Make Terminal Payments

Feature: Payments

Editions: Cloud, Corporate, Store

The Terminal Payments API endpoints provide a way to initiate and complete payment actions through terminal devices.

Terminal payments are for card-present payments, where a customer taps or enters their credit card into a payment terminal. The payment terminal is configured to a backend payment processor that handles the payment details. POS Server – through specific payment features, such as *Fortis Payments* – integrates with the processor to perform the payment transactions.

As payment processors use different ways to access payment terminals, the POS API uses a standard method of working with the terminals. This method is:

- The caller starts a payment transaction through the `terminalPayment.start` endpoint.
- The start endpoint will return one of three types of results:
  - The payment, if the payment was immediately processed during the API call.
  - An error, if the payment failed validation or was not successfully processed.
  - A *continuation*, indicating that the caller must query for an updated payment status. The continuation includes the amount of time to wait before querying for the status.
- The caller must call the `terminalPayment.continue` endpoint to query the updated payment status, if a continuation was returned in the initial call.
- Like the start endpoint, the continue endpoint will return one of three results:
  - The payment, if the payment was successfully processed.
  - An error, if the payment was not successfully processed.
  - A continuation, indicating that the caller must re-query for an updated payment status.
- For a successful payment authorization, another call is required. The caller should call the `terminalPayment.capture` endpoint to capture the authorized payment. Again, the capture endpoint will return either the successful payment, an error, or a continuation.

Permissions

These endpoints require the following permissions:

- `MakeSaleTerminalPayment` to make a SALE (immediate) payment.
- `MakeRefundTerminalPayment` to make a REFUND (immediate) payment.
- `AuthorizeTerminalPayment` to make an AUTHORIZE payment authorization.
- `CapturePayment` to capture a payment authorization.
- `VoidPayment` to void a payment or payment authorization.
- `ListPaymentTerminals` to view a list of payment terminals for the current location.

a) List Terminals

Return a list of payment terminals configured for the POS server location:

```
query MyQuery {
  terminalPayment {
    terminals {
      id
      name
    }
  }
}
```

The caller should should typically call this endpoint to retrieve a list of available terminals. This can be done in a settings page, for example, to allow a user to choose the terminal that will process payments.

b) Get Capabilities

Return the capabilities supported by the currently configured payment processor:

```
query MyQuery {
  terminalPayment {
    capabilities {
      canAuthorize
      canBlindRefund
      canDirectRefund
      refNoMaxLength
    }
  }
}
```

The caller should query this endpoint to determine the type of operations that are supported by the processor, to ensure that it only calls the operations that are supported. Payment processors may vary in their ability to perform the following types of operations:

- Whether they support payment authorization and capture;

- Whether they support *blind* refunds – refunds where the ID of the payment to refund *is not* known;
- Whether they support *direct* refunds – refunds where the ID of the payment to refund *is* known;
- The maximum field length of a payment's `refNo` field.

c) Start Payment

Start a payment operation:

```
mutation MyMutation {
  terminalPayment {
    start(
      startRequest: {
        type: AUTHORIZE,
        terminalId: "11ed07ba8af1eb32897e4482",
        amount: "3.99",
        correlationId: "abcdef",
        refNo: "P1003",
        sale: {
          id: "1000",
          seq: 1,
          subTotal: "3.95",
          tax: "0.04",
          lines: {
            sku: "GUM",
            description: "Chewing gum",
            unitPrice: "3.95",
            quantity: 1,
            uom: "EA",
            commodityCode: null
          }
        }
      }) {
      terminalId
      status
      continuation {
        code
        retrySeconds
      }
      error {
        type
        message
```

```
        providerMessage
        isPaymentInUnknownState
      }
      payment {
        id
        saleId
        refNo
        requestedAmount
        amount
        tipAmount
        authCode
      }
    }
  }
}
```

This example shows a payment authorization, with start request type set to `AUTHORIZE`. The requests for `SALE` and `REFUND` payment operations are identical, other than the type input parameter.

