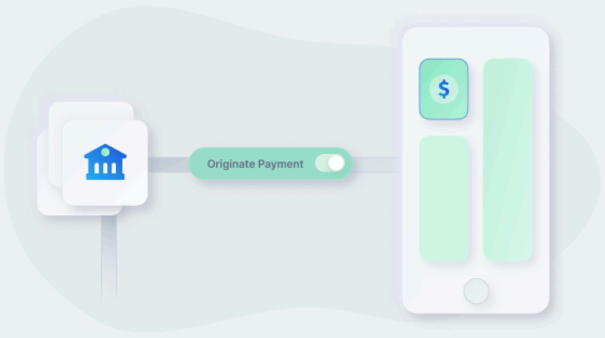
cross river | Docs

Tutorials

1. Tutorials

Need help using our APIs?

These step-by-step guides will show you how to set up and integrate key features.



Our tutorials are here to help you use Cross River's APIs and platform features in real-world scenarios. Whether you're setting up account onboarding, instant payments, or webhooks, these guides will take you from idea to implementation quickly and easily. Each tutorial comes with sample requests, code snippets, and best practices. This way, you can speed up your development and avoid any guesswork.

Build a sample app incorporating several different products in a single how-to.

Customer Management

- Onboard a customer
- Add a beneficial owner
- Add ID details

International payments

- Send an international payment

Wires

- Send a wire payment
- Send a drawdown request
- Respond to a drawdown request
- Simulate an inbound wire

2. Customer management

Our customer management tutorials explain how to use our APIs to:

- **Onboard a customer**: Create a basic customer record in our system. Most products and services require a customer record for each of your customers.
- **Add a beneficial owner**: Associate a beneficial owner with a business customer record or use the beneficial owner field for BIN sponsorship.
- **Add ID details**: Add the details of an identification document such as a driver's license or passport to a customer record.

2.1. Onboard a customer

Before you use banking services offered at Cross River, you need to create a customer record for each of your customers. The customer onboarding process includes creating a customer record and adding relevant customer information.

All customers are automatically scanned for regulatory compliance purposes, so you need to register for relevant webhooks events to receive customer record status updates.

In this tutorial, you'll learn how to:

✅ Register the relevant webhooks

✅ Create a new customer

✅ Add relevant customer details, including customer address and phone number

✅ Retrieve information about a customer

> If you are new to customer management, we recommend you read our **customer management** documentation.
>
> The tutorial assumes you have a knowledge of APIs and how they work. Refer to **API basics** for more details.

The tutorial uses these API endpoints:

API	Description
POST /core/v1/cm/customers	Create a new customer record
GET /core/v1/cm/customers/{customerId}	Retrieve customer information
POST /core/v1/cm/customers/{customerId}/addresses	Add or edit the customer address
POST /core/v1/cm/customers/{customerId}/phones	Add or edit the customer phone number

The tutorial uses these webhooks:

Webhook	Description
`Core.Customer.Onboarded`	Notifies you that the customer record has been created
`Core.Customer.Ofac.Changed`	Notifies you that a customer's OFAC status has been updated
`Core.Customer.PepScan.Changed`	Notifies you that a customer's PEP status has been updated

Before you begin

Make sure you have:

- **API credentials**
- Partner ID

Register the relevant webhook events

To receive the webhook events for this tutorial you need to **register** each specific webhook event type. Once you are registered, the event objects are sent to the registered URLs.

The event object contains a list of resource identifiers used to download details on each event.

After the request to onboard a customer is submitted, you need to stay informed of the result of the compliance scan and that the customer record is created.

Register the customer

The customer record contains customer information. The customer record supports both classifications of types *Personal* or *Business*. Once registered, a customer can be associated with one or more accounts.

In this tutorial, we'll onboard Peter Griffin. First, let's create a customer record. We are registering Peter as a *Personal* customer.

> To onboard a business customer, first create a personal customer record for the *primary owner* of that business.

1. Call `POST /core/v1/cm/customers`. For this call these attributes are required:

partnerId	Your unique partner ID
`classification`	The customer classification. Either: • Personal • Business
`name`	The object containing the customer name details
`profile`	The customer banking profile. This includes: • reg0 • Politically exposed person (PEP) • Customer risk level • Tax ID and Tax ID type

> **IMPORTANT**
> We highly recommend you include an **idempotency key** in your request header to provide duplicate protection in the event of a failure.

Curl

```
POST /core/v1/cm/customers
{
  "partnerId": "6e80b097-693c-4592-8440-02f345335bbf",
  "name": {
    "firstName": "Peter",
    "lastName": "Griffin"
  },
  "classification": "Personal",
  "profile": {
    "regO": false,
    "politicallyExposedPerson": false,
    "taxIdType": "Ssn",
    "taxId": "119988776",
    "birthDate": "1953-09-22",
    "riskRating": "Low"
  }
}
```

The customer record is created.

The unique customer ID is the first line of the response body.

Onboarding a customer response body

```json
{
  "id": "9052b6a5-3f09-41d1-b526-ade80104eb79",
  "cifNumber": "32653745014",
  "classification": "Personal",
  "status": "Active",
  "ofac": "Pending",
  "pepScan": "Pending",
  "name": {
    "firstName": "Peter",
    "lastName": "Griffin",
    "fullName": "Peter Griffin"
  },
  "profile": {
    "regO": false,
    "politicallyExposedPerson": false,
    "enableBackupWithholding": false,
    "taxIdType": "Ssn",
    "taxId": "119988776",
    "birthDate": "1953-09-22",
    "riskRating": "Low"
  },
  "createdAt": "2021-01-25T17:55:24.4422582-05:00",
  "lastModifiedAt": "2021-01-25T17:55:24.4432543-05:00",
  "partnerId": "6e80b097-693c-4592-8440-02f345335bbf"
}
```

2 The `Core.Customer.Onboarded` webhook event is triggered when the customer record is created.

Customer onboarded event

```
{
  "id": "3d9c5e1a-623b-4cf2-812e-ade80105048e",
  "eventName": "Core.Customer.Onboarded",
  "status": "Pending",
  "partnerId": "30dee145-b6a2-4058-8dc3-ac4000dee91f",
  "createdAt": "2021-11-22T10:50:20.553-05:00",
  "resources": [
    "core/v1/cm/customers/9052b6a5-3f09-41d1-b526-ade80104eb79"
  ],
  "details": []
}
```

Cross River scans a new customer record for OFAC and PEP compliance.

> Sometimes the results of the scan require a review by the our Anti-Money Laundering (AML) team.

On scan completion, `Core.Customer.Ofac.Changed` and `Core.Customer.PepScan.Changed` webhook events return with a status. Customers with no compliance issues show a `Clear` status.

Customer OFAC scan changed

```
{
  "id": "45db5592-56ee-41f5-85e7-ade8010571ce",
  "eventName": "Core.Customer.Ofac.Changed",
  "status": "Pending",
  "partnerId": "30dee145-b6a2-4058-8dc3-ac4000dee91f",
  "createdAt": "2021-11-22T10:51:53.657-05:00",
  "resources": [
    "core/v1/cm/customers/9052b6a5-3f09-41d1-b526-ade80104eb79"
  ],
  "details": []
}
```

Customer PEP scan changed

```
{
  "id": "a5000831-3e01-4231-9ec2-ade8010571c5",
  "eventName": "Core.Customer.PepScan.Changed",
  "status": "Pending",
  "partnerId": "30dee145-b6a2-4058-8dc3-ac4000dee91f",
  "createdAt": "2021-11-22T10:51:53.597-05:00",
  "resources": [
    "core/v1/cm/customers/9052b6a5-3f09-41d1-b526-ade80104eb79"
  ],
  "details": []
}
```

3 To retrieve the results of the OFAC and PEP scans, call `GET core/v1/cm/customers/{id}`, where `id` is the customer ID in the `resources` attribute of both events.

In this example, the customer ID is **9052b6a5-3f09-41d1-b526-ade80104eb79**

Get customer details

```
{
  "id": "9052b6a5-3f09-41d1-b526-ade80104eb79",
  "cifNumber": "32653745014",
  "classification": "Personal",
  "status": "Active",
  "ofac": "Clear",
  "pepScan": "Clear",
  "name": {
    "firstName": "Peter",
    "lastName": "Griffin",
    "fullName": "Peter Griffin"
  },
  "profile": {
    "regO": false,
    "citizenshipCountryCode": "US",
    "politicallyExposedPerson": false,
    "enableBackupWithholding": false,
    "taxIdType": "Ssn",
    "taxId": "119988776",
    "birthDate": "1953-09-22",
    "riskRating": "Low"
  },
  "createdAt": "2021-11-22T10:49:58.843-05:00",
  "lastModifiedAt": "2021-11-22T10:51:49.5299112-05:00",
  "partnerId": "30dee145-b6a2-4058-8dc3-ac4000dee91f",
  "dueDiligence": {
    "annualIncome": 0
  }
}
```

Add details to the customer record

Next, you need to add the customer address and phone number. You can also add the customer email and identification information. Once you add this initial information, if needed you add more information, such as a mailing address. This tutorial will cover adding the address and phone number only. These details are considered *primary* information. If you add a second address, this is *secondary*.

1. Use the customer ID returned when you create the customer record to call `POST /core/v1/cm/customers/{id}/addresses` to add the customer address. The first address you add is the *primary* address.

   Add customer address

   ```
   POST /core/v1/cm/customers/9052b6a5-3f09-41d1-b526-ade80104eb79/addresse
   {
     "addressType": "Home",
     "classification": "Residential",
     "isPrimary": true,
     "street1": "123 Any St",
     "city": "Anywhere",
     "state": "NY",
     "postalCode": "12345",
     "countryCode": "US"
   }
   ```

2. Call `POST /core/v1/cm/customers/{id}/phones` to add the customer phone number. The first phone number you add is the *primary* phone number.

   Add customer phone number

   ```
    POST /core/v1/cm/customers/9052b6a5-3f09-41d1-b526-ade80104eb79/phones
   {
     "isPrimary": true,
     "phoneType": "Mobile",
     "phoneNumber": "2015552345"
   }
   ```

The onboarding process is considered complete when the address and phone records have been added.

2.2. Add a beneficial owner

A beneficial owner is a person who owns 25% or more of a business. For regulatory reasons, you need to create a Beneficial Owner resource in a Business customer record for each beneficial owner of that business. Each beneficial owner has a *personal* customer record that includes a unique customer ID. The `ownerCustomerId` attribute in the call described here references the *personal* customer ID for each beneficial owner. You usually add beneficial owner resources to the business customer record immediately after onboarding a business.

> If you are a Cross River partner using our **partner-managed issuing** to provide commercial card services (and not to individual cardholders) use this endpoint to associate individual cardholder customer records with their employer business customer record.

In this tutorial, you'll learn how to:

✅ Update a COS customer record with a beneficial owner resource.

> If you are new to customer management, we recommend you read our **customer onboarding** documentation.
>
> The tutorial assumes you have a knowledge of APIs and how they work. Refer to **API basics** for more details.

The tutorial uses this API endpoint:

API	Description
POST /core/v1/cm/customers/{customerId}/beneficial-owners	Adds a beneficial owner resource to customer details.

Before you begin

Make sure you have:

- **API credentials**
- Customer ID of the business customer record to be updated (only available once the business is **onboarded**).
- Customer ID of the beneficial owner personal customer record (only available once the beneficial owner is **onboarded**) or the customer ID of the business customer record to associate an individual cardholder customer record with their employer customer record.
- Beneficial owner title as it appears in their customer record

Add beneficial owner resource to record

When you onboard a business customer, you'll need to add any beneficial owners as resources for the business. Add the beneficial owner to the existing customer record.

If you're using {{BIN}} to provide credit card services for businesses, you need to add each cardholder as a beneficial owner resource to the business customer record. In that case, the `ownerTitle` for each cardholder is **Auth User**.

