ENCODED

secure automated payments

Encoded Gateway API

Developer Guide

Title	Encoded Gateway API - Developer Guide
Document No	ENCODED/MSC/19-091
Classification	PUBLIC
Revision	3.2
Date	2023-10-03
Author	Adam Bromage-Hughes
Approver	Frank Pocklington

Contents

Introduction

The purpose of this document is to be read in conjunction with the Encoded Gateway API Specification Document, to provide additional guidance and implementation notes for implementers.

Any major changes to the Encoded Gateway API will be communicated to all partners, and implementers should always ensure that they have the most up-to-date copy by making a request to Encoded's Service Desk.

API Specification Documentation

The API specification documentation, in OpenAPI format, can be found at the following URL:

https://sit.encoded.services/api/v1/docs

Changelogs

Changelogs for the Encoded Gateway API and associated services can be found at the following URL:

https://wiki.encoded.support/changelogs

URLs

The API will be hosted at a base URL of

```
https://[env].encoded.services/api/[version]/
```

where *[env]* is the environment targeted and *[version]* is the version targeted. Your IP will need to be whitelisted to access this service.

The following environments are currently available:

1. **prod**
   Production environment for live transacting.

2. **sit**
   System Integration Testing environment, for implementors to test their implementation and perform test transactions.

The following versions are currently available:

- **v1**

Authentication

Authentication is via OAuth 2.0 Client Credentials Grant Type, which will return JWT access tokens. The JWT access token should be provided in the Authorization header for all requests to the Encoded Gateway API.

The Authentication Server will be hosted at the URL of

```
https://[env].encoded.services/auth/oauth/token
```

where *[env]* is the environment targeted (prod, sit, etc).

To authorise and receive a token for access to the Gateway API, a request must be made with the Client ID and Client Secret provided as a HTTP Basic Auth Header, along with the Grant Type provided a query parameter, as in the example request and response below.

```
POST /auth/oauth/token HTTP/1.1
Host: sit.encoded.services
Authorization: Basic QWxhZGRpbjpPcGVuU2VzYW1l
grant_type=client_credentials
```


```
{
    "access_token":
"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOlsib2F1dGgyLXJlc291cmNlIl0sInNjb3BlIjpbInNlc3Npb25fbGltaXRlZF9yZWFkIl0sImV4cCI6MTYyMDY5MzgwMiwidXNlcmNvZGUiOjI1ODEyOSwiYXV0aG9yaXRpZXMiOlsiUk9MRV9BUElfU1VCU0NSSUJFUiJdLCJqdGkiOiJ4eGxHdzRWUkJybkExYnVHM3FhcTV3aTR0U00iLCJjbGllbnRfaWQiOiJBc3RyYWxUZWNoRGV2In0.hqO_0scdeP8yE7h_LMIHKzXMGpinvffAtsoQrL5eoo8hcizHFFFmSrnwSGciwZWEsp1D98f-qpexeN2vjdEok1ItaGMlibz7hVspMGNNleLwBevjIJOWi3Nf-RAUZc0flOsTz7AKAbGph-j5mdfjtxXe5rp61ArPpoe9bRx8tubEcAvgSqIBJtwl6VYqBdkBjK6ytkBUabyWbA3eKZB-gZ4QDMugOt6Sv2afjyNTD42pPj0FFGOrm2MZZH6YDLgSuqi0enhKbba3h5BWR4S4i78RrexlxufjrbFUs-9OZjkRh1r8QGPWqYGGq8jMEWsnMH35kI43n9Pdd99S_sbNAw",
    "token_type": "bearer",
    "expires_in": 21599,
    "scopes": "session_limited_read",
    "usercode": 258129,
    "jti": "xxlGw4VRBrnA1buG3qaq5wi4tSM",
    "tokenType": "bearer",
    "expiration": "May 11, 2021 1:43:22 AM",
    "scope": [
        "session_limited_read"
    ],
    "additionalInformation": {
```

```
        "usercode": 258129,
        "jti": "xxlGw4VRBrnA1buG3qaq5wi4tSM"
    }
}
```


The returned token should then be provided to all requests to the Encoded Gateway API as an Authorization Bearer header:


```
GET /api/1/transactions/4b502950-8801-4112-ba15-88f8eb280525
Authorization: Bearer
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOlsib2F1dGgyLXJlc291cmNlIl0
sInNjb3BlIjpbInNlc3Npb25fbGltaXRlZF9yZWFkIl0sImV4cCI6MTYyMDY5MzgwMiwidXN
lcmNvZGUiOjI1ODEyOSwiYXV0aG9yaXRpZXMiOlsiUk9MRV9BUElfU1VCU0NSSUJFUiJdLCJ
qdGkiOiJ4eGxHdzRWUkJybkExYnVHM3FhcTV3aTR0U00iLCJjbGllbnRfaWQiOiJBc3RyYWx
UZWNoRGV2In0.hqO_0scdeP8yE7h_LMIHKzXMGpinvffAtsoQrL5eoo8hcizHFFFmSrnwSGc
iwZWEsp1D98f-qpexeN2vjdEok1ItaGMlibz7hVspMGNNleLwBevjIJOWi3Nf-RAUZc0flOs
Tz7AKAbGph-j5mdfjtxXe5rp61ArPpoe9bRx8tubEcAvgSqIBJtwl6VYqBdkBjK6ytkBUaby
WbA3eKZB-gZ4QDMugOt6Sv2afjyNTD42pPj0FFGOrm2MZZH6YDLgSuqi0enhKbba3h5BWR4S
4i78RrexlxufjrbFUs-9OZjkRh1r8QGPWqYGGq8jMEWsnMH35kI43n9Pdd99S_sbNAw
```


Authentication for Hosted Payment Fields

When utilising Hosted Payment Fields (see Hosted Payment Fields), it is required to pass a JWT to the browser in order to initialise the HPF Javascript library. Doing so exposes the JWT to the front-end, and no longer guarantees that the JWT used for accessing the protected resources is private.

To protect from this, it is necessary to create a “limited use” JWT which is then used by the Hosted Payment Fields Javascript library. Note that the library will ONLY accept a limited use JWT.

To authorise and receive a token for access to the Hosted Payment Fields Javascript LibraryI, a request must be made with the Client ID and Client Secret provided as a HTTP Basic Auth Header, along with the Grant Type provided as a query parameter, as with the usual method of authentication. Additional to this, a Scope of “*session_limited_read”* must be provided, along with the *session_id* of the Session created prior to initialising the Hosted Payment Fields Javascript Library., as in the example request below:

```
POST /auth/oauth/token HTTP/1.1
Host: sit.encoded.services
```

```
Authorization: Basic QWxhZGRpbjpPcGVuU2VzYW1l
grant_type=client_credentials
scope=session_limited_read
session_id=10b3ca21-d9fd-4030-b5f5-fb45f220a6dd
```


```
{
    "access_token":
"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOlsib2F1dGgyLXJlc291cmNlIl0sInNjb3BlIjpbInNlc3Npb25fbGltaXRlZF9yZWFkIl0sImV4cCI6MTYyMDY5MzgwMiwidXNlcmNvZGUiOjI1ODEyOSwiYXV0aG9yaXRpZXMiOlsiUk9MRV9BUElfU1VCU0NSSUJFUiJdLCJqdGkiOiJ4eGxHdzRWUkJybkExYnVHM3FhcTV3aTR0U00iLCJjbGllbnRfaWQiOiJBc3RyYWxUZWNoRGV2In0.hqO_0scdeP8yE7h_LMIHKzXMGpinvffAtsoQrL5eoo8hcizHFFFmSrnwSGciwZWEsp1D98f-qpexeN2vjdEok1ItaGMlibz7hVspMGNNleLwBevjIJOWi3Nf-RAUZc0flOsTz7AKAbGph-j5mdfjtxXe5rp61ArPpoe9bRx8tubEcAvgSqIBJtwl6VYqBdkBjK6ytkBUabyWbA3eKZB-gZ4QDMugOt6Sv2afjyNTD42pPj0FFGOrm2MZZH6YDLgSuqi0enhKbba3h5BWR4S4i78Rrexlxufjrb FUs-9OZjkRh1r8QGPWqYGGq8jMEWsnMH35kI43n9Pdd99S_sbNAw",
    "token_type": "bearer",
    "expires_in": 21599,
    "scopes": "session_limited_read",
    "usercode": 258129,
    "jti": "xxlGw4VRBrnA1buG3qaq5wi4tSM",
    "tokenType": "bearer",
    "expiration": "May 11, 2021 1:43:22 AM",
    "scope": [
        "session_limited_read"
    ],
    "additionalInformation": {
        "usercode": 258129,
        "jti": "xxlGw4VRBrnA1buG3qaq5wi4tSM"
    }
}
```

Resource References

The API allows for the use of resource reference where passing the full object may not be necessary or preferred. A resource reference has the following structure:

```
{
  "object": "transaction",
  "id": "4b502950-8801-4112-ba15-88f8eb280525",
  "links": {
                                                                    "self":
"https://sit.encoded.services/api/1/transactions/4b502950-8801-4112-ba15
-88f8eb280525"
  }
}
```

The *object* and *id* fields together uniquely identify a specific resource and can therefore be used to reference that resource in its entirety. The *links* field is not required when sending the resource reference as part of a request, but will be available when the resource reference makes up part of a response from the API. The *links* field's value will be an object containing links to allow the API user to access the referenced resource directly.

Merchant Accounts

A Merchant Account is a type of bank account that is specifically used for accepting customer payments via a credit or debit card, or an alternative or local payment method such as Apple Pay or Bancontact respectively. You can get a Merchant Account from an entity called an Acquirer.

The Gateway API is an Acquirer agnostic platform, which means that transactions can be processed through any Acquirer that Encoded is connected to. If you do not already have a relationship with an Acquirer, or are looking to move Acquirers, Encoded can make recommendations.

Payment Orchestration

The Gateway API is also what is known as a Payment Orchestration Platform, which means that it can make smart decisions about how and where to route transactions to be processed. As such, you can have multiple Merchant Accounts with multiple Acquirers, and allow Encoded's Gateway API to route transactions to the underlying Acquirer which has the best outcome for that transaction; whether that be lowest cost, highest acceptance rate, or any other potential outcome.

This is an automatic process which is governed by configurable rulesets within the Gateway API. These rules can be configured either via an API (coming soon), via the Merchant Management Portal (coming soon), or through consultation with your Implementation Manager.

Explicit Merchant Account Selection

We also provide the flexibility for merchants to make their own decisions on the most appropriate Merchant Account to use to process each transaction, and provide an optional API field - *transactionRequest.merchantAccountId* - to do so. If you wish to take advantage of this, please speak to your Implementation Manager who will provide you with a list of Merchant Account IDs.

Orders

An order within the Gateway API allows you to link all related transactions that relate to a single order between the merchant and the end customer. An order can contain multiple individual transactions that either capture or refund funds from and to the end customer.

The majority of orders will have either one or two transactions, where either the *pay* action has been used to perform a transaction that both authorises and captures the funds immediately, or where an *authorise* transaction is followed by a *capture* transaction to authorise and capture the funds as two individual transactions, respectively. The former is more likely when the related good or service is available immediately, and the latter when the good or service may be provided at a future date, at which point the authorised transaction is captured (for example, when the product will be dispatched at a later date once available or in stock).

An order can be created individually from a transaction, or can be created during the initial transaction, however there is no requirement for an order to be created for a transaction, and all transactions can be processed on an individual (not linked to an order) basis. Once a transaction has been created on an individual basis, an order can not be retroactively applied to it.

An *order.id* is generated when the order is created. Additionally, a merchant defined reference can be provided via *order.ref* for reconciliation with any merchant ordering system.

An order must be created if you wish to take advantage of the Hosted Payment Pages

Transactions

Transactions in the Gateway API represent each individual transaction; the status of the transaction, along with both the request made and the response received. There are a number of transaction *action*s that can be performed. These are:

- **authorise**
  Performs an authorisation for the proposed transaction. A successful response from the acquirer indicates that the payment source provided is valid and that the funds are available, however no funds are transferred at this time. The acquirer may reserve the funds for a period of time. An authorisation can be captured at a later time.

- **capture**
  Performs a capture of a previous authorisation. A successful response from the acquirer indicates that the funds will be moved from the payer's account to the merchant's account. When performing a capture, you must provide a previously successful *transaction* as the source, as well as the same Order used for the previous authorisation action.

- **pay**
  Performs an authorisation and capture in a single transaction. A successful response from the acquirer indicates that the payment source provided is valid, the funds are available, and that the funds will be moved from the payer's account to the merchant's account.

- **refund**
  Performs a refund of funds from the merchant's account to the payer's account. A successful response from the acquirer indicates that the funds will be moved from the merchant's account to the payer's account. When performing a refund, the source may be a Card, Token, or a previously successful Transaction, depending on the underlying gateway provider.

- **void**
  Performs a void of a previous transaction, which will attempt to cancel the previous transaction. Whether a void will be successful depends on a number of factors, such as the amount of time between the initial transaction and the void attempt, the underlying acquirer, the transaction source, etc. It may be necessary to perform a refund if the void was not successful.

- **verify**
  Performs a verification of the source. The verification method used is that which is supported by the underlying payment gateway and acquirer.

When performing a transaction, a source of the funds needs to be provided via *transaction.source*. There are currently six supported source types. These are:

- **card**
  The source of funds is a debit or credit card where the card details (the PAN, expiry date, security code, etc) are provided directly.

- **token**
  The source of funds is a token that has been previously created. A token is a reference to a previously used and stored debit or credit card. To create a token, a transaction will be performed with a card source, which will be provided along with the *transaction.source.card.tokenisation* object to indicate that the card details provided should be stored as a token for future use.

- **transaction**
  The source of funds is a previously performed transaction (where supported by the underlying acquirer). This is required for performing a *capture* transaction against a previously performed *authorise* transaction

- **session**
  The source of funds is a session in which card details have been collected for use directly by Encoded. This source type would be used, as an example, when performing a Hosted Payment Fields transaction.

- **google_pay**
  The source of funds is a Google Pay token.

- **apple_pay**
  The source of funds is an Apple Pay token.

A *transaction.id* is generated when the transaction is created. Additionally, a merchant defined reference can be provided via *transaction.ref* for reconciliation with any merchant ordering system.

When a transaction is performed by POSTing a *transaction.request* object to the */transactions* endpoint, the response polls until either: the transaction reaches a *processed* or *challenged* status; or, the poll time provided as a query parameter is met. If the poll time is met, the transaction is returned in the *processing* state. Additional requests can be made to */transactions/{transactionId}* which follow the same rules.