> Refunds also optionally take a `refundPaymentId` parameter. If specified, the refund will be a *direct* refund against the identified payment transaction; if not specified, the refund is a *blind* refund.
>
> Consult the `capabilities` endpoint to determine which type(s) of refunds that the current payment processor supports. Payment processors may also differ on how they implement each type of refund.

Other Parameters

The endpoint also takes the following input parameters:

- A terminal ID indicating which terminal to scan/process the credit card.
- The amount of the payment.
- A reference number for the payment.
    - This is typically the caller's internal identifier for the payment.
    - This value must be unique, as most payment processors will often validate it to help prevent duplicate payment transactions.

  - Use the `capabilities` endpoint to query the maximum text length of this value.
- A sale object containing the details of the sale.
  - While optional, it is highly recommended to provide this information as it enables benefits such a as Level 3 process integration.
- A *correlation ID* that can be used for logging.
  - The caller should generate a unique value that can be used to identify the payment attempt.
  - This value can be used to correlate log records across multiple systems, making it easier to identify potential problems.
  - For example, the same ID can be used to identify the payment operation at the caller, within POS server, and within the payment processor itself.

Response

Like the other terminal payment endpoints, this endpoint returns either a payment value, an error value, or a continuation value. The caller should be prepared to handle any one of these states.

The following is an example of a continuation value response:

```
{
  "data": {
    "terminalPayment": {
      "start": {
        "terminalId": "11ed07ba8af1eb32897e4482",
        "status": "CONTINUE",
        "continuation": {
          "code": "5cb5705a-1496-4997-8275-99c508b2c0ea",
          "retrySeconds": 2
        },
        "error": null,
        "payment": null
      }
    }
  }
}
```

There are two values in the response that should be considered when calling the `continue` endpoint:

- The `code` must be passed to the continue endpoint.
- The `retrySeconds` indicates the the optimal time to wait (in seconds) before calling the endpoint.

d) Continue Payment

Polls the payment processor for updated status about a payment operation:

```
mutation MyMutation {
  terminalPayment {
    continue(
      continueRequest: {
        code: "5cb5705a-1496-4997-8275-99c508b2c0ea"
        correlationId: "ghijkl",
        sale: {
          id: "1000",
          seq: 1,
          subTotal: "3.95",
          tax: "0.04",
          lines: {
            sku: "GUM",
            description: "Chewing gum",
            unitPrice: "3.95",
            quantity: 1,
            uom: "EA",
            commodityCode: null
          }
        }
      })
    {
      terminalId
      status
      continuation {
        code
        retrySeconds
      }
      error {
        type
        message
        providerMessage
        isPaymentInUnknownState
      }
      payment {
        id
        saleId
```

```
        refNo
        requestedAmount
        amount
        tipAmount
        authCode
      }
    }
  }
}
```

The key parameter that you must pass is the continuation `code` that was returned in the call to the `start` endpoint.

Other Parameters

The endpoint also takes the following input parameters:

- A sale object containing the details of the sale.
  - This should be the same sale information that you originally supplied to the `start` endpoint.
- A *correlation ID* that can be used for logging.
  - Like the other endpoints, the correlation ID is a unique value that can be used to identify the continuation operation in log records that are created across multiple systems.

Response:

Like the other terminal payment endpoints, this endpoint returns either a payment value, an error value, or a continuation value. The caller should be prepared to handle any one of these states.

> A continuation call may return another continuation in the event that the payment operations is still being processed (e.g. waiting for input from the user). It may take several calls to the `continue` endpoint to complete the operation.

