07:00.000</localtime>
  <receipt_id>1469</receipt_id>
  <sale_type/>
  <taxes>
    <name>Tax</name>
    <value>1.08</value>
  </taxes>
  <staff>
    <id>2006</id>
    <has_original_id>true</has_original_id>
  </staff>
  <products>
    <seat>1</seat>
    <quantity>1</quantity>
    <discounts/>
    <price>5.99</price>
    <name>Sandwich</name>
    <modifiers/>
  </products>
  <products>
    <seat>1</seat>
    <quantity>1</quantity>
    <discounts/>
    <price>2.49</price>
    <name>Soda</name>
    <modifiers/>
  </products>
  <products>
    <seat>2</seat>
    <quantity>1</quantity>
    <discounts/>
    <price>4.99</price>
    <name>Salad</name>
    <modifiers/>
  </products>
  <consumer_id>11-1</consumer_id>
  <payment_methods/>
  <shop_id>1535509869033384</shop_id>
  <total>14.55</total>
  <is_void>false</is_void>
  <discounts/>
  <subtotal>13.47</subtotal>
```

### JSON Data

```
{
  "root": {
    "localtime": "2020-02-07T21:07:00.000",
    "receipt_id": "1469",
    "sale_type": "",
    "taxes": {
      "name": "Tax",
      "value": "1.08"
    },
    "staff": {
      "id": "2006",
      "has_original_id": "true"
    },
    "products": [
      {
        "seat": "1",
        "quantity": "1",
        "discounts": "",
        "price": "5.99",
        "name": "Sandwich",
        "modifiers": ""
      },
      {
        "seat": "1",
        "quantity": "1",
        "discounts": "",
        "price": "2.49",
        "name": "Soda",
        "modifiers": ""
      },
      {
        "seat": "2",
        "quantity": "1",
        "discounts": "",
        "price": "4.99",
        "name": "Salad",
        "modifiers": ""
      }
    ],
    "consumer_id": "11-1",
    "payment_methods": "",
    "shop_id": "1535509869033384",
    "total": "14.55",
    "is_void": "false",
    "discounts": "",
    "subtotal": "13.47"
  }
}
```

## Testing Endpoint

<https://qr-cert.simpletabcloud.com/tomcat>

Log in and use the test utility:

User: simpletabcloud Pass: 963687

<https://qr-cert.simpletabcloud.com/tomcat/web/#/home/command-test>

## Production Endpoint

<https://qr.simpletabcloud.com/tomcat>

EVO PROPERTY