Authorisation & Capture

There are two individual aspects to a payment transaction, known as the Authorisation and the Capture. These processes can either be performed at once (with a *pay* action), or individually with an *authorise* action followed by a *capture* action.

When an *authorise* action is performed, the gateway sends the card details to the issuer, who checks the card details provided and, if valid, will hold the funds requested if they are available. The merchant will be provided with an authorisation code and can then collect the funds associated with that authorisation within 7 days by performing a *capture* action. When a *capture* is performed, the transaction ID of the *authorise* action must be provided as part of the request.

An example Authorisation & Capture flow is shown below. Firstly, an *authorise* action is performed:

```
POST /transactions
{
    "object": "transaction.request",
    "action": "authorise",
    "ref": "trans-1234",
    "amount": 20.54,
    "currency": "GBP",
    "source": {
        "object": "source",
        "card": {
            "object": "card",
            "pan": "4444333322221111",
            "expiry": "2022-10",
            "securityCode": "111"
        }
    }
}
```

Which receives the following response:

```
{
  "object": "transaction",
  "id": "4b502950-8801-4112-ba15-88f8eb280525",
  "creationDate": "2019-07-01T00:00:00Z",
  "status": "processed",
```

```
  "request": {
    "object": "transaction.request",
    "id": "4b502950-8801-4112-ba15-88f8eb280525"
  },
  "response": {
    "object": "transaction.response",
    "id": "4b502950-8801-4112-ba15-88f8eb280525",
    "result": {
      "resultType": "accepted",
      "resultCode": "accepted",
      "message": "Authorised.",
      "authCode": "123456",
      "authDate": "2019-07-01T00:00:00Z"
    }
  },
  "links": {
    "self":
"https://sit.encoded.services/api/1/transactions/4b502950-8801-4112-ba15
-88f8eb280525"
  }
}
```

The response to the *authorise* action returns an *id* field, which will be used for the subsequent *capture* action:

```
POST /transactions
{
    "object": "transaction.request",
    "action": "capture",
    "ref": "trans-1234",
    "amount": 20.54,
    "currency": "GBP",
    "source": {
        "object": "source",
        "transaction": {
            "object": "transaction",
            "id": "4b502950-8801-4112-ba15-88f8eb280525"
        }
    }
}
```

The *capture* is performed referencing the previous transaction as the source. This receives the following response indicating that the capture was successful:


```
{
  "object": "transaction",
  "id": "2156d9ef-a16a-4e43-b5d2-bfdcb6c9afae",
  "creationDate": "2019-07-01T00:00:00Z",
  "status": "processed",
  "request": {
    "object": "transaction.request",
    "id": "2156d9ef-a16a-4e43-b5d2-bfdcb6c9afae"
  },
  "response": {
    "object": "transaction.response",
    "id": "2156d9ef-a16a-4e43-b5d2-bfdcb6c9afae"
    "result": {
      "resultType": "accepted",
      "resultCode": "accepted",
      "message": "Captured."
    }
  },
  "links": {
    "self":
"https://sit.encoded.services/api/1/transactions/2156d9ef-a16a-4e43-b5d2
-bfdcb6c9afae"
  }
}
```


Voiding an Authorisation

If you have an outstanding authorisation that will not be captured, you must void the authorisation as soon as possible. This can be performed with a *void* action, passing in the transaction as the source (as with performing a capture).

Partial Capture

If you perform a partial capture on an authorisation - capturing a lower amount then was authorised - there is no need to additionally void the remaining amount.

Excessive Capture

The gateway does not currently support excessive capture. If the capture amount will be higher than the authorised amount, there are two possible scenarios:

1. Void the existing authorisation, and perform a new authorisation for the entire higher amount.
2. Perform a second authorisation of the remaining amount (the difference between the previous authorised amount and the intended capture amount), and then capture both authorisation when required.

Customers

The Gateway API allows for Customer objects to be created that represent the end customer. These Customer objects can be attached to Orders, Transactions and Tokens. This is useful functionality to allow these resources to be tied together, and searched and loaded on a per-customer basis.

Customer objects can either be created explicitly or implicitly during the creation of an Order, Transaction or Token.

Adding New Customers

Customers can be added explicitly via the Encoded Gateway API. To do this, a POST request can be made to */customers* with an Array of Customer objects:

```
[{
  "object": "customer",
  "ref": "12345-19850701",
  "title": "Mr",
  "forename": "John",
  "surname": "Doe",
  "dateOfBirth": "1985-07-01",
  "contact": {
    "object": "contact",
    "address": {
      "object": "address",
      "title": "Mr",
      "forename": "John",
      "surname": "Doe",
      "postcode": "AB1 2CD",
    }
  },
  "attributes": {
    "account": [
    {
      "accountNumber": "100001234",
      "paymentAmount": 100.00,
      "arrearsAmount": 100.00,
      "tokenisationEnabled": true,
      "creditCardsEnabled": true,
      "defaultToken": "2156d9ef-a16a-4e43-b5d2-bfdcb6c9afae"
    },
    {
```

```
      "accountNumber": "100009999",
      "paymentAmount": 450.25,
      "arrearsAmount": 100.00,
      "tokenisationEnabled": true,
      "creditCardsEnabled": true,
      "defaultToken": "2156d9ef-a16a-4e43-b5d2-bfdcb6c9afae"
    },
  ]
}]
```

Shown above is a Customer object example with non-required fields removed.
The *ref* field shown should be a unique identifier from the merchant for the customer, such as a unique customer or account ID.

The *attributes* field allows for custom attributes to be saved against the Customer object. This is shown in more detail in the *Custom Attributes* section below.

Updating Existing Customers

In order to update an existing Customer resource, the existing Customer resource must first be identified directly via the *id,* or searched for via the *ref*, or any attribute contained within the *attributes* Object. There are two main methods for doing this:

1. When a customer resource is created, it is generated a unique *id* which is included in the response object. This can be used to identify the resource directly..
2. Customer records can be searched by including the targeted reference in the query string, which returns an Array of matching Customer resources. The correct Customer resource should be identified and the *id* used to directly access the resource.

Once the *id* of the Customer resource is known, a PUT request can be made to /customers/{customerId} along with the Customer object to replace the existing object.

```
GET /customers?page=0&results=1&attributes.account.accountNumber=12345
[{
  "object": "customer",
  "id": "7ff16945-db98-4216-af7d-cf6094ae5f61",
  "ref": "12345-19850701",
  "title": "Mr",
  "forename": "John",
  "surname": "Doe",
  "dateOfBirth": "1985-07-01",
```

```
  "contact": {
    "object": "contact",
    "address": {
      "object": "address",
      "title": "Mr",
      "forename": "John",
      "surname": "Doe",
      "postcode": "AB1 2CD",
    }
  },
  "attributes": {
    "account:" {
      "accountNumber": "12345",
      "paymentAmount": 100.00,
      "arrearsAmount": 100.00,
      "tokenisationEnabled": true,
      "creditCardsEnabled": true
    }
  }
}]

PUT /customers/7ff16945-db98-4216-af7d-cf6094ae5f61
{
  "object": "customer",
  "ref": "12345-19850701",
  "title": "Mr",
  "forename": "John",
  "surname": "Doe",
  "dateOfBirth": "1985-07-01",
  "contact": {
    "object": "contact",
    "address": {
      "object": "address",
      "title": "Mr",
      "forename": "John",
      "surname": "Doe",
      "postcode": "AB1 2CD",
    }
  },
  "attributes":
    "account:" {
      "accountNumber": "12345",
      "paymentAmount": 150.00,
```

```
      "arrearsAmount": 100.00,
      "tokenisationEnabled": true,
      "creditCardsEnabled": true,
      "defaultToken": "2156d9ef-a16a-4e43-b5d2-bfdcb6c9afae"
    }
  }
}
```

The above example searches for an existing customer who has a custom attribute of *account.accountNumber* with the value 12345, and updates their payment amount (another custom attribute) from 100.00 to 150.00.

Deleting Existing Customers

The same method is applicable for deleting existing customers. A customer resource can be deleted by making a DELETE request to /customers/{customerId}.

Be aware that tokens associated with that customer resource will also subsequently be deleted.

Custom Attributes

The Customer object allows attributes to be provided to extend what information can be provided about a customer. This has many applicable use cases, such as setting specific feature flags for certain customers (whether this customer is allowed to be with credit cards, as an example), default payment amounts, preferred tokens for scheduled payments, etc. Below shows an example of setting some additional account information, some default payment values, and setting some feature flags.

```
"attributes": {
  "account": {
    "accountNumber": "100001234",
    "paymentAmount": 800.00,
    "arrearsAmount": 100.00,
    "tokenisationEnabled": true,
    "creditCardsEnabled": true,
    "defaultToken": "2156d9ef-a16a-4e43-b5d2-bfdcb6c9afae"
  }
}
```

This object can also be provided as an Array if required.

```
"attributes": {
  "account": [
    {
      "accountNumber": "100001234",
      "paymentAmount": 100.00,
      "arrearsAmount": 100.00,
      "tokenisationEnabled": true,
      "creditCardsEnabled": true,
      "defaultToken": "2156d9ef-a16a-4e43-b5d2-bfdcb6c9afae"
    },
    {
      "accountNumber": "100009999",
      "paymentAmount": 450.25,
      "arrearsAmount": 100.00,
      "tokenisationEnabled": true,
      "creditCardsEnabled": true,
      "defaultToken": "2156d9ef-a16a-4e43-b5d2-bfdcb6c9afae"
    },
  ]
}
```

Custom attributes support Strings, Numbers, Booleans, Objects and Arrays as values.

Tokens

Tokenisation allows merchants to store payment details in exchange for a token. The token can be identified via a token ID and subsequently used to perform new transactions without having to recollect all (or any) of the sensitive card details from the cardholder. This is a useful feature to help implementers and merchants to reduce their PCI compliance burden by reducing the scope of their cardholder data environment. It should be noted that this is distinct from performing a follow-up transaction - for example, a capture following an authorisation - using the transaction ID of the original transaction, and is targeted at implementers who wish to make new transactions at a future date without having to recollect card details.

Tokens can be created from any transaction that provides a *card* as the transaction source. This can be achieved by including a *tokenisation* object alongside the *card* object, as in the below example:

```
"card": {
  "object": "card",
  "pan": "4444333322221111",
  "expiry": "2022-10",
  "securityCode": "111",
  "tokenisation": {
    "object": "tokenisation",
    "agreement": "card_on_file",
    "ref": "token-1234"
  }
}
```

The above example indicates that we wish to tokenise the card details provided with an *agreement* type of **card_on_file,** and a *ref* of **token-1234**. The agreement type indicates how we intend to use the token, and will be one of:

- **card_on_file**
  A transaction using a stored card for a fixed/variable amount which is not part of a scheduled/regular agreement but where the Cardholder themselves initiates the payment. For example, a stored card transaction initiated by the cardholder directly via an E-com or IVR channel, or indirectly (by verbally authorising) via a virtual terminal.

- **recurring**
  A transaction in a series of transactions processed for the purchase of goods/services provided at regular/fixed intervals. For example, a stored card

transaction performed for an ongoing service such as payment towards utility bills, subscriptions, etc.

- **instalment**
  A transaction in a series of transactions processed over a set period and number of payments for a single purchase of goods/services. For example, a stored card transaction performed for a one-off service/good, such as making monthly payments towards a large purchase.

- **unscheduled**
  A transaction using a stored credential for a fixed/variable amount which is not part of a scheduled/regular agreement but where the Cardholder has provided consent for the Merchant to initiate one or more future transactions. For example, a stored card transaction performed as a secondary means of payment when a primary means (such as Direct Debit) has failed.

The **authorise**, **capture**, **pay**, and **verify** *actions* can be used to create a token. Using the **verify** action means that only basic verification of the card will be performed, and does heighten the risk that a subsequent transaction performed on that token may fail.

Once the transaction has finished processing, the *transaction.response* will contain the *token* object for any token created as a result of that transaction being processed.

Once a token has been created, it can subsequently be provided as the *source* for any future transaction request to attempt to collect the funds from the original card.

Retrieving Tokens for a Customer

The best method of retrieving a token is via an associated Customer. To do this, make a GET request to /customers/{customerId}/tokens to retrieve an array of all tokens associated with the Customer.

```
GET /customers/7ff16945-db98-4216-af7d-cf6094ae5f61/tokens
[{
  "object": "token",
  "id": "09f50370-8d25-43b7-8250-2b2e56b360bd",
  "creationDate": "2019-07-01T00:00:00Z",
  "ref": "token-1234",
  "pan": "465858******6034",
  "expiry": "2020-05",
  "securityCode": "***",
  "issuer": {
```

```
    "object": "issuer",
    "id": "465858",
    "scheme": "Visa",
    "type": "Credit",
    "brand": "Barclays Bank Plc",
    "level": "Classic",
    "country": "GBR"
  },
  "agreement": "card_on_file",
  "billingCustomer": {
    "object": "customer",
    "id": "7ff16945-db98-4216-af7d-cf6094ae5f61",
    "links": {
      "self":
"https://sit.encoded.services/api/1/customer/7ff16945-db98-4216-af7d-cf6
094ae5f61"
    }
  }
}]
```

Retrieving all Tokens

An array of all existing token resources can be retrieved by making a GET request to /tokens along with suitable paging query parameters.

Making a Token Payment

Below is an example of a transaction request being made with a Token as the source of funds.

```
POST /transactions
{
  "object": "transaction.request",
  "action": "pay",
  "ref": "12345-19850701/20190801-1",
  "currency": "GBP",
  "amount": 100.00,
  "source": {
    "object": "source",
    "token": {
      "object": "token",
```

```
      "id": "09f50370-8d25-43b7-8250-2b2e56b360bd"
    }
  },
  "billingCustomer": {
    "object": "customer",
    "id": "7ff16945-db98-4216-af7d-cf6094ae5f61",
  }
}
```

Alternative Payment Methods (APMs)

APMs are alternative ways of making payment through the Gateway other than directly via a debit or credit card or a stored card token. The Encoded Gateway offers the following APMs.

Apple Pay

Apple Pay is the one way to pay. It replaces your physical cards and cash with an easier, safer, more secure and private payment method — whether you're in a shop, on a website or in an app. It's money, made modern.

Set up Apple Pay

Apple Developer Account

Firstly, you will need to set up an Apple Developer account. If you do not already have one, follow the link and click on Account in the top right corner. An Apple Developer account is a paid account which has a cost of approximately £80 per year as at July 2023. If you already have an Apple Developer account, skip to the next step.

Create a merchant identifier

A merchant identifier uniquely identifies you to Apple Pay as a merchant who is able to accept payments. A merchant identifier never expires, and you can use the same one for multiple apps.

1. Within your Apple Developer account, in "Certificates, IDs & Profiles", click Identifiers in the sidebar, then click the add button (+) on the top left. Select Merchant IDs, then click Continue.

2. Enter a useful description that will help you to identify what the merchant identifier is being used for.

3. Enter your merchant identifier name in the Identifier section. It is strongly recommended that you use a descriptive identifier including the environment and domain that you will use it within. For example: `merchant.services.encoded.prod`

4. Review the settings, then click Register.

Alternatively, you can create a merchant identifier in Xcode.

Create a payment processing certificate

A payment processing certificate is associated with your merchant identifier and used to encrypt payment information. The payment processing certificate expires every 25 months. If the certificate is revoked, you can recreate it.

Creating a payment processing certificate is a three step process; generating a Certificate Signing Request (CSR) from Encoded - either via an API request or via the Merchant Management Portal (coming soon), generating the Apple Pay Payment Processing Certificate within the Apple Developer account, and then uploading the Apple Pay Payment Processing Certificate to Encoded - again either via an API request or via the Merchant Management Portal (coming soon).

Generate a certificate signing request

To generate a certificate signing request, use the API endpoint along with the merchant identifier created in the above steps. An example request:

```
POST /applepay/signing-requests
{
    "merchantIdentifier": "merchant.services.encoded.prod"
}
```

We will then generate a certificate signing request on your behalf, along with an ID that can be used to later provide us with the Apple Pay Payment Processing Certificate.

You will receive the following example response:

```
{
    "id": "4fefad24-a6ce-41ba-a222-c823a31e7961",
    "csr": "-----BEGIN CERTIFICATE REQUEST-----
MIHZMIGBAgEAMB8xEDAOBgNVBAoMB0VuY29kZWQxCzAJBgNVBAYTAkdCMFkwEwYHKoZIzj0C
AQYIKoZIzj0DAQcDQgAE/XoGBUX8cuCCRovY1bxARjp5sl9hsk6yaBG/J3rBQoI9GmDpc6k4
HbQfWaUo/3mNMxbAU6v09TMUGx1c43brTaAAMAoGCCqGSM49BAMCA0cAMEQCIEI6RssKkutV
bmIdfwAXQj8c+087uV+cNf2AHJcgcLS1AiB682f8sWjhDc5G11kKGh/FHTEXkAYKULW60b6B
DliOmg== -----END CERTIFICATE REQUEST-----",
    "links":  {
        "self":
"https://sit.encoded.services/api/1/applepay/certificates/4fefad24-a6ce-
41ba-a222-c823a31e7961"
    }
}
```

The content provided to you in the *csr* field should be saved as a `.cer` file to later be provided to Apple Pay. Additionally, the link provided in the *links.self* field will be the endpoint that you will later use to provide Apple Pay Payment Processing Certificate back to Encoded.

Create a payment processing certificate

To create an Apple Pay Payment Processing Certificate, you need to provide the generated CSR to Apple.

1. Within your Apple Developer account, in “Certificates, IDs & Profiles”, click Identifiers in the sidebar, then select “Merchant IDs” on the drop-down on the right hand side of the screen.

2. Select the merchant identifier that was created in the above steps.

3. In the Apple Pay Payment Processing Certificate section - make sure you’re not in the Apple Pay Merchant Identity Certificate section - select Create Certificate.

4. Respond No to the question about processing in China and select Continue.

5. Upload the .csr file from the above step and select Continue.

6. Select Download to receive your Apple Pay Payment Processing Certificate in `.cer` format.

Upload the payment processing certificate

The Apple Pay Payment Processing Certificate must then be provided to Encoded in PEM format to the endpoint previously provided in the *signing-requests* response. An example request:

```
POST /applepay/certificates/4fefad24-a6ce-41ba-a222-c823a31e7961
{
    "certificate": "-----BEGIN CERTIFICATE-----
MIIEezCCBCGgAwIBAgIIDM6aT+Ea5xYwCgYIKoZIzj0EAwIwgYAxNDAyBgNVBAMMK0FwcGxl
IFdvcmxkd2lkZSBEZXZlbG9wZXIgUmVsYXRpb25zIENBIC0gRzIxJjAkBgNVBAsMHUFwcGxl
IENlcnRpZmljYXRpb24gQXV0aG9yaXR5MRMwEQYDVQQKDApBcHBsZSBJbmMuMQswCQYDVQQG
EwJVUzAeFw0yMzA1MjQxMjEwMzlaFw0yNTA2MjIxMjEwMzhaMIGsMS0wKwYKCZImiZPyLGQB
AQwdbWVyY2hhbnQuc2VydmljZXMuZW5jb2RlZC5kZXYxQzBBBgNVBAMMOkFwcGxlIFBheSBQ
YXltZW50IFByb2Nlc3Npbmc6bWVyY2hhbnQuc2VydmljZXMuZW5jb2RlZC5kZXYxEzARBgNV
BAsMCkpQNlMzWUZRNDYxFDASBgNVBAoMC0VuY29kZWQgTHRkMQswCQYDVQQGEwJHQjBZMBMG
ByqGSM49AgEGCCqGSM49AwEHA0IABH8KdeLni4Nx5aW3u4b2Axhyt2Nb5Rdn/loI+HWY+X/6
hCHO9Png4EIVe2HgYguGL6dN+zpi2YojYLespk5UocWjggJVMIICUTAMBgNVHRMBAf8EAjAA
MB8GA1UdIwQYMBaAFIS2hMw6hmJyFlmU6BqjvUjfOt8LMEcGCCsGAQUFBwEBBDswOTA3Bggr
BgEFBQcwAYYraHR0cDovL29jc3AuYXBwbGUuY29tL29jc3AwNC1hcHBsZXd3ZHJjYTIwMTCC
AR0GA1UdIASCARQwggEQMIIBDAYJKoZIhvdjZAUBMIH+MIHDBggrBgEFBQcCAjCBtgyBs1Jl
bGlhbmNlIG9uIHRoaXMgY2VydGlmaWNhdGUgYnkgYW55IHBhcnR5IGFzc3VtZXMgYWNjZXB0
YW5jZSBvZiB0aGUgdGhlbiBhcHBsaWNhYmxlIHN0YW5kYXJkIHRlcm1zIGFuZCBjb25kaXRp
```

```
b25zIG9mIHVzZSwgY2VydGlmaWNhdGUgcG9saWN5IGFuZCBjZXJ0aWZpY2F0aW9uIHByYWN0
aWNlIHN0YXRlbWVudHMuMDYGCCsGAQUFBwIBFipodHRwOi8vd3d3LmFwcGxlLmNvbS9jZXJ0
aWZpY2F0ZWF1dGhvcml0eS8wNgYDVR0fBC8wLTAroCmgJ4YlaHR0cDovL2NybC5hcHBsZS5j
b20vYXBwbGV3d2RyY2EyLmNybDAdBgNVHQ4EFgQU4gztoVmaREZ+pQTBLn5xebNpm0MwDgYD
VR0PAQH/BAQDAgMoME8GCSqGSIb3Y2QGIARCDEAzOEQzMjhCMzk5NzQ0MTU4QjgxNDg0N0Ex
QkM5QkIzRjNFRUNGQTJEQURGMTNERTI1Mzg5OUIxRUE2RDQ3QzM3MAoGCCqGSM49BAMCA0gA
MEUCIQDrrNeql2HlX2Xjx9a7Z5XJ+VVzPV4/cMLX1VjU8/YUkAIgOWvxwFVCf7KhZ2T4U7kU
BOy7B7cVKK1ZW7MchMygsX8= -----END CERTIFICATE-----"
}
```


Configure Apple Pay on the web

If you are using Apple Pay on the web, you will need to additionally register and verify your domain, and create a Merchant Identity Certificate. You will need to perform these tasks within your Apple Developer account, and will subsequently need to ensure that your frontend implementation makes use of the Merchant Identity Certificate created in this step.

Checkout Implementation

Please see the Apple Pay API documentation for information on how to integrate Apple Pay into your checkout.

- Apple Pay on the web
- PassKit (Apple Pay and Wallet) in app

Using the Apple Pay API

When creating the ApplePayPaymentRequest, you must set the values of *supportedNetworks* and *merchantCapabilities* to values supported by Encoded.

You can define which payment schemes are supported. Encoded currently supports American Express, Discovery, Mastercard and Visa.

```
const supportedNetworks = ["amex", "discover", "masterCard", "visa"];
```

You will also need to define the merchant capabilities.

```
const merchantCapabilities = ["supports3DS"];
```

Receiving the Payment Token from Apple Pay

Once your checkout has completed the Apple Pay process, an ApplePayPayment object will be returned to your application in the `ApplePaySession.onpaymentauthorized` event handler. The following fields will need to be sent to the Encoded Gateway:

```
ApplePayPayment.token
```

An example of the value contained in this field is below:


```
{"paymentData":{"data":"/U4zEWwhShc9GCiJorjt1WOvIC/Hx6dwdUGLLQUjtbIsG4Uj
tN61ovF1Y0Qm43zeXn1xogIcTVoG8qd0hoWzdqo8/8HqkWPgYwo+CKlPgh3ipCGy+GdPEEw9
0RXdESZ8Wwe1ZhGtojKsA8SNg4plPVNPGb9kGb0uFPQQ3Sbg7794rn1RmOFcuE7+3VnYJeB7
lvNoV7h2q2i90OWTXnliYy/IpVBV1uA3DBKHqzeAsZ25vuIWi6HCZBPlG8cMrL0BbHdNw7Mu
/ghIsYLWyOSrOQpmIGLDZ8CAVxVWSOLeQ/JR1ZuvxjThaeA+wrTchfZiV+incoe2pJJhbx2d
CZ+dmsps3Ne9LxYGxxI10RnZpDDoNBUwG7cOX5j04OGoneGJIc+2IjTV7GH0osfZ","signa
ture":"MIAGCSqGSIb3DQEHAqCAMIACAQExDTALBglghkgBZQMEAgEwgAYJKoZIhvcNAQcBA
ACggDCCA+MwggOIoAMCAQICCEwwQUlRnVQ2MAoGCCqGSM49BAMCMHoxLjAsBgNVBAMMJUFwc
GxlIEFwcGxpY2F0aW9uIEludGVncmF0aW9uIENBIC0gRzMxJjAkBgNVBAsMHUFwcGxlIENlc
nRpZmljYXRpb24gQXV0aG9yaXR5MRMwEQYDVQQKDApBcHBsZSBJbmMuMQswCQYDVQQGEwJVU
zAeFw0xOTA1MTgwMTMyNTdaFw0yNDA1MTYwMTMyNTdaMF8xJTAjBgNVBAMMHGVjYy1zbXAtY
nJva2VyLXNpZ25fVUM0LVBST0QxFDASBgNVBAsMC2lPUyBTeXN0ZW1zMRMwEQYDVQQKDApBc
HBsZSBJbmMuMQswCQYDVQQGEwJVUzBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABMIVd+3r1
seyIY9o3XCQoSGNx7C9bywoPYRgldlK9KVBG4NCDtgR80B+gzMfHFTD9+syINa61dTv9JKJi
T58DxOjggIRMIICDTAMBgNVHRMBAf8EAjAAMB8GA1UdIwQYMBaAFCPyScRPk+TvJ+bE9ihsP
6K7/S5LMEUGCCsGAQUFBwEBBDkwNzA1BggrBgEFBQcwAYYpaHR0cDovL29jc3AuYXBwbGUuY
29tL29jc3AwNC1hcHBsZWFpY2EzMDIwggEdBgNVHSAEggEUMIIBEDCCAQwGCSqGSIb3Y2QFA
TCB/jCBwwYIKwYBBQUHAgIwgbYMgbNSZWxpYW5jZSBvbiB0aGlzIGNlcnRpZmljYXRlIGJ5I
GFueSBwYXJ0eSBhc3N1bWVzIGFjY2VwdGFuY2Ugb2YgdGhlIHRoZW4gYXBwbGljYWJsZSBzd
GFuZGFyZCB0ZXJtcyBhbmQgY29uZGl0aW9ucyBvZiB1c2UsIGNlcnRpZmljYXRlIHBvbGlje
SBhbmQgY2VydGlmaWNhdGlvbiBwcmFjdGljZSBzdGF0ZW1lbnRzLjA2BggrBgEFBQcCARYqa
HR0cDovL3d3dy5hcHBsZS5jb20vY2VydGlmaWNhdGVhdXRob3JpdHkvMDQGA1UdHwQtMCswK
aAnoCWGI2h0dHA6Ly9jcmwuYXBwbGUuY29tL2FwcGxlYWljYTMuY3JsMB0GA1UdDgQWBBSUV
9tv1XSBhomJdi9+V4UH55tYJDAOBgNVHQ8BAf8EBAMCB4AwDwYJKoZIhvdjZAYdBAIFADAKB
ggqhkjOPQQDAgNJADBGAiEAvglXH+ceHnNbVeWvrLTHL+tEXzAYUiLHJRACth69b1UCIQDRi
zUKXdbdbrF0YDWxHrLOh8+j5q9svYOAiQ3ILN2qYzCCAu4wggJ1oAMCAQICCEltL786mNqXM
AoGCCqGSM49BAMCMGcxGzAZBgNVBAMMEkFwcGxlIFJvb3QgQ0EgLSBHMzEmMCQGA1UECwwdQ
XBwbGUgQ2VydGlmaWNhdGlvbiBBdXRob3JpdHkxEzARBgNVBAoMCkFwcGxlIEluYy4xCzAJB
gNVBAYTAlVTMB4XDTE0MDUwNjIzNDYzMFoXDTI5MDUwNjIzNDYzMFowejEuMCwGA1UEAwwlQ
XBwbGUgQXBwbGljYXRpb24gSW50ZWdyYXRpb24gQ0EgLSBHMzEmMCQGA1UECwwdQXBwbGUgQ
2VydGlmaWNhdGlvbiBBdXRob3JpdHkxEzARBgNVBAoMCkFwcGxlIEluYy4xCzAJBgNVBAYTA
lVTMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE8BcRhBnXZIXVGl4lgQd26ICi7957rk3gj
```

```
fxLk+EzVtVmWzWuItCXdg0iTnu6CP12F86Iy3a7ZnC+yOgphP9URaOB9zCB9DBGBggrBgEFB
QcBAQQ6MDgwNgYIKwYBBQUHMAGGKmh0dHA6Ly9vY3NwLmFwcGxlLmNvbS9vY3NwMDQtYXBwb
GVyb290Y2FnMzAdBgNVHQ4EFgQUI/JJxE+T5O8n5sT2KGw/orv9LkswDwYDVR0TAQH/BAUwA
wEB/zAfBgNVHSMEGDAWgBS7sN6hWDOImqSKmd6+veuv2sskqzA3BgNVHR8EMDAuMCygKqAoh
iZodHRwOi8vY3JsLmFwcGxlLmNvbS9hcHBsZXJvb3RjYWczLmNybDAOBgNVHQ8BAf8EBAMCA
QYwEAYKKoZIhvdjZAYCDgQCBQAwCgYIKoZIzj0EAwIDZwAwZAIwOs9yg1EWmbGG+zXDVspiv
/QX7dkPdU2ijr7xnIFeQreJ+Jj3m1mfmNVBDY+d6cL+AjAyLdVEIbCjBXdsXfM4O5Bn/Rd8L
CFtlk/GcmmCEm9U+Hp9G5nLmwmJIWEGmQ8Jkh0AADGCAYkwggGFAgEBMIGGMHoxLjAsBgNVB
AMMJUFwcGxlIEFwcGxpY2F0aW9uIEludGVncmF0aW9uIENBIC0gRzMxJjAkBgNVBAsMHUFwc
GxlIENlcnRpZmljYXRpb24gQXV0aG9yaXR5MRMwEQYDVQQKDApBcHBsZSBJbmMuMQswCQYDV
QQGEwJVUwIITDBBSVGdVDYwCwYJYIZIAWUDBAIBoIGTMBgGCSqGSIb3DQEJAzELBgkqhkiG9
w0BBwEwHAYJKoZIhvcNAQkFMQ8XDTIzMDcwMzE0MzAxMVowKAYJKoZIhvcNAQk0MRswGTALB
glghkgBZQMEAgGhCgYIKoZIzj0EAwIwLwYJKoZIhvcNAQkEMSIEINRuT6QZlwurSiEyAKJ6f
J0RRujEuseHS7s+IvKZNsDwMAoGCCqGSM49BAMCBEgwRgIhAMOwACcuWbaC1wY0w8saXsU9O
s94a+w7O456xIJ6OC6CAiEAolr3LxqAr3v2HGnR6RUvfr67XLNHGMATQ7j1uPlzUjkAAAAAA
AA=","header":{"publicKeyHash":"kNfaKFR9Oad5u80zOYLcK9ZWw4T5ZY8XdroU4l8S
Y6k=","ephemeralPublicKey":"MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEU95jmBYL
B7aGq1Z7IyMMTbE4PPXKj6Q8xZ9IvnuysurrIGVuUBWU3IlT+O5xqRCvh9SBWxRxg6PK0ace
eaLSOA==","transactionId":"a5a00ce5542237beddd513108b5d14232276ebb238de0
1d0f816cce442630ffd"},"version":"EC_v1"},"paymentMethod":{"displayName":
"MasterCard
0049","network":"MasterCard","type":"credit"},"transactionIdentifier":"A
5A00CE5542237BEDDD513108B5D14232276EBB238DE01D0F816CCE442630FFD"}
```


You must base64 this value prior to sending to Encoded. Example below:


```
eyJwYXltZW50RGF0YSI6eyJkYXRhIjoiL1U0ekVXd2hTaGM5R0NpSm9yanQxV092SUMvSHg2
ZHdkVUdMTFFVanRiSXNHNFVqdE42MW92RjFZMFFtNDN6ZVhuMXhvZ0ljVFZvRzhxZDBob1d6
ZHFvOC84SHFrV1BnWXdvK0NLbFBnaDNpcENHeStHZFBFRXc5MFJYZEVTWjhXd2UxWmhHdG9q
S3NBOFNOZzRwbFBWTlBHYjlrR2IwdUZQUVEzU2JnNzc5NHJuMVJtT0ZjdUU3KzNWbllKZUI3
bHZOb1Y3aDJxMmk5ME9XVFhubGlZeS9JcFZCVjF1QTNEQktIcXplQXNaMjV2dUlXaTZIQ1pC
UGxHOGNNckwwQmJIZE53N011L2doSXNZTFd5T1NyT1FwbUlHTERaOENBVnhWV1NPTGVRL0pS
MVp1dnhqVGhhZUErd3JUY2hmWmlWK2luY29lMnBKSmhieDJkQ1orZG1zcHMzTmU5THhZR3h4
STEwUm5acEREb05CVXdHN2NPWDVqMDRPR29uZUdKSWMrMklqVFY3R0gwb3NmWiIsInNpZ25h
dHVyZSI6Ik1JQUdDU3FHU0liM0RRRUhBcUNBTUlBQ0FRRXhEVEFMQmdsZ2hrZ0JaUU1FQWdF
d2dBWUpLb1pJaHZjTkFRY0JBQUNnZ0RDQ0ErTXdnZ09Jb0FNQ0FRSUNDRXd3UVVsUm5WUTJN
QW9HQ0NxR1NNNDlCQU1DTUhveExqQXNCZ05WQkFNTUpVRndjR3hsSUVGd2NHeHBZMkYwYVc5
dUlFbHVkR1ZuY21GMGFXOXVJRU5CSUMwZ1J6TXhKakFrQmdOVkJBc01IVUZ3Y0d4bElFTmxj
blJwWm1sallYUnBiMjRnUVhWMGFHOXlhWFI1TVJNd0VRWURWUVFLREEwQmNIQnNaU0JKYm1N
dU1Rc3dDUVlEVlFRR0V3SlZVekFlRncweE9UQTFNVGd3TVRNeU5UZGFGdzB5TkRBMU1UWXdN
VE15TlRkYU1GOHhKVEFqQmdOVkJBTU1IR1ZqWXkxemJYQXRZbkp2YTJWeUxYTnBaMjVmVlVN
MExWQlNUMFF4RkRBU0JnTlZCQXNNQzJsUFV5QlRlWE4wWlcxek1STXdFUVlEVlFRS0RBcEJj
```


SEJzWlNCSmJtTXVNUXN3Q1FZRFZRUUdFd0pWVXpCWk1CTUdCeXFHU000OUFnRUdDQ3FHU000
OUF3RUhBMElBQk1JVmQrM3Ixc2V5SVk5bzNYQ1FvU0dOeDdDOWJ5d29QWVJnbGRsSzlLVkJH
NE5DRHRnUjgwQitnek1mSEZURDkrc3lJTmE2MWRUdjlKS0ppVDU4RHhPamdnSVJNSUlDRFRB
TUJnTlZIUk1CQWY4RUFqQUFNQjhHQTFVZEl3UVlNQmFBRkNQeVNjUlBrK1R2SitiRTlpaHNQ
Nks3L1M1TE1FVUdDQ3NHQVFVRkJ3RUJCRGt3TnpBMUJnZ3JCZ0VGQlFjd0FZWXBhSFIwY0Rv
dkwyOWpjM0F1WVhCd2JHVXVZMjl0TDI5amMzQXdOQzFoY0hCc1pXRnBZMkV6TURJd2dnRWRC
Z05WSFNBRWdnRVVNSUlCRURDQ0FRd0dDU3FHU0liM1kyUUZBVENCL2pDQnd3WUlLd1lCQlFV
SEFnSXdnYllNZ2JOU1pXeHBZVzVqWlNCdmJpQjBhR2x6SUdObGNuUnBabWxqWVhSbElHSjVJ
R0Z1ZVNCd1lYSjBlU0JoYzNOMWJXVnpJR0ZqWTJWd2RHRnVZMlVnYjJZZ2RHaGxJSFJvWlc0
Z1lYQndiR2xqWVdKc1pTQnpkR0Z1WkdGeVpDQjBaWEp0Y3lCaGJtUWdZMjl1WkdsMGFXOXVj
eUJ2WmlCMWMyVXNJR05sY25ScFptbGpZWFJsSUhCdmJHbGplU0JoYm1RZ1kyVnlkR2xtYVdO
aGRHbHZiaUJ3Y21GamRHbGpaU0J6ZEdGMFpXMWxiblJ6TGpBMkJnZ3JCZ0VGQlFjQ0FSWXFh
SFIwY0RvdkwzZDNkeTVoY0hCc1pTNWpiMjB2WTJWeWRHbG1hV05oZEdWaGRYUm9iM0pwZEhr
dk1EUUdBMVVkSHdRdE1Dc3dLYUFub0NXR0kyaDBkSEE2THk5amNtd3VZWEJ3YkdVdVkyOXRM
MkZ3Y0d4bFlXbGpZVE11WTNKc01CMEdBMVVkRGdRV0JCU1VWOXR2MVhTQmhvbUpkaTkrVjRV
SDU1dFlKREFPQmdOVkhROEJBZjhFQkFNQ0I0QXdEd1lKS29aSWh2ZGpaQVlkQkFJRkFEQUtC
Z2dxaGtqT1BRUURBZ05KQURCR0FpRUF2Z2xYSCtjZUhuTmJWZVd2ckxUSEwrdEVYekFZVWlM
SEpSQUN0aDY5YjFVQ0lRRFJpelVLWGRiZGJyRjBZRFd4SHJMT2g4K2o1cTlzdllPQWlRM0lM
TjJxWXpDQ0F1NHdnZ0oxb0FNQ0FRSUNDRWx0TDc4Nm1OcVhNQW9HQ0NxR1NNNDlCQU1DTUdj
eEd6QVpCZ05WQkFNTUVrRndjR3hsSUZKdmIzUWdRMEVnTFNCSE16RW1NQ1FHQTFVRUN3d2RR
WEJ3YkdVZ1EyVnlkR2xtYVdOaGRHbHZiaUJCZFhSb2IzSnBkSGt4RXpBUkJnTlZCQW9NQ2tG
d2NHeGxJRWx1WXk0eEN6QUpCZ05WQkFZVEFsVlRNQjRYRFRFME1EVXdOakl6TkRZek1Gb1hE
VEk1TURVd05qSXpORFl6TUZvd2VqRXVNQ3dHQTFVRUF3d2xRWEJ3YkdVZ1FYQndiR2xqWVhS
cGIyNGdTVzUwWldkeVlYUnBiMjRnUTBFZ0xTQkhNekVtTUNRR0ExVUVDd3dkUVhCd2JHVWdR
MlZ5ZEdsbWFXTmhkR2x2YmlCQmRYUm9iM0pwZEhreEV6QVJCZ05WQkFvTUNrRndjR3hsSUVs
dVl5NHhDekFKQmdOVkJBWVRBbFZUTUZrd0V3WUhLb1pJemowQ0FRWUlLb1pJemowREFRY0RR
Z0FFOEJjUmhCblhaSVhWR2w0bGdRZDI2SUNpNzk1N3JrM2dqZnhMaytFelZ0Vm1Xeld1SXRD
WGRnMGlUbnU2Q1AxMkY4Nkl5M2E3Wm5DK3lPZ3BoUDlVUmFPQjl6Q0I5REJHQmdnckJnRUZC
UWNCQVFRNk1EZ3dOZ1lJS3dZQkJRVUhNQUdHS21oMGRIQTZMeTl2WTNOd0xtRndjR3hsTG1O
dmJTOXZZM053TURRdFlYQndiR1Z5YjI5MFkyRm5NekFkQmdOVkhRNEVGZ1FVSS9KSnhFK1Q1
TzhuNXNUMktHdy9vcnY5TGtzd0R3WURWUjBUQVFIL0JBVXdBd0VCL3pBZkJnTlZIU01FR0RB
V2dCUzdzTjZoV0RPSW1xU0ttZDYrdmV1djJzc2txekEzQmdOVkhSOEVNREF1TUN5Z0txQW9o
aVpvZEhSd09pOHZZM0pzTGtzd0R3WURWUjBUQVFIL0JBVXdBd0VCL3pBZkJnTlZIU01FR0RB
T0JnTlZIUThCQWY4RUJBTUNBUVl3RUFZS0tvWklodmRqWkFZQ0RnUUNCUUF3Q2dZSUtvWkl6
ajBFQXdJRFp3QXdaQUl3T3M5eWcxRVdtYkdHK3pYRFZzcGl2L1FYN2RrUGRVMmlqcjd4bklG
ZVFyZUorSmozbTFtZm1OVkJEWStkNmNMK0FqQXlMZFZFSWJDakJYZHNYZk00TzVCbi9SZDhM
Q0Z0bGsvR2NtbUNFbTlVK0hwOUc1bkxtd21KSVdFR21ROEpraDBBQURHQ0FZa3dnZ0dGQWdF
Qk1JR0dNSG94TGpBc0JnTlZCQU1NSlVGd2NHeGxJRUZ3Y0d4cFkyRjBhVzl1SUVsdWRHVm5j
bUYwYVc5dUlFTkJJQzBnUnpNeEpqQWtCZ05WQkFzTUhVRndjR3hsSUVObGNuUnBabWxqWVhS
cGIyNGdRWFYwYUc5eWFYUjVNUk13RVFZRFZRUUtEQXBCY0hCc1pTQkpibU11TVFzd0NRWURW
UVFHRXdKVlV3SUlUREJCU1ZHZFZEWXdDd1lKWUlaSUFXVURCQUlCb0lHVE1CZ0dDU3FHU0li
M0RRRUpBekVMQmdrcWhraUc5dzBCQndFd0hBWUpLb1pJaHZjTkFRa0ZNUThYRFRJek1EY3dN
ekUwTXpBeE1Wb3dLQVlKS29aSWh2Y05BUWswTVJzd0dUQUxCZ2xnaGtnQlpRTUVBZ0doQ2dZ

```
SUtvWkl6ajBFQXdJd0x3WUpLb1pJaHZjTkFRa0VNU0lFSU5SdVQ2UVpsd3VyU2lFeUFLSjZm
SjBSUnVqRXVzZUhTN3MrSXZLWk5zRHdNQW9HQ0NxR1NNNDlCQU1DQkVnd1JnSWhBTU93QUNj
dVdiYUMxd1kwdzhzYVhzVTlPczk0YSt3N080NTZ4SUo2T0M2Q0FpRUFvbHIzTHhxQXIzdjJI
R25SNlJVdmZyNjdYTE5IR01BVFE3ajF1UGx6VWprQUFBQUFBQUE9IiwiaGVhZGVyIjp7InB1
YmxpY0tleUhhc2giOiJrTmZhS0ZSOU9hZDV1ODB6T1lMY0s5Wld3NFQ1Wlk4WGRyb1U0bDhT
WTZrPSIsImVwaGVtZXJhbFB1YmxpY0tleSI6Ik1Ga3dFd1lIS29aSXpqMENBUVlJS29aSXpq
MERBUWNEUWdBRVU5NWptQllMQjdhR3ExWjdJeU1NVGJFNFBQWEtqNlE4eFo5SXZudXlzdXJy
SUdWdVVCV1UzSWxUK081eHFSQ3ZoOVNCV3hSeGc2UEswYWNlZWFMU09BPT0iLCJ0cmFuc2Fj
dGlvbklkIjoiYTVhMDBjZTU1NDIyMzdiZWRkZDUxMzEwOGI1ZDE0MjMyMjc2ZWJiMjM4ZGUw
MWQwZjgxNmNjZTQ0MjYzMGZmZCJ9LCJ2ZXJzaW9uIjoiRUNfdjEifSwicGF5bWVudE1ldGhv
ZCI6eyJkaXNwbGF5TmFtZSI6Ik1hc3RlckNhcmQgMDA0OSIsIm5ldHdvcmsiOiJNYXN0ZXJD
YXJkIiwidHlwZSI6ImNyZWRpdCJ9LCJ0cmFuc2FjdGlvbklkZW50aWZpZXIiOiJBNUEwMENF
NTU0MjIzN0JFRERENTEzMTA4QjVEMTQyMzIyNzZFQkIyMzhERTAxRDBGODE2Q0NFNDQyNjMw
RkZEIn0=
```


The ApplePayPayment object also can potentially contain billing and shipping data. If you wish to use this data alongside the payment, you will need to convert the data returned into the equivalent Encoded Gateway API fields and send them down as a part of the transaction in the *billingCustomer* and *shippingCustomer* fields.

Submitting the Payment

You will then need to send this token, along the rest of your standard transaction fields, to the Encoded Gateway as an *apple_pay* source.

An example *pay* action with the *apple_pay* source:

```
POST /transactions
{
    "object": "transaction.request",
    "action": "pay",
    "ref": "trans-1234",
    "amount": 20.54,
    "currency": "GBP",
    "source": {
        "object": "source",
        "apple_pay": {
            "object":"apple_pay",
            "token":
"dkwyOWpjM0F1WVhCd2JHVXVZMjl0TDI5amMzQXdOQzFoY0hCc1pXRnBZMkV6TURJd2dnRWR
CZ05WSFNBRWdnRVVNSUlCRURDQ0FRd0dDU3FHU0liM1kyUUZBVENCL2pDQnd3WUlLd1lCQlF
VSEFnSXdnYllNZ2JOU1pXeHBZVzVqWlNCdmJpQjBhR2x6SUdObGNuUnBabWxqWVhSbElHSjV
JR0Z1ZVNCd1lYSjBlU0JoYzNOMWJXVnpJR0ZqWTJWd2RHRnVZMlVnYjJZZ2RHaGxJSFJvWlc
```

0Z1lYQndiR2xqWVdKc1pTQnpkR0Z1WkdGeVpDQjBaWEp0Y3lCaGJtUWdZMjl1WkdsMGFXOXV
jeUJ2WmlCMWMyVXNJR05sY25ScFptbGpZWFJsSUhCdmJHbGplU0JoYm1RZ1kyVnlkR2xtYVd
OaGRHbHZiaUJ3Y21GamRHbGpaU0J6ZEdGMFpXMWxiblJ6TGpBMkJnZ3JCZ0VGQlFjQ0FSWXF
hSFIwY0RvdkwzZDNkeTVoY0hCc1pTNWpiMjB2WTJWeWRHbG1hV05oZEdWaGRYUm9iM0pwZEh
rdk1EUUdBMVVkSHdRdE1Dc3dLYUFub0NXR0kyaDBkSEE2THk5amNtd3VZWEJ3YkdVdVkyOXR
MMkZ3Y0d4bFlXbGpZVE11WTNKc01CMEdBMVVkRGdRV0JCU1VWOXR2MVhTQmhvbUpkaTkrVjR
VSDU1dFlKREFPQmdOVkhROEJBZjhFQkFNQ0I0QXdEd1lKS29aSWh2ZGpaQVlkQkFJRkFEQUt
CZ2dxaGtqT1BRUURBZ05KQURCR0FpRUF2Z2xYSCtjZUhuTmJWZVd2ckxUSEwrdEVYekFZVWl
MSEpSQUN0aDY5YjFVQ0lRRFJpelVLWGRiZGJyRjBZRFd4SHJMT2g4K2o1cTlzdllPQWlRM0l
MTjJxWXpDQ0F1NHdnZ0oxb0FNQ0FRSUNDRWx0TDc4Nm1OcVhNQW9HQ0NxR1NNNDlCQU1DTUd
jeEd6QVpCZ05WQkFNTUVrRndjR3hsSUZKdmIzUWdRMEVnTFNCSE16RW1NQ1FHQTFVRUN3d2R
RWEJ3YkdVZ1EyVnlkR2xtYVdOaGRHbHZiaUJCZFhSb2IzSnBkSGt4RXpBUkJnTlZCQW9NQ2t
Gd2NHeGxJRWx1WXk0eEN6QUpCZ05WQkFZVEFsVlRNQjRYRFRFME1EVXdOakl6TkRZek1Gb1h
EVEk1TURVd05qSXpORl6TUZvd2VqRXVNQ3dHQTFVRUF3d2xRWEJ3YkdVZ1FYQndiR2xqWVh
ScGIyNGdTVzUwWldkeVlYUnBiMjRnUTBFZ0xTQkhNekVtTUNRR0ExVUVDd3dkUVhCd2JHVWd
RMlZ5ZEdsbWFXTmhkR2x2YmlCQmRYUm9iM0pwZEhreEV6QVJCZ05WQkFvTUNrRndjR3hsSUV
sdVl5NHhDekFKQmdOVkJBWVRBbFZUTUZrd0V3WUhLb1pJemowQ0FRWUlLb1pJemowREFRY0R
RZ0FFOEJjUmhCblhaSVhWR2w0bGdRZDI2SUNpNzk1N3JrM2dqZnhMaytFelZ0Vm1Xeld1SXR
DWGRnMGlUbnU2Q1AxMkY4Nkl5M2E3Wm5DK3lPZ3BoUDlVUmFPQjl6Q0I5REJHQmdnckJnRUZ
CUWNCQVFRNk1EZ3dOZ1lJS3dZQkJRVUhNQUdHS21oMGRIQTZMeTl2WTNOd0xtRndjR3hsTG1
OdmJTOXZZM053TURRdFlYQndiR1Z5YjI5MFkyRm5NekFkQmdOVkhRNEVGZ1FVSS9KSnhFK1Q
1TzhuNXNUMktHdy9vcnY5TGtzd0R3WURWUjBUQVFIL0JBVXdBd0VCL3pBZkJnTlZIU01FR0R
BV2dCUzdzTjZoV0RPSW1xU0ttZDYrdmV1djJzc2txekEzQmdOVkhSOEVNREF1TUN5Z0txQW9
oaVpvZEhSd09pOHZZM0pzTG1Gd2NHeGxMbU52YlM5aGNIQnNaWEp2YjNSallXY3pMbU55YkR
BT0JnTlZIUThCQWY4RUJBTUNBUVl3RUFZS0tvWklodmRqWkFZQ0RnUUNCUUF3Q2dZSUtvWkl
6ajBFQXdJRFp3QXdaQUl3T3M5eWcxRVdtYkdHK3pYRFZzcGl2L1FYN2RrUGRVMmlqcjd4bkl
GZVFyZUorSmozbTFtZm1OVkJEWStkNmNMK0FqQXlMZFZFSWJDakJYZHNYZk00TzVCbi9SZDh
MQ0Z0bGsvR2NtbUNFbTlVK0hwOUc1bkxtd21KSVdFR21ROEpraDBBQURHQ0FZa3dnZ0dGQWd
FQk1JR0dNSG94TGpBc0JnTlZCQU1NSlVGd2NHeGxJRUZ3Y0d4cFkyRjBhVzl1SUVsdWRHVm5
jbUYwYVc5dUlFTkJJQzBnUnpNeEpqQWtCZ05WQkFzTUhVRndjR3hsSUVObGNuUnBabWxqWVh
ScGIyNGdRWFYwYUc5eWFYUjVNUk13RVFZRFZRUUtEQXBCY0hCc1pTQkpibU11TVFzd0NRWUR
WUVFHRXdKVlV3SUlUREJCU1ZHZFZEWXdDd1lKWUlaSUFXVURCQUlCb0lHVE1CZ0dDU3FHU0l
iM0RRRUpBekVMQmdrcWhraUc5dzBCQndFd0hBWUpLb1pJaHZjTkFRa0ZNUThYRFRJek1EY3d
NekUwTXpBeE1Wb3dLQVlKS29aSWh2Y05BUWswTVJzd0dUQUxCZ2xnaGtnQlpRTUVBZ0doQ2d
ZSUtvWkl6ajBFQXdJd0x3WUpLb1pJaHZjTkFRa0VNU0lFSU5SdVQ2UVpsd3VyU2lFeUFLSjZ
mSjBSUnVqRXVzZUhTN3MrSXZLWk5zRHdNQW9HQ0NxR1NNNDlCQU1DQkVnd1JnSWhBTU93QUN
jdVdiYUMxd1kwdzhzYVhzVTlPczk0YSt3N080NTZ4SUo2T0M2Q0FpRUFvbHIzTHhxQXIzdjJ
IR25SNlJVdmZyNjdYTE5IR01BVFE3ajF1UGx6VWprQUFBQUFBQUE9IiwiaGVhZGVyIjp7InB
1YmxpY0tleUhhc2giOiJrTmZhS0ZSOU9hZDV1ODB6T1lMY0s5Wld3NFQ1Wlk4WGRyb1U0bDh
TWTZrPSIsImVwaGVtZXJhbFB1YmxpY0tleSI6Ik1Ga3dFd1lIS29aSXpqMENBUVlJS29aSXp
qMERBUWNEUWdBRVU5NWptQllMQjdhR3ExWjdJeU1NVGJFNFBQWEtqNlE4eFo5SXZudXlzdXJ
ySUdWdVVCV1UzSWxUK081eHFSQ3ZoOVNCV3hSeGc2UEswYWNlZWFMU09BPT0iLCJ0cmFuc2F
jdGlvbklkIjoiYTVhMDBjZTU1NDIyMzdiZWRkZDUxMzEwOGI1ZDE0MjMyMjc2ZWJiMjM4ZGU

```
wMWQwZjgxNmNjZTQ0MjYzMGZmZCJ9LCJ2ZXJzaW9uIjoiRUNfdjEifSwicGF5bWVudE1ldGh
vZCI6eyJkaXNwbGF5TmFtZSI6Ik1hc3RlckNhcmQgMDA0OSIsIm5ldHdvcmsiOiJNYXN0ZXJ
DYXJkIiwidHlwZSI6ImNyZWRpdCJ9LCJ0cmFuc2FjdGlvbklkZW50aWZpZXIiOiJBNUEwMEN
FNTU0MjIzN0JFRERENTEzMTA4QjVEMTQyMzIyNzZFQkIyMzhERTAxRDBGODE2Q0NFNDQyNjM
wRkZEIn0="
        }
    }
}
```


Google Pay™

Google Pay is the fast, simple way to pay on sites, in apps, and in stores using the cards saved to your Google Account.

Google Pay makes it easy for your customers to complete their purchase on your checkout. A customer with a compatible mobile device or web browser can be offered Google Pay as a payment option, quickly and easily selecting the cards already saved within their Google Pay account.

Checkout Implementation

Please see the Google Pay API documentation for information on how to integrate Google Pay into your checkout.

Android

- Android Developer Documentation
- Android Integration Checklist
- Android Brand Guidelines

Web

- Web Developer Documentation
- Web Integration Checklist
- Web Brand Guidelines

Using the Google Pay API

When requesting a payment token for your payment provider, you must set the *type* to **'PAYMENT_GATEWAY'**, the *gateway* to **'encoded'**, and the *gatewayMerchantId* to your Client ID for the Encoded Gateway.

```
const tokenizationSpecification = {
  type: 'PAYMENT_GATEWAY',
```

```
  parameters: {
    'gateway': 'encoded',
    'gatewayMerchantId': 'ENCODED_GATEWAY_CLIENT_ID'
  }
}
```

You will also need to define the card networks accepted by your site. Encoded currently supports VISA, MASTERCARD and AMEX.

```
const allowedCardNetworks = ["AMEX", "MASTERCARD", "VISA"];
```

You will also need to define which authentication methods are accepted by your site. Encoded currently supports both PAN_ONLY and CRYPTOGRAM_3DS:

```
const allowedCardAuthMethods = ["PAN_ONLY", "CRYPTOGRAM_3DS"];
```

Receiving the Payment Token from Google Pay

Once your checkout has completed the Google Pay process, an encrypted payload will be returned to your application in the PaymentData response. The following fields will need to be sent to the Encoded Gateway:

```
paymentData.paymentMethodData.tokenizationData.token
```

An example of the value contained in this field is below:

```
{
  "protocolVersion":"ECv2",
"signature":"MEUCIG39tbaQPwJe28U+UMsJmxUBUWSkwlOv9Ibohacer+CoAiEA8Wuq3lL
UCwLQ06D2kErxaMg3b/oLDFbd2gcFze1zDqU\u003d",
  "intermediateSigningKey":{
    "signedKey":
"{\"keyExpiration\":\"1542394027316\",\"keyValue\":\"MFkwEwYHKoZIzj0CAQY
IKoZIzj0DAQcDQgAE/1+3HBVSbdv+j7NaArdgMyoSAM43yRydzqdg1TxodSzA96Dj4Mc1EiK
roxxunavVIvdxGnJeFViTzFvzFRxyCw\\u003d\\u003d\"}",
    "signatures":
["MEYCIQDcXCoB4fYJF3EolxrE2zB+7THZCfKA7cWxSztKceXTCgIhAN/d5eBgx/1A6qKBdH
0IS7/aQ7dO4MuEt26OrLCUxZnl"]
  },

"signedMessage":"{\"tag\":\"TjkIKzIOvCrFvjf7/aeeL8/FZJ3tigaNnerag68hIaw\
```

```
\u003d\",\"ephemeralPublicKey\":\"BLJoTmxP2z7M2N6JmaN786aJcT/L/OJfuJKQdI
XcceuBBZ00sf5nm2+snxAJxeJ4HYFTdNH4MOJrH58GNDJ9lJw\\u003d\",\"encryptedMe
ssage\":\"mleAf23XkKjj\"}"
}
```


You must base64 this value prior to sending to Encoded. Example below:


```
ewogICJwcm90b2NvbFZlcnNpb24iOiJFQ3YyIiwKInNpZ25hdHVyZSI6Ik1FVUNJRzM5dGJh
UVB3SmUyOFUrVU1zSm14VUJVV1Nrd2xPdjlJYm9oYWNlcitDb0FpRUE4V3VxM2xMVUN3TFEw
NkQya0VyeGFNZzNiL29MREZiZDJnY0Z6ZTF6RHFVXHUwMDNkIiwKICAiaW50ZXJtZWRpYXRl
U2lnbmluZ0tleSI6ewogICAgInNpZ25lZEtleSI6ICJ7XCJrZXlFeHBpcmF0aW9uXCI6XCIx
NTQyMzk0MDI3MzE2XCIsXCJrZXlwYWx1ZVwiOlwiTUZrd0V3WUhLb1pJemowQ0FRWUlLb1pJ
emowREFRY0RRZ0FFLzErM0hCVlNiZHYrajdOYUFyZGdNeW9TQU00M3lSeWR6cWRnMVR4b2RT
ekE5NkRqNE1jMUVpS3JveHh1bmF2Vkl2ZHhHbkplRlZpVHpGdnpGUnh5Q3dcXHUwMDNkXFx1
MDAzZFwifSIsCiAgICAic2lnbmF0dXJlcyI6IFsiTUVZQ0lRRGNYQ29CNGZZSkYzRW9seHJF
MnpCKzdUSFpDZktBN2NXeFN6dEtjZVhUQ2dJaEFOL2Q1ZUJneC8xQTZxS0JkSDBJUzcvYVE3
ZE80TXVFdDI2T3JMQ1V4Wm5sIl0KICB9LAogICJzaWduZWRNZXNzYWdlIjoie1widGFnXCI6
XCJUamtJS3pJT3ZDckZ2amY3L2FlZUw4L0ZaSjN0aWdhTm5lcmFnNjhoSWF3XFx1MDAzZFwi
LFwiZXBoZW1lcmFsUHVibGljS2V5XCI6XCJCTEpvVG14UDJ6N00yTjZKbWFONzg2YUpjVC9M
L09KZnVKS1FkSVhjY2V1QkJaMDBzZjVubTIrc254QUp4ZUo0SFlGVGROSDRNT0pySDU4R05E
SjlsSndcXHUwMDNkXCIsXCJlbmNyeXB0ZWRNZXNzYWdlXCI6XCJtbGVBZjIzWGtLampcIn0i
Cn0=
```


Submitting the Payment

You will then need to send this token, along the rest of your standard transaction fields, to the Encoded Gateway as a google_pay source.

An example *pay* action with the *google_pay* source:

```
POST /transactions
{
    "object": "transaction.request",
    "action": "pay",
    "ref": "trans-1234",
    "amount": 20.54,
    "currency": "GBP",
    "source": {
        "object": "source",
        "google_pay": {
```

```
            "object":"google_pay",
            "token":
"ewogICJwcm90b2NvbFZlcnNpb24iOiJFQ3YyIiwKICAic2lnbmF0dXJlIjoiTUVRQ0lINlE
0T3dRMGpBY2VGRWtHRjBKSUQ2c0pOWHhPRWk0cittQTdiaVJ4cUJRQWlBb25kcW9VcFUvYmR
zckFPcFpJc3JIUVM5bndpaU53T3JyMjRSeVBlSEEwUVx1MDAzZFx1MDAzZCIsCiAgImludGV
ybWVkaWF0ZVNpZ25pbmdLZXkiOnsKICAgICJzaWduZWRLZXkiOiAie1wia2V5RXhwaXJhdGl
vblwiOlwiMTU0MjMyMzM5MzE0N1wiLFwia2V5VmFsdWVcIjpcIk1Ga3dFd1lIS29aSXpqMEN
BUVlJS29aSXpqMERBUWNEUWdBRS8xKzNIQlZTYmR2K2o3TmFBcmRnTXlvU0FNNDN5UnlkenF
kZzFUeG9kU3pBOTZEajRNYzFFaUtyb3h4dW5hdlZJdmR4R25KZUZWaVR6RnZ6RlJ4eUN3XFx
1MDAzZFxcdTAwM2RcIn0iLAogICAgInNpZ25hdHVyZXMiOiBbIk1FWUNJUUNPMkVJaTQ4czh
WVEgraWxNRXBvWExGZmt4QXdIamZQU0NWRUQvUURTSG1RSWhBTExKbXJVbE5BWThoRFFSVi9
5MWlLWkdzV3BlTm1JUCt6K3RDUUhReFAwdiJdCiAgfSwKICAic2lnbmVkTWVzc2FnZSI6Int
cInRhZ1wiOlwianBHejFGMUJjb2kvZkNOeEk5bjdRcnN3N2k3S0hyR3RUZjNOclJjbHQrVVx
cdTAwM2RcIixcImVwaGVtZXJhbFB1YmxpY0tleVwiOlwiQkphdHlGdkZQUEQyMWw4L3VMUDQ
2VGExaHNLSG5kZjhaK3RBZ2srREVQUWdZVGtoSHkxOWNGM2gvYlhzMHRXVG1adG5ObSt2bFZ
yS2JSVTlLOCs3Y1pzXFx1MDAzZFwiLFwiZW5jcnlwdGVkTWVzc2FnZVwiOlwibUtPb1h3aTh
PYXZaXCJ9Igp9"
        }
    }
}
```


If you have chosen *PAN_ONLY* as an allowed authentication method, then you may receive a challenge response following the transaction request. Please see the EMV 3-D Secure (3DS2) section for details on how to deal with a 3DS2 challenge response.

EMV 3-D Secure (3DS2)

3DS2 provides many improvements over 3DS1 and provides additional assurances around the identity of the cardholder, protecting both merchants and cardholders from credit and debit card fraud. 3DS2 works by taking the cardholder through a number of different types of authentication flows based on the perceived risk of the transactions.

The Gateway API will determine the correct 3DS version to perform based on the configuration of the merchant account.

3DS2 is performed in a similar manner to 3DS1, through a number of steps:

1. **Initiate 3D Secure Transaction**
   A transaction is initiated with the *platformType* field set to "*ecom"* and the *threeDSecure* field set to *true*.
2. **Receive Challenge**
   The Gateway API responds to a transaction request with a 3DS2 challenge.
3. **Redirect to ACS**
   The implementer redirects the cardholder's browser to the Gateway API Access Control Server (ACS) for the URL provided. The ACS will then potentially take the cardholder through a number of challenges.
4. **Process ACS Response**
   The ACS will redirect the cardholder's browser back to the implementer who must process the authentication response.
5. **Send Challenge Response**
   The implementer sends the information received from the ACS in the authentication response to the Gateway API. The transaction will then continue to be processed.

Initiate 3D Secure Transaction

A transaction is performed with the *platformType* set to *"ecom"* and the *threeDSecure* field set to *true*.

```
POST /transactions
{
    "object": "transaction.request",
    "action": "authorise",
    "ref": "trans-1234",
    "amount": 20.54,
    "currency": "GBP",
    "platformType": "ecom",
    "threeDSecure": true,
```

```
    "source": {
        "object": "source",
        "card": {
            "object": "card",
            "pan": "4444333322221111",
            "expiry": "2022-10",
            "securityCode": "111"
        }
    }
}
```

Receive Challenge

When a supported transaction is initiated and the merchant account is configured for e-commerce transactions and 3DS2, the Gateway API may respond to the transaction request with a challenge via the transaction.challenge object. The challenge object (if a 3DS challenge) will contain all of the information required to send to the Gateway API ACS. Below is an example response that includes a 3DS2 challenge:

```
{
    "object": "transaction",
    "id": "4b502950-8801-4112-ba15-88f8eb280525",
    "creationDate": "2020-07-01T00:00:00Z",
    "status": "challenged",
    "challenge": {
        "object": "transaction.challenge",
        "id": "1",
        "threeDSecure": {
            "object": "transaction.challenge.threeDSecure",
            "v2": {
                "object": "transaction.challenge.threeDSecure.v2",
                "acsUrl":
"https://sit.encoded.services/emv3ds/acs/init/4b502950-8801-4112-ba15-88
f8eb280525"
            }
        },
        "links": {
            "self":
"https://sit.encoded.services/api/v1/transactions/4b502950-8801-4112-ba1
5-88f8eb280525/challenge/1"
        }
```

```
    },
    "attributes": {},
    "links": {
        "self":
"https://sit.encoded.services/api/v1/transactions/d9d97dd1-d12b-4071-97c
7-ab8c7022dad6"
    }
}
```


In the example above, the transaction has returned with a *status* of **challenged**. The challenge itself is contained within the *challenge* object. The *challenge* object contains a *threeDSecure.v2* object which contains the fields necessary to perform the 3DSecure authentication. These are:

- **acsUrl**
  The URL of the Gateway API Access Control Server (ACS). The implementer will perform a POST to this URL within the cardholder's browser.

Redirect to ACS

The implementer must redirect the cardholder's browser to the Gateway API Access Control Server at the URL provided. The implementer must provide the callback URL for the ACS to send the authentication response to once the process is complete. This is provided as part of the POST request, with the name *returnUrl*. Additionally, the implementer may provider the size of the challenge window. This is provided as part of the POST request with the name *challengeWindowSize*.

Below is an example form that can be constructed by the implementer to perform the redirection to the Gateway API ACS:

```
<form name="3dsRedirect"
action="https://sit.encoded.services/emv3ds/acs/init/4b502950-8801-4112-
ba15-88f8eb280525"
      method="POST"
      accept-charset="UTF-8">
  <input type="hidden"
         name="returnUrl"
         value="https://implementer.com/3ds/callback"/>
  <input type="hidden"
         name="challengeWindowSize"
         value="05"/>
  <input type="submit" value="Click  here to continue" class="button">
```

```
</form>
```

Process ACS Response

Once the authentication process at the Gateway API ACS is complete, the ACS will return an authentication response to the URL provided in the *returnUrl* field, as a HTTP Post. The fields returned will include:

- **dsTransId**
  A unique ID to reference the authorisation attempt.
- **authenticationValue**
  The authentication value provided by the ACS.
- **eci**
  The electronic commerce indicator.
- **transStatus**
  The status of the authorisation attempt.
- **messageVersion**
  The 3DS2 message version.

Send Challenge Response

When the challenge was made, the *transaction.challenge* object contained a link to the challenge response endpoint. The implementer will send a challenge response object to this endpoint, containing the information received from the ACS, as in the example below:

```
POST /transactions/4b502950-8801-4112-ba15-88f8eb280525/challenge/1
{
  "object": "transaction.challenge.response",
  "threeDSecure": {
    "object": "transaction.challenge.response.threeDSecure",
    "v2": {
      "object": "transaction.challenge.response.threeDSecure.v2",
      "dsTransId": "8b167401-2c2e-4f5c-80e0-6dcbf68737cd",
      "authenticationValue": "MTIzNDU2Nzg5MDEyMzQ1OTcyODM=",
      "eci": "05",
      "transStatus": "Y",
      "messageVersion": "2.2.0"
    }
  }
}
```

The challenge endpoint will then respond with a *transaction* object, as if we had just sent a transaction request to the Gateway API. The *transaction* returned will likely have a *transaction.status* of **processed** and will contain a *transaction.response* object.

Multiple Challenges

In some circumstances, the *transaction* returned will again have a *transaction.status* of **challenged**. This indicates that an additional challenge has been requested. This can occur in situations where the issuer has initially performed a 3DS2 frictionless flow, but following the transaction request being sent has subsequently decided to escalate the transaction and request a 3DS2 challenge flow. You should code your implementation accordingly to account for potentially multiple challenge requests.

In the above instance, the *transaction.challenge* field will contain the most recent *challenge* object. The *transaction.challenges* field contains an array of all challenges that have been issued during the life of the transaction.

Hosted Payment Pages

Hosted Payment Pages provide a simple solution for accepting card payments, by providing a drop-in payment form that can handle all aspects of the transaction process on the frontend with little development resource required by the implementer. The solution is also eligible for the lowest level of PCI Compliance - SAQ-A.

Hosted Payment Pages work by using the Gateway API to create an Order, which provides a URL to a hosted payment page capable of presenting the ecom user with all available payment methods, and handling all aspects of the transaction.

Create an Order

The Gateway API will be used to create an Order. This allows the order details to be provided ahead of any transaction being performed, and sets the payment amount, currency, billing customer information, and any required Hosted Payment Pages configuration.

The Hosted Payment Pages configuration can be provided as part of the Order creation, setting the *action* and *returnUrl,* as well as any configuration required to handle tokens.

If no *action* is supplied, *pay* is used as the default action.

Tokenisation

In order to support Tokens, you must provide a valid Customer Object in the *billingCustomer* field that has associated Tokens, along with a *hpp.tokens* Object as part of the Hosted Payment Pages configuration.

The *tokens.enabled* field sets whether any current tokens should be displayed to the user as a valid payment method.

The *tokens.tokenisation* field should be provided with a Tokenisation Object, and setting this will provide the user with the option to save their card details for future use when making a new card transaction.

Example

Below is an example request and response. Only the pertinent information required in the *Order* object has been shown.

Request:

```
POST /orders
{
   "object":"order",
   "ref":"ORD-0372837",
   "description":"Payment Order",
   "currency":"GBP",
   "totalAmount":9.60,
   "billingCustomer":{
      "object":"customer",
      "id":"3fa85f64-5717-4562-b3fc-2c963f66afa6"
   },
   "hpp":{
      "action": "authorise",
      "returnUrl":"https://myurl.com/ecom-response",
      "tokens":{
         "enabled":true,
         "tokenisation":{
            "object":"tokenisation",
            "agreement":"card_on_file",
            "ref":"token-1234"
         }
      }
   }
}
```

Response:

```
{
   "object":"order",
   "id":"3fa85f64-5717-4562-b3fc-2c963f66afa6",
   "creationDate":"2019-07-01T00:00:00Z",
   "ref":"ORD-0372837",
   "description":"Payment Order",
   "currency":"GBP",
   "totalAmount":9.60,
   "billingCustomer":{
      "object":"customer",
      "id":"3fa85f64-5717-4562-b3fc-2c963f66afa6",
      "creationDate":"2019-07-01T00:00:00Z",
      "ref":"CUST-0094637",
```

```
      "title":"Mr",
      "forename":"John",
      "surname":"Doe",
      "dateOfBirth":"1985-07-01",
      "contact":{
         "object":"contact",
         "address":{
            "object":"address",
            "title":"Mr",
            "forename":"John",
            "surname":"Doe",
            "postcode":"AB1 2CD",
            "country":"GBR"
         },
         "email":"email@example.com"
      },
      "links":{

"self":"https://sit.encoded.services/api/v1/customers/96d7d98a-46fd-4d3b
-a15b-b03c116ae949",

"tokens":"https://sit.encoded.services/api/v1/customers/96d7d98a-46fd-4d
3b-a15b-b03c116ae949/tokens"
      }
   },
   "hpp":{
      "action": "authorise",
      "returnUrl":"https://myurl.com/ecom-response",
      "tokens":{
         "enabled":true,
         "tokenisation":{
            "object":"tokenisation",
            "agreement":"card_on_file",
            "ref":"token-1234"
         }
      }
   },
   "links":{

"self":"https://sit.encoded.services/api/v1/orders/96d7d98a-46fd-4d3b-a1
5b-b03c116ae949",
      "hpp":{
```

```
"v1":"https://sit.encoded.services/hpp/v1/96d7d98a-46fd-4d3b-a15b-b03c116ae949"
      }
   }
}
```

Displaying the Hosted Payment Page

The order will respond with a URL in the *links.hpp.v1* field. This URL is the access URL for the Hosted Payment Page specifically created and configured for this Order.

This URL will likely be opened within an iframe embedded on the parent page, and will be responsible for displaying the available payment methods to the user, collecting payment information, and processing the transaction.

Below is an example HTML page containing the Hosted Payment Page.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Payment Form</title>
</head>
<body>
   <iframe
src="https://sit.encoded.services/hpp/v1/96d7d98a-46fd-4d3b-a15b-b03c116ae949"
           frameborder="0"
           width="700"
           height="500"
           style="overflow:hidden"
           sandbox="allow-top-navigation allow-scripts allow-forms"></iframe>
</body>
</html>
```

Note that the *sandbox* attribute has been set on the *iframe* element, and allows *allow-top-navigation*. This allows the *returnUrl* callbacks to be targeted at the top page.

Handling The Response

Once the user has completed the transaction, a HTTP POST call-back will be performed to the *returnUrl* provided. The POST will contain the following form fields:

Field Name	Field Values	Notes
orderId	The ID of the Order that was processed.	This can be used to tie up the call-back with the Order ID provided when creating the Order in the server back-end.
transactionId	The ID of the Transaction that was processed.	This can be used to look-up all additional information relating to the transaction via the Gateway API.
result	One of: ● accepted ● declined ● error	
authCode	If accepted, the authorisation code of the transaction.	This is a reference that can be displayed to the user, and that can be provided back to their issuing bank if required.
tokenId	The ID of any token that may have been created as part of this transaction.	Supplied only if a token was created as part of this transaction. Allows additional token management steps to be performed.

If further information is required, a lookup can be performed against the Gateway API for the full transaction details. See Transactions for more information.

The implementer should display an appropriate page to the user to inform them of the result of the transaction and provide any further information that may be required.

Token Management

The *tokenId* field indicates whether a Token was created as part of this transaction and supplied the ID of the created Token. This is useful if the implementer wants to perform additional token management functionality.

For example, if a *Customer* may only have one active Token on their account, the implementer may wish to use the Gateway API to remove any previously held tokens.

Hosted Payment Fields

In order to be eligible for the lowest level of PCI Compliance - SAQ-A - a payment form must have all elements of the payment page hosted securely by a third-party Level 1 PCI DSS Compliant Service Provider.

It is not enough to simply POST the card details to the service provider. If any part of the page can be modified in any way to intercept or interrupt the gathering and submission of the card details, then the payment form will fall under SAQ-AEP, a far more stringent set of requirements.

An iframe provides a suitable level of protection for the payment form to comply with SAQ-A, but provides a reduced experience for usability and flexibility to the implementers. An iframe may not fit in with the existing user interface, and may not provide the flexibility required to accept additional fields at the same time.

Hosted Payment Fields attempts to bridge the gap between compliance, and the usability and flexibility of the payment form, by rendering each individual input in its own iframe. This allows much more flexibility for implementers to style their payment forms whilst not having to worry about dealing with sensitive card details.

Generating a Payment Session

The Gateway API will be used to create a Payment Session. This provides a Session ID which can be used to collect card details via various methods, and then complete the transaction via the Gateway API.

When creating a Payment Session, the fields which will make up the payment session can be specified. This is useful for situations where you may not require all of the fields, for example when collecting only the stored card to use alongside a token object. Below shows an example payment session being created for all available fields.

Request:

```
HTTP POST https://sit.encoded.services/api/v1/sessions
{
  "object": "session",
  "fields": ["pan", "expiryDate", "securityCode"]
}
```

Response:

```
{
   "object": "session",
```

```
   "id": "10b3ca21-d9fd-4030-b5f5-fb45f220a6dd",
   "creationDate": "2020-09-24T09:43:18.625Z",
   "lastUpdated": "2020-09-24T09:43:18.625Z",
   "sessionFields": [
       {
           "object": "sessionField",
           "name": "pan",
           "value": "",
           "state": "unset",
           "selected": false,
           "additionalParams": {},
           "links": {
               "iframeURL":
"https://sit.encoded.services/hpf/v1/10b3ca21-d9fd-4030-b5f5-fb45f220a6dd/pan"
           }
       },
       {
           "object": "sessionField",
           "name": "expiryDate",
           "value": "",
           "state": "unset",
           "selected": false,
           "additionalParams": {},
           "links": {
               "iframeURL":
"https://sit.encoded.services/hpf/v1/10b3ca21-d9fd-4030-b5f5-fb45f220a6dd/expiryDate"
           }
       },
       {
           "object": "sessionField",
           "name": "securityCode",
           "value": "",
           "state": "unset",
           "selected": false,
           "additionalParams": {},
           "links": {
               "iframeURL":
"https://sit.encoded.services/hpf/v1/10b3ca21-d9fd-4030-b5f5-fb45f220a6dd/securityCode"
           }
       }
   ]
}
```


The response returned provides all of the details currently for this session. The Session ID (returned as the *id* field) is required to use Hosted Payment Fields. Once the Session ID has been generated, this can then be used in the next step to generate the Hosted Payment Field iframes via a Javascript library.

Generate a Session Limited JWT

You must then generate a Session Limited JWT that is safe to send to the browser to initialise the Hosted Payment Fields Javascript. See Authentication for Hosted Payment Fields for details.

Generate the Hosted Payment Fields

Hosted Payment Fields are applied to an existing HTML payment form by using Javascript to generate the iframes and apply them to existing `<div>` containers in the payment form.

A Javascript module is provided by Encoded which creates, initialises and interacts with the Hosted Payment Fields. The latest version is available at:

```
https://[env].encoded.services/assets/js/hpf/hpf-2.1.0.min.js
```

Here is an example basic HTML payment form with Hosted Payment Fields:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Payment Form</title>

    <script src="https://sit.encoded.services/assets/js/hpf/hpf-2.1.0.min.js"></script>
    <script>
      HostedPaymentFields.initialise(JWT_AUTH_TOKEN, {
        sessionId: "10b3ca21-d9fd-4030-b5f5-fb45f220a6dd",
        form: {
          id: "payment-form",
          fields: {
            pan: {
              id: "pan"
            },
            expiry: {
              id: "expiry"
            },
            securityCode: {
              id: "securityCode"
            }
          }
        },
        onEvent: function (event) {
          // Handle the event
        }
```

```
      });
    </script>
  </head>
  <body>
    <form id="payment-form" action="/make-payment" method="post">
      <label for="pan">Card Number</label>
      <div id="pan"></div>

      <label for="expiry">Expiry Date</label>
      <div id="expiry"></div>

      <label for="securityCode">Security Code</label>
      <div id="securityCode"></div>

      <input type="submit" value="Make Payment" />
    </form>
  </body>
</html>
```

The Hosted Payment Fields Javascript module will generate an iframe containing the correct input element within each of the `<div>` containers.

Interact with the Hosted Payment Fields

The Hosted Payment Fields Javascript modules will generate and dispatch synthetic events, which are used to provide the implementer with real-time feedback. This is the primary mechanism with which the implementer can update the frontend in response to input being entered by the user and validated by the Hosted Payment Fields client.

All events dispatched use jQuery event namespace notation, with a namespace of *encodedHpf.* So as an example, for the *initialisationStart* event below, the *type* field of the *Event* object would be *initialisationStart.encodedHpf.*

A full list of events can be found at Appendix 1 - Hosted Payment Fields Events.

Below is a sample of JS to add an event listener for a specific synthetic event.

```
document.addEventListener('fieldStatusChange.encodedHpf', function(theEvent) {
    var target = theEvent.target;
    var detail = theEvent.detail;
    // Add code here.
});
```

Sync Hosted Payment Fields with the Payment Session

At the point where you are ready to perform the transaction - once all fields have been completed by the user and have passed validation - the Hosted Payment Fields must be synced to ensure that the card data is saved against the Payment Session generated.

To do this, you must generate and dispatch a synthetic event from the payment form element supplied when initialising the Hosted Payment Fields, which will be received by the Javascript module, and will trigger the sync process. You may wish to perform this action once the payment form's Submit button has been clicked by the user.

For more information about this event, please refer to the *paymentSessionSyncRequest* in Appendix 1 - Hosted Payment Fields Events.

Below is a sample of JS for generating and dispatching the event:

```
const syncRequest = new CustomEvent('paymentSessionSyncRequest.encodedHpf');
document.getElementById('payment-form').dispatchEvent(syncRequest);
```

You will then receive events to provide feedback on the sync process. If the Payment Session sync was successful, you may then submit the payment.

Submitting the Payment

The Gateway API will then be used to perform the transaction in the normal way, supplying the Session ID as the source of the transaction.

```
POST /transactions
{
   "object": "transaction.request",
   "action": "pay",
   "amount": 1.00,
   "currency": "GBP",
   "source": {
       "object": "source",
       "session": {
           "object": "session",
           "id": "10b3ca21-d9fd-4030-b5f5-fb45f220a6dd"
       }
   },
   "billingCustomer": {
     "object": "customer",
     "forename": "test"
   }
}
```

Which provides the following response:


```
{
   "object": "transaction",
   "id": "b015dd96-a8e5-46e1-b390-af39d0b93960",
   "creationDate": "2020-09-24T10:14:20Z",
   "status": "processed",
   "request": {
       "object": "transaction.request",
       "id": "700ff349-4038-4063-ad45-715ce9d8a48c",
       "creationDate": "2020-09-24T10:14:20Z",
       "action": "pay",
       "order": {
           "object": "order",
           "id": "b13dcdd5-c4c1-4b55-bf8c-9628ff2f5ac5",
           "creationDate": "2020-09-24T10:14:21Z",
           "ref": "b13dcdd5-c4c1-4b55-bf8c-9628ff2f5ac5",
           "currency": "GBP",
           "totalAmount": 1.00,
           "pendingAmount": 0.00,
           "billingCustomer": {
               "object": "customer",
               "id": "c92edae3-7c45-445e-9e56-c5d8a0bde52c",
               "links": {
                   "self":
"https://sit.encoded.services/api/v1/customers/c92edae3-7c45-445e-9e56-c5d8a0bde52c"
               }
           },
           "links": {
               "self":
"https://sit.encoded.services/api/v1/orders/b13dcdd5-c4c1-4b55-bf8c-9628ff2f5ac5"
           }
       },
       "currency": "GBP",
       "amount": 1.00,
       "source": {
           "object": "source"
       },
       "billingCustomer": {
           "object": "customer",
           "id": "c92edae3-7c45-445e-9e56-c5d8a0bde52c",
           "links": {
               "self":
"https://sit.encoded.services/api/v1/customers/c92edae3-7c45-445e-9e56-c5d8a0bde52c"
           }
       },
       "links": {
           "self":
"https://sit.encoded.services/api/v1/transactions/b015dd96-a8e5-46e1-b390-af39d0b93960/r
```

```
equest"
        }
    },
    "response": {
        "object": "transaction.response",
        "id": "913805d4-7ffa-46a8-a9fd-0a64b66c4e60",
        "result": {
            "object": "result",
            "resultType": "accepted",
            "resultCode": "APPROVED",
            "message": "Approved"
        },
        "auth": {
            "object": "auth",
            "code": "006348",
            "date": "2020-09-24T10:14:21.208Z"
        }
    },
    "links": {
        "self":
"https://sit.encoded.services/api/v1/transactions/b015dd96-a8e5-46e1-b390-af39d0b93960"
    },
    "attributes": {}
}
```


Styling

A number of styling options can be applied to the input elements within the generated iframe via the Hosted Payments Fields JS.

Styles can be applied at multiple levels (either at the form level or the field level) and across different statuses (to allow different stylings when a specific status is set for an input field).

Styles are cascading, so that more specific styles override more generic styles. For example, if a grey background is set at the form level, and a white background set under the pan field,, all input elements will have a grey background except for the pan input field, which will have a white background.

The following is an example of the initialisation with styling elements included:

```
HostedPaymentFields.initialise(JWT_AUTH_TOKEN, {
    sessionId: "10b3ca21-d9fd-4030-b5f5-fb45f220a6dd",
    form: {
        id: "payment-form",
        fields: {
            pan: {
```

```
            id: "pan",
            style: {
                default: {
                    height: "30px"
                }
            }
        },
        expiry: {
            id: "expiry",
            style: {
                default: {
                    height: "20px"
                }
            }
        },
        securityCode: {
            id: "securityCode",
            style: {
                default: {
                    height: "20px"
                }
            }
        }
    },
    style: {
        default: {
            font: {
                family: "Tacoma, sans-serif",
                size: "10px",
                stretch: "normal",
                weight: "normal",
                style: "normal",
                variant: "normal"
            },
            background: {
                color: "transparent"
            }
        },
        invalid: {
            background: {
                color: "red"
            }
        }
    }
},
onEvent: function (event) {
    // Handle the event
}
});
```

The following objects and properties exist:

Style

Property	Type	Description
default	StatusStyle	Styles applied by default. These styles are shown for all statuses, unless specifically overridden by a specific status's style.
unset	StatusStyle	Styles applied to input elements with the "unset" status.
invalid	StatusStyle	Styles applied to input elements with the "invalid" status.
valid	StatusStyle	Styles applied to input elements with the "valid" status.

StatusStyle

Property	Type	Description
background	BackgroundStyle	Styles applied to the background of the input element.
color	String	The colour of the input element.
font	FontStyle	Styles applied to the font of the input element.
height	String	The height of the input element.
margin	MarginPaddingStyle	Styles applied to the margins of the input element.
padding	MarginPaddingStyle	Styles applied to the padding of the input element.

BackgroundStyle

Property	Type	Description
clip	String	The clip for the input element.
color	String	The colour of the input element.

FontStyle

Property	Type	Description
family	String	The font families for the input element.

size	String	The font size for the input element.
stretch	String	The font stretch for the input element.
style	String	The font style for the input element.
weight	String	The font weight for the input element.
variant	String	The font variants for the input element.

MarginPaddingStyle

Property	Type	Description
top	String	The top margin/padding for the input element.
right	String	The right margin/padding for the input element.
bottom	String	The bottom margin/padding for the input element.
left	String	The left margin/padding for the input element.

Notifications

To support asynchronous payment flows, a notification call-back can be provided once a transaction has been processed. The notification call-back will be performed to a single URL that can be set at either the account or user level.

The notification call-back will be sent as a HTTP POST request in JSON format to the configured endpoint for either the account or user. An example notification is below:

```
{
  "object": "notification",
  "id": "9799936c-27be-432d-be84-e1249c7cac9e",
  "creationDate": "2021-12-17T10:53:08Z",
  "user": "Encoded1",
  "environment": "prod",
  "payload": {
      "object": "transaction",
      "id": "c7b66223-927d-41fd-afcc-a006d4cecb06"
      ...
  },
  "signature":
"OTA4ZTE1YjNkMzhjNzc4ODUyNTZkOGQ0ZDI5YjgzOTMzYjQ3MTQzM2EwYzNjZWFjZWUxZDE
zOTdmYmJhM2FhMQ=="
}
```

The notification call-back provides a *Notification* object, which contains the date that the notification was performed, the user who performed the action that generated the call-back, the environment that the action was performed on, and the payload of the notification (which is usually the created object).

The *Notification* object also contains a signature, which is a SHA256 HMAC in Base64 representation. This is used to ensure that the notification received came from Encoded and that the key information contained within has not been tampered with. Different payload objects will have differing algorithms for calculating the HMAC signature, detailed below.

Transactions

Notification call-backs can be sent when a transaction has been processed. The resulting payload will be the *Transaction* object.

The signature for this payload can be calculated as follows:

1. **Create the string that will be hashed.**

   This can be done by combining the following fields from the *Notification* object and *Transaction* object:

```
notification_id:user:environment:transaction_id:amount_in_decimal:currency:resultType
```

   From the above example, this would give us a string of:

```
9799936c-27be-432d-be84-e1249c7cac9e:Encoded1:prod:c7b66223-927d-41fd-afcc-a006d4cecb06:10.00:GBP:accepted
```

2. **Perform a SHA256 HMAC on the created string, using the HMAC Key provided.**

   Assuming a secret key of 'secret', performing a SHA256 HMAC on the above string would output:

```
908e15b3d38c77885256d8d4d29b83933b471433a0c3ceacee1d1397fbba3aa1
```

3. **Encode the resulting string as Base64.**

   The above string encoded as Base64 would output:

```
OTA4ZTE1YjNkMzhjNzc4ODUyNTZkOGQ0ZDI5YjgzOTMzYjQ3MTQzM2EwYzNjZWFjZWUxZDEzOTdmYmJhM2FhMQ==
```

4. **Compare the computed string with the *signature* included in the *Notification* object.**

Address Verification Service

The Address Verification Service (AVS) is a security feature that compares the billing address provided as part of the transaction request against the billing address details held by the card issuer. The merchant account can be configured to automatically decline transactions that have failed the AVS check.

In order to support AVS, a *billingCustomer* object must be provided as part of the transaction request, and must contain at least the *billingCustomer.contact.address.postcode* field.

AVS is not supported by all acquirers or card issuers.

Response Codes

The following response codes are available as part of the transaction response.

Response codes provided are translations of the responses received from the underlying gateway and/or card issuer once a transaction has processed. Not all gateways and card issuers will support all response codes. Many gateways and card issuers provide only basic response codes. Responses from the underlying gateway and/or card issuer without a specific response code will be mapped to the closest generic response code.

Code	Description
Validation	
invalid_merchant_account	The merchant account is incorrectly configured. Contact Encoded Support to rectify.
invalid_card_number	The card number supplied is invalid. In this circumstance, the card number provided
invalid_expiry_date	The expiry date supplied is invalid.
invalid_security_code	The security code supplied is invalid.
requires_cardholder_name	If no cardholder name has been provided as part of the billing address for card schemes or transaction types that mandate a cardholder name being provided
insufficient_billing_address	If insufficient billing address details have been provided for card schemes or transaction types that mandate certain address details being provided
invalid_email	The email supplied is invalid.
invalid_phone	The phone number supplied is invalid.
invalid_amount	The amount supplied is invalid.
invalid_currency	The currency supplied is invalid.
invalid_source_transaction	The source transaction referenced could not be found.
expired_source_transaction	The source transaction referenced has expired.

requires_source_transaction	A transaction has been attempted that requires a source transaction.
excessively_captured	The transaction attempts to capture more than was authorised
excessively_refunded	The transaction attempts to refund more than was captured.
cannot_refund	Can not perform a refund on this type of transaction.
cannot_void	Can not perform a void on this type of transaction.
unsupported_currency	Unsupported currency.
expired_card	The card has expired.
requires_security_code	The security code is required.
already_settled	The transaction has already been settled and can not be voided.
already_voided	The transaction has been voided.
invalid_3ds	The 3DS details provided were invalid.
requires_3ds	The transaction was attempted without 3DSecure.
invalid_apple_pay_token	Invalid Apple Pay token has been provided.
invalid_google_pay_token	Invalid Google Pay token has been provided.
expired_google_pay_token	An expired Google Pay token has been provided.
unsupported_source	The source provided for the transaction is unsupported for your account. This could be because it is unsupported by the underlying acquirer and/or gateway configured.
unsupported_recurrence	The type of recurrence attempted is invalid for this card
invalid_request	Generic invalid request response.
Declined	
declined	Non-specific declined response.
declined_insufficient_funds	Declined due to insufficient funds.

referred_retain_card	The card should be retained.
blocked_blacklisted	The card is on a blacklist.
excessively_authed	Card authorisation attempt limit reached.
excessively_declined	The card has reached a limit of declines for the card scheme and should continue to be declined.
cancelled_continuous_authority	The continuous authority for this card has been cancelled.
incorrect_security_code	The security code provided is incorrect.
incorrect_expiry_date	The expiry date provided is incorrect.
blocked_fraud	Blocked due to anti-fraud checks
referred	Referral.
declined_3ds	Declined by the 3DS2 authentication server prior to the transaction being processed.
Errors	
error_comms	Error communicating with the issuer.
error_system	Error in the system.
error_unknown	An unknown error has occurred.
timeout	The request has timed out.
no_response	The issuer did not respond.

Test Cards

The following test cards can be used to perform test transactions against the SIT environment.

You must use the specified Security Code to guarantee an approved transaction.

Card Number	Scheme	Type	Expiry	Security Code
4000000000000002	Visa	Standard	Any Valid	123
4462030000000000	Visa	Prepaid	Any Valid	444
5555555555554444	Mastercard	Standard	Any Valid	321
5597507644910558	Mastercard	Prepaid	Any Valid	888
340001916255521	American Express	Standard	Any Valid	1234

To force a declined transaction, please use Security Code 999.

Appendix 1 - Hosted Payment Fields Events

Events generated by the Hosted Payment Fields have the following fields available, which are detailed in the table below for each Event type:

- type
- target
- detail

Normal Events

Type	Target	Description	Detail
initialisationStart	DOM Document	Dispatched when the initialisation of the Hosted Payment Fields has started.	N/A
initialisationReady	Payment Form Element	Dispatched when the initialisation of the Hosted Payment Fields has completed, and the fields are ready for user entry. An implementer may wish to hide the payment form from the user until this event has been received.	N/A
fieldStatusChange	Payment Field Element	Dispatched when the validation status of a payment field is updated by user input.	FieldStatusChangeObject
issuerArrived	Payment Field Element	Dispatched when Issuer information becomes available for the payment card information entered by the user. This will likely be in response to the first six digits of the PAN being entered by the user.	IssuerObject
issuerCleared	Payment Field	Dispatched when Issuer information is no longer	N/A

	Element	available. This will likely be in response to the PAN being cleared or removed by the user.	
paymentSessionFieldSync	Payment Form Element	Dispatched after a *paymentSessionSyncRequest* event has been dispatched by the implementor. Indicates that the field specified has been synced with the Payment Session.	FieldSyncSuccessObject
paymentSessionSyncComplete	Payment Form Element	Dispatched after a *paymentSessionSyncRequest* event has been dispatched by the implementor. Indicates that the Payment Session sync process has completed. The implementer should ensure that all payment fields are still valid following this process.	SessionSyncCompleteObject

Error Events

Type	Target	Description	Detail
configurationError	DOM Document	Dispatched after initialisationStarted and before initialisationReady has been received. Indicates that a configuration error has occurred, and the Hosted Payment Fields have been unable to initialise.	Array of ConfigurationErrorObject
unrecognisedBrowser	Payment Form Element	Dispatched after initialisationStarted and before initialisationReady has been received. Indicates that the user is using a browser that is not recognised by Hosted Payment Fields. The Hosted Payment Fields may still work, but has not been tested nor is supported in the current browser.	BrowserInfoObject

unsupportedBrowser	Payment Field Element	Dispatched after initialisationStarted and before initialisationReady has been received. Indicates that the user is using a browser that does not work with Hosted Payment Fields.	BrowserInfoObject
paymentSessionError	Payment Field Element	Dispatched after initialisationStarted and before initialisationReady has been received. Indicates that the Payment Session for the ID provided during initialisation could not be loaded.	ErrorObject
fieldInitialisationError	Payment Field Element	Dispatched after initialisationStarted and before initialisationReady has been received. Indicates that one of the payment fields could not be initialised.	N/A
fieldGetIssuerError	Payment Field Element	Dispatched when Issuer information was attempted to be retrieved, but could not due to an error.	N/A
fieldUpdateError	Payment Field Element	Dispatched when a field could not be updated due to an error.	N/A
paymentSessionSyncFailure	Payment Field Element	Dispatched	FieldSyncFailureObject

Objects

FieldStatusChangeObject

Property	Type	Description
field	String	The name of the field that's status has changed.
state	String	The new state of the field. One of: valid, invalid, unset.
container	DOM Element	The DOM Element for the Div container of the field.
type	String	The type of event that triggered the status change. One of: blur, focus.

IssuerObject

Property	Type	Description
type	String	The type of card. One of: DEBIT, CREDIT, CHARGE_CARD
scheme	String	The scheme of the card: For example: VISA, MASTERCARD, AMERICAN_EXPRESS
brand	String	The brand of the card. Usually the issuing bank. For example: BARCLAYS, MONZO, HSBC
level	String	The level of the card. Used to identify standard, corporate, platinum, etc.
country	String	Two digit country code of the issuing country.
accepted	Boolean	Whether this type of card is accepted for the merchants configuration within the Gateway API.

infringement	Array of Strings	If not accepted, which of the properties has caused the card not to be accepted. For example: Type, Scheme, Country, etc. Useful to be able to provide feedback to the user such as “Sorry, but we do not accept credit cards.”

FieldSyncSuccessObject

Property	Type	Description
field	FieldStatusObject	The field status object.

FieldStatusObject

Property	Type	Description
type	String	The name of the field
state	String	The new state of the field. One of: valid, invalid, unset.

SessionSyncCompleteObject

Property	Type	Description
pan	String	The state of the PAN. One of: valid, invalid, unset.
expiry	String	The state of the expiry date. One of: valid, invalid, unset.
securityCode	String	The state of the security code. One of: valid, invalid, unset.

ConfigurationErrorObject

Property	Type	Description
id	String	The configuration option that was provided.
reason	String	The reason that the configuration option was invalid.

BrowserInfoObject

Property	Type	Description
browser	BrowserObject	The browser object

BrowserObject

Property	Type	Description
name	String	The name of the browser.
version	BrowserVersionObject	The browser version object

BrowserVersionObject

Property	Type	Description
current	String	The current version of the browser.

supported	String	The lowest supported version of the browser.

ErrorObject

Property	Type	Description
reason	String	A single description outlining the reason for the error.

FieldSyncFailureObject

Property	Type	Description
field	FieldErrorStatusObject	The field error status object.

FieldErrorStatusObject

Property	Type	Description
type	String	The name of the field.
reason	String	The reason for the error.

Version History

Version	Status	Date	Author	Approved	Notes
0.1	Draft	2020-01-14	ABH		Submitted for approval
1.0	Release	2020-01-15	ABH	FHP	Approved
1.1	Draft	2020-02-14	ABH		Added Authorisation & Capture section. Clarified use of Tokens.
1.1	Release	2020-02-19	ABH	FHP	Approved.
1.2	Draft	2020-02-20	ABH		Added response codes.
1.2	Release	2020-02-20	ABH	FHP	Approved.
1.3	Draft	2020-06-24	ABH		Added information about Customer object creation and using custom attributes. Updated Token section with more information about how they are tied to Customer objects. Added page breaks between each section.
1.3	Release	2020-06-24	ABH	FHP	Approved.
1.4	Draft	2020-08-03	ABH		Add 3DS2 information.
1.4	Release	2020-08-06	ABH	FHP	Approved.
1.5	Release	2021-05-10	ABH	FHP	Incorporate HPF into combined document
1.6	Release	2021-12-15	ABH	FHP	Update documentation on Hosted Payment Pages to make the flow clearer.
1.7	Release	2022-01-12	ABH	FHP	Added notification call-back documentation. Removed references to RAML and replaced with URL to online documentation.
1.8	Draft	2022-03-02	ABH		Added HPFv2.
2.0	Release	2023-03-16	ABH	FHP	Added Google Pay implementation details. Removed 3DSv1. Added new result codes for Google Pay, and for declined due to 3DS decline.
2.1	Release	2023-07-13	ABH	FHP	Added multiple 3DS2 challenge functionality.
3.0	Draft	2023-07-14	ABH		• Added Apple Pay implementation details. • Updated 3DS2 to include messageVersion • Added changelogs link and updated Service Desk URL.
3.0	Release	2023-07-17	ABH	JC	Approved.
3.1	Release	2023-08-01	ABH	FHP	• Added action to Hosted Payment Pages.
3.2	Release	2023-10-03	ABH	ABH	• Added section on Merchant Accounts, and explicit Merchant Account Selection.