Upon a successful payment operation, the detailed payment information is returned in the response like the following:

```
{
  "data": {
    "terminalPayment": {
      "continue": {
```

```
        "terminalId": "11ed07ba8af1eb32897e4482",
        "status": "OK",
        "continuation": null,
        "error": null,
        "payment": {
          "id": "11ed256a6ea7390aa07d5728",
          "saleId": "1000",
          "refNo": "P1003",
          "requestedAmount": 3.99,
          "amount": 3.99,
          "tipAmount": 0,
          "authCode": "256a6e"
        }
      }
    }
  }
}
```

e) Capture Payment

Capture (complete) a payment authorization:

```
mutation MyMutation {
  terminalPayment {
    capture(
      captureRequest: {
        id: "11ed256a6ea7390aa07d5728"
        amount: "3.99"
        correlationId: "mnopqr",
        sale: {
          id: "1000",
          seq: 1,
          subTotal: "3.95",
          tax: "0.04",
          lines: {
            sku: "GUM",
            description: "Chewing gum",
            unitPrice: "3.95",
            quantity: 1,
            uom: "EA",
            commodityCode: null
```

```
        }
      }
    })
  {
    terminalId
    status
    continuation {
      code
      retrySeconds
    }
    error {
      type
      message
      providerMessage
      isPaymentInUnknownState
    }
    payment {
      id
      saleId
      refNo
      requestedAmount
      amount
      tipAmount
      authCode
    }
  }
 }
}
```

The key parameters that you must pass are the payment ID generated by the `start` endpoint, and the amount of funds that you want to capture.

> The amount of funds to capture should be less than or equal to the amount that was authorized in the original payment authorization.

Other Parameters

The endpoint also takes the following input parameters:

- A sale object containing the details of the sale.
  - This should be the same sale information that you originally supplied to the `start` endpoint.
- A *correlation ID* that can be used for logging.
  - Like the other endpoints, the correlation ID is a unique value that can be used to identify the capture operation in log records that are created across multiple systems.

Response:

Like the other terminal payment endpoints, this endpoint returns either a payment value, an error value, or a continuation value. The caller should be prepared to handle any one of these states.

> While it is not common, a payment processor may return a continuation for a capture operation. The caller must be able to support this type of scenario, and use the `continue` endpoint to poll for the status of the capture operation.

Upon a successful payment operation, the detailed payment information is returned in the response like the following:


```
{
  "data": {
    "terminalPayment": {
      "capture": {
        "terminalId": "11ed07ba8af1eb32897e4482",
        "status": "OK",
        "continuation": null,
        "error": null,
        "payment": {
          "id": "11ed256a6ea7390aa07d5728",
          "saleId": "1000",
          "refNo": "P1003",
          "requestedAmount": 3.99,
          "amount": 3.99,
          "tipAmount": 0,
          "authCode": "256a6e"
```

```
        }
      }
    }
}
}
```

f) Void Payment

Void (cancel) a payment or payment authorization:

```
mutation MyMutation {
  terminalPayment {
    void(
      voidRequest: {
        id: "11ed256a6ea7390aa07d5728",
        correlationId: "stuvwx"
      })
    {
      terminalId
      status
      continuation {
        code
        retrySeconds
      }
      error {
        type
        message
        providerMessage
        isPaymentInUnknownState
      }
      payment {
        id
        saleId
        refNo
        requestedAmount
        amount
        tipAmount
        authCode
      }
    }
  }
```

}

Voids are supported only for payments that are not posted yet in the payment processor. The endpoint will return an error if a void against a posted payment is attempted.

The key parameter that you must pass is the payment ID generated by the `start` endpoint.

Other Parameters

The endpoint also takes the following input parameters:

- A *correlation ID* that can be used for logging.
    - Like the other endpoints, the correlation ID is a unique value that can be used to identify the void operation in log records that are created across the caller, POS server and the payment processor.

Response:

Like the other terminal payment endpoints, this endpoint returns either a payment value, an error value, or a continuation value. The caller should be prepared to handle any one of these states.

> While it is not common, a payment processor may return a continuation for a void operation. The caller must be able to support this type of scenario, and use the `continue` endpoint to poll for the status of the void operation.