In this tutorial, you'll add the CEO of a business as a beneficial owner resource to an existing business customer record.

To add identification information

1. Call `POST /core/v1/cm/customers/{customerId}/beneficial-owners`. The attributes below are required.

Detail	Request attribute name	Value used in the sample request (not valid)
CR customer ID for the business customer record being updated (provided when the customer was onboarded to COS)	`customerId`	**e2a76a89-b5ff-46f1-bf4d-494579d7bb71**
Beneficial owner CR customer ID (provided when the beneficial owner was onboarded to COS)	`ownerCustomerId`	**83454c7a-e5d3-4f3a-8633-11dd49389de2**
Title or role of the beneficial owner	`ownerTitle`	**CEO**

IMPORTANT
We highly recommend you include an **idempotency key** in your request header to provide duplicate protection in the event of a failure.

Curl

```
curl -X POST
--header 'Content-Type: application/json'
--header 'Accept: application/json'
--header 'Authorization: Bearer <token>'
-d '{ \
  "ownerCustomerId": "83454c7a-e5d3-4f3a-8633-11dd49389de2",
  "ownerTitle": "CEO",
}' 'https://sandbox.crbcos.com/Core/v1/cm/customers/e2a76a89-b5ff-46f1-b
```

2 A successful API call returns a JSON response confirming the beneficial owner added in COS. The `id` attribute indicates the COS ID of the beneficial owner record. In the sample below, the ID is **6d441e6d-cd94-4be1-bd30-3890fcad3100**. Yours will be different.

3

Response sample

```
{
  "id": "6d441e6d-cd94-4be1-bd30-3890fcad3100",
  "customerId": "e2a76a89-b5ff-46f1-bf4d-494579d7bb71",
  "ownerCustomerId": "83454c7a-e5d3-4f3a-8633-11dd49389de2",
  "status": "Active",
  "ownerTitle": "CEO",
  "partnerId": "0497b3ba-ab2f-4537-8257-4af39d7a5cf7",
  "createdAt": "2024-01-07T08:13:08.464Z",
  "lastModifiedAt": "2024-01-07T08:13:08.464Z",
}
```

4

2.3. Add ID details

Update a customer record with physical ID metadata or other identification information.

> If you are new to customer management, we recommend you read our **customer onboarding** documentation.
>
> The tutorial assumes you have a knowledge of APIs and how they work. Refer to **API basics** for more details.

The tutorial uses this API endpoint:

API	Description
POST /core/v1/cm/customers/{customerId}/identifications	Adds unique identification metadata of customer-identifying documents to customer details.

Before you begin

Make sure you have:

- **API credentials**
- Customer ID (only available once the customer is **onboarded**)
- The necessary details for the identification you are adding to the customer record. Depending on the type of identification, you might need:
    - ID serial number
    - Expiration date
    - Details of the issuing authority
    - Other information

Register relevant webhook events

To receive the webhook events for this tutorial both partner accounts need to **register** each specific webhook event type. Once you are registered, the event objects are sent to the registered URLs.

The event object contains a list of resource identifiers used to download details on each event.

Add ID information to existing record

You might want or need to add a drivers license, passport or other identification to an existing customer record. If your agreement with Cross River includes {{BIN}}, you need to add the card processor unique identifier for that customer to their COS record.

In this tutorial, you'll add a driver's license to Peter Griffin's customer record.

1. Call `POST /core/v1/cm/customers/{customerId}/identifications`. The attributes below are required. A full list of attributes is found here.

Detail	Request attribute name	Value to use in the request
Cross River customer ID (provided when the customer was onboarded to COS)	`customerId`	ID from the **onboard a customer** tutorial
True if the ID is the primary ID for this customer. Otherwise false.	`isPrimary`	**true**
License ID number	`idNumber`	**123 456 789**
ID type	`idType`	**DriversLicense**
License Issue date	`issuedDate`	**2022-07-15**
License Expiration date	`expDate`	**2029-07-14**
License Issuing authority	`issuingAuthority`	**New York DMV**
License Issuing country code	`issuingCountryCode`	**US**

IMPORTANT

We highly recommend you include an **idempotency key** in your request header to provide duplicate protection in the event of a failure.

Sample request

```
curl -X POST
--header 'Content-Type: application/json'
--header 'Accept: application/json'
--header 'Authorization: Bearer <token>'
-d '{ \
  "isPrimary": true,
  "idNumber": "123 456 789",
  "idType": "DriversLicense",
  "issuedDate": "2022-07-15",
  "expDate": "2029-07-14",
  "verifiedDate": "2023-05-21",
  "issuingAuthority": "NY DMV",
  "issuingStateOrProvince": "NY",
  "issuingCountryCode": "US",
}' 'https://sandbox.crbcos.com/Core/v1/cm/customers/b71cf966-7896-40a0-8
```

2 A successful API call returns a JSON response confirming the identification details added in COS. The `id` attribute indicates the COS ID of that specific identification item. In the sample below, the ID is **6d3add34-0a1e-4a4e-9c7f-b00a0091e308**. Yours will be different.

3

Response sample

```
{
  "id": "6d3add34-0a1e-4a4e-9c7f-b00a0091e308",
  "customerId": "b71cf966-7896-40a0-88c5-af5g0138fc8c",
  "isPrimary": true,
  "idNumber": "123 456 789",
  "status": "Active",
  "idType": "DriversLicense",
  "issuedDate": "2022-07-15",
  "expDate": "2029-07-14",
  "issuingAuthority": "NY DMV",
  "issuingStateOrProvince": "NY",
  "issuingCountryCode": "US",
  "createdAt": "2023-05-22T04:51:09.4656083-04:00",
  "lastModifiedAt": "2023-05-22T04:51:09.4656083-04:00",
  "partnerId": "cd9c12f4-7691-424a-b38b-af5b0134c611"
}
```

IMPORTANT

It's up to you to keep this information up to date.

3. Accounts

Our accounts tutorials explain how to use our APIs to:

- **Open an account**: Create a deposit account.
- **Open a subledger**: Create a subledger (virtual account).
- **Withdraw CD funds early**: Withdraw funds from a certificate of deposit before it reaches maturity.

3.1. Open an account

CR provides a number of different types of accounts, including check and savings, Certificates of Deposit (CDs or time deposit accounts), and more.

To open any kind of account you must have a valid product ID for the type of account you want to open, and a customer ID (onboarded customer record ID) for the account holder. Note that the account holder must have at least one address and phone number in their customer record, and their OFAC status must be *Clear*. In addition, the classification of the customer must match the configured classification for the product. For example, only business customers can be added to a business product.

In this tutorial, you'll learn how to:

✅ Open a customer master account

> If you are new to account management we recommend you read the **accounts** documentation before starting this tutorial.
>
> The tutorial assumes you have a knowledge of APIs and how they work. Refer to **API basics** for more details.

The tutorial uses this API endpoint:

API	Description
POST /core/v1/dda/accounts	Opens a deposit account for a customer

The tutorial uses these webhooks:

Webhook	Description
`Core.Account.Opened`	A new account was opened

Before you begin

Make sure you have:

- **API credentials**
- Partner ID
- Customer ID (only available once the customer is **onboarded**)
- **Product ID** (defines the type of account being opened)

> **IMPORTANT**
> We highly recommend you include an idempotency key in your request header to provide duplicate protection in the event of a failure. Read more about **idempotency keys**.

Register the relevant webhook events

To receive the webhook events for this tutorial both partner accounts need to **register** each specific webhook event type. Once you are registered, the event objects are sent to the registered URLs.

The event object contains a list of resource identifiers used to download details on each event.

Open an account

This tutorial shows you how to create a deposit account. In this scenario, you'll open an account for Jana Parker, a customer who you successfully onboarded.

> Closed accounts can only be re-opened by the CR Ops teams. To re-open an account please open a **support ticket.**

To open an account

1. Call `POST /v1/dda/accounts`. For this call, some **attributes** are required.

POST /core/v1/dda/accounts request

```json
{
  "customerId": "59e3bc15-bbec-4990-88e9-a9a600d3296c",
  "productId": "44015e68-1afb-40fc-9497-abc1014f52da",
  "title": "Jana Parker",
  "statementAddress": {
    "street1": "257 Dalton Groves",
    "city": "Barton City",
    "state": "MI",
    "postalCode": "48705",
    "countryCode": "US"
  }
}
```

2 A successful API call returns a JSON response with the details of the new account.

POST /core/v1/dda/accounts response

```json
{
  "customerId": "59e3bc15-bbec-4990-88e9-a9a600d3296c",
  "productId": "44015e68-1afb-40fc-9497-abc1014f52da",
  "title": "Jana Parker",
  "statementAddress": {
    "street1": "257 Dalton Groves",
    "city": "Barton City",
    "state": "MI",
    "postalCode": "48705",
    "countryCode": "US"
  }
}
```

3 The `accountNumber` field provides the account number for the new account.

4 In the response example, the account is classified as `Personal` because the user configured the product classification as **Personal**. The account classification always matches the configured product classification.

5 The account status is automatically updated to `Active`, and is immediately available for use. This triggers the `Core.Account.Opened` event.

Core.Account.Opened Event Details

```
{
  "id": "259cdbca-6a89-4af8-a50e-ada3010fb13f",
  "eventName": "Core.Account.Opened",
  "status": "Pending",
  "partnerId": "e6c3824a-377f-44d5-a2f6-a9a600c9b37e",
  "createdAt": "2021-01-26T09:48:10.1011462-05:00",
  "resources": [
    "core/v1/dda/accounts/2235223803"
  ],
  "details": []
}
```

3.2. Open a subledger

Subledgers can be **opened** under any master account, such as a checking or savings account.

The only information required to open a subledger is the master account number and Title field. The Title is typically set to the name of the partner. Subledgers also support a beneficiary profile for storing additional data such as the name and address associated with the subledger. Your Integration manager will assist you with any questions related to the data used to populate these fields, as they can vary by use case and program.

Once opened, the subledger is ready within a few seconds.

Curl

```
POST /core/v1/dda/subaccounts
{
  "masterAccountNumber": "2001231234",
  "title": "Acme Co",
  "beneficiary": {
    "referenceId": "ABC789",
    "entityName": "Acme Co",
    "streetAddress1": "400 Business Street",
    "streetAddress2": "Suite 123",
    "city": "New York",
    "state": "NY",
    "postalCode": "10025",
    "countryCode": "US",
    "phoneNumber": "2015551234",
    "emailAddress": "acme@test.com",
    "notes": "Testing 123"
  }
}
```

3.3. Withdraw CD funds early

In this tutorial, you'll learn how to:

✅ Request early withdrawal

✅ Determine how much to withdraw from the account after the penalty fee is deducted

> This tutorial assumes you have a knowledge of APIs and how they work. For more information on sending API calls, see **API basics**.
>
> Learn more about **CDs**.

The tutorial uses these endpoints:

API	Description
POST /core/v1/dda/accounts/{accountNumber}/time-deposit/early-withdraw	This endpoint requests early withdrawal from a time deposit account
GET /core/v1/accounts/{accountNumber}	This endpoint queries a specific account for information about it

The tutorial uses these webhooks:

Webhook	Description
`Core.TimeDeposit.Withdrawn`	Shows information about the account the early withdrawal request was made to

Before you begin

Make sure you have:

- **API credentials**

- Account number

1 Register relevant webhook events

To receive the webhook events for this tutorial both partner accounts need to **register** each specific webhook event type. Once you are registered, the event objects are sent to the registered URLs.

The event object contains a list of resource identifiers used to download details on each event.

2 Request time deposit early withdrawal

Before an account holder can actually withdraw funds from a time account that hasn't reached maturity, you have to make a request (for the account holder) to make that early withdrawal. In the request you also indicate if the withdrawal will trigger a penalty fee or not. Cross River charges a penalty fee for early withdrawal, although in some cases it can be waived. For example, Cross River time deposit accounts have no provisions for partial withdrawals. However, in certain circumstances, an account holder can be allowed to make an early withdrawal of the money in the account without penalty, which is similar to a partial withdrawal.

Note that a request for early withdrawal does not remove funds from the account. That is a separate call.

In this tutorial, you will request an early withdrawal with a penalty fee.

To request early withdrawal

Call `POST /v1/dda/accounts/{accountNumber}/time-deposit/early-withdraw`. For this call, some/all **attributes** are required.
Set the `waivePenaltyFee` attribute to *false.*

Curl

```
): https://sandbox.crbcos.com/core/v1/dda/accounts/2434508988/time-depos
{
  "waivePenaltyFee": false
}
```

A successful API call returns a JSON response with the details of account, including the total funds to be withdrawn and the amount of penalty fee charged. In our example, the total deposit amount is **1,000,000.00**. The penalty fee amount is **4,109.59**.

JSON

```json
{
  "timeDeposit": {
    "masterAccountNumber": "2434508988",
    "status": "EarlyWithdrawal",
    "minFundingAmount": 50000000,
    "maxFundingAmount": 1000000000,
    "autoClose": true,
    "allowBumpUp": true,
    "waivePenaltyFee": false,
    "fundingDays": 2,
    "gracePeriodDays": 2,
    "currentRate": 0,
    "maturityMonths": 12,
    "rates": [
      {
        "months": 6,
        "rate": 5
      },
      {
        "months": 6,
        "rate": 2.5
      }
    ],
    "penalties": [
      {
        "months": 3,
        "feeDays": 30
      },
      {
        "months": 9,
        "feeDays": 90
      }
    ],
    "fundingExpDate": "2023-01-17T00:00:00-05:00",
    "fundingDate": "2023-01-12T00:00:00-05:00",
    "startDate": "2023-01-12T00:00:00-05:00",
    "depositAmount": 100000000,
    "depositCurrency": "usd",
    "maturityDate": "2024-01-11T00:00:00-05:00",
    "earlyWithdrawnAt": "2023-03-08T09:57:00.005123-05:00",
    "rolloverDate": "2024-01-16T00:00:00-05:00",
    "createdAt": "2023-01-12T16:45:45.96-05:00",
    "productId": "3c4e1c34-3544-4236-93a5-af110157167c",
    "partnerId": "1e5d3f04-ae24-4af6-9e30-aecf012b99dd",
    "lastModifiedAt": "2023-03-08T09:57:00.0208451-05:00"
    },
    "penaltyAccountNumber": "20020020028",
    "penaltyFeeAmount": 410959
  }
```

To see amount available

1. Call `GET /core/v1/accounts/{accountNumber}` to see the balance that remains in the account after deducting the penalty. This is the sum you will withdraw. You must transfer the entire amount. Transfer the funds as you would any money transfer.
2. Withdrawal of the funds triggers the `Core.TimeDeposit.Withdrawn` webhook event.

Core.TimeDeposit.Withdrawn webhook event

```
  {
    "id": "90cef2a8-f3e1-41ff-9394-afbf00f8a406",
    "eventName": "Core.TimeDeposit.Withdrawn",
    "status": "Pending",
    "partnerId": "1e5d3f04-ae24-4af6-9e30-aecf012b99dd",
    "createdAt": "2023-03-08T10:05:16.423-05:00",
    "resources": [
      "core/v1/dda/accounts/2434508988/time-deposit"
    ],
    "details": [
      {
        "masterAccountNumber": "2434508988",
        "status": "EarlyWithdrawal",
        "depositAmount": "100023362",
        "maturityDate": "3/14/2023",
        "rolloverDate": "3/19/2023",
        "currentRate": "0.0"
      }
    ]
  }
```

4. Cards

Our cards tutorials explain how to use our APIs to:

- **Create a card**: Create and order a new card.
- **Activate a card**: Change the card status to Active.
- **Simulate card management**: Test the system with our card simulation endpoints.

4.1. Create a card

In this tutorial, you'll learn how to:

✅ Create and order a new card

> If you are new to card issuing we recommend you read the **Cards** concept pages before starting this tutorial.
>
> The tutorial assumes you have a knowledge of APIs and how they work. For more information on sending API calls, see **API basics**.

The tutorial uses this API endpoint:

API	Description
POST /cardmanagement/v1/cards	Requests creation of a new debit card

The tutorial uses these webhooks.

Webhook	Description
`Cards.Card.Created`	Debit card created

Before you begin

Make sure you have:

- **API credentials**
- Cardholder's account number
- Customer ID (you get this when you create a customer)
- Configuration ID

Register relevant webhook events

To receive the webhook events for this tutorial both partner accounts need to **register** each specific webhook event type. Once you are registered, the event objects are sent to the registered URLs.

The event object contains a list of resource identifiers used to download details on each event.

Order a new card

In this tutorial you will learn how to order/create a new debit card by creating a card for John Smith.

1 Call `POST /cardmanagement/v1/cards`. For this call, some **attributes** are required.

Sample create a card

```
curl --location 'https://sandbox.crbcos.com/cardmanagement/v1/cards' \
--header 'Idempotency-key: b06ab5cc-553a-4fe4-ad78-751b5ff81f0e' \
--data-raw '{
    "accountNumber": "158560897007",
    "customerId": "e2599a17-d1e2-476c-9672-b2ff008fa575",
    "configurationId": "b2251a28-c218-4e2e-a787-b2f700ef0ecd",
    "firstName": "Daly",
    "lastName": "Khol",
    "phone": {
        "phoneType": "Home",
        "phoneNumber": "7185551234"
    },
    "emailAddress": "dkhol@gmail.com",
    "shippingAddress": {
        "street1": "1 Roshar Ave",
        "city": "Roshar",
        "state": "NY",
        "postalCode": "10001",
        "countryCode": "US"
    },
    "billingAddress": {
        "street1": "1 Roshar Ave",
        "city": "Roshar",
        "state": "NY",
        "postalCode": "10001",
        "countryCode": "US"
    },
    "nameOnCard": "Daly Khol",
    "shippingType": "Normal",
   "clientIdentifier": "b06ab5cc-553a-4fe4-ad78-751b5ff81f00"
}
```

2 A successful API call returns a JSON response with the details of the new card. The card status will be unactivated until you activate the card. The `id` is the card ID, in this example **8c6a53e0-83ad-4b76-b446-b300006a3e6a** which you need to **activate the card**.

Sample order a card response

```
{
    "id": "8c6a53e0-83ad-4b76-b446-b300006a3e6a",
    "productId": "57146944-b145-4326-884d-b2f700ecf688",
    "partnerId": "19222b81-0e1e-452d-a842-b2f1011c16f3",
    "accountNumber": "158560897007",
    "status": "Unactivated",
    "statusReasonCode": "NotSet",
    "processorCardStatus": "I",
    "firstName": "Daly",
    "lastName": "Khol",
    "shippingAddress": {
        "street1": "1 Roshar Ave",
        "city": "Roshar",
        "state": "NY",
        "postalCode": "10001",
        "countryCode": "US"
    },
    "billingAddress": {
        "street1": "1 Roshar Ave",
        "city": "Roshar",
        "state": "NY",
        "postalCode": "10001",
        "countryCode": "US"
    },
    "phone": {
        "phoneType": "Home",
        "phoneNumber": "7185551234"
    },
    "emailAddress": "dkhol@gmail.com",
    "nameOnCard": "Daly Khol",
    "isPinSet": false,
    "adminBlocked": false,
    "fraudSuspect": false,
    "configurationId": "b2251a28-c218-4e2e-a787-b2f700ef0ecd",
    "category": "Debit",
    "paymentInstrument": "PhysicalCombo",
    "processor": "i2c",
    "shippingType": "Normal",
    "orderStatus": "OrderPending",
    "replacementStatus": "NotApplicable",
    "customerId": "e2599a17-d1e2-476c-9672-b2ff008fa575",
    "clientIdentifier": "b06ab5cc-553a-4fe4-ad78-751b5ff81f00",
    "createdAt": "2025-06-18T02:26:49.6169047-04:00",
```

3 When the card is generated at the processor, the `Cards.Card.created` webhook event fires.

Sample Cards.Card.Created event

```
{
  "id": "cea6c5b5-13e7-4c2f-ba75-afce01110fe5",
  "eventName": "Cards.Card.Created",
  "status": "Pending",
  "partnerId": "cd9c12f4-7691-424a-b38b-af5b0134c611",
  "createdAt": "2023-02-13T04:42:29.4869172-05:00",
  "resources": [
    "cardmanagement/v1/cards/8c6a53e0-83ad-4b76-b446-b300006a3e6a"
  ],
  "details": [
    {
      "cardId": "8c6a53e0-83ad-4b76-b446-b300006a3e6a",
      "status": "Unactivated",
      "statusReasonCode": "NotSet"
    }
  ]
}
```

4.2. Activate a card

In this tutorial, you'll learn how to:

✅ Activate a debit card using the card ID

> If you are new to card issuing we recommend you read the **Cards** concept pages before starting this tutorial.
>
> The tutorial assumes you have a knowledge of APIs and how they work. For more information on sending API calls, see **API basics**.

The tutorial uses this API endpoint:

API	Description
POST /cardmanagement/v1/cards/{id}/activate	Activates the card using the card ID

The tutorial uses this webhook event:

Webhook	Description
`Cards.Card.Activated`	Debit card activated

Before you begin

Make sure you have:

- **API credentials**
- Card ID from when you **created the card**

Register relevant webhook events

To receive the webhook events for this tutorial both partner accounts need to **register** each specific webhook event type. Once you are registered, the event objects are sent to the registered URLs.

The event object contains a list of resource identifiers used to download details on each event.

Activate the card

Activate the card you created in the **create a card** tutorial.

1. Call `POST /cardmanagement/v1/cards/{id}/activate`. Set the `id` attribute to the card ID. In the sample below the ID is **8c6a53e0-83ad-4b76-b446-b300006a3e6a**.

   Sample activate card request

   ```
   curl --location --globoff --request POST 'https://sandbox.crbcos.com/car
   --header 'Idempotency-key: a5b1b129-421e-441b-9661-ec1c0396965e' \
   --data ''
   ```

2. A successful API call returns a JSON response with the details of the card. The card `status` is now **Active**. The `OrderStatus` is **Completed**.

Sample activate card response

```
{
    "id": "8c6a53e0-83ad-4b76-b446-b300006a3e6a",
    "productId": "57146944-b145-4326-884d-b2f700ecf688",
    "partnerId": "19222b81-0e1e-452d-a842-b2f1011c16f3",
    "processorCardId": "827277613081398",
    "accountNumber": "158560897007",
    "status": "Active",
    "statusReasonCode": "NotSet",
    "processorCardStatus": "B",
    "firstName": "Daly",
    "lastName": "Khol",
    "shippingAddress": {
        "street1": "1 Roshar Ave",
        "city": "Roshar",
        "state": "NY",
        "postalCode": "10001",
        "countryCode": "US"
    },
    "billingAddress": {
        "street1": "1 Roshar Ave",
        "city": "Roshar",
        "state": "NY",
        "postalCode": "10001",
        "countryCode": "US"
    },
    "phone": {
        "phoneType": "Home",
        "phoneNumber": "7185551234"
    },
    "emailAddress": "dkhol@gmail.com",
    "nameOnCard": "Daly Khol",
    "panLastFour": "4558",
    "isPinSet": false,
    "expirationDate": "2028-06-18",
    "adminBlocked": false,
    "fraudSuspect": false,
    "configurationId": "b2251a28-c218-4e2e-a787-b2f700ef0ecd",
    "category": "Debit",
    "paymentInstrument": "PhysicalCombo",
    "processor": "i2c",
    "shippingType": "Normal",
    "orderStatus": "Completed",
    "replacementStatus": "NotApplicable",
    "customerId": "e2599a17-d1e2-476c-9672-b2ff008fa575",
    "clientIdentifier": "b06ab5cc-553a-4fe4-ad78-751b5ff81f00",
    "processorCustomerId": "0CIF7L2973U98WT4E652",
    "createdAt": "2025-06-18T02:26:49.617-04:00",
    "initialActivation": "2025-06-18T00:00:00-04:00",
    "activatedAt": "2025-06-18T02:27:58.8133927-04:00",
    "lastModifiedAt": "2025-06-18T02:27:58.8133927-04:00"
}
```

3 When the card status changes to `Active`, the `Cards.Card.Activated` webhook event fires.

Cards.Card.Activated webhook event

```
{
  "id": "95ac33ca-3d7f-49aa-bfc2-afce01113310",
  "eventName": "Cards.Card.Activated",
  "status": "Pending",
  "partnerId": "19222b81-0e1e-452d-a842-b2f1011c16f3",
  "createdAt": "2023-02-13T04:44:31.282287-05:00",
  "resources": [
    "cardmanagement/v1/cards/8c6a53e0-83ad-4b76-b446-b300006a3e6a"
  ],
  "details": [
    {
      "cardId": "8c6a53e0-83ad-4b76-b446-b300006a3e6a",
      "status": "Active",
      "statusReasonCode": "NotSet"
    }
  ]
}
```

4.3. Simulate card management

In this tutorial, you'll learn how to:

✅ Simulate these card management actions:

- Clearing
- Reversal
- Incremental transaction
- Single message clearing
- Single message reversal

> If you are new to card issuing we recommend you read the **Cards** concept pages before starting this tutorial.
>
> The tutorial assumes you have a knowledge of APIs and how they work. For more information on sending API calls, see **API basics**.

The tutorial uses these API endpoints:

API	Description
POST /v1/simulate/authorization	Simulates merchant authorization
POST /v1/simulate/clearing	Simulates replacement of a memo post with a core transaction
POST /v1/simulate/reversal	Simulates reversal of an authorization
POST /v1/simulate/incremental	Simulates an incremental transaction reflected in the deposit account activity
POST /v1/simulate/single-message-clearing	Simulations the message type for cardholder verification to authorize and clear a transaction
POST /v1/simulate/single-message-reversal	Simulates a single message clearing transaction for which you can create a reversal transaction

Before you begin

Make sure you have:

- **API credentials**
- Client ID

Register relevant webhook events

To receive the webhook events for this tutorial both partner accounts need to **register** each specific webhook event type. Once you are registered, the event objects are sent to the registered URLs.

The event object contains a list of resource identifiers used to download details on each event.

Simulate card management

The `simulate` endpoints within Card Management allow you to simulate most of the transaction scenarios you would experience in a live environment.

To simulate authorization

Call **POST /v1/simulate/authorization**. This endpoint simulates a merchant authorization that affects the available balance on the deposit account.

Simulate authorization request

```
curl --location 'https://sandbox.crbcos.com/cardmanagement/v1/simulate/author
--data '{
  "amount": 1999,
  "cardId": "8c6a53e0-83ad-4b76-b446-b300006a3e6a",
  "merchantCountryCode": "US",
  "merchantName": "StreamFlix",
  "merchantStreet": "123 Any St",
  "merchantCity": "Anywhere",
  "merchantState": "NY",
  "merchantPostalCode": "10001",
  "cardNetwork": "Visa",
  "additionalAmounts": [],
  "processingCode": "BillPayment"
}'
```

The response includes the `retrievalReferenceNumber` and the `cardId`. You need these values to replace the memo post with a core transaction to post the activity record to the account, using the endpoint `POST /v1/simulate/clearing` as explained below.

Simulate authorization response

```
{
    "retrievalReferenceNumber": "644371423186",
    "cardId": "8c6a53e0-83ad-4b76-b446-b300006a3e6a"
}
```

To simulate clearing

In a real transaction, when the authorization request is made, a memo post reflects the account and impacts the available balance. However, the memo post remains until the `clearing` endpoint is called. Calling the **POST /v1/simulate/clearing** endpoint replaces the memo post with a core transaction to post the activity record to the account. In this simulation, authorization is not complete and the transaction does not post on the account until the `simulate/clearing` endpoint is called using the `retrievalReferenceNumber` and `cardId`.

Simulate clearing request

```
curl --location 'https://sandbox.crbcos.com/cardmanagement/v1/simulate/cleari
--data '{
  "cardId": "8c6a53e0-83ad-4b76-b446-b300006a3e6a",
  "retrievalReferenceNumber": "644371423186",
  "processingCode": "BillPayment"
}'
```

To simulate reversal

This endpoint is used to reverse an existing authorization using the `retrievalReferenceNumber` returned in the response to the **POST /v1/simulate/reversal** endpoint. This reverses the original authorization placed on the account and removes it from the deposit account activity. The amount in this request can be any value and does not have to be the same amount as the original clearing transaction, which allows for a partial amount reversal.

Simulate reversal request

```
curl --location 'https://sandbox.crbcos.com/cardmanagement/v1/simulate/revers
--data '{
  "cardId": "8c6a53e0-83ad-4b76-b446-b300006a3e6a",
  "retrievalReferenceNumber": "654492282074",
  "amount": "123"
}'
```

To simulate a incremental transaction

This endpoint, used with the `retrievalReferenceNumber` returned by the **POST /v1/simulate/incremental** endpoint, creates an incremental transaction reflected in the deposit account activity. An example of an incremental transaction is when a hotel adds an additional charge for an item charged to your room.

Simulate incremental request

```
curl --location 'https://sandbox.crbcos.com/cardmanagement/v1/simulate/increm
--data '{
  "cardId": "8c6a53e0-83ad-4b76-b446-b300006a3e6a",
  "retrievalReferenceNumber": "654492282074",
  "amount": "123"
}'
```

To simulate single message clearing

A *single message* is a message type that uses real-time verification and requires the cardholder to enter their PIN to authorize and clear their transaction. An example of a transaction that uses the **POST /v1/simulate/single-message-clearing** enpoint is an ATM withdrawal.

Simulate single message clearing request

```
curl --location 'https://sandbox.crbcos.com/cardmanagement/v1/simulate/single
--data '{
  "processingCode": "Purchase",
  "cardId": "8c6a53e0-83ad-4b76-b446-b300006a3e6a",
  "amount": "22",
  "merchantCountryCode": "US",
  "merchantName": "StreamFlix",
  "merchantStreet": "123 Any St",
  "merchantCity": "Anywhere",
  "merchantState": "NY",
  "merchantPostalCode": "10001",
  "cardNetwork": "Visa",
  "additionalAmounts": [
    {
      "amountType": "Unknown",
      "accountType": "NotSpecified",
      "amount": "10"
    }
  ]
}

'
```

To simulate single message reversal

This endpoint allows you to create a reversal transaction to a single message clearing transaction. For example, use the **POST /v1/simulate/single-message-reversal** endpoint to simulate a purchase and then simulate a return with the `single-message-reversal`.

Simulate single message reversal request

```
curl --location 'https://sandbox.crbcos.com/cardmanagement/v1/simulate/single
--data '{
  "amount": "22",
  "processingCode": "Purchase",
  "cardId": "8c6a53e0-83ad-4b76-b446-b300006a3e6a",
  "merchantCountryCode": "US",
  "merchantName": "StreamFlix",
  "merchantStreet": "123 Any St",
  "merchantCity": "Anywhere",
  "merchantState": "NY",
  "merchantPostalCode": "10001",
  "cardNetwork": "Visa",
  "additionalAmounts": [
    {
      "amountType": "Unknown",
      "accountType": "NotSpecified",
      "amount": "10"
    }
  ]
}
'
```

5. Instant payments

Our cards tutorials explain how to use our APIs to:

- **Send an instant payment**: Originate a payment using RTP, FedNow, CRNow or network interoperability.
- **Get service info**: Find out the instant payment services a specific financial institution supports, per network (RTP or FedNow).
- **Set payment expiration**: Set your own custom expiration time for retrying a payment when a receiving financial institution is down.
- **Fraud reporting**: Each instant payment network has specific requirements for reporting suspected fraud, with its own criteria for what qualifies as reportable fraud

5.1. Send an instant payment

In this tutorial, you'll learn how to

✅ Originate an instant payment credit transfer from your master account.

✅ Simulate a rejected credit transfer.

> If you are new to instant payments, check out the **instant payments** overview.
>
> This tutorial assumes familiarity with APIs. For more information, visit **API basics**.

Before you begin

Make sure you have:

- **API credentials**
- Your partner ID
- Your master account number

You should be familiar with these terms:

Financial institution (FI)	The bank or credit union facilitating the transfer of funds between parties
Debtor	The account initiating and sending a payment through the Instant Payments network
Creditor	The account receiving funds through the Instant Payments network

This tutorial uses this API endpoint:

POST /rtp/v1/payments	Transfers funds between banks in real time via an Instant Payments network

Webhooks used in this tutorial:

`Rtp.Payment.Sent`	**Payment successfully sent to the receiving institution; funds are available in the recipient's account**
`Rtp.Payment.Received`	Payment received and posted to an account in COS
`Rtp.Payment.Rejected`	Payment rejected by the receiving institution or Instant Payments network.

Register for relevant webhook events

- To receive webhook events, **register** for each webhook event type. Events are sent to the URLs you register.
- The event object includes resource identifiers that provide details on each event.

Originate a credit transfer

Instant payment credit transfers allow funds to move from your account to the recipient's account in real time. Transfers can originate from master accounts or subledgers.

- The **debtor** sends the payment, and the **creditor** receives it.
- Cross River supports three network platforms:
  - RTP® via The Clearing House (TCH)
  - FedNow®
  - CRNow

In this tutorial, you are the debtor. You will send an outbound credit transfer from your master account to the creditor using a specified network platform.

Possible Outcomes:

- **Accepted Transfer:** Funds are posted to the recipient's account.
  - **Status: ACTC** (Accepted)

- **Rejected Transfer:** Payment is rejected by the creditor FI or the network. No funds are transferred.
    - **Status: RJCT** (Rejected)
- **No Response:** The transfer times out, resulting in rejection.
    - **Status: RJCT** (Rejected)
- **Accepted Without Posting:** The payment is accepted but funds are not immediately posted. The creditor FI can accept or reject the transfer within 24 hours.
    - **Status: ACWP** (Accepted Without Posting)

> Cross River cannot confirm how the receiving FI displays payment details. Contact the recipient FI directly for specifics on how the payment will appear.

> **IMPORTANT**
> We strongly recommend that you include an **idempotency key** in the request header to prevent duplicate payments in case of a failure.

Initiating a credit transfer

- Use `POST /rtp/v1/payments` to initiate a credit transfer.
- Some attributes are required for the call. Refer to the full list of **attributes**.
- In this tutorial, you will send $175 to C. Brown.

> Amounts in API calls and responses are written without decimal points (e.g., $175 is written as 17500).

Example request: originate a credit transfer

```
POST /Rtp/v1/payments

{
  "accountNumber": "2553179843",
  "amount": 17500,
  "creditor": {
    "routingNumber": "011000138",
    "accountNumber": "456789000",
    "name": "C Brown",
    "addressStreetName": "Main St",
    "addressBuildingNumber": "34",
    "addressCity": "New York",
    "addressState": "NY",
    "addressPostalCode": "12345",
    "addressCountry": "US"
  },
  "purpose": "gift money"
}
```

A successful API call returns a JSON response containing the details of the originated credit transfer.

Sample Rtp.Payment.Sent event

```json
{
  "id": "7af80bc3-4f1c-4842-b60e-ad9400fb59db",
  "eventName": "Rtp.Payment.Sent",
  "status": "Pending",
  "partnerId": "bd7a716f-6349-43ef-89cd-aa2200f15977",
  "createdAt": "2021-08-30T11:15:08.623-04:00",
  "resources": [
    "rtp/v1/payments/7b5f4bfb-8595-452b-914e-ad9400f7b8e3"
  ],
  "details": [
    {
      "paymentId": "7b5f4bfb-8595-452b-914e-ad9400f7b8e3",
      "paymentType": "CreditTransfer",
      "resultCode": "OK",
      "resultAdditionalInfo": null,
      "coreTransactionId": "3e04e9b9-e80f-4f5f-9a0b-b04a011d23df",
      "memoPostId": "4820a916-f062-49ad-9519-b04a011d2372",
      "accountNumber": "2553179843",
      "postingCode": "OK",
      "rtpTransactionStatus": "ACTC",
      "purpose": null,
      "awaitingResponse": "False",
      "referencedPaymentId": null
    }
  ]
}
```

When a credit transfer (**pacs.008**) is sent to the instant payments network, the `Rtp.Payment.Sent` webhook fires. The event body includes the **payment ID** in the details section (e.g., **7b5f4bfb-8595-452b-914e-ad9400f7b8e3**).

Payment status webhooks

- `Rtp.Payment.Sent` – Confirms the payment was sent.
- `Rtp.Payment.Received` – Confirms the payment was received and posted.

These webhooks provide real-time updates on the status of your credit transfer.

Simulate rejected credit transfers

You can simulate a rejected response from the creditor:

1. Call `POST /rtp/v1/payments` (same as initiating a credit transfer).
2. Add `reject:` before the creditor's `name` (see line 9 in the request example).

Example request: rejected credit transfer

```
POST /v1/payments

{
  "accountNumber": "2553179843",
  "amount": 15000,
  "creditor": {
    "routingNumber": "011000138",
    "accountNumber": "456789000",
    "name": "reject:Cleveland Brown",
    "addressStreetName": "Main St",
    "addressBuildingNumber": "34",
    "addressCity": "New York",
    "addressState": "NY",
    "addressPostalCode": "00093",
    "addressCountry": "US"
  },
  "purpose": "gift money"
}
```

The response example shows a `status` of **Rejected** on line 13.

Example response: rejected credit transfer

```
{
    "id": "7b5f4bfb-8595-452b-914e-ad9400f7b8e3",
    "originalPaymentId": "7b5f4bfb-8595-452b-914e-ad9400f7b8e3",
    "referenceId": "R24200354Y055",
    "accountNumber": "2553179843",
    "amount": 15000,
    "operatorCoreTransactionId": "2da7c99c-5804-4757-98c1-ad9400fa6c2c",
    "memoPostId": "7b5f4bfb-8595-452b-914e-ad9400f7b8e3",
    "memoPostRemovedAt": "2021-08-30T11:14:44.973-04:00",
    "direction": "Outbound",
    "status": "Rejected",
    "paymentType": "CreditTransfer",
    "source": "Api",
    "transactionAccountContext": "Rejected",
    "rtpTransactionStatus": "RJCT",
    "debtor": {
        "routingNumber": "021214891",
        "accountNumber": "2553179843",
        "name": "P Griffin",
        "addressStreetName": "Main Street",
        "addressBuildingNumber": "31",
        "addressCity": "New York",
        "addressState": "NY",
        "addressPostalCode": "00093",
        "addressCountry": "US"
    },
    "creditor": {
        "routingNumber": "011000138",
        "accountNumber": "456789000",
        "name": "C Brown",
        "addressStreetName": "Main St",
        "addressBuildingNumber": "34",
        "addressCity": "New York",
        "addressState": "NY",
        "addressPostalCode": "00093",
        "addressCountry": "US"
    },
    "network": {
        "messageDefId": "pacs.008.001.08",
        "businessMessageId": "B20210830021214273T1BQPZ97287285414",
        "messageId": "M20210830021214273T1BEML46024873029",
        "createdAt": "2021-08-30T11:01:55.74-04:00",
        "numberOfTransactions": 1,
        "interbankSettlementAmount": 15000,
        "currency": "USD",
          "interbankSettlementDate": "2021-08-30",
          "settlementMethod": "CLRG",
          "clearingSystemCode": "TCH",
          "instructionId": "20210830021214273T1B4S0534677157734",
          "endToEndId": "9a2638dc8cbe48f18d8cad9400f7b8e3",
          "transactionId": "20210830021214273T1B4S0534677157734",
          "clearingSystemRef": "001",
          "fromParticipantId": "021214273T1",
          "toParticipantId": "990000001S1",
          "instructingRoutingNumber": "021214891",
          "instructedRoutingNumber": "011000138",
          "headerCreationDate": "2021-08-30T11:11:45.373-04:00",
          "messageCreationDateTime": "2021-08-30T11:01:55.74-04:00"
      },
      "confirmedStatus": {
          "messageStatusReportId": "2da7c99c-5804-4757-98c1-ad9400fa6c2c",
          "acceptedDateTime": "2021-08-30T11:11:45.4-04:00",
          "transactionStatus": "ACTC"
      },
      "serviceLevelCode": "SDVA",
      "localInstrumentProprietary": "STANDARD",
      "categoryPurpose": "CONSUMER",
      "currency": "USD",
      "chargeBearer": "SLEV",
      "purpose": "gift money",
      "wasRefunded": false,
      "wasPaid": false,
      "createdAt": "2021-08-30T11:01:55.74-04:00",
      "completedAt": "2021-08-30T11:14:45.05-04:00",
      "processingStartedAt": "2021-08-30T11:11:45.247-04:00",
      "postingCode": "OK",
      "productId": "13362d99-f04e-455b-9363-abc10151c27c",
      "partnerId": "bd7a716f-6349-43ef-89cd-aa2200f15977",
      "lastModifiedAt": "2021-08-30T11:14:45.0505825-04:00",
      "sentAt": "2021-08-30T11:11:45.3733333-04:00",
      "sendAttemptCount": 1,
      "result": {
          "code": "OK"
      },
      "discounts": [],
      "awaitingResponse": false
  }
```

The `Rtp.Payment.Rejected` event is triggered when a credit transfer is rejected.

- The `details` object in the event contains the **payment ID** from the response body (e.g., **7b5f4bfb-8595-452b-914e-ad9400f7b8e3**).
- The `resultCode` in the details object provides the rejection reason.
    - Example: **AC06** – Account is blocked.

Sample Rtp.Payment.Rejected event

```
{
  "id": "06afb069-dc90-48ea-b642-b04a0126580f",
  "eventName": "Rtp.Payment.Rejected",
  "status": "Pending",
  "partnerId": "1e5d3f04-ae24-4af6-9e30-aecf012b99dd",
  "createdAt": "2023-07-25T13:51:40.417-04:00",
  "resources": [
    "rtp/v1/payments/7b5f4bfb-8595-452b-914e-ad9400f7b8e3"
  ],
  "details": [
    {
      "paymentId": "7b5f4bfb-8595-452b-914e-ad9400f7b8e3",
      "paymentType": "CreditTransfer",
      "resultCode": "AC06",
      "resultAdditionalInfo": null,
      "postingCode": "RES",
      "rtpTransactionStatus": null,
      "purpose": null,
      "referencedPaymentId": null
    }
  ]
}
```

5.2. Get service info

In this tutorial, you'll learn how to:

✅ Identify which Instant Payments receive services are available to creditor banks through **RTP® via TCH or FedNow®.**

> If you are new to instant payments, check out the **instant payments** overview.
>
> This tutorial assumes familiarity with APIs. For more information, visit **API basics**.

Before you begin

Make sure you have:

- **API credentials**

API endpoints used in this tutorial:

API	Description
GET /rtp/v1/directory	Returns available Instant Payments receive services

> For FedNow, banks must register each routing transit number (RTN) individually. Banks may have partial RTN registration.

Access the directory

You'll check the available services for **JP Morgan Chase Bank**. The API call returns a list of service codes. If the bank supports both **RTP via TCH** and **FedNow**, codes for each network will be returned.

- Call `GET /rtp/v1/directory?{queryParam}={queryParamValue}`.

- Include at least one query parameter to filter and control the data returned.

`filter.name`	The official name of the financial institution (FI)
`filter.routingNumber`	The routing number of the recipient account *(Best practice: Filter by this value for accuracy)*
`filter.participantId`	The participant ID of the FI
`filter.institutionRoutingNumber`	The routing number of the FI

Enter routing number **021000021** as either `routingNumber` or `institutionRoutingNumber` to retrieve results for the financial institution.

Sample query using routing number

```
curl -X GET /rtp/v1/directory

'https://sandbox.crbcos.com/Rtp/v1/directory?filter.routingNumber=021000021'
```

A successful API call returns a **JSON response** listing the supported Instant Payments services for routing number **021000021**, associated with **JP Morgan Chase Bank**.

- The `receiveServices` object contains the service codes.
- The `networkPlatform` attribute specifies the Instant Payments network (e.g., RTP® or FedNow®).
- If the bank participates in both networks, the response includes a separate list of services for each.

Sample directory response with both networks

```
{
      "networkPlatform": "TCH",
      "routingNumber": "021000021",
      "participantId": "200000020T1",
      "name": "JPMorgan Chase",
      "institutionRoutingNumber": "200000021",
      "institutionName": "JPMorgan Chase",
      "receiveServices": [
        "ACK",
        "CRDT",
        "RFI",
        "RFIR",
        "RFP",
        "RFPR",
        "RFRF",
        "RFRFR",
        "RMT"
      ],
      "receivingConnection": "JPMC",
      "participantActivationDate": "11/13/2017 12:00:00 AM",
      "extractionDateTime": "9/29/2022 7:00:00 AM",
      "lastModifiedAt": "2022-09-29T14:23:01.9180766-04:00",
      "networkPlatform": "TCH",
      "onlineStatus": "SignedOn",
      "onlineStatusChangedAt": "2022-09-15T12:22:04.9180766-04:00"
    },
    {
      "routingNumber": "021000021",
      "name": "JPMORGAN CHASE BANK, NA",
      "receiveServices": [
        "CTSR"
      ],
      "lastModifiedAt": "2022-09-29T14:23:01.9180766-04:00",
      "networkPlatform": "FedNow",
      "onlineStatus": "SignedOn",
      "onlineStatusChangedAt": "2022-09-15T12:22:04.9180766-04:00"
    }
```

Even if a financial institution lists a specific service, it does not guarantee that the creditor or debtor account at that institution is eligible for it.

Available Instant Payments services for TCH

Code	Description	Details
CRDT	Credit Transfer	Permits receipt of Credit Transfer (pacs.008)
RFP	Request for Payment	Permits receipt of Request for Payment (pain.013)
ACK	Payment Acknowledgement	Permits receipt of Payment Acknowledgement (camt.035)
RMT	Remittance Advice	Permits receipt of Remittance Advice (remt.001)
RFI	Request for Information	Permits receipt of Request for Information (camt.026)
RFRF	Request for Return of Funds	Permits receipt of Request for Return of Funds (camt.056)
RFPR	Request for Payment Response	Permits receipt of Request for Payment Response (pain.014)
RFIR	Request for Information Response	Permits receipt of Request for Information Response (camt.028)
RFRFR	Request for Return of Funds Response	Permits receipt of Request for Return of Funds Response (camt.029)

Available Instant Payments services for FedNow

Code	Description	Details
CTRO	Credit Transfer Receive Only	Indicates a FedNow participant is enabled to receive but not send customer credit transfer messages.
CTRO	Credit Transfer Receive Send	Indicates a FedNow participant is enabled to send and receive customer credit transfer messages.
RFPR	Request For Payment Receive	Indicates a FedNow participant is enabled to receive request for payment messages.

5.3. Set payment expiration

Cross River queues instant payments to be sent later if for some reason the receiving bank is offline and cannot receive the credit transfer immediately.

In this tutorial, you'll learn how to:

✅ Set an expiration time for how long a payment remains in the queue before being cancelled

> If you are new to instant payments, check out the **instant payments** overview.
>
> This tutorial assumes familiarity with APIs. For more information, visit **API basics**.

Before you begin

Make sure you have:

- **API credentials**
- Your partner ID
- Your master account number

The tutorial uses these API endpoints:

POST /rtp/v1/payments	**Transfers funds between banks in real time via an Instant Payments network**
POST /rtp/v1/payments/{id}/payment-request/cancel	Cancels a payment request if a credit transfer or payment request response has not been received.

The tutorial uses these webhooks:

`Rtp.Payment.Queued`	**Payment queued for sending when the receiving financial institution comes back online**
`Rtp.Payment.Canceled`	Payment canceled

Register for relevant webhook events

- To receive webhook events, **register** for each webhook event type. Events are sent to the URLs you register.
- The event object includes resource identifiers that provide details on each event.

Set queued payment expiration time

Prepare to send an instant payment as described in the **send an instant payment** tutorial.

Define your expiration time:

- Set a specific date and time (`queuedPaymentExpiresAt`)
- Define an expiration time in seconds (`queuedPaymentExpiresAfterInSeconds`)

If both are provided, the system prioritizes the **time in seconds** and ignores the date and time.

Payments have a default queue expiration of 3 days.

If the expiration is reached, Cross River cancels the payment and triggers an `Rtp.Payment.Canceled` webhook.

By date and time

- Use the `queuedPaymentExpiresAt` attribute in your payment request.
- Format: **yyyy-mm-ddThh:mm:ss** (US Eastern Time).
- The value must be a future date and time.
- This field is optional.

For reference, the sample request includes this attribute on row 17.

Sample payment request with date and time expiration queuing

```
POST /v1/payments
{
"accountNumber": "2553179843",
"amount": 15000,
"creditor": {
"routingNumber": "011000138",
"accountNumber": "456789000",
"name": "Cleveland Brown",
"addressStreetName": "Spooner St",
"addressBuildingNumber": "34",
"addressCity": "Quahog",
"addressState": "RI",
"addressPostalCode": "00093",
"addressCountry": "US"
"addressCountry": "US"
},
"queuedPaymentExpiresAt": "2023-02-19T08:22:17.512Z"
}
```

By time in seconds

- Use the `queuedPaymentExpiresAfterInSeconds` attribute in your payment request.
- Enter the value in seconds (whole number).
- The countdown starts when the payment is queued.
- A value of **0** cancels the payment immediately if the RDFI is offline.
- This field is optional.

For reference, the sample request includes this attribute on row 17.

Sample payment request with queuing expiration in seconds

```
POST /v1/payments
{
"accountNumber": "2553179843",
"amount": 15000,
"creditor": {
"routingNumber": "011000138",
"accountNumber": "456789000",
"name": "Cleveland Brown",
"addressStreetName": "Spooner St",
"addressBuildingNumber": "34",
"addressCity": "Quahog",
"addressState": "RI",
"addressPostalCode": "00093",
"addressCountry": "US"
"addressCountry": "US"
},
"queuedPaymentExpiresAfterInSeconds": 3600
}
```

Cancel a queued payment

Cancel a queued payment anytime using the `POST /v1/payments/{paymentId}/cancel` endpoint.

Test queuing in sandbox

To simulate offline participants, three participants alternate between online and offline every 60 minutes.

1. Register for the `Rtp.Payment.Queued` webhook.
2. Submit a payment using one of the following routing numbers:
    - **000000010**
    - **000000017**
    - **244084264**

3. If the participant is offline, the `Rtp.Payment.Queued` webhook fires, and the payment status changes to **Queued**.
4. Once the participant returns online, the payment resumes normal processing.

If the payment isn’t queued, continue submitting until the participant cycles offline.

5.4. Fraud reporting

Each instant payment network has specific requirements for reporting suspected fraud, with its own criteria for what qualifies as reportable fraud.

You *must* notify Cross River about any suspected fraudulent transaction.

Reporting procedure

For *all* instant payment networks, if an activity meets the network's definition of a fraudulent payment, you must notify Cross River as follows:

1. Send an email to **iprops.support@crossriver.com** with *at least* the following information, per payment network:

FedNow	RTP	CRNow
A notice that a **Reportable Transfer** occurred over FedNow	A notice that an **unauthorized transaction** occurred over RTP	A notice that a **fraudulent payment** occurred over CRNow
Date of transaction	Date of transaction	Date of transaction
Payment ID or COS reference ID	Payment ID or COS reference ID	Payment ID or COS reference ID
Amount	Amount	Amount
Customer name	Customer name	CRNow customer name payment was sent to/from
Other financial institution that is party to the transaction	Receiving financial institution	
FraudClassifer model type code		
Party associated with the Reportable Transfer. Valid values are: • **S**: Sender is the suspected fraudulent party • **R**: Receiver is the suspected fraudulent party • **B**: Both Sender and Receiver are suspected fraudulent parties		

2. In accordance with the thresholds outlined in the **Incident Report Submission Summary**, send an Incident Report, if warranted, to **unusualactivityreferrals@crossriverbank.com**. Use the **Incident Report form template**.

Reporting requirements

Each instant payments network has explicit reporting requirements.

FedNow

> **Summary**
>
> If a funds transfer made through the FedNow network is later identified as potentially fraudulent, you as a Cross River partner must report it to Cross River, enabling us to notify FedNow.

The FedNow Service requires participants to report *Reportable Transfers* sent over the network. A Reportable Transfer is defined as:

> ***"Any funds transfer completed, in part, through the FedNow Service based on a payment order sent or received by a FedNow Participant that was authorized by the sender at the time of submission but was later determined to potentially involve fraudulent activity. The FedNow Participant must have a good-faith belief that the transaction resulted from fraudulent activity."***

You *must* ensure compliance with this requirement by reporting any Reportable Transfers to Cross River, regardless of whether you are the sender or receiver of the transfer.

FraudClassifer model type codes

The table below presents the various FedNow FraudClassifier model type codes.

Fraud type code	Code name	Fraud type	Description	Notes/examples
FC00	Transaction is not fraudulent	None	If a Reportable Transfer was previously reported and is later determined not to be the result of fraudulent activity, use FC00 to indicate Not Fraud.	N/A
FC01	Authorized party was manipulated	Product and Services Fraud	A situation involving a transfer of funds in exchange for a product or service, irrespective of the nature of the	Examples include rental scams, travel scams, lottery scams, tech support scams, home repair scams, home alarm scams, free trial scams, brain booster scams, gold coin scams, etc. Example: Paul has been wanting a

Fraud type code	Code name	Fraud type	Description	Notes/examples
			relationship between the two parties, whereby the receiver of the funds does not deliver the product or service or delivers a grossly inferior product or service than advertised or promised.	puppy and found a great deal online. For $75, he can get a chocolate lab puppy, including delivery. He needs to send $75 to the information provided in the ad. Excited about this great deal, he sends the money but never received the puppy.
FC02	Authorized party was manipulated	Relationship and Trust Fraud	A situation involving a transfer of funds to a trusted party or an imposter acting as a trusted or authorized party, where there is no expectation or promise of goods or services in exchange for the transferred funds; the seemingly trustworthy party can be an existing or emerging relationship or a party pretending	Examples include IRS imposter scams, Social Security imposter scams, sheriff's office scams (jury duty), romance scams, grandparent scams, utility scams, fake debt collections, duplicate payment scams, etc. Example: Joan developed a relationship with Fred online. A day before the first meeting in person, Fred asked Joan to send him $10,000 to get out of serious trouble. Joan sent the money to Fred. Fred does not show up for their meeting and Joan never hears from him again.

Fraud type code	Code name	Fraud type	Description	Notes/examples
			to be an authority or reputable company.	
FC03	Authorized party acted fraudulently	Embezzlement	Theft or misuse of funds legally placed in one's trust or belonging to one's employer.	This would include situations involving agents acting for others. Example: Tina, the Treasurer, had the ability to initiate payments at her company. Tina instructs the company's FIs account to her personal account.
FC04	Authorized party acted fraudulently	Synthetic Identity Fraud (SIF)	The use of a combination of personally identifiable information (PII) to fabricate a person or entity to commit a dishonest act for personal or financial gain.	Use of a false identification to create an account with the intent to commit fraud. Example: Fred opens a deposit account under a fabricated identity. Fred uses the account to collect payments from his fraudulent scheme, wires the amount to an offshore account, and leaves the account dormant.
FC05	Authorized party acted fraudulently	False claim	An intentional lie or deception to receive a payment or avoid a payment obligation.	Informing a consumer of a false situation to obtain funds (e.g., overdue utility bill/disconnect; child/grandchild in prison). Example: Betsy orders online and makes the payment electronically. Days after she received the goods, she calls her bank, reports the

Fraud type code	Code name	Fraud type	Description	Notes/examples
				purchases as fraudulent, and seeks a refund.
FC06	Unauthorized party took over account	Compromised Credentials	Account login information, intended only for an authorized party is obtained by an unauthorized party.	Account login information (e.g., ID/password) allows one to access an account and is not specific to personal information, contact information, etc. Access to personal information, contact information, etc. would be classified under Impersonated Authorized Party. Example: Using Greg's login ID and password, Frank gains full access to Greg's online bank account. Frank then proceeds to initiate several transfers through Greg's bank and the FedNow Service from Greg's account to an account at different bank.
FC07	Unauthorized party modified payment information	Compromised Credentials	Unauthorized Party has obtained access to a payment instruction “in process” and modified it to redirect funds to an account they have access to.	Account login information (e.g., ID/password) allows one to access an account and is not specific to personal information, contact information, etc. Access to personal information, contact information etc. would be classified under Impersonated Authorized

Fraud type code	Code name	Fraud type	Description	Notes/examples
				Party Example: Steve set up an online recurring bill payment from his bank account while his roommate was nearby. His roommate later authenticated into Steve's account using Steve's ID/password and modified the recurring payment. Upon processing of the payment, funds were redirected to an account under the roommate's control
FC08	Unauthorized party modified payment information	Impersonated Authorized Party	A person or organization who does not have authorized credentials but has enough information to authenticate as the Authorized Party.	Authorized credentials in this context include account login information (e.g., ID/password) that allows one to access an account and is not specific to personal information, contact information, etc. Example: Jim scheduled an online bill payment. A day later, Jake represented himself as Jim by successfully answering verification questions asked by the call center associate. Jake (as Jim) then instructed the call center associate to modify the scheduled payment, redirecting it to an account in his control. Upon

Fraud type code	Code name	Fraud type	Description	Notes/examples
				receipt of the payment, Jake withdrew the funds

The Clearing House (RTP)

Summary

You as a Cross River partner must report to Cross River any funds transfer using the RTP network that a network user learns afterwards was not authorized by the sender, so we can notify The Clearing House.

We will request that the funds receiver return the funds, flagging the request as due to suspected fraudulent activity.

Under The Clearing House (TCH) RTP Operating Rule II.G.2, Participants must report fraudulent activity involving the RTP System to TCH and the other Participant involved, following the RTP Technical Specifications and Risk Management and Fraud Control Requirements.

Section 5 of the Risk Management and Fraud Control Requirements states:

> ***"A Participant must report any instance of fraudulent activity or suspected fraudulent activity to TCH subject to and in accordance with the RTP Operating Rules and other procedures established by TCH from time to time."***

A fraudulent RTP Payment is a payment the Sending Participant determines was unauthorized by the Sender ("Unauthorized Payment") based on an investigation of how it was initiated.

A payment authorized by the Sender but induced under false pretenses does not qualify as an Unauthorized Payment under this rule.

You *must* report any suspected unauthorized RTP Payments to Cross River. We take the necessary steps, including notifying the Receiving Participant and submitting a Request for Return of Funds message with the "FRAD" reason code to request a return.

Your timely reporting ensures compliance with TCH rules and facilitates an efficient fraud response.

CRNow (book transfers)

> **Summary**
>
> You as a Cross River partner must report any funds transfer made through Cross River's CRNow (book transfers) network that is identified as fraudulent or unauthorized.

Cross River's procedures require participants to report fraudulent activity involving the CRNow system.

A fraudulent CRNow (book transfers) payment is defined as:

> ***"Any funds transfer completed through the CRNow Service, based on any Payment sent or received by a CRNow Participant, that resulted from fraudulent or unauthorized activity."***

You *must* ensure timely reporting of such transactions to Cross River.

Incident report submission summary

Partners must escalate the following **unusual activity** to **Cross River Bank (CRB)** via an **Incident Report (IR):**

- Criminal violations involving insider abuse of any amount.
- Criminal violations of $5,000 or more when a suspect can be identified.
- Criminal violations of $25,000 or more, regardless of whether a suspect is identified.

- Transactions aggregating $5,000 or more, if the Partner knows, suspects, or has reason to suspect that the transaction:
    - May involve money laundering or other illegal activity.
    - Is designed to evade the BSA or its regulations.
    - Lacks a clear business purpose or is inconsistent with the customer’s expected activity, with no reasonable explanation after reviewing available facts.

Incident reporting process

- As a partner, maintain a documented incident reporting process to notify Cross River of potentially unusual or suspicious activity.
- Escalate any unusual activity within 5 days of detection by submitting a CRB Incident Report to **unusualactivityreferrals@crossriverbank.com**.
- Include any relevant supporting documentation in your report.

Timely reporting is critical to ensure compliance and mitigate financial crime risks.

Incident report template

Download our MS Word incident report template as an example:

Incident report template.docx

6. Card payments

Our card payments tutorials explain how to use our APIs to:

- **Send a push transaction**: Sign up a card and send funds to that card.
- **Send a pull transaction**: Sign up a card and pull funds from that card.
- **Set up iFrame**: Set up a PCI-compliant iFrame. This function will be deprecated. See the **template-based way** of generating iFrame code.

6.1. Send a push transaction

In this tutorial, you'll learn how to:

✅ Sign up a card

✅ Send a push payment

> This tutorial assumes you have a knowledge of APIs and how they work. Refer to the **API basics** for more details.

The tutorial uses these API endpoints:

API	Description
POST /api/Card	Signs up a card
POST /api/transaction	Sends a payment

The tutorial uses these webhooks:

Webhook	Description
`CardAuthorized`	• Reports when a card authorization attempt is completed • Shows you the status of the card
`Transaction`	Reports a transaction

Before you begin

Make sure you have:

- **API credentials**
- `requestId` - your unique reference ID
- `cardToken` - the token you received when you registered the card

- The dollar amount to send
- Enough funds in your account
- The name of the person or organization originating the push transaction

Register relevant webhook events

Register to receive the webhooks available for P2C. These webhooks report relevant events back to your system in real-time. This keeps you up to date on each transaction.

Sign up a card

To start a transaction you must first register a card. We use different endpoints to be able to secure your customer's debit card numbers. To register a card, send Cross River the card number. The card number is instantly converted into a token. Cross River doesn't save any card number information. Doing this asynchronously allows Cross River to store your customer's card information. This ensures that you don't need to retrieve the card information again. It also keeps your customer's data secure.

1. Call `POST /api/Card`.
2. When a card is registered, Cross River validates the card with the relevant card network. This makes sure the card is a valid card and that it is allowed to receive payments.
3. The `CardAuthorized` webhook is triggered.

Sample request POST /api/Card

```
{
   "RequestId":"49af65c0-f815-4f49-ba8d-b67bf1b125f4",
   "FirstName":"Joseph",
   "LastName":"Roll",
   "OwnerExternalId":"4444",
   "Address1":"123 Main Street",
   "City":"Venice",
   "State":"CA",
   "ZipCode":"10989",
   "CountryCode":"US",
   "PhoneNumber":"5555559275",
   "Email":"{youremail}",
   "CreditCardNumber":"full card number",
   "ExpirationMonth":"06",
   "ExpirationYear":"20",
   "CCV":"420"
}
```

Start a payment

When you send funds directly to a debit card, we call this a push payment, or a push-to-card (P2C) transaction.

1. Authenticate into the Cross River system. This returns a token to you.
2. Use the token to call `POST /api/transaction.`
3. Cross River sends this request to the card networks. The networks direct the transaction to the right bank.
4. The issuing bank (of the registered debit card) receives the transaction.
5. The issuing bank either authorizes or declines the transaction.
6. If the bank authorizes the transaction, your payee will receive the funds.

Sample request POST /api/transaction

How to originate an Account Funding Transaction (AFT)

In this tutorial, you'll learn how to:

✅ Register the relevant webhooks

✅ Sign up a card

✅ Start a pull transaction

> This tutorial assumes you have a knowledge of APIs and how they work. Refer to the **API basics** for more details.

An Account Funding Transaction (AFT) lets you pull funds from a debit card for specific purposes. You can use an AFT to fund a wallet or a prepaid card. Or, you can use it to initiate a person to person (P2P) transfer through the card network rails. AFTs are not allowed for buying goods and services.

The tutorial uses these API endpoints:

API	Description
POST /api/Card	Signs up a card
POST /api/PullTransaction/	Starts a pull payment

The tutorial uses these webhooks:

Webhook	Description
`CardAuthorized`	• Reports when a card authorization attempt is completed • Shows you the status of the card
`Transaction`	Reports a transaction

Before you begin

Make sure you have:

- **API credentials**
- `requestId` - your unique reference ID
- `cardToken` - the token you received when you signed up the card
- The dollar amount to send

Register relevant webhook events

Register to receive the webhooks available for P2C. These webhooks report relevant events back to your system in real-time. This keeps you up to date on each transaction.

Sign up a card

To start a transaction you must first register a card. We use different endpoints to be able to secure your customer's debit card numbers. To register a card, send Cross River the card number. The card number is instantly converted into a token. Cross River doesn't save any card number information. Doing this asynchronously allows Cross River to store your customer's card information. This ensures that you don't need to retrieve the card information again. It also keeps your customer's data secure.

1. Call `POST /api/Card.`
2. When a card is registered, Cross River validates the card with the relevant card network. This makes sure the card is a valid card and that it is allowed to receive payments.

3. The `CardAuthorized` webhook is triggered.

Sample request POST /api/Card

```
{
   "RequestId":"49af65c0-f815-4f49-ba8d-b67bf1b125f4",
   "FirstName":"Joseph",
   "LastName":"Roll",
   "OwnerExternalId":"4444",
   "Address1":"123 Main Street",
   "City":"Venice",
   "State":"CA",
   "ZipCode":"10989",
   "CountryCode":"US",
   "PhoneNumber":"5555559275",
   "Email":"{youremail}",
   "CreditCardNumber":"full card number",
   "ExpirationMonth":"06",
   "ExpirationYear":"20",
   "CCV":"420"
}
```

Start a pull transaction

When you start a transfer, you are requesting funds (as the recipient) from a payor.

1. Authenticate into the Cross River system. This returns a token to you.
2. Use the token to call `POST /api/PullTransaction`.
3. Cross River sends this request to the card networks. The networks route the transaction to the appropriate bank.
4. The issuing bank (of the registered debit card) receives the transaction.
5. The issuing bank either authorizes or declines the transaction.
6. If the transaction is authorized, Cross River responds to your API call as shown below.

Sample request POST /api/PullTransaction

```
curl -X POST --header 'Content-Type: application/json'
--header 'Accept: application/json'
--header 'Authorization: Bearer 12AAB60C01ED32A0C18D44111A753BF3CE180BC716FEC
   "requestId": "89GVA07C-FJ8X-42NV5-9A00C9-6MS5AQ3D7100",
   "cardToken":"KR7D47MNJIL2R39OZNRVPC0DIS82ES8JE6J7VU",
   "amount": 400
 }
```

Sample response POST /api/PullTransaction

```
{
  "version": "1.0.0.0",
  "result": {
    "transactionRequestId": "7CE5A0DC-253B-4AB5-9CC9-6E9BE63D7100",
    "amount": 400,
    "transactionRequestedAt": "2022-08-03T21:09:13.5882004Z",
    "transactionStatus": "Succeeded",
    "errorDescription": null,
    "creditCardId": "KR7D47MNJIL2R39OZNRVPC0DIS82ES8JE6J7VU",
    "railId": "TabaPay",
    "network": "MasterCard",
    "retrievalReferenceId": "8abcb1fa-51f1-4b2d-9998-5948877bdcc0",
    "actualTransactionDoneAt": "2022-08-03T21:09:13.937108Z",
    "requestApproved": true,
    "responseReceived": true,
    "responseCode": "00",
    "responseDescription": "Approved",
    "traceNumber": "xxx",
    "error": null,
    "requesterName": null,
    "requesterMcc": null
  },
  "isSuccessfull": true,
  "isSuccessful": true
}
```

6.3. Set up iFrame

How to set up your iFrame to integrate with P2C

As a Cross River merchant partner, suppose you prefer not to hold the (PII) of your customers. This could include, for example, customer debit card numbers and security details. iFrames offer a solution to PII problem. The Cross River iFrame gives you a simple interface to capture debit card details. You can easily add an iFrame to your website.

Your customer's card details pass through directly to Cross River. They are not stored on your servers.

In this tutorial, you'll learn how to:

✅ Register the relevant webhooks

✅ Set up an iFrame to integrate with P2C

> This tutorial assumes you have a knowledge of APIs and how they work. Refer to the **API basics** for more details.

The tutorial uses these API endpoints:

API	Description
POST /api/V2/iFrameConfiguration/GenerateOtcSignupCard	Add an iFrame into your site

The tutorial uses these webhooks:

Webhook	Description
`CardAuthorized`	• Reports when a card authorization attempt is completed • Shows you the status of the card

Before you begin

Make sure you have:

- **API credentials**
- Contacted the **Integration Team** to register the domains where you'll be embedding the iFrame

Register relevant webhook events

Register to receive the webhooks available for P2C. These webhooks report relevant events back to your system in real-time. This keeps you up to date on each transaction.

The `CardAuthorized` webhook reports when a new card was authorized on our system.

> **No webhook received**
>
> Were you able to sign up for a card but didn't get a webhook? The webhook function call might not be returning a 200 ("success") message to Cross River.

Set up your iFrame

1. Authenticate into the Cross River system. This returns a token to you.
2. Use the token to call `POST api/IframeConfiguration/BuildSignupCardUrl`. The endpoint will return a response containing a unique URL. That URL is valid for 5 minutes.
3. Pass the URL into the `src` attribute of the iFrame.

Attributes for building the iFrame

requestId **string**	**The GUID that enables the application to link request with response**
customerReferenceNumber string	A 4-digit code, assigned by the merchant, to identify the cardholder
domain string	The name of a valid top-level internet domain where consumers will use the iFrame. For example, myfintech.com. This domain must appear in the Cross River allowlist.
successContinueNavigationPoint string	The landing page you are directed to if the sign up was successful
failureContinueNavigationPoint string	The landing page you are directed if the sign-up wasn't succeessful
firstName string	Cardholder's first name
lastName string	Cardholder's last name
address1 string	Primary location details of cardholder. For instance, street name, house or building number, and PO box. Maximum 255 characters.
address2 string	Secondary location details of cardholder. For instance, number of apartment or floor. Maximum 255 characters.
city string	City full name
state string	2-letter code of the cardholder's state
countryCode string	2-letter Country code (2 letters)
zipCode string	ZIP code

requestId **string**	**The GUID that enables the application to link request with response**
email string	A valid cardholder email address, for example, me@mailprovider.com
phoneNumber string	Phone number, no dashes required
showOptionalFields boolean	True if optional fields will appear in the iFrame, otherwise false

Sample request

```
{
  "requestId": "89clha9s-27ci-90ck-7jcs-8lksic02cjag",
  "customerReferenceNumber": "string",
  "domain": "string",
  "successContinueNavigationPoint": "string",
  "failureContinueNavigationPoint": "string",
  "firstName": "Joseph",
  "lastName": "Roll",
  "address1": "123 Main Street",
  "address2": "string",
  "city": "Venice",
  "state": "CA",
  "countryCode": "string",
  "zipCode": "66666",
  "email": "{email}",
  "phoneNumber": "string",
  "showOptionalFields": true
}
```

Response attributes

result string	**Result of whether or not the source URL was added to our server. Success or fail.**
isSuccessful boolean	True if a unique URL was built successfully, otherwise false

Sample response

```
{
  "version": "1.0.0.0",
  "result": "Success",
  "isSuccessfull": true
}
```

7. International payments

In this tutorial, you'll learn how to:

✅ Get an estimate of the exchange rate for a cross border payment

✅ Get a list of fields required to get a valid executable quote for the payment

✅ Request an executable quote

✅ Send a payment

✅ Deal with returned and rejected payments

✅ Send an international payment using COS Explorer

> If you are new to International Payments we recommend you read **International payments** before starting this tutorial.
>
> This tutorial assumes you have a knowledge of APIs and how they work. For more information on sending API calls, refer to **API basics**.

The tutorial uses these API endpoints:

API	Description
GET /international/v1/estimates	Returns the estimated cost of sending an international payment, including the exchange rate
GET /international/v1/meta/quote-requirements	Returns a list of country-specific fields you need to submit when requesting a quote
POST /international/v1/quotes	Requests an executable payment quote for sending either USD or foreign currency before actually sending funds
POST /international/v1/payments	Executes a payment quote to send funds internationally

Before you begin

Make sure you have:

- **API credentials**
- Account number of the sending Cross River account
- Sending account configured by Cross River to send International Payments
- Fees configured by Cross River according to your signed agreement

Register relevant webhook events

To receive the webhook events for this tutorial you need to **register** each specific webhook event type. Once you are registered, the event objects are sent to the registered URLs.

A basic event object contains a list of resource identifiers used to download details on each event. An extended event object contains more details.

For this tutorial register for this webhook event:

Event Name	Description
`International.Payment.Sent`	Funds have been sent via a wire payment to the receivers bank.

Send an international payment from a Cross River account

To send a payment cross border, you must make several API calls:

1. **Get an estimate of the fees and exchange rate**.
   This call gives a general idea of how much the money transfer will cost the customer. We recommend you call this endpoint but it's not required.
2. **Determine the required fields for getting a quote for a payment to a specific country**.
   Depending on the country, the required information can differ. It's important to know which values you must supply for the quote call to complete without errors.
3. **Request a quote for the exchange**.
   This call returns a `quoteId` that is required to make the actual payment. The quote is usually good for 30 seconds.
4. **Originate the payment**.
   Include the `quoteId` to use the API to send the payment.

> Money amounts in API calls and responses are written without a decimal point between the dollars and the cents.

To get an exchange rate estimate

1 Call the **GET /international/v1/estimates** endpoint. For this call, you must supply your Cross River `accountNumber`, the desired `currency` of the received payment, and either amount in USD you plan to send (`fromAmount`) or the amount in the foreign currency you want sent (`toAmount`).

In this example, the account number is **158560897007**, the currency is Great British Pounds (**GBP**), and the from amount is 10.00 USD (**1000**).

IMPORTANT

You must have a value for either a `fromAmount` or a `toAmount`, but not both.

Sample request

```
curl --location 'https://sandbox.crbcos.com/international/v1/estimates?a
--data ''
```

2 A successful API call returns a JSON response with the details of the estimate. This estimate is non-binding and only gives you an approximate idea of what the exchange rate will be.

In this example, we provided the `fromAmount`. The `toAmount` returned is **7.89** GBP at an exchange rate of **0.7889** USD to the GBP. In addition, you can see that a regular transaction costs **1.29** USD while a priority transaction (SWIFT) costs **14.00** USD.

Sample response

```
{
    "accountNumber": "158560897007",
    "currency": "gbp",
    "fromAmount": 1000,
    "toAmount": 789,
    "exchangeRate": "0.7889",
    "regularTransactionFeeAmount": 129,
    "priorityTransactionFeeAmount": 1400
}
```

To get a list of required fields for a quote

1 Call `GET /International/v1/meta/quote-requirements.` For this call, you must supply values for all possible attributes. None are optional.

In this example, we provide the following values:

- `currency` : **GBP** (Great Britain Pounds)
- `beneficiaryCountry` : **US** (United States)
- `bankCountryCode` : **GB** (Great Britain)
- `entityType` : **Individual** (what legal entity is receiving the payment)
- `priority` : If no value is supplied, the default is false meaning not via SWIFT.

Sample request

```
curl --location 'https://sandbox.crbcos.com/International/v1/meta/quote-
--data ''
```

2 A successful API call returns a JSON response with a list of required fields/attributes you must provide values for when you call `POST /International/v1/quotes`.

The attributes in the response are required for the quote that will have the parameters as defined in this request. The response values describe the required responses. For example, for `lastName` the value is **^([^0-9]{2,255})$**, indicating that regular expression characters are permitted, up to 255 characters maximum. The `bankCountryCode` is **GB**, as provided in the request. Attributes regarding entities and FIs refer to the beneficiary only

Sample response

```
[
    {
        "firstName": "^([^0-9]{2,255})$",
        "lastName": "^([^0-9]{2,255})$",
        "currency": "gbp",
        "address": "^.{1,255}",
        "city": "^.{1,255}",
        "postalCode": "^.{1,12}$",
        "countryCode": "US",
        "routingCodeType1": "SortCode",
        "routingCodeValue1": "^\\d{6}$",
        "bankName": "^.{1,255}",
        "bankCountryCode": "GB",
        "receiverAccountNumber": "^\\d{8}$",
        "entityType": "Individual",
        "paymentNetwork": "Regular",
        "priority": false
    }
]
```

To request an international payment quote

1. Call `POST /International/v1/quotes`. For this call, you *must* supply values for the fields returned in the the `GET /International/v1/meta/quote-requirements` call.

2

Sample request

```
curl --location 'https://sandbox.crbcos.com/International/v1/quotes' \
--data '{
  "currency": "gbp",
  "accountNumber": "158560897007",
  "fromAmount": "500",
  "toAmount": "",
  "beneficiary": {
    "firstName": "Jon",
    "lastName": "Smith",

    "fullName": "JonSmith",

    "birthDate": "2001-06-18T13:05:09.015Z",
    "address": "1 Street",
    "city": "Winfield",
    "stateProvince": "",
    "postalCode": "GB12345",
    "countryCode": "GB",
    "entityType": "Individual"
  },
  "beneficiaryFi": {
    "bankName": "Bank UK",
    "bankCountryCode": "GB",
    "bankAddress": "1 Avenue",
    "bankAccountType": "Checking",

    "routingCodeType1": "SortCode",
    "routingCodeValue1": "123456789",
    "routingCodeType2": "aba",
    "routingCodeValue2": "123456789",
    "bicSwift": "TGCLGB99",
    "iban": "GB33BUKB20201555555555"
  },
  "priority": true,


  "purpose": "SRV"
}'
```

3 A successful API call returns a JSON response with a quote ID in the `id` field and information about the exchange rate. You need the quote ID to make the payment. The quote is valid for 30 seconds.

In this example, the quote ID is **f710a42a-e03a-47b8-a415-b3050061085e**.

JSON

```json
{
    "id": "f710a42a-e03a-47b8-a415-b3050061085e",
    "accountNumber": "158560897007",
    "currency": "gbp",
    "beneficiary": {
        "firstName": "Jon",
        "lastName": "Smith",
        "birthDate": "2001-06-18T00:00:00-04:00",
        "address": "1 Street",
        "city": "Winfield",
        "postalCode": "GB12345",
        "countryCode": "GB",
        "entityType": "Individual"
    },
    "beneficiaryFi": {
        "bankName": "Bank UK",
        "bankCountryCode": "GB",
        "bankAddress": "1 Avenue",
        "bankAccountType": "Checking",
        "routingCodeType1": "SortCode",
        "routingCodeValue1": "123456789",
        "routingCodeType2": "ABA",
        "routingCodeValue2": "123456789",
        "bicSwift": "TGCLGB99",
        "iban": "GB33BUKB20201555555555"
    },
    "fromAmount": 500,
    "toAmount": 394,
    "transactionFee": 100,
    "conversionRate": 0.7889,
    "estimatedDeliveryDate": "2025-06-23T00:00:00-04:00",
    "expiresAt": "2025-06-23T01:54:18.8357042-04:00",
    "status": "Created",
    "priority": true,
    "paymentNetwork": "Priority",
    "purpose": "SRV"
}
```

To send an international payment

1. Call `POST /International/v1/payments`. For this call, you must supply the quote ID from the `id` field returned in the the `POST /International/v1/quotes` call. In this example, the quote ID is **f710a42a-e03a-47b8-a415-b3050061085e**, which we received in the response. You can add a client identifier if you like.

   Sample request

   ```
   curl --location 'https://sandbox.crbcos.com/International/v1/payments' \
   --data '{
       "quoteId": "f710a42a-e03a-47b8-a415-b3050061085e"
   }'
   ```

2. A successful API call returns a JSON response with the payment ID in the `id` field and information about the payment. In this example, the payment ID is **f710a42a-e03a-47b8-a415-b3050061085e**.

Sample response

```
{
    "id": "f710a42a-e03a-47b8-a415-b3050061085e",
    "partnerId": "19222b81-0e1e-452d-a842-b2f1011c16f3",
    "productId": "57146944-b145-4326-884d-b2f700ecf688",
    "quoteId": "1a7c9fa9-10fb-4db8-a620-b3050061ca0e",
    "fromCurrency": "usd",
    "toCurrency": "gbp",
    "fromAmount": 500,
    "toAmount": 394,
    "accountNumber": "158560897007",
    "estimatedDeliveryDate": "2025-06-23T00:00:00-04:00",
    "originator": {
        "firstName": "Brandon",
        "lastName": "Sanderson",
        "fullName": "Brandon Sanderson",
        "address": "400 Kelby St",
        "city": "Fort Lee",
        "stateProvince": "NJ",
        "postalCode": "07024",
        "countryCode": "US",
        "entityType": "Individual"
    },
    "beneficiary": {
        "firstName": "Jon",
        "lastName": "Smith",
        "fullName": "Jon Smith",
        "birthDate": "2001-06-18T00:00:00-04:00",
        "address": "1 Street",
        "city": "Winfield",
        "postalCode": "GB12345",
        "countryCode": "GB",
        "entityType": "Individual"
    },
    "beneficiaryFi": {
        "bankName": "Bank UK",
        "bankCountryCode": "GB",
        "bankAddress": "1 Avenue",
        "bankAccountType": "Checking",
        "routingCodeType1": "SortCode",
        "routingCodeValue1": "123456789",
        "routingCodeType2": "ABA",
        "routingCodeValue2": "123456789",
        "bicSwift": "TGCLGB99",
        "iban": "GB33BUKB20201555555555"
    },
      "status": "Created",
      "purpose": "SRV",
      "paymentType": "Transfer",
      "direction": "Outbound",
      "priority": true,
      "feeAmount": 100,
      "feeCurrency": "usd",
      "source": "Api",
      "createdAt": "2025-06-23T01:56:22.770927-04:00",
      "lastModifiedAt": "2025-06-23T01:56:22.8905947-04:00",
      "limitsEligