Oracle Checkout Hosted Fields Integration Guide

December 2025, Version 1.4

Public

Purpose statement

This document provides an overview of features and enhancements included in releases oracle-checkout-web 25.0.1 & oracle-checkout-iOS 25.0.0. It is intended solely to help you assess the business benefits of upgrading to oracle-checkout-web 25.0.1 & oracle-checkout-iOS 25.0.0 and planning for implementing and upgrading the product features described.

Disclaimer

Document Revision History

VERSION	DATE	DESCRIPTION OF CHANGE
1.0	2025-10-16	Initial version
1.1	2025-11-13	Added additional payment data parameters into Additional Data for the Payment Response.
1.2	2025-12-09	New Optional Parameter “Platform” in the Payments API.
1.3	2026-01-15	Updated the fields and added webhook examples for AUTHORISATION_EXPIRE, CAPTURE, CAPTURE_FAILURE.
1.4	2026-05-07	Added the instructions to use JWE Token

Table of Contents

ORACLE CHECKOUT HOSTED FIELDS PAYMENT API 7

BEFORE YOU BEGIN 7

PAY WITH ENCRYPTED CARD DETAILS 9
REQUEST 9
RESPONSE 10
PAY WITH ENCRYPTED CARD DETAILS AND SAVE CARD TO PROFILE 13
REQUEST 13
RESPONSE 14
PAY WITH APPLE PAY TOKEN 18
REQUEST 18
RESPONSE 19
PAY WITH GOOGLE ANDROID PAY TOKEN 20
REQUEST 20
RESPONSE 21
PAY WITH ORACLE PAYMENT TOKEN (CARD ON PROFILE) 22
REQUEST 22
RESPONSE 23
MANUAL CAPTURE 24
REQUEST 24
RESPONSE 24
REFUND/CANCEL/VOID 26
REQUEST 26
RESPONSE 26
PARTIAL REFUND/ AUTHORIZATION ADJUSTMENT 27
REQUEST 27
RESPONSE 27
GET TOKENS ON PROFILE 29
REQUEST 29
RESPONSE 29
DELETE TOKEN ON PROFILE 31
REQUEST 31
RESPONSE 31
WEBHOOK SUBSCRIPTION CONFIGURATION: 32
AUTHORISATION WEBHOOK 33
AUTHORISATION ADJUSTMENT WEBHOOK 34
AUTHORISATION EXPIRE WEBHOOK 35
TOKEN CREATION WEBHOOK 36
CANCELLATION WEBHOOK 37
CAPTURE WEBHOOK 38

Capture Failure Webhook 39
Refund Webhook 40
Refund Failure Webhook 41
Authentication 42
API Endpoints 43

PARAMETERS 44

Header Parameters 44
Request Parameters 44
Sample Testing Values 50
Error Codes 51

INTEGRATING WEB-HOSTED CHECKOUT FOR RETRIEVING ENCRYPTED CARD DETAILS 52

Step 1: Add the Custom Card Form to Your HTML 52
Step 2: Include the Encryption Script in Your Page 53
Step 3: Setup Configuration and Handle Payment Submission 54
makeOraclePayment function definition 55
getAccessToken method definitionscope for RBE environment 56
Step 4: Trigger the Flow on Button Pay 57

ENCRYPTED CARD EXAMPLE 59

INTEGRATING IOS APP FOR RETRIEVING ENCRYPTED CARD DETAILS 60

Step 1: Import the library 60
Step 2: Add validations 60
Step 3: Encrypt card fields 61
Step 4: Retrieve a session token 62
Step 5: Make a payment 63

INTEGRATING GOOGLE ANDROID APP FOR RETRIEVING ENCRYPTED CARD DETAILS 64

Step 1: Import the library 64
Step 2: Validation and Encryption 64
Step 3: Making Payments 67
Step 4: Getting Access Tokens 67

CARD ON FILE 68

Web 68
Save Card 68
List Saved Cards 69

MAKING A PAYMENT USING A SAVED TOKEN (PAYVIATOKEN) 71
DELETING A SAVED CARD (DELETECARD) 73
IOS 75
SAVE CARD 75
LIST SAVED CARDS 77
MAKING A PAYMENT USING A SAVED TOKEN (PAYVIATOKEN) 79
DELETING A SAVED CARD (DELETECARD) 81
GOOGLE ANDROID 82
LIST SAVED CARDS 82
MAKING A PAYMENT USING A SAVED TOKEN (PAYVIATOKEN) 83
DELETING A SAVED CARD (DELETECARD) 85
PAYMENT GATEWAY CONFIGURATION FOR GOOGLE PAY 86
TO SUCCESSFULLY ENABLE GOOGLE PAY TRANSACTIONS, SPECIFIC CONFIGURATION STEPS ARE REQUIRED TO ENSURE SHOPPER PAYMENT CREDENTIALS ARE SECURELY ENCRYPTED AND PROCESSED. 86

INTEGRATION GUIDE FOR JWE API SUPPORT 87

CONNECT WITH US 93

Oracle Checkout Hosted Fields Payment API

Before you begin

The following requirements must be fulfilled before integrating with Oracle Payment Cloud Service Online/e-commerce Payments:

1. General Requirements
   a. Oracle must be the designated Payment Provider.
   b. An Oracle Account Representative must be contacted to enable Online/e-commerce payment configurations. The organization must be onboarded to Oracle Payment Cloud Service and have access to a test environment.

2. Developer Setup Requirements
   a. The integrator must be registered as a Service Provider in Oracle Payment Cloud Service (to enable channel creation).
   b. The integrator must provide:
      i. Email addresses for users requiring Oracle Payment Cloud Service credentials
      ii. Webhook endpoint and HMAC key to receive payment events
   c. Oracle will share the following:
      i. **Channel Code** (defines transaction handling attributes)
      ii. **Location reference(s)** (identifiers for customer Payment locations. It would be used for making payments reflect the payment location.
      iii. **Organization reference(s)** (identifiers for customer organization.

Generate the Client Credentials Key

Authentication

OAuth 2.0 – Client Credentials

To authenticate API calls, use the OAuth 2.0 **Client Credentials** flow. Your service must obtain an access token from Oracle Identity Cloud Service (IDCS) using the **Client ID** and **Client Secret** provided for your environment.

Token Endpoint:

POST https://idcs-0e86f01b7f094720af224335e43f84eb.identity.oraclecloud.com/oauth2/v1/token

HTTP Headers:

Authorization: Basic <base64(client_id:client_secret)>

Content-Type: application/x-www-form-urlencoded;charset=UTF-8

Request Body:

Use the appropriate environment code in the scope:

ORACLE

grant_type=client_credentials&scope=OPOP_<environment>/opop.payments

Generating the Basic Authorization Header

Before calling the token endpoint, you must generate the Base64-encoded value for the Authorization: Basic header

1): Concatenate the Client ID and Client Secret

Format:

clientId + ":" + clientSecret

Example:

12345:ABCDE

2): Base64-encode the resulting string

Command:

echo -n '12345:ABCDE' | base64

Output:

MTIzNDpBQkNERQ==

3): Use the encoded value in the Authorization header

Final header format:

Authorization: Basic MTIzNDpBQkNERQ==

Postman Script (Optional):

To automatically extract the token and store it in Postman collection variables:

```
const response = pm.response.json();
const access = response.access_token;
pm.collectionVariables.set("access_token", access);
```

ORACLE

Pay With Encrypted Card Details

Request


```
{
"transactionDetails": {
    "currency": "USD",
    "amount": 1000
},
  "paymentMethod": {
    "type": "scheme",
    "encryptedCardNumber": "test_4111111111111111",
    "encryptedExpiryMonth": "test_03",
    "encryptedExpiryYear": "test_2030",
    "encryptedSecurityCode": "test_737"
  },
  "billingAddress": {
    "city": "New York",
    "country": "US",
    "houseNumberOrName": "123",
    "postalCode": "10001",
    "stateOrProvince": "NY",
    "street": "123 Main Street"
  },
  "returnUrl": "https://your-company.com/...",
  "transactionReferences": {
    "clientTransactionId": "123456"
  },
  "transactionType": "011",
  "channelCode": "ch123456",
  "locationReference": "string",
  "organizationReference": "string",
  "captureType": "",

   "captureDelayHours": 4,
  "shopperInteraction": "Ecommerce",
```

```
                "tokenization": {
    "savePaymentMethod": false,
  }
      "platform": "Web"
}
```


Response


```
{
"transactionDetails": {
    "currency": "USD",
    "amount": 1000
},
"resultCode": "Authorised",
  "transactionType": "011",
  "cardAlias": "string",
  "paymentAccountReference": "string",
  "transactionReferences": {
    "clientTransactionId": "123456",
    "opcsTransactionId": "ABC1234DEF",
    "pspReference": "P8PJ89XK2MBFKQV5",
    "requestDateTime": "2024-03-12T16:04:16.296651Z",
    "responseDateTime": "2025-03-17T12:34:00.330757699Z"
},
  "additionalData": {
        "refusalReasonRaw": "AUTHORISED",
        "eci": "N/A",
        "acquirerAccountCode": "TestPmmAcquirerAccount",
        "xid": "N/A",
        "fraudRiskLevel": "veryLow",
        "recurringProcessingModel": "CardOnFile",
        "threeDAuthenticated": "false",
        "paymentMethodVariant": "visa",
        "issuerBin": "41111111",

        "payoutEligible": "Y",
```

```
                    "fraudManualReview": "false",
  "threeDOffered": "false",
  "threeDOfferedResponse": "N/A",
  "authorisationMid": "50",
  "bankAccount.iban": null,
  "cavv": "N/A",
  "bankAccount.ownerName": null,
  "fundsAvailability": "I",
  "authorisedAmountCurrency": "USD",
  "threeDAuthenticatedResponse": "N/A",
  "threeds2.cardEnrolled": "false",
  "avsResultRaw": "2",
  "retry.attempt1.rawResponse": "AUTHORISED",
  "paymentMethod": "visa",
  "avsResult": "2 Neither postal code nor address match",
  "cardSummary": "1142",
  "retry.attempt1.avsResultRaw": "2",
  "networkTxReference": "958237262264735",
  "expiryDate": "3/2030",
  "cavvAlgorithm": "N/A",
  "cardBin": "411111",
  "alias": "H480138306895004",
  "cvcResultRaw": "U",
  "merchantReference": "1234_wwwwwww",
  "acquirerReference": "5C5QV2L5RVV",
  "cardIssuingCountry": "NL",
  "liabilityShift": "false",
  "fraudResultType": "GREEN",
  "authCode": "099844",
  "cardHolderName": "Checkout Shopper PlaceHolder",
  "adjustAuthorisationData":
"BQABAQB+Lsp4c3bXpSfp81CS0qWuqzFrMQZEhT3BRTuJSDLiHFNw/JMOLr/3ffrJCE5zkU3/hcTRmIQNS3lprT1Rx
6GRoajZkRHINquM3sZl4gsPjxmvQ+8rOBHaoatfNBxq+pQxeaKm9gKqXEC+lCJf6HcfO5hcYFvzmmE9RWBIvNUnQ5i
ioz/VjFMVSsg5x0U0loRLRar/qajOeBo8PTkkSnoI9V1NS2zTsnQoxipNf+zJv6gvOez+kgH1Jk5tHoHLT+aJJl3+1
WThG+Q8pznU92KWhXYIZfH+BJUI7+EdI08+W2WMvSjRNx+fclT0n6SDjUZtdjO2FSg1TiRcuGmUUNsyDG/26JHPHBV
NDOQZWgAAMWBOqkV3Xb8YqUJb5SqkebCAvom9JYg1K+EJAQfqCqj+AcpBXj7mXnazZCkuL89oqzT5iCSAw4j58UKHE
```

i5+jEx0J3vRE13F+JiSCxJ6gMOhfQbgRkrp9k34sSL/H8fznnVcfERboEfBeTc8Ra4i7Vdor/0Ryogors1XSnLieFR
Gbib+G2Vb4Fq0YuPwdsMYg+hTkWMb9oKrT2RdSTMd+q5jAb1DtOms”,

"isCardCommercial": "unknown",

"PaymentAccountReference": "cSTmgHcgFFy7kpV6JurDFd5fjMtOc",

"retry.attempt1.acquirerAccount": "TestPmmAcquirerAccount",

"cardIssuingBank": "BANK",

"retry.attempt1.acquirer": "TestPmmAcquirer",

"authorisedAmountValue": "20000",

"issuerCountry": "NL",

"cvcResult": "5 Issuer not certified for CVC/CVV",

"retry.attempt1.responseCode": "Approved",

"aliasType": "Default",

"retry.attempt1.shopperInteraction": "Ecommerce",

"cardPaymentMethod": "visa",

"acquirerCode": "TestPmmAcquirer"

}

ORACLE

Pay With Encrypted Card Details and Save Card to Profile

Note: Saving a card to a profile without a payment uses the same spec as below except that the amount is zero in the request and the location reference is not required.

Request


```
{
  "transactionDetails": {
    "currency": "USD",
    "amount": 1000
  },
  "paymentMethod": {
    "type": "scheme",
    "encryptedCardNumber": "test_4111111111111111",
    "encryptedExpiryMonth": "test_03",
    "encryptedExpiryYear": "test_2030",
    "encryptedSecurityCode": "test_737"
  },
  "billingAddress": {
    "city": "New York",
    "country": "US",
    "houseNumberOrName": "123",
    "postalCode": "10001",
    "stateOrProvince": "NY",
    "street": "123 Main Street"
  },
  "returnUrl": "https://your-company.com/...",
  "transactionReferences": {
    "clientTransactionId": "123456"
  },
  "transactionType": "011",
  "channelCode": "ch123456",
  "locationReference": "string",
  "organizationReference": "string",
  "captureType": "",
  "captureDelayHours": 4,
```

```
                "shopperInteraction": "Ecommerce",
  "tokenization": {
    "shopperReference": "shopperReference",
"savePaymentMethod": true,
    "savedPaymentMethodUsage": "CardOnFile"
  },
      "platform": "Web"
}
```


Response


```
{
"transactionDetails": {
    "currency": "USD",
    "amount": 1000
},
"resultCode": "Authorised",
  "transactionType": "011",
  "cardAlias": "string",
  "paymentAccountReference": "string",
  "transactionReferences": {
    "clientTransactionId": "123456",
    "opcsTransactionId": "ABC1234DEF",
    "pspReference": "P8PJ89XK2MBFKQV5",
    "requestDateTime": "2024-03-12T16:04:16.296651Z",
    "responseDateTime": "2025-03-17T12:34:00.330757699Z"
},
"tokenization": {
    "shopperReference": "YOUR_SHOPPER_REFERENCE",
    "oraclePayToken": "M5N7TQ4TG5PFWR50",
    "savedPaymentMethodOperationType": "created"
  },
  "additionalData": {
        "refusalReasonRaw": "AUTHORISED",
        "eci": "N/A",
        "acquirerAccountCode": "TestPmmAcquirerAccount",
```

```
                    "xid": "N/A",
    "fraudRiskLevel": "veryLow",
    "recurringProcessingModel": "CardOnFile",
    "threeDAuthenticated": "false",
    "paymentMethodVariant": "visa",
    "issuerBin": "41111111",

    "payoutEligible": "Y",
    "fraudManualReview": "false",
    "threeDOffered": "false",
    "threeDOfferedResponse": "N/A",
    "authorisationMid": "50",
    "bankAccount.iban": null,
    "cavv": "N/A",
    "bankAccount.ownerName": null,
    "fundsAvailability": "I",
    "authorisedAmountCurrency": "USD",
    "threeDAuthenticatedResponse": "N/A",
    "threeds2.cardEnrolled": "false",
    "avsResultRaw": "2",
    "retry.attempt1.rawResponse": "AUTHORISED",
    "paymentMethod": "visa",
    "avsResult": "2 Neither postal code nor address match",
    "cardSummary": "1142",
    "retry.attempt1.avsResultRaw": "2",
    "networkTxReference": "958237262264735",
    "expiryDate": "3/2030",
    "cavvAlgorithm": "N/A",
    "cardBin": "411111",
    "alias": "H480138306895004",
    "cvcResultRaw": "U",
    "merchantReference": "1234_wwwwwww",
    "acquirerReference": "5C5QV2L5RVV",
    "cardIssuingCountry": "NL",
    "liabilityShift": "false",
```

```
                    "fraudResultType": "GREEN",
      "authCode": "099844",
      "cardHolderName": "Checkout Shopper PlaceHolder",
      "adjustAuthorisationData":
    "BQABAQB+Lsp4c3bXpSfp81CS0qWuqzFrMQZEhT3BRTuJSDLiHFNw/JMOLr/3ffrJCE5zkU3/hcTRmIQNS3lprT1Rx
    6GRoajZkRHINquM3sZl4gsPjxmvQ+8rOBHaoatfNBxq+pQxeaKm9gKqXEC+lCJf6HcfO5hcYFvzmmE9RWBIvNUnQ5i
    ioz/VjFMVSsg5x0U0loRLRar/qajOeBo8PTkkSnoI9V1NS2zTsnQoxipNf+zJv6gvOez+kgH1Jk5tHoHLT+aJJl3+1
    WThG+Q8pznU92KWhXYIZfH+BJUI7+EdI08+W2WMvSjRNx+fclT0n6SDjUZtdjO2FSg1TiRcuGmUUNsyDG/26JHPHBV
    NDOQZWgAAMWBOqkV3Xb8YqUJb5SqkebCAvom9JYg1K+EJAQfqCqj+AcpBXj7mXnazZCkuL89oqzT5iCSAw4j58UKHE
    i5+jEx0J3vRE13F+JiSCxJ6gMOhfQbgRkrp9k34sSL/H8fznnVcfERboEfBeTc8Ra4i7Vdor/0Ryogors1XSnLieFR
    Gbib+G2Vb4Fq0YuPwdsMYg+hTkWMb9oKrT2RdSTMd+q5jAb1DtOms",
      "isCardCommercial": "unknown",
      "PaymentAccountReference": "cSTmgHcgFFy7kpV6JurDFd5fjMtOc",
      "retry.attempt1.acquirerAccount": "TestPmmAcquirerAccount",
      "cardIssuingBank": "BANK",
      "retry.attempt1.acquirer": "TestPmmAcquirer",
      "authorisedAmountValue": "20000",
      "issuerCountry": "NL",
      "cvcResult": "5 Issuer not certified for CVC/CVV",
      "retry.attempt1.responseCode": "Approved",
      "aliasType": "Default",
      "retry.attempt1.shopperInteraction": "Ecommerce",
      "cardPaymentMethod": "visa",
     "acquirerCode": "TestPmmAcquirer"
  }
}
```

ORACLE

Pay with Apple Pay Token

Request


```
{
  "transactionDetails": {
    "currency": "USD",
    "amount": 829
  },
  "billingAddress": {
    "city": "Toronto",
    "country": "CA",
    "houseNumberOrName": "123",
    "postalCode": "M5H2N2",
    "stateOrProvince": "ON",
    "street": "123 Front Street"
  },
  "paymentMethod": {
    "applePayToken": "exampleToken",
    "type": "applepay"
  },
  "shopperReference": "shopperReference",
  "transactionReferences": {
    "clientTransactionId": "123456",
  },
  "transactionType": "011",
"channelCode": "ch123456",
  "locationReference": "string",
  "organizationReference": "string",
  "captureType": "",
  "captureDelayHours": 4,
  "shopperInteraction": "Ecommerce",
  "platform": "iOS"
}
```

ORACLE

Response

Response


```
{
"transactionDetails": {
    "currency": "USD",
    "amount": 1000
},
"resultCode": "Authorised",
  "transactionType": "011",
  "cardAlias": "string",
  "paymentAccountReference": "string",
  "transactionReferences": {
    "clientTransactionId": "123456",
    "opcsTransactionId": "ABC1234DEF",
    "pspReference": "P8PJ89XK2MBFKQV5",
    "requestDateTime": "2024-03-12T16:04:16.296651Z",
    "responseDateTime": "2025-03-17T12:34:00.330757699Z"
},
  "additionalData": {
    "cvcResult": "1 Matches",
    "authCode": "065696",
    "avsResult": "4 AVS not supported for this card type",
    "avsResultRaw": "4",
    "cvcResultRaw": "M",
    "refusalReasonRaw": "AUTHORISED",
    "acquirerCode": "TestPmmAcquirer",
    "acquirerReference": "8PQMP9VIE9N"
},
}
```

ORACLE

Pay with Google Android Pay Token

Request


```
{
"transactionDetails": {
    "currency": "USD",
    "amount": 1000
},
  "billingAddress": {
    "city": "Los Angeles",
    "country": "US",
    "houseNumberOrName": "456",
    "postalCode": "90001",
    "stateOrProvince": "CA",
    "street": "456 Sunset Blvd"
  },
  "paymentMethod": {
    "googlePayToken": "exampleToken",
    "type": "googlepay"
  },
  "shopperReference": "shopperReference",
  "transactionReferences": {
    "clientTransactionId": "123456",
  },
  "transactionType": "011",
  "channelCode": "ch123456",
  "locationReference": "string",
  "organizationReference": "string",
  "captureType": "",
  "captureDelayHours": 4,
  "shopperInteraction": "Ecommerce"
  "platform": "Android"
}
```

ORACLE

Response


```
{
"transactionDetails": {
    "currency": "USD",
    "amount": 1000
},
"resultCode": "Authorised",
  "transactionType": "011",
  "cardAlias": "string",
  "paymentAccountReference": "string",
  "transactionReferences": {
    "clientTransactionId": "123456",
    "opcsTransactionId": "ABC1234DEF",
    "pspReference": "P8PJ89XK2MBFKQV5",
    "requestDateTime": "2024-03-12T16:04:16.296651Z",
    "responseDateTime": "2025-03-17T12:34:00.330757699Z"
},
  "additionalData": {
    "cvcResult": "1 Matches",
    "authCode": "065696",     "avsResult": "4 AVS not
supported for this card type",
    "avsResultRaw": "4",
    "cvcResultRaw": "M",
    "refusalReasonRaw": "AUTHORISED",
    "acquirerCode": "TestPmmAcquirer",
    "acquirerReference": "8PQMP9VIE9N"
},
}
```

ORACLE

Pay with Oracle Payment Token (Card on Profile)

Request


```
{
"transactionDetails": {
    "currency": "USD",
    "amount": 1000
},
  "paymentMethod": {
    "type": "oracleToken",
    "oraclePayToken": "M5N7TQ4TG5PFWR50",
    "encryptedSecurityCode": "test_737"
  },
  "billingAddress": {
    "city": "Chicago",
    "country": "US",
    "houseNumberOrName": "789",
    "postalCode": "60601",
    "stateOrProvince": "IL",
    "street": "789 Wacker Drive"
  },
  "transactionReferences": {
    "clientTransactionId": "123456",
  },
  "transactionType": "011",
  "channelCode": "ch123456",
  "locationReference": "string",
  "organizationReference": "string",
  "captureType": "",
  "captureDelayHours": 4,
  "shopperInteraction": "ContAuth"
  "shopperReference": "shopperReference"
}
```

ORACLE

Response


```
{
"transactionDetails": {
    "currency": "USD",
    "amount": 1000
},
"resultCode": "Authorised",
"transactionType": "011",
  "transactionReferences": {
    "clientTransactionId": "123456",
    "opcsTransactionId": "ABC1234DEF",
    "pspReference": "P8PJ89XK2MBFKQV5",
    "requestDateTime": "2024-03-12T16:04:16.296651Z",
    "responseDateTime": "2025-03-17T12:34:00.330757699Z"
},
  "additionalData": {
    "cvcResult": "1 Matches",
    "authCode": "065696",
    "avsResult": "4 AVS not supported for this card type",
    "avsResultRaw": "4",
"cvcResultRaw": "M",
    "refusalReasonRaw": "AUTHORISED",
    "acquirerCode": "TestPmmAcquirer",
    "acquirerReference": "8PQMP9VIE9N"
},
}
```

ORACLE

Manual Capture

Request


```
{
  "transactionType": "003",
  "channelCode": "ch123456",
  "organizationReference": "string",
"transactionReferences": {
    "clientTransactionId": "123456",
    "originalOPCSTransactionId": "FED6543CBA",
    "requestDateTime": "2024-03-12T16:04:16.296651Z",
},
}
```


Response


```
{
“channelCode”: “ch123456”,
    "result": {
        "authorizedAmount": 843,
        "currency": "USD",
        "resultCode": "SUCCESS"
    },
    "transactionDetails": {
        "amount": 843,
        "currency": "USD"
    },
    "transactionReferences": {

"clientTransactionId": "123456",
"opcsTransactionId": "ABC1234DEF",

"originalOpcsTransactionId": "FED6543CBA",
"originalPspReference": "VBTSRJJQ35GSKTV5",

"pspReference": "M2JLCNW3T3TPBC75",
```

```
                "requestDateTime": "2024-02-05T10:44:16.296651Z",
"responseDateTime": "2025-03-17T12:56:29.335747724Z"
    },
    "transactionType": "003"
}
```

ORACLE

Refund/Cancel/Void

If the transaction has been captured, a full amount refund will be issued; else the transaction will be a cancel/void.

Request


```
{
"transactionType": "006",
"transactionReference": {
"clientTransactionId": "CHLoc1GSVI0026REF",
"originalOpcsTransactionId": "12345ABCDE",
"requestDateTime" : "2025-09-09T13:52:44.000Z"
},
"channelCode": "Test_123456",
"organizationReference": "string"
}
```


Response


```
{
"channelCode": " Test_123456",
"result": {
"resultCode": "RECEIVED"
},
"transactionReferences": {
"clientTransactionId": "CHLoc1GSVI0026REF",
"opcsTransactionId": "tHAH5mdkmU",
"originalOpcsTransactionId": "12345ABCDE",
"originalPspReference": "S6R6PZ5FBBCDCQV5",
"pspReference": "BVQZ35BSHWZWKCV5",
"requestDateTime" : "2025-09-09T13:52:44.000Z"
"responseDateTime": "2025-09-09T14:52:44.000Z "
},
"transactionType": "006"
}
```

ORACLE

Partial Refund/ Authorization Adjustment

If the transaction has not been captured, then an authorization adjustment will be made; else a partial refund will be performed.

Request


```
{
"transactionType": "007",
"transactionDetails": {
"amount": 100,
"currency": "USD"
},
"transactionReferences":
{
"clientTransactionId": "ORC_ABC00009PR",
"originalOpcsTransactionId": "123ABC456DEF890GHI",
"requestDateTime" : "2023-11-09T13:52:44.000Z"
},
"channelCode": "ORC_ABC00009",
"organizationReference": "string"
}
```


Response


```
{
"channelCode": " ORC_ABC00009",
"result": {
"resultCode": "RECEIVED"
},
"transactionDetails": {
"amount": 500,
"currency": "USD"
},
"transactionReferences":
{
"clientTransactionId": "partialrefund ",
"opcsTransactionId": "P8aHmfk6XP",
"originalOpcsTransactionId": "123ABC456DEF890GHI ",
"originalPspReference": "D5V5FB8QSZL3JM65",
```

```
"pspReference": "NDFMVV4ZLWRNK375",
"requestDateTime": "2025-01-29T16:18:44.000Z",
"responseDateTime": "2025-09-11T08:25:47.116448747Z"
},
"transactionType": "007"
}
```

ORACLE

Get Tokens on Profile

Request


```
{
  "shopperReference": "shopperReference",
  "channelCode": "ch123456",
  "organizationReference": "string",
"transactionReferences": {
    "clientTransactionId": "123456"
},
}
```


Response


```
{
  "channelCode": "ch123456",
  "organizationReference": "string",
      "shopperReference": "YOUR_SHOPPER_REFERENCE",
    "transactionReferences": {
    "clientTransactionId": "123456",
    "opcsTransactionId": "ABC1234DEF",
    "requestDateTime": "2024-03-12T16:04:16.296651Z",
    "responseDateTime": "2025-03-17T12:34:00.330757699Z"
     },
      "storedPaymentMethods": [
       {
"brandName": "cup",
"channelCode": "ccextHC5SC6",
"expiryMonth": "10",
"expiryYear": "2030",
"holderName": "J. Smith",
"lastFour": "0000",
"oraclePayToken": "4E05B5F10737BEC72C44350DC2B5928FAEA3F3C1",
"type": "scheme"
        },
       {
"brandName": "cup",
```

```
"channelCode": "ccextHC5SC6",
"expiryMonth": "10",
"expiryYear": "2030",
"holderName": "J. Smith",
"lastFour": "0000",
"oraclePayToken": "4E05B5F10737BEC72C44350DC2B5928FAEA3F3C1",
"type": "scheme"
          },
],
}
```

ORACLE

Delete Token on Profile

Request


```
{
  "shopperReference": "shopperReference",
  "channelCode": "ch123456",
  "organizationReference": "string",
  "transactionReferences": {
    "clientTransactionId": "123456"
},
  "oraclePayToken": "4E05B5F10737BEC72C44350DC2B5928FAEA3F3C1"
}
```


Response


```
{
      "channelCode": "ch123456",
      "organizationReference": "string",
      "shopperReference": "YOUR_SHOPPER_REFERENCE",
      "transactionReferences": {
            "clientTransactionId": "123456",
             "opcsTransactionId": "ABC1234DEF",
             "requestDateTime": "2024-03-12T16:04:16.296651Z",
          "responseDateTime": "2025-03-17T12:34:00.330757699Z"
     },
      “resultCode”: “SUCCESS”
}
```

ORACLE

Webhook Subscription Configuration:

Before Oracle can configure webhook delivery for your integration, the following information must be provided by the customer.

1. General Requirements

a. You must determine which payment events you want to receive from Oracle Payment Cloud Service (OPCS).

b. You must provide a secure, publicly accessible HTTPS endpoint to receive webhook notifications.

c. Your system must support at least one of the supported authentication methods (Basic, Bearer, or OAuth-based).

2. Webhook Setup Requirements

You need to provide the following details:

a. Events to Subscribe To

Specify the list of event types you want to receive (Supported Events)

- AUTHORISATION,
- AUTHORISATION_ADJUSTMENT,
- AUTHORISATION_EXPIRE,
- CONTRACT_CREATED
- CAPTURE
- CAPTURE_FAILURE
- REVERSAL
- REFUNDED
- REFUND_FAILURE

b. Notification Endpoint

Provide the full URL of the endpoint where Oracle Payments will send webhook messages.

c. Authentication Method

Choose one of the supported authentication mechanisms and provide the required information:

i. Basic Authentication

Username, Password

ii. Bearer Token Authentication

Static Bearer token

iii. OAuth-Based Authentication (Basic_With_Bearer)

OAuth token URL,Username,Password,(Oracle Paymentswill use these credentials to fetch the Bearer token.)

ORACLE

Authorisation Webhook


```
{
  "live":"false",
  "event":"AUTHORISATION",
  "eventDateTime":"2023-09-11T11:52:15+02:00",
  "eventSuccess":"true",
  "eventMessage":"",
  "opcsTransactionId":"123456ABCD",
  "clientTransactionId":"123456",
  "pspReference":"A456C987ZZ",
  "channelCode":"CHCTS001",
  "amount":{
    "currency":"USD",
    "value":100
  },
  "pspAdditionalData":{
    "cardBin":54133,
    "cardIssuingCurrency":"USD",
    "cardIssuingBank":"null",
    "cardPaymentMethod":"mc",
    "cardIssuingCountry":"US",
    "fundingSource":"CREDIT",
    "tokenTxVariant":"null",
    "untokenisedCardSummary":"null",
    "cardHolderName":"TEST",
    "shopperCountry":"US",
    "acquirerCode":"TestPmmAcquirer",
    "cvcResultRaw":"Unknown",
    "avsResultRaw":"Unknown",
    "refusalReasonRaw":"APPROVED",
    "threeDOfferedResponse":"null",
    "threeDAuthenticatedResponse":"null"
  }
}
```

ORACLE

Authorisation Adjustment Webhook


```
{
  "live":"false",
  "event":"AUTHORISATION_ADJUSTMENT",
  "eventDateTime":"2025-05-13T14:57:34.000+0000",
  "eventSuccess":true,
  "eventMessage":"Acquirer Error",
  "opcsTransactionId":"3DyY7K9uzq",
  "originalOpcsTransactionId":"3DyY7K9uzq",
  "pspReference":"VRCJBX79632WQT65",
  "channelCode":"ccorcAOMKY9",
  "orginalPspReference":"JPXRPZ5RPXFH9PT5",
  "clientTransactionId":"partialrefundrbe_after004--12May2025",
  "amount":{
    "value":3325,
    "currency":"USD"
  }
}
```

ORACLE

Authorisation Expire Webhook


```
{
  "live":"false",
  "event":"AUTHORISATION_EXPIRE",
  "eventDateTime":"2025-05-21T03:57:34.000+0000",
  "eventSuccess":true,
  "eventMessage":"Authorisation Expired",
  "opcsTransactionId":"3DyY7K9uzq",
  "originalOpcsTransactionId":"3DyY7K9uzq",
  "pspReference":"VRCJBX79632WQT65",
  "channelCode":"ccorcAOMKY9",
  "orginalPspReference":"JPXRPZ5RPXFH9PT5",
  "clientTransactionId":"auth_expire--21May2025",
  "amount":{
    "value":36,
    "currency":"USD"
  }
}
```

ORACLE

Token Creation Webhook


```
{
  "live":"false",
  "event":"CONTRACT_CREATED",
  "eventDateTime":"2025-05-23T10:28:47.000+0000",
  "eventSuccess":true,
  "eventMessage":"",
  "pspReference":"AAAAAAAAAAAAAAAAKpZ+zpcUmsyZlu1bFYGNrD6NJ+UscYNCZG+pBS3uNjU=",
  "orginalPspReference":"XVFLWRMJGWVV9ST5",
  "opcsTransactionId":"DPl1HQI2Ea",
  "opcsToken":"AAAAAAAAAAAAAAAAKpZ+zpcUmsyZlu1bFYGNrD6NJ+UscYNCZG+pBS3uNjU=",
  "clientTransactionId":"test1234",
  "shopperReference":"shopper_1",
  "channelCode":"ccextU7CJ8O",
  "amount":{
    "value":300,
    "currency":"USD"
  }
}
```

ORACLE

Cancellation Webhook


```
{
  "live":"false",
  "event":"REVERSAL",
  "eventDateTime":"2025-04-16T15:36:54.000+0000",
  "eventSuccess”: true,
  "eventMessage":"",
  "originalOpcsTransactionId":"BditTy1tjs",
  "opcsTransactionId":"BditTy1tjs",
  "clientTransactionId":"Happy_Path_002_08thApr2025",
  "pspReference":"DQZ9Z394CNTZJMV5",
  "orginalPspReference":"V8JW44RLQSWQQ2V5",
  "channelCode":"ccorcAOMKY9",
  "amount":{
    "value":0,
    "currency":"EUR"
  },
  "shopperData":{
    "shopperCountry":"US"
  }
}
```

ORACLE

Capture Webhook


```
{
  "live":"false",
  "event":"CAPTURE",
  "eventDateTime":"2025-05-20T13:39:43.000+0000",
  "eventSuccess":true,
  "eventMessage":"Amount captured",
  "originalOpcsTransactionId":"dFvKV3AHdM",
  "opcsTransactionId":"iHY6ShyhTB",
  "clientTransactionId":"capture--20May2025",
  "pspReference":"ZQFS6XKVT3XW24V5",
  "orginalPspReference":"C8K69VNG6GH45G75",
  "channelCode":"ccorcAOMKY9",
  "amount":{
    "value":501,
    "currency":"USD"
  }
}
```

ORACLE

Capture Failure Webhook


```
{
  "live":"false",
  "event":"CAPTURE_FAILURE",
  "eventDateTime":"2025-05-20T15:55:43.000+0000",
  "eventSuccess":true,
  "eventMessage":"Amount capture failed",
  "originalOpcsTransactionId":"dFvKV3AHdM",
  "opcsTransactionId":"iHY6ShyhTB",
  "clientTransactionId":"capture_failed--20May2025",
  "pspReference":"ZQFS6XKVT3XW24V5",
  "orginalPspReference":"C8K69VNG6GH45G75",
  "channelCode":"ccorcAOMKY9",
  "amount":{
    "value":1001,
    "currency":"USD"
  }
}
```

ORACLE

Refund Webhook


```
{
  "live":"false",
  "event":"REFUNDED",
  "eventDateTime":"2025-05-09T17:39:43.000+0000",
  "eventSuccess":false,
  "eventMessage":"Requested refund amount too high",
  "originalOpcsTransactionId":"dFvKV3AHdM",
  "opcsTransactionId":"iHY6ShyhTB",
  "clientTransactionId":"partialrefundrbe_afterEH004--09May2025",
  "pspReference":"ZQFS6XKVT3XW24V5",
  "orginalPspReference":"C8K69VNG6GH45G75",
  "channelCode":"ccorcAOMKY9",
  "amount":{
    "value":6436,
    "currency":"USD"
  }
}
```

ORACLE

Refund Failure Webhook


```
{
  "live":"false",
  "event":"REFUND_FAILURE",
  "eventDateTime":"2025-04-09T13:28:49.000+0000",
  "eventSuccess":true,
  "eventMessage":"Refund Failed",
  "originalOpcsTransactionId":"x87BXSLpOZ",
  "opcsTransactionId":"x87BXSLpOZ",
  "pspReference":"GDBPQ2ZKB6N49W65",
  "orginalPspReference":"FJ9LQPCF7FQRX2V5",
  "channelCode":"ccextU7CJ8O",
  "amount":{
    "value":119900,
    "currency":"USD"
  }
}
```

ORACLE

Authentication

OAuth 2.0 Client Credentials Grant

Token URL:

https://idcs-0e86f01b7f094720af224335e43f84eb.identity.oraclecloud.com/oauth2/v1/token

Request Method: POST

Headers:

Authorization: Basic <base64(client_id:client_secret)>
Content-Type: application/x-www-form-urlencoded;charset=UTF-8

Request Body:

grant_type=client_credentials&scope=OPOP_COMMON/opop.payments

Postman Script to Extract Token:

javascript
const response = pm.response.json(); const access =
response.access_token;
pm.collectionVariables.set("access_token", access);

Usage: Use the obtained token as: Authorization: Bearer {{access_token}}

ORACLE

API Endpoints

Make Payment

Endpoint: https://stage.opcs.ocs.oraclecloud.com/api/v1/oracle/makePayment

Method: POST

Description: Process payments using various payment methods including encrypted card details, Apple Pay, Google Pay, and Oracle tokens.

Fetch All Cards

Endpoint: https://stage.opcs.ocs.oraclecloud.com/api/v1/oracle/fetchAllCards

Method: POST

Description: Retrieve all stored payment methods for a specific shopper.

Delete Saved Card

Endpoint: https://stage.opcs.ocs.oraclecloud.com/api/v1/oracle/removeCardDetails

Method: POST

Description: Remove a stored payment method from a shopper's profile.

Pay Via Token

Endpoint: https://stage.opcs.ocs.oraclecloud.com/api/v1/oracle/payViaToken

Method: POST

Description: Process payments using previously stored Oracle payment tokens.

ORACLE

Parameters

Header Parameters

Parameter Name	Required?	Description
AuthToken	Y	A base64 encoded combination of your clientId and clientSecret

Request Parameters

`api/v1/oracle/makePayment`

Parameter Name	Required?	Data Type	Description
transactionType	Y	Number	The type of transaction being performed. Type "011" e-commerce sale and capture
transactionDetails.amount	Y	Number	Transaction amount in minor units.
transactionDetails.currency	Y	String	Transaction Currency, 3character ISO currency code
locationReference	Y (except for saving card on profile)	String	Organization's reference for the location. This is captured in Oracle Payment Cloud Service during onboarding when locations are added via API or the UI.
organizationReference	Y	String	A unique reference for the customer to designate traffic.
channelCode	Y	String	Reference for the channel of the ecommerce transaction will pass through
transactionReferences. clientTransactionID	Y	String	ID generated by the client for their reference. This will be added to the merchantReference when the transaction is created

ORACLE

paymentMethod.type	Y	string	The payment method type. Supported values: "scheme" (card), "applepay", "googlepay","oracletoken".

paymentMethod.encryptedCardNumber	Y (when type is scheme)	String	Encrypted card number.
paymentMethod.encryptedExpiryMonth	Y (when type is scheme)	String	Encrypted card expiry month
paymentMethod.encryptedExpiryYear	Y (when type is scheme)	String	Encrypted card expiry year
paymentMethod.encryptedSecurityCode	Y (when type is scheme)	String	Encrypted CVV or CVC.
paymentMethod.encryptedCard	Y (when type is scheme)	String	Encrypted JWE token Either all four (paymentMethod.encryptedCardNumber, paymentMethod.encryptedExpiryMonth, paymentMethod.encryptedExpiryYear, paymentMethod.encryptedSecurityCode) are required OR PaymentMethod.encryptedCard is required.
paymentMethod.applePayToken	Y (when type is applepay)	String	Encrypted Apple Pay token, the stringified and base64 encoded paymentData you retrieved from the Apple framework. Max length: 10000
paymentMethod.googlePayToken	Y (when type is googlePay)	String	Encrypted Google Pay token, Max length: 10000 The token that you obtained from the Google Pay API PaymentData response.

paymentMethod.oraclePayToken	Y (when type is oracleToken)	String	A token that represents a previously saved card or payment method. This is returned by the system after tokenization and is used to initiate future payments without needing full card details again.
billingAddress.city	N	String	The billing address city is passed in the payment request. (Billing address based on customer configuration)
billingAddress.country	N	String	Country, The value format must adhere to the ISO 3166-1 alpha-2 standard.
billingAddress.houseNumberOrName	N	String	The number or name of the house.
billingAddress.postalCode	Y	String	A maximum of five digits for an address in the US, or a maximum of ten characters for an address in all other countries/regions.
billingAddress.stateOrProvince	N	String	Required for shoppers from the US or Canada. Provide the child element stateOrProvince as defined in ISO 3166-2. For example, **CA** in the US or **ON** for Canada. AVS does not validate stateOrProvince.
billingAddress.street	N	String	The name of the street.
returnUrl	N	String	The URL to return to in case of a redirection.
captureType	N	String	Defines how and when the transaction will be captured. Supported values: automatic, delayed_Automatic, manual, If not provided, the default settings configured at the merchant account level will be applied.
captureDelayHours	N	Integer	Number of hours to delay automatic capture after authorization. Only applicable if captureType = delayedAutomatic. Valid range: 1 to 672.

ORACLE

shopperInteraction	Y	String	Specifies the sales channel through which the shopper gives their card details (For example, Ecommerce)
tokenization.savePaymentMethod	Y	boolean	Indicates whether the payment method should be stored for future use. Set to true to enable tokenization. Tokenization is only supported when using encrypted card details. It should be set to false When using Apple Pay or Google Pay.
tokenization.savedPaymentMethodUsage	Y (when savePaymentMethod is set to true)	String	Specifies the intended use of the saved payment method. Common values: (Ex: CardOnFile)
tokenization.shopperReference	Y(when savePaymentMethod is set to true)	String	A unique ID representing the shopper. Used to associate saved payment methods with the correct customer for future transactions. Min length: 3Max length: 256
shopperReference	Y(when saving a card or paying with oraclepaytoken)	String	A unique ID representing the shopper. Used to pay with saved payment methods with the correct customer. Min length: 3Max length: 256
platform	N (Y for Android and iOS)	String	Defines the platform from which the payment request is made. Allowed values: Web Android iOS Default value: 'Web' if nothing is passed

ORACLE

`api/v1/oracle/fetchAllCards`

Parameter Name	Required?	Data Type	Description
organizationReference	Y	String	A unique reference for the customer to designate traffic.
channelCode	Y	String	Reference for the channel of the e-commerce transaction will page through
transactionReferences.clientTransactionID	Y	String	ID generated by the client for their reference. This will be added to the merchantReference when the transaction is created
shopperReference	Y	String	A unique ID representing the shopper. Used to fetch saved payment methods with the correct customer. Min length: 3Max length: 256

ORACLE

/api/v1/oracle/removeCardDetails

Parameter Name	Required?	Data Type	Description
organizationReference	Y	String	A unique reference for the customer to designate traffic.
channelCode	Y	String	Reference for the channel of the e-commerce transaction will page through
transactionReferences.clientTransactionID	Y	String	ID generated by the client for their reference. This will be added to the merchantReference when the transaction is created
shopperReference	Y	String	A unique ID representing the shopper. Used to remove saved payment methods with the correct customer. Min length: 3Max length: 256
opcsToken	Y	String	A token that represents a previously saved card or payment method. This is returned by the system after tokenization and is used to initiate the removal of saved card without needing full card details again.

ORACLE

Sample Testing Values


```
"channelCode": "ccextHC5SC6",
"organizationReference": "OPCS_Test",
"locationReference": "7433",
"paymentMethod": {
  "type": "scheme",
  "encryptedCardNumber": "test_36006666333344",
  "encryptedExpiryMonth": "test_03",
  "encryptedExpiryYear": "test_2030",
  "encryptedSecurityCode": "test_737"
}
```

ORACLE

Error Codes

Error Code	Error Category	Possible Reasons
00000-<PAYMENT_PROVIDER_CODE>	UNKNOWN	Unknown
00001-<PAYMENT_PROVIDER_CODE>	INVALD_PAYLOAD	• Negative amount • Invalid currency • Invalid characters • Mandatory fields not completed
00002-<PAYMENT_PROVIDER_CODE>	MISSING_LOOKUP	• Failed to find location • Failed to find channel
00004-<PAYMENT_PROVIDER_CODE>	AUTHENTICATION_ERRORS	• Failed to authenticate
00006-<PAYMENT_PROVIDER_CODE>	INFRASTRUCTURE_ERRORS	• Infrastructure error

ORACLE

Integrating Web-Hosted Checkout for Retrieving Encrypted Card Details

The following steps show how to integrate the web component card form that encrypts the card on the client side.

Step 1: Add the Custom Card Form to Your HTML

Add the following HTML to your checkout or payment form. It renders the cardholder name, card number, expiry date, and CVV inputs.

```
<div id="customCard-container">
            <label class="field-group">
              <span class="field-label">Name on Card:</span>
              <input type="text" id="cardHolderName" class="field-input"
placeholder="Enter name on card" required>
              <div id="cardHolderNameError" class="field-error"
                style="display:none;color:#dc3545;font-size:13px;margin-top:2px;">Please
enter the name on card.</div>
            </label>
            <label class="field-group">
              <span class="field-label">Card number:</span>
              <span class="field-input" data-cse="encryptedCardNumber"></span>
              <img id="card-brand-icon" src="" alt="Brand" />
            </label>
            <div class="expiry-cvc">
              <label class="field-group">
                <span class="field-label">Expiry date:</span>
                <span class="field-input" data-cse="encryptedExpiryDate"></span>
              </label>
              <label class="field-group">
                <span class="field-label">CVV/CVC:</span>
                <span class="field-input" data-cse="encryptedSecurityCode"></span>
              </label>
            </div>           </div>
```

For custom CSS, you can do it as follows:

```
        <div id="cardHolderNameError" class="field-error"
                style="display:none;color:#dc3545;font-size:13px;margin-top:2px;">Please
enter the name on card.</div>
```

ORACLE

Step 2: Include the Encryption Script in Your Page

Place the JavaScript encryption SDK at the end of your <body> tag:

```
<script rel="stylesheet"
src="https://checkoutshoppertest.cdn.adyen.com/checkoutshopper/sdk/6.13.0/adyen.js"
    integrity="sha384-CBUu7OtS6E6Ys1RtLZpGr3lwO0ofNycTs2DRYgEJMuYNMm7SjxUz/4NOzoztMjkw"
    crossorigin="anonymous">
</script>
```

Define this clientKey and the clientIdClientSecret on the top of the file as a constant:

```
const clientKey = "TBD“ const
clientIdClientSecret = ”TBD"
```

ORACLE

Step 3: Setup Configuration and Handle Payment Submission

```
const configuration =
{   clientKey:
clientKey,
environment: "test",
amount: {     value:
1000,     currency:
"EUR",
  },   locale: "nl-
NL",   countryCode:
"US",
//The full /paymentMethods response object from your server. Contains the payment
methods configured in your account.
//No need to call onSubmit method if you are calling a server side API for making a
Payment
  onSubmit: async (state, component, actions) => {
try {
      console.log("this is state", state);

      // Make the Oracle payment API call
      const result = await makeOraclePayment(state.data.makeOraclePayment, holderName);
      // If the payment request fails, or if an unexpected error occurs.
      if (!result.resultCode) {
actions.reject();
return;
      }
      onPaymentCompleted: (result, component) => {
console.info(result, component);
      },
        onPaymentFailed: (result, component) => {
console.info(result, component);
        },
          onError: (error, component) => {
            console.error(error.name, error.message, error.stack, component);
          },
};
```

ORACLE

Add the definition of makeOraclePayment() and getAccessToken() Functions. Do not add these methods if you are calling a server-side API for making a Payment:

makeOraclePayment function definition

```
async function makeOraclePayment(cardData, holderName) {
try {
    // First get the access token
    const tokenResponse = await getAccessToken();


    // Prepare the payment payload
const paymentPayload = {
"transactionDetails": {
        "amount": amount, // Use the global amount variable
        "currency": "USD"
      },
      "paymentMethod": {
        "type": "scheme",
        "encryptedCardNumber": cardData.paymentMethod.encryptedCardNumber,
        "encryptedExpiryMonth": cardData.paymentMethod.encryptedExpiryMonth,
        "encryptedExpiryYear": cardData.paymentMethod.encryptedExpiryYear,
        "encryptedSecurityCode": cardData.paymentMethod.encryptedSecurityCode
},
      "transactionReferences": {
        "clientTransactionId": “1233r”
      },
      "channelCode": "ccorcAOMKY9",
      "organizationReference": "QA",
      "locationReference": "TC0052",
      "captureType": "delayed_Automatic",
      "captureDelayHours": 1,
    };

    // Make the payment API call const
response = await
fetch("https://stage.opcs.ocs.oraclecloud.com/api/v1/oracle/makePayment",
{       method: "POST",       headers: {
        'Content-Type': 'application/json',
        'Authorization': 'Bearer ' + tokenResponse.access_token
      },
      body: JSON.stringify(paymentPayload)
    });
    if (!response.ok) {
      console.error("HTTP error:", await response.json());
throw new Error(`HTTP error! status: ${response.status}`);
}      const result = await response.json();
console.log("Payment result:", result);     return result;

  } catch (error) {
    console.error("Error making Oracle payment:", error);
throw error;
  }
};
```

ORACLE

getAccessToken method definition
scope for RBE environment

```
async function getAccessToken() {
const params = {
    grant_type: "client_credentials",
scope: " OPOP_COMMON/opop.payments"
  };
  const searchParams = new URLSearchParams();
for (const key in params) {
    searchParams.append(key, params[key]);
  }
  const headers = new Headers();
  headers.append('Content-Type', 'application/x-www-form-urlencoded; charset=UTF-8');
headers.append('Authorization', `Basic ${clientIdClientSecret}`);

  const requestOptions = {
method: 'POST',
headers: headers,
body: searchParams
  };
  const response = await fetch("https://idcs-
0e86f01b7f094720af224335e43f84eb.identity.oraclecloud.com/oauth2/v1/token",
requestOptions);

  if (!response.ok) {
    throw new Error(`Failed to get access token:
${response.status}`);   }
  return await response.json();
}
```

ORACLE

Step 4: Trigger the Flow on Button Pay

Call this action by embedding the click to checkout:

```
 async function initCheckout() {
checkout = await AdyenCheckout({
clientKey: clientKey,
environment: "test",     locale:
"en-US",     countryCode: "US",
amount: {       value: amount,
currency: "USD",
    },
  });
  customCard = new CustomCard(checkout, {
type: "card",
    brands: ["mc", "visa", "amex", "bcmc", "maestro"],
styles: {       error: {         color: "red",
      },
validated: {
color: "green",
      },
placeholder: {
color: "#d8d8d8",
      },
    },     //
Events
    onSubmit: handleOnSubmit,
    // this is required for Encrytping the card
onChange: function (state) {
      if (state.data && state.data.paymentMethod &&
state.data.paymentMethod.encryptedCardNumber &&
state.data.paymentMethod.encryptedExpiryMonth &&
state.data.paymentMethod.encryptedExpiryYear &&
state.data.paymentMethod.encryptedSecurityCode) {         //
Log encrypted data when field becomes valid
console.log("Valid field - Encrypted data:", {
          encryptedCardNumber: state.data.paymentMethod.encryptedCardNumber,
encryptedExpiryMonth: state.data.paymentMethod.encryptedExpiryMonth,
encryptedExpiryYear: state.data.paymentMethod.encryptedExpiryYear,
encryptedSecurityCode: state.data.paymentMethod.encryptedSecurityCode,
brand: state.data.paymentMethod.brand
        });
}
      validateCardForm(state);
    },
    onFieldValid: function (state) {
validateCardForm(state);
    },
    onBrand: function (event) {
      const icon = document.getElementById("card-brand-icon");
if (event.brand && event.brand !== "unknown") {
icon.src = `https://checkoutshopper-
test.adyen.com/checkoutshopper/images/logos/${event.brand}.svg`;
icon.style.display = "block";
      } else {
        icon.style.display = "none";
      }
    },
    onError: function () { },     onFocus:
function () { },     onBinValue: function
```

ORACLE

```
(bin) { },     onBinLookup: function
(callbackObj) { },
  }).mount("#customCard-container");
   document.getElementById("payButton").addEventListener("click", async (e) => {
const name = document.getElementById("cardHolderName").value.trim();       let
hasError = false;

    if (!name) {
      document.getElementById("cardHolderNameError").style.display = "block";
hasError = true;     } else {
      document.getElementById("cardHolderNameError").style.display = "none";
}      if (hasError) {
      e.preventDefault();
return;
    }
    if (!customCard) {
      console.error("Card component not initialized yet");
return;
    }

    // Add loading state
    const payButton = document.getElementById("payButton");
const originalText = payButton.textContent;
payButton.disabled = true;
    payButton.textContent = "Processing...";
     try
{
      customCard.submit();
} catch (error) {
      console.error("Payment submission error:", error);
payButton.disabled = false;       payButton.textContent
= originalText;
    }
  });

  // Hide errors on input
document
    .getElementById("cardHolderName")
.addEventListener("input", function () {
if (this.value.trim()) {
        document.getElementById("cardHolderNameError").style.display = "none";
      }
      validateCardForm(customCard.state || { isValid: false });
    });
}
```

ORACLE

Encrypted Card Example

The result looks as follows:

```
{

   "encryptedCardNumber":"18$MT6ppy0FAMVMLH...",

   "encryptedExpiryMonth":"18$MT6ppy0FAMVMLH...",

   "encryptedExpiryYear":"18$MT6ppy0FAMVMLH...",

   "encryptedSecurityCode":"18$MT6ppy0FAMVMLH...",

}
```

ORACLE

Integrating iOS App for Retrieving Encrypted Card Details

The following steps show how to integrate the required step to make a payment to your existing iOS application.

Step 1: Import the library

Follow Apple's Adding Package Dependencies to Your App guide on how to add a Swift Package dependency.

Use https://github.com/Adyen/adyen-ios as the repository URL.

Specify the version to be at least **4.9.0**.

Step 2: Add validations

As a best practice, add iOS application validations for the fields Card Number, Card Expiry Date, and Card Security Code. These validations can be executed while the field is being edited or before making the call to make the payment.

The following classes and methods are available for the validations. The three fields are type string and the Expiry Date is the format *MMDD*.

```
//This import is required import
AdyenCard

//cardNumber: String
let isCardValid = cardValidator.isValid(cardNumber)
//cardCvv: String
let isCvvValid = securityCodeValidator.isValid(cardCvv)
//cardExpiryDate: String //
with format MMDD
let isExpirationDateValid = expiryValidator.isValid(cardExpiryDate)
//Validation before the call to make the payment. No need to call this method if you are
calling a server side API for making a Payment.
if isCardValid && isCvvValid && isExpirationDateValid && !cardHolderName.isEmpty {
makePayment(...)
}
```

ORACLE

Step 3: Encrypt card fields

When making a payment, you must encrypt the credit card information.

Use the following method to encrypt the information. The three fields are strings, the expiry month should be in the format *MM* and the expiry year in the format *YYYY*. Provide the public key for the encryption when the client is onboarded.

```
//These imports are required
import AdyenCard import
AdyenEncryption

// Create the card object to perform the encryption
// expiryMonth: String with format MM
// expiryYear: String with format YYYY
let cardObject = Card(number: cardNumber, securityCode: cardCvv, expiryMonth: month,
expiryYear: year) do {
    //Encrypt credit data     //PUBLIC_KEY: string     let encryptedCard
= try CardEncryptor.encrypt(card: cardObject, with:
PUBLIC_KEY)
} catch let error {
    //Handle error in encryption
}
```

ORACLE

Step 4: Retrieve a session token

Once the card information is validated and encrypted, send the payment using the makePayment endpoint.

To call the makePayment endpoint, you need a session token. Retrieve the token by using the corresponding API and the assigned Client Secret ID.

```
var request = URLRequest(url: TOKEN_ENDPOINT)
request.httpMethod = "POST"
request.setValue("application/x-www-form-urlencoded;charset=UTF-8",
forHTTPHeaderField: "Content-Type")
request.setValue("Basic \(CLIENT_SECRET_ID)", forHTTPHeaderField: "Authorization")
let body = "grant_type=client_credentials&scope= OPOP_COMMON/opop.payments"

request.httpBody = body.data(using: .utf8)

URLSession.shared.dataTask(with: request) { data, response, error in
if let error = error {
        print("Token request failed:
\(error)")         return     }

    if let httpResponse = response as? HTTPURLResponse {
print("HTTP Status Code: \(httpResponse.statusCode)")
print("Response Headers: \(httpResponse.allHeaderFields)")     }

    guard let data = data else {
        print("No data received for token")
return     }

    if let jsonObject = try? JSONSerialization.jsonObject(with: data, options:
[]),          let prettyData = try? JSONSerialization.data(withJSONObject:
jsonObject, options: .prettyPrinted),
         let jsonString = String(data: prettyData, encoding: .utf8) {
print("JSON Response: \n\(jsonString)")
    } else {                 print("Failed to
decode JSON")
    }

    if let json = try? JSONSerialization.jsonObject(with: data) as? [String:
Any],          let token = json["access_token"] as? String {
self.sessionToken = token
    }
}.resume()
```

ORACLE

Step 5: Make a payment

After getting the session token, send it in an authorization header as a bearer token. The card fields should be the ones encrypted in previous steps.

```
var request = URLRequest(url: MAKE_PAYMENT_ENDPOINT) var result = ""
request.httpMethod = "POST"
request.setValue("application/json", forHTTPHeaderField: "Content-Type")
// Use the session toke retrieved in the previous step for authorization
request.setValue("Bearer \(self.sessionToken)", forHTTPHeaderField: "Authorization")
// Build your request body considering that the encrypted fields for the card
information should come from the object generated in previous steps        var
body: [String: Any] = [



        "captureType": "automatic",
        "captureDelayHours": 0,
"transactionDetails": [
            "amount": AMOUNT TO PAY,
            "currency": "USD"
        ],
        "paymentMethod": [
            "type": "scheme",
            "encryptedCardNumber": encryptedCard.number,
            "encryptedExpiryMonth": encryptedCard.expiryMonth,
            "encryptedExpiryYear": encryptedCard.expiryYear,
            "encryptedSecurityCode": encryptedCard.securityCode
        ],
        "billingAddress": [
             "postalCode": POSTAL CODE LINKED TO THE CARD,
        ],
        "returnUrl": RETURNING URL,
        "transactionReferences": [
             "clientTransactionId": CLIENT TRANSACTION ID
        ],
        "channelCode": YOUR ASSIGNED CHANNEL CODE,
        "locationReference": YOUR ASSIGNED LOCATION REFERENCE,
        "organizationReference": YOUR ORGANIZATION REFERENCE,
        "shopperInteraction":"Ecommerce"
]
          do
{
     request.httpBody = try JSONSerialization.data(withJSONObject: body, options: [])
} catch {
     print("Failed to encode payment body: \(error)")
}

let (data, _) = try await URLSession.shared.data(for: request)

let rawResponse = String(data: data, encoding: .utf8) ?? "Could not convert to string"

print("Raw response:\n\(rawResponse)")

if let jsonObject = try? JSONSerialization.jsonObject(with: data, options: []),
let prettyData = try? JSONSerialization.data(withJSONObject: jsonObject, options:
.prettyPrinted),
```

ORACLE

```
    let jsonString = String(data: prettyData, encoding: .utf8) {
print("Pretty-printed JSON:\n\(jsonString)")       result =
"Make Payment Result: \(jsonString)"
} else {
      print("Failed to decode or pretty-print JSON.")
}
          return
result
```

Integrating Google Android App for Retrieving Encrypted Card Details

Step 1: Import the library

To use the Android Module, add the client-side encryption dependency in the build.gradle file by implementing `com.adyen.checkout:cse:5.13.0`.

Step 2: Validation and Encryption

As a best practice, add Android application validations for the fields Card Number, Card Expiry Date, and Card Security Code. These validations can be executed while the field is being edited or before making the call to make the payment.

Use the following classes and methods to perform validations. The three fields are type string and the Expiry Date has the format *MMDD*.

ORACLE

```
import com.adyen.checkout.cse.CardEncrypter import
com.adyen.checkout.cse.EncryptionException import
com.adyen.checkout.cse.UnencryptedCard
private val binding: DialogEncryptedInputBinding get() = requireNotNull(_binding)
// DialogEncryptedInputBinding is the hosted field component private
val clientEncryptionPublicKey ="clientEncryptionPublicKey"
val cardNumber = binding.editTextCardNumber.text.toString().replace("\\D".toRegex(), "")
val expiryDate = binding.editTextExpiryDate.text.toString().replace("/", "") val cvc =
binding.editTextCvc.text.toString().trim() val saveCard =
binding.checkBoxSaveCard.isChecked
// Validate card inputs
val validationResult = validateCardInputs(cardNumber, expiryDate,
cvc) lifecycleScope.launch { try {
val encryptedCard = withContext(Dispatchers.IO) {
// Validate public key format
validateClientEncryptionPublicKey(clientEncryptionPublicKey)
// Build unencrypted card object
val unencryptedCard = UnencryptedCard.Builder()
.setNumber(cardNumber)
.setExpiryDate(expiryMonth, expiryYear)
.setCvc(cvc)
.build()
// Encrypt using Adyen's CardEncrypter
CardEncrypter.encryptFields(unencryptedCard, clientEncryptionPublicKey)
}
// Load dynamic settings for the payment request val
paymentRequest = apiSettingsManager.loadSettings() //
Call PaymentService with encrypted card data val
paymentResult = paymentService.processPayment(
paymentRequest = paymentRequest,
encryptedCardNumber = encryptedCard.encryptedCardNumber ?: "",
encryptedExpiryMonth = encryptedCard.encryptedExpiryMonth ?: "",
encryptedExpiryYear = encryptedCard.encryptedExpiryYear ?: "",
encryptedSecurityCode = encryptedCard.encryptedSecurityCode ?: "", saveCard
= saveCard,
)
} catch (e: EncryptionException) {} catch (e: Exception) {
} finally {
} } fun validateCardDetails(cardNumber: String, expiryDate: String, cvc:
String): ValidationResult { if (cardNumber.isEmpty()) {
return ValidationResult(false, "Please enter a card number")
if (cardNumber.length !in 13..19) {
```

ORACLE

```
return ValidationResult(false, "Card number must be between 13-19 digits")
} }
if (expiryDate.length != 4) {
return ValidationResult(false, "Please enter expiry date as MM/YY")
}
val month = expiryDate.substring(0, 2).toIntOrNull() if
(month == null || month !in 1..12) {
return ValidationResult(false, "Invalid expiry month")
}
if (cvc.isEmpty()) {
return ValidationResult(false, "Please enter CVC")
}
if (cvc.length !in 3..4) {
return ValidationResult(false, "CVC must be 3-4 digits")
}
return ValidationResult(true, "Valid card details")
}
```

ORACLE

Step 3: Making Payments

Once the card information is validated and encrypted, send the payment using the makePayment endpoint.

To call the makePayment endpoint, you need an access token. Retrieve the token using the following API and the assigned Client Secret ID. You do not need to call this method when calling a server-side API for making a payment.

Step 4: Getting Access Tokens

```
suspend fun getAccessToken(){
val header="headerVal" try { val
url = "https://idcs-
0e86f01b7f094720af224335e43f84eb.identity.oraclecloud.com/oauth2/v1/token"
val body = "grant_type=client_credentials&scope= OPOP_COMMON/opop.payments"
val request = Request.Builder()
.url(url)
.addHeader("Authorization", "Basic $header")
.addHeader("Content-Type", "application/x-www-form-urlencoded;charset=UTF-8")
.post(body.toRequestBody("application/x-www-form-urlencoded".toMediaType()))
.build() }
return access_tokensuspend fun processPayment(){
val accessToken = getAccessToken()
val requestJson = JSONObject().apply {put("paymentMethod", JSONObject().apply {
put("type", "scheme")
put("encryptedCardNumber", encryptedCardNumber)
put("encryptedExpiryMonth", encryptedExpiryMonth)
put("encryptedExpiryYear", encryptedExpiryYear)
put("encryptedSecurityCode", encryptedSecurityCode)
//*** add others fields as part of request body
}) }
val request = Request.Builder()
.url(PAYMENT_ENDPOINT)
.post(requestJson.toString().toRequestBody("application/json".toMediaType()))
.addHeader("Content-Type", "application/json")
.addHeader("Authorization", "Bearer $accessToken")
.addHeader("Accept", "application/json").build()
// Execute request
val response = httpClient.newCall(request).execute()}
```

ORACLE

Card On File

This section explains, for Web, iOS, and Android, how to support tokenized card usage, enabling customers to:

- Save a card: /api/v1/oracle/makePayment
- View a list of saved cards: /api/v1/oracle/fetchAllCards
- Pay using a saved card: /api/v1/oracle/payViaToken
- Delete a saved card: /api/v1/oracle/removeCardDetails

Web

Save Card

Update your existing *makeOraclePayment* function by including a tokenization object in the request body. Add the following code to your request payload:

```
"tokenization": {
  "savePaymentMethod": savePaymentMethod, // boolean value
  "savedPaymentMethodUsage": "CardOnFile", // static
  "shopperReference": shopperReference || "test-shopper-333" // unique user
identifier
}
```

Retrieve the value of the “Save Card” checkbox by adding the following code at the beginning of the *makeOraclePayment* function:

```
const saveCardCheckbox = document.getElementById("saveCardCheckbox");
const savePaymentMethod = saveCardCheckbox ? saveCardCheckbox.checked :true
```

ORACLE

List Saved Cards

This step retrieves cards that were previously tokenized during successful payments and is required for presenting these options to the customer.

Prerequisites

- A valid shopperReference stored in the session (you should store this after login or account selection).
- Access to the client credentials for fetching a valid Bearer token.

How to Use It

When the saved cards dialog or modal is opened, call the *getSavedCardsForDisplay* function. Store the returned list in a local array (for example, *apiSavedCards*) and use your UI components to render the saved cards.

Be sure to associate the corresponding token values (*opcsToken*) with each "Pay" button so they can be used to initiate tokenized payments.

```
async function getSavedCardsForDisplay () {
        try {
          const shopperReference = sessionStorage.getItem("shopperReference") ||
"test-shopper-333";
          // Get access token first
          const tokenResponse = await getAccessTokenForDialog();
          const paymentMethodsBody = {
            "channelCode": "ccextU7CJ8O",
            "organizationReference": "RGBU_Ref",
            "locationReference": "TEST REF HS",
            "shopperReference": shopperReference,
            "transactionReferences": {
              "clientTransactionId": `Hi-techDemo-1234-${Date.now()}`
            } "
          };
```

ORACLE

```
const response = await
fetch("https://stage.opcs.ocs.oraclecloud.com/api/v1/oracle/fetchAllCards", {
            method: "POST",
            headers: {
              'Content-Type': 'application/json',
              'Authorization': 'Bearer ' + tokenResponse.access_token
            },
            body: JSON.stringify(paymentMethodsBody)

        });

          const data = await response.json();
          console.log("API Response for dialog:", data);

          if (data.savedPaymentMethods && data.savedPaymentMethods.length > 0) {
            apiSavedCards = data.savedPaymentMethods.map(card => ({
              brand: card.brandName ? card.brandName.toLowerCase() : 'card',
              last4: card.lastFour || '0000',
              expiry: card.expiryMonth && card.expiryYear ?
`${card.expiryMonth}/${card.expiryYear}` : 'N/A',
              opcsToken: card.opcsToken,
              brandName: card.brandName,
              holderName: card.holderName
            }));
          } else {
            apiSavedCards = [];
          }

          return apiSavedCards;
        } catch (error) {
          console.error("Error fetching saved cards for dialog:", error);
          apiSavedCards = [];
          return [];
        }
```

ORACLE

Making a Payment Using a Saved Token (payViaToken)

After retrieving saved cards using the *fetchSavedCards* API, you can enable users to select a card and initiate a payment with its *opcsToken*.
The *payViaToken* function uses this token to securely process the payment, eliminating the need to collect card details again from the user.

Prerequisites
You must have the opcsToken from the selected saved card.
An active session with the shopperReference stored (e.g., in sessionStorage).
A valid access token via client credentials.

How to Use It
When a user selects a saved card from the list, invoke the payViaToken(opcsToken) function to initiate the payment.
This can be triggered directly from a "Pay" button within your dialog or form.
Be sure to implement loading indicators and error handling to provide a smooth and user-friendly experience.

```
async function payViaToken(token) {
  try {
    const loaderElement = document.getElementById('loader');
    if (loaderElement) {
      loaderElement.style.display = 'block';
    }
    const tokenValue = await getAccessToken();

    const paymentBody = {
      "transactionType": "011",
      "transactionDetails": {
        "amount": amount,
        "currency": "USD"
      },
      "paymentMethod": {
        "type": "oracletoken",
        "oraclePayToken": token
      },
      "billingAddress": {
        "city": "New York",
        "country": "US",
        "street": "123 Main Street",
        "houseNumberOrName": "123",
        "postalCode": "10001",
        "stateOrProvince": "NY"
      },
      "returnUrl": "https://www.oracle.com",
      "transactionReferences": {
        "clientTransactionId": `token-payment-${Date.now()}`
      },
      "channelCode": "ccextU7CJ8O",
      "organizationReference": "RGBU_Ref",
      "locationReference": "TEST REF HS",
      "shopperInteraction": "Ecommerce",
      "captureType": "Manual",
      "shopperReference": shopperReference || "test-shopper-333"
    };
```

ORACLE

```
const response = await
fetch("https://stage.opcs.ocs.oraclecloud.com/api/v1/oracle/makePayment", {
      method: "POST",
      headers: {
        'Content-Type': 'application/json',
        'Authorization': 'Bearer ' + tokenValue.access_token
      },
      body: JSON.stringify(paymentBody)
    });

    const data = await response.json();
    const loaderElementSuccess = document.getElementById('loader');
    if (loaderElementSuccess) {
      loaderElementSuccess.style.display = 'none';
    }

    if (data.resultCode === 'Authorised' || data.resultCode === 'Success') {
      window.location.href = "./success.html";
    } else {
      alert("Unable to make payment");
    }
  } catch (error) {
    console.error("Error paying via token:", error);
    const loaderElementError = document.getElementById('loader');
    if (loaderElementError) {
      loaderElementError.style.display = 'none';
    }
    alert("Payment failed. Please try again.");  }}
```

ORACLE

Deleting a Saved Card (deleteCard)

To enable users to manage their saved cards, provide a "Delete Card" option. When a user selects this option, the *removeCardDetails* endpoint removes the specified card using its associated OPCS token.

Prerequisites
You must have the opcsToken from the selected saved card.
A valid access token via client credentials.

How to Use It

- Call deleteCard(opcsToken, cardIndex) when the delete icon is clicked.
- Ensure the opcsToken is unique to the card and provided by fetchSavedCards().
- cardIndex is used to update the UI.

```
async function deleteCard(opcsToken, cardIndex) {
  if (!confirm("Are you sure you want to delete this card?")) {
    return;
  }

  const deleteBtn = document.querySelectorAll(".delete-card-btn")[cardIndex];
  deleteBtn.disabled = true;
  deleteBtn.textContent = "⏳"; // Optional: loading state

  try {
    await attemptDelete(opcsToken, cardIndex);
    alert("Card deleted successfully!");
  } catch (error) {
    console.warn("First attempt failed, retrying...", error);
    try {
      await attemptDelete(opcsToken, cardIndex);
      alert("Card deleted successfully on retry!");
    } catch (finalError) {
      console.error("Delete failed after retry:", finalError);
      alert("Failed to delete card. Please try again.");
    }
  } finally {
    deleteBtn.disabled = false;
    deleteBtn.textContent = "";  }}
```

attemptDelete() Helper Function:

```
async function attemptDelete(opcsToken, cardIndex) {
  const shopperReference = sessionStorage.getItem("shopperReference") || "test-
shopper-333";
  const tokenResponse = await getAccessToken();
const removeCardBody = {
    channelCode: "ccextU7CJ8O",
    locationReference: "TEST REF HS",
    organizationReference: "RGBU_Ref",
    shopperReference,
    transactionReferences: {
      clientTransactionId: `delete-${Date.now()}`
    },
    opcsToken: opcsToken
  };

  const response = await
fetch("https://stage.opcs.ocs.oraclecloud.com/api/v1/oracle/removeCardDetails", {
    method: "POST",
    headers: {
      'Content-Type': 'application/json',
      'Authorization': 'Bearer ' + tokenResponse.access_token
    },
    body: JSON.stringify(removeCardBody)
  });

  const data = await response.json();
  console.log("Remove card response:", data);

  if (response.ok) {
    // Update local card list
    apiSavedCards.splice(cardIndex, 1);
    if (selectedCardIndex >= apiSavedCards.length) {
      selectedCardIndex = Math.max(0, apiSavedCards.length - 1);
    }

    // UI updates after delete
    if (apiSavedCards.length > 0) {
      renderMangedCards(); // Stay on Manage page
    } else {
      showNewCardPage(); // Go to "Add New Card" page
    }
  } else {
    throw new Error(data.message || "Failed to delete card");  }}
```

ORACLE

iOS

Save Card

To save a card on file, a payment needs to be executed. To make a payment, the credit card information should be encrypted.

The following method is available to execute this step. The 3 fields are strings, the Expiry Month should be in the format MM, and the Expiry Year in the format YYYY. The public key required for the encryption should be given when the client is onboarded.

```
//These imports are required

import AdyenCard

import AdyenEncryption

// Create the card object to perform the encryption

// expiryMonth: String with format MM

// expiryYear: String with format YYYY

let cardObject = Card(number: cardNumber, securityCode: cardCvv, expiryMonth: month,
expiryYear: year)

do {

    //Encrypt credit data

    //PUBLIC_KEY: string

    let encryptedCard = try CardEncryptor.encrypt(card: cardObject, with: PUBLIC_KEY)

} catch let error {

    //Handle error in encryption
```

The encrypted card information should be used to make the payment.

```
var request = URLRequest(url: MAKE_PAYMENT_ENDPOINT)

var result = ""

request.httpMethod = "POST"

request.setValue("application/json", forHTTPHeaderField: "Content-Type")

// Use the session toke retrieved in the previous step for authorization

request.setValue("Bearer \(AUTHENTICATION_SESSION_TOKEN)", f

orHTTPHeaderField: "Authorization")
```

ORACLE

```
// Build your request body considering that the encrypted fields for the card information
should come from the object generated in previous steps

var body: [String: Any] = [

        "captureType": "automatic",

        "captureDelayHours": 0,

        "transactionDetails": [

            "amount": AMOUNT TO PAY,

            "currency": "USD"

        ],

        "paymentMethod": [

            "type": "scheme",

            "encryptedCardNumber": encryptedCard.number,

            "encryptedExpiryMonth": encryptedCard.expiryMonth,

            "encryptedExpiryYear": encryptedCard.expiryYear,

            "encryptedSecurityCode": encryptedCard.securityCode

        ],

        "billingAddress": [

             "postalCode": POSTAL CODE LINKED TO THE CARD,

        ],

        "returnUrl": RETURNING URL,

        "transactionReferences": [

             "clientTransactionId": CLIENT TRANSACTION ID

        ],

        "channelCode": YOUR ASSIGNED CHANNEL CODE,

        "locationReference": YOUR ASSIGNED LOCATION REFERENCE,

        "organizationReference": YOUR ORGANIZATION REFERENCE,

        "shopperInteraction":"Ecommerce",

        "tokenization" = [

                "savePaymentMethod": true,

                "savedPaymentMethodUsage": "CardOnFile",

                "shopperReference": SHOPPER REFERENCE TO RETRIEVE COF IN FUTURE STEPS
```

ORACLE

```
do {
     request.httpBody = try JSONSerialization.data(withJSONObject: body, options: [])
} catch {
     print ("Failed to encode payment body: \(error)")
}
let (data, _) = try await URLSession.shared.data(for: request)
let rawResponse = String(data: data, encoding: .utf8) ?? "❌ Could not convert to string"
print("📄 Raw response:\n\(rawResponse)")
if let jsonObject = try? JSONSerialization.jsonObject(with: data, options: []),
    let prettyData = try? JSONSerialization.data(withJSONObject: jsonObject, options:
.prettyPrinted),
    let jsonString = String(data: prettyData, encoding: .utf8) {
      print("Pretty-printed JSON:\n\(jsonString)")
      result = "Make Payment Result: \(jsonString)"
} else {
      print("Failed to decode or pretty-print JSON.")
}
```

List Saved Cards

This step retrieves cards that were previously tokenized during successful payments and is required for presenting these options to the customer.

Prerequisites

- A valid shopperReference stored in the session (you should store this after login or account selection).
- Access to the client credentials for fetching a valid Bearer token.

How to Use It

To pay with a card on file, retrieve the list of the previous saved cards. This is done by calling the fetchAllCards endpoint. The Shopper Reference field should be the value used to tokenize and save the cards in the previous step.

ORACLE

```
var request = URLRequest(url: FETCH_ALL_CARDS_ENDPOINT)
request.httpMethod = "POST"
request.setValue("application/json", forHTTPHeaderField: "Content-Type")
request.setValue("Bearer \(AUTHENTICATION_SESSION_TOKEN)", forHTTPHeaderField:
"Authorization")

var body: [String: Any] = [
        "organizationReference": YOUR ASSIGNED ORGANIZATION REFERENCE,
        "channelCode": YOUR ASSIGNED CHANNEL CODE,
        "transactionReferences": [
            "clientTransactionID": CLIENT TRANSACTION ID
        ],
        "shopperReference": PREVIOUS SHOPPER REFERENCE
]
```

```
   do {
    request.httpBody = try JSONSerialization.data(withJSONObject: body, options: [])
} catch {
    print("Failed to encode payment body: \(error)")
}
let (data, _) = try await URLSession.shared.data(for: request)

let rawResponse = String(data: data, encoding: .utf8) ?? "Could not convert to string"

print("Raw response:\n\(rawResponse)")
if let jsonObject = try? JSONSerialization.jsonObject(with: data, options: []),
    let prettyData = try? JSONSerialization.data(withJSONObject: jsonObject, options:
.prettyPrinted),
    let jsonString = String(data: prettyData, encoding: .utf8) {
     print("Pretty-printed JSON:\n\(jsonString)")
     let jsonDict = jsonObject as? [String: Any]
     cardsOnFile = jsonDict!["savedPaymentMethods"] as! [[String: String]]
} else {
     print("Failed to decode or pretty-print JSON.")
}
```

ORACLE

Making a Payment Using a Saved Token (payViaToken)

After retrieving saved cards using the *fetchSavedCards* API, you can enable users to select a card and initiate a payment with its *opcsToken*.
The *payViaToken* function uses this token to securely process the payment, eliminating the need to collect card details again from the user.

Prerequisites
You must have the opcsToken from the selected saved card.
An active session with the shopperReference stored (e.g., in sessionStorage).
A valid access token via client credentials.

How to Use It

Once a card from the cards on file list is selected, use the field opcsToken to make the payment.

```
var request = URLRequest(url: MAKE_PAYMENT_ENDPOINT)

var result = ""

request.httpMethod = "POST"

request.setValue("application/json", forHTTPHeaderField: "Content-Type")

// Use the session toke retrieved in the previous step for authorization

request.setValue("Bearer \(AUTHENTICATION_SESSION_TOKEN)", forHTTPHeaderField:
"Authorization")
```

ORACLE

```
var body: [String: Any] = [
        "captureType": "automatic",
        "captureDelayHours": 0,
        "transactionDetails": [
            "amount": AMOUNT TO PAY,
            "currency": "USD"
        ],        "paymentMethod": [
            "type": "oracletoken",
            "oraclePayToken": OPCS TOKEN FROM SELECTED CARD
        ],
        "returnUrl": RETURNING URL,
        "transactionReferences": [
             "clientTransactionId": CLIENT TRANSACTION ID
        ],
        "channelCode": YOUR ASSIGNED CHANNEL CODE,
        "locationReference": YOUR ASSIGNED LOCATION REFERENCE,
        "organizationReference": YOUR ORGANIZATION REFERENCE,
        "shopperInteraction":"Ecommerce"
]
do {
     request.httpBody = try JSONSerialization.data(withJSONObject: body, options: [])
} catch {
     print("Failed to encode payment body: \(error)")
}
let (data, _) = try await URLSession.shared.data(for: request)
let rawResponse = String(data: data, encoding: .utf8) ?? "Could not convert to
string"
print("raw response:\n\(rawResponse)")
if let jsonObject = try? JSONSerialization.jsonObject(with: data, options: []),
    let prettyData = try? JSONSerialization.data(withJSONObject: jsonObject, options:
.prettyPrinted),
    let jsonString = String(data: prettyData, encoding: .utf8) {
      print("Pretty-printed JSON:\n\(jsonString)")
      result = "Make Payment Result: \(jsonString)"
} else {
      print(" Failed to decode or pretty-print JSON.")
```

ORACLE

Deleting a Saved Card (deleteCard)

To enable users to manage their saved cards, provide a "Delete Card" option. When a user selects this option, the *removeCardDetails* endpoint removes the specified card using its associated OPCS token.

Prerequisites
You must have the opcsToken from the selected saved card.
A valid access token via client credentials.

How to Use It

A card should be selected to delete, and its OPCS token is needed to perform this action.

```
var request = URLRequest(url: DELETE ENDPOINT)

var result = ""

request.httpMethod = "POST"

request.setValue("application/json", forHTTPHeaderField: "Content-Type")

// Use the session toke retrieved in the previous step for authorization

request.setValue("Bearer \(AUTHENTICATION_SESSION_TOKEN)", forHTTPHeaderField: "Authorization")

var body: [String: Any] = [

        "transactionReferences": [

             "clientTransactionId": CLIENT TRANSACTION ID

        ],

        "channelCode": YOUR ASSIGNED CHANNEL CODE,

        "organizationReference": YOUR ORGANIZATION REFERENCE,

        "shopperReference": YOUR SHOPPER REFERENCE,

        "opcsToken": TOKEN TO DELETE

]

do {

     request.httpBody = try JSONSerialization.data(withJSONObject: body, options: [])

} catch {

     print("Failed to encode payment body: \(error)")

}

let (data, _) = try await URLSession.shared.data(for: request)

let rawResponse = String(data: data, encoding: .utf8) ?? "Could not convert to string"

print(" Raw response:\n\(rawResponse)")

if let jsonObject = try? JSONSerialization.jsonObject(with: data, options: []),

    let prettyData = try? JSONSerialization.data(withJSONObject: jsonObject, options:
.prettyPrinted),

    let jsonString = String(data: prettyData, encoding: .utf8) {

      print("Pretty-printed JSON:\n\(jsonString)")

      result = "Make Payment Result: \(jsonString)"

} else {

      print(" Failed to decode or pretty-print JSON.")

}

 return result
```

ORACLE

Google Android

List Saved Cards

This step retrieves cards that were previously tokenized during successful payments and is required for presenting these options to the customer.

Prerequisites

- A valid shopperReference stored in the session (you should store this after login or account selection).
- Access to the client credentials for fetching a valid Bearer token.

How to Use It

To get the list of all the saved cards saved for the shopperRefernce below are the steps: Please call the access Token Api before making api call

```
FetchAllCards
val requestJson = JSONObject().apply {
               put("channelCode", settings.channelCode)
               put("organizationReference", settings.organizationReference)
                put("shopperReference", settings.shopperReference)
                put("transactionReferences", JSONObject().apply {
                put("clientTransactionId", dynamicClientTransactionId)
                 })
            }val request = Request.Builder()
.url(FETCH_CARDS_ENDPOINT)
.post(requestJson.toString().toRequestBody("application/json".toMediaType()))
.addHeader("Content-Type", "application/json")
.addHeader("Authorization", "Bearer $accessToken")
.addHeader("Accept", "application/json")
.build()
```

ORACLE

Making a Payment Using a Saved Token (payViaToken)

After retrieving saved cards using the *fetchSavedCards* API, you can enable users to select a card and initiate a payment with its *opcsToken*.
The *payViaToken* function uses this token to securely process the payment, eliminating the need to collect card details again from the user.

Prerequisites
You must have the opcsToken from the selected saved card.
An active session with the shopperReference stored (e.g., in sessionStorage).
A valid access token via client credentials.

```
private fun processSavedCardPayment(savedCard: SavedPaymentMethod) {
     lifecycleScope.launch {
           try {
               // Load API settings
               val settings = apiSettingsManager.loadSettings() // load
               // Call makePayment API with opcsToken
               val result = paymentService.makePayment(
                   settings = settings,
                   opcsToken = savedCard.opcsToken,
                   amount = 200,
                   currency = "USD")
```

ORACLE

Follow these steps to make a payment via Token from the list of saved cards. Call the access Token API before making api call.

```
suspend fun makePayment(
        settings:ApiRequestSettings,
       opcsToken:String,
        amount:Int=200,
        currency:String="USD"
):Result<PaymentResponse> =
withContext(Dispatchers.IO) {
    try {
// Build payment request JSON according to your specification
        val requestJson = JSONObject().apply {
            put("transactionType", "011")
            put("transactionDetails", JSONObject().apply {
                put("amount", amount)
                put("currency", currency)
            })
            put("paymentMethod", JSONObject().apply {
                put("type", "oracletoken")
                put("oraclePayToken", opcsToken)
            })
           //Add other Details as part of the Api Spec
// Create OkHttp request
        val request = Request.Builder()
                .url("https://stage.opcs.ocs.oraclecloud.com/api/v1/oracle/makePayment")

.post(requestJson.toString().toRequestBody("application/json".toMediaType()))
                .addHeader("Content-Type", "application/json")
                .addHeader("Accept", "application/json")
                .addHeader("Authorization", "Bearer $accessToken")
                .build()
// Execute request
        val response = httpClient.newCall(request).execute()
    } catch (e:Exception){
         //Log Exception
        Result.failure(e)}}
```

ORACLE

Deleting a Saved Card (deleteCard)

To enable users to manage their saved cards, provide a "Delete Card" option. When a user selects this option, the *removeCardDetails* endpoint removes the specified card using its associated OPCS token.

Prerequisites
You must have the opcsToken from the selected saved card.
A valid access token via client credentials.

How to Use It

Follow these steps to remove a card from the saved card list. Call the access Token API before making api call.

```
suspend fun removeCard(
        opcsToken: String,
        settings: ApiRequestSettings
    ): Result<Unit> = withContext(Dispatchers.IO) {
                JSONObject().apply {
                put("opcsToken", opcsToken)
                 put("shopperReference", settings.shopperReference)
                   //Add other fields as part of API Spec
}
val request = Request.Builder()
                .url(REMOVE_CARD_ENDPOINT)

.post(requestJson.toString().toRequestBody("application/json".toMediaType()))
                .addHeader("Content-Type", "application/json")
                .addHeader("Accept", "application/json")
                .addHeader("Authorization", "Bearer $accessToken")
                .build()
            // Execute request
            val response = httpClient.newCall(request).execute()
}
```

ORACLE

Payment Gateway Configuration for Google Pay

To successfully enable Google Pay transactions, specific configuration steps are required to ensure shopper payment credentials are securely encrypted and processed.

Tokenization Specification Configuration

Google Pay uses a tokenizationSpecification object to determine how payment data is securely handled. You must configure this object to use Oracle public keys by setting the correct parameters. The specification requires:

1. Setting Adyen as the payment gateway
2. Providing the exact merchant account name supplied for your integration

Failure to configure these parameters correctly will result in payment tokens that Adyen cannot decrypt, causing all Google Pay payment attempts to fail.

Example Configuration

```
const tokenizationSpecification = {
type: 'PAYMENT_GATEWAY',
parameters: {
'gateway': 'adyen',
'gatewayMerchantId': 'Oracle_FBGIU_US_RBI_Prod3'
}
};
```

gateway: Must be set to 'adyen'

gatewayMerchantId: Must be set to your merchant account name (Oracle_FBGIU_US_RBI_Prod3)

ORACLE

Integration Guide for JWE API Support

Overview

JWE token encryption is supported as the single-card-data encryption path for scheme payments. In this flow, the cardholder details are encrypted on the client side, a JWE token is generated, and that token is sent to our backend API.

We provide the following needed to integrate:

- the X.509 certificate (Oracle will share it securely via SharePoint)
- the backend payment API
  Endpoint: https://opop.us-ashburn-1.ocs.oraclecloud.com/api/v1/oracle/makePayment

Parameters

Client-Side Implementation

Use the certificate that has been provided by Oracle and generate the token with a standard JWE

Generate JWE Token using the following script:

```
import * as jose from 'jose';

export async function generateJWEToken({
  x509Certificate,
  cardNumber,
  expiryMonth,
  expiryYear,
  cvc,
}) {
  if (!x509Certificate) {
    throw new Error('x509Certificate is required');
  }
  const publicKey = await jose.importX509(x509Certificate, 'RSA-OAEP-256');
  const payload = JSON.stringify({
    number: cardNumber,
    expiryMonth,
    expiryYear,
    cvc,
    generationtime: new Date().toISOString(),
  });
```

ORACLE

```
 const jweToken = await new jose.CompactEncrypt(
   new TextEncoder().encode(payload)
 )
   .setProtectedHeader({ alg: 'RSA-OAEP-256', enc: 'A256GCM', version: '1' })
   .encrypt(publicKey);

 return jweToken;
}
```

Handle JWE Token Payment Submission:

Endpoint: https://opop.us-ashburn-1.ocs.oraclecloud.com/api/v1/oracle/makePayment

```
async function handleJweSubmit(state, component) {
 try {
   const loaderElement = document.getElementById('loader');
   if (loaderElement) {
     loaderElement.style.display = 'block';
   }
   const tokenResponse = await getAccessToken();

   const jweToken = await generateJWEToken({
     x509Certificate: window.X509_CERTIFICATE,
     cardNumber: paymentMethod.cardNumber,
     expiryMonth: paymentMethod.expiryMonth,
     expiryYear: paymentMethod.expiryYear,
     cvc: paymentMethod.cvc
   });

   const paymentPayload = {
     ...state.data,
     paymentMethod: {
       type: 'scheme',
       encryptedCard: jweToken
     }
   };

   const response = await fetch('https://opop.us-ashburn-1.ocs.oraclecloud.com/api/v1/oracle/makePayment/', {
     method: 'POST',
     headers: {
       'Content-Type': 'application/json',
       'Authorization': 'Bearer ' + tokenResponse.access_token
     },
```

ORACLE

```
    body: JSON.stringify(paymentPayload)
  });

  if (!response.ok) {
    const errorData = await response.json();
    console.error('HTTP error:', errorData);
    throw new Error(`HTTP error! status: ${response.status}`);
  }

  const result = await response.json();
  console.log('JWE payment result:', result);

  if (loaderElement) {
    loaderElement.style.display = 'none';
  }

  if (result.resultCode === 'Authorised' || result.resultCode === 'Success') {
    window.location.href = './success.html';
  } else if (result.resultCode === 'Pending' || result.resultCode === 'Received') {
    window.location.href = './pending.html';
  } else {
    window.location.href = './failed.html';
  }
 } catch (error) {
  console.error('JWE submit error:', error);
  const loaderElement = document.getElementById('loader');
  if (loaderElement) {
    loaderElement.style.display = 'none';
  }
  window.location.href = './failed.html';
 }
}
```

JWE Token Payment Payload:

```
const JWEPayload= {
 "transactionType": "011",
 "transactionDetails": {
   "amount": amount,
   "currency": "USD"
 },
 "paymentMethod": {
```

ORACLE

```
    "type": "scheme",
    "encryptedCard": "<JWE_TOKEN>"
  },
  "billingAddress": {
    "city": "your_city",
    "country": "US",
    "houseNumberOrName": "your_house_number",
    "postalCode": "your_postal_code",
    "stateOrProvince": "your_state",
    "street": "your_street"
  },
  "returnUrl": "https://your-return-url.com",
  "transactionReferences": {
    "clientTransactionId": `apay-${Date.now()}`
  },
  "channelCode": "your_channel_code",
  "organizationReference": "your_organization_reference",
  "locationReference": "your_location_reference",
  "captureType": "Manual",
  "shopperInteraction": "Ecommerce",
  "tokenization": {
    "savePaymentMethod": savePaymentMethod.toString(),
    "savedPaymentMethodUsage": "CardOnFile",
    "shopperReference": "your_shopper_reference"
  },
  "shopperReference": "your_shopper_reference"
};
```

Token Expiration

A generated JWE token is valid for up to 24 hours.
It is recommended to use the token immediately after generation. Tokens that have expired will be rejected during payment processing.

Payment Method Payload Rules
When using type: "scheme", exactly one of the following approaches must be used.

Option 1 — Field-level Encryption

```
{
  "paymentMethod": {
```

ORACLE

```
    "type": "scheme",
    "encryptedCardNumber": "test_36006666333344",
    "encryptedExpiryMonth": "test_03",
    "encryptedExpiryYear": "test_2030",
    "encryptedSecurityCode": "test_737"
  }
}
```

Option 2 — JWE Token

```
{
  "paymentMethod": {
    "type": "scheme",
    "encryptedCard": "<JWE_TOKEN>"
  }
}
```

Invalid Example — Mixed Approaches

```
{
  "paymentMethod": {
    "type": "scheme",
    "encryptedCard": "<JWE_TOKEN>",
    "encryptedCardNumber": "test_36006666333344"
  }
}
```

An Error will be thrown in case both the approaches are mixed:

```
Provide either paymentMethod.encryptedCard OR
encryptedCardNumber/encryptedExpiryMonth/encryptedExpiryYear/encryptedSecurityCode, not both
```

Constraints

The `encryptedCard` field is mutually exclusive with the following fields:

- `encryptedCardNumber`
- `encryptedExpiryMonth`
- `encryptedExpiryYear`
- `encryptedSecurityCode`

Requests that include both formats will be rejected.
The `type` field must remain:

```
"type": "scheme"
```

Best Practices

Use a Single Encryption Strategy

Each request must use only one encryption format. Mixing formats within the same request is not supported.

Payload Validation

It is recommended to validate requests before sending:

- Ensure that either `encryptedCard` is present, or all required encrypted card fields are present.
- Ensure that both formats are never included in the same request.

Token Expiration Handling

If a request fails due to token expiration:

- Generate a new JWE token
- Retry the request

ORACLE

Connect with us

Call +**1.800.ORACLE1** or visit **oracle.com**. Outside North America, find your local office at: **oracle.com/contact**.

blogs.oracle.com facebook.com/oracle twitter.com/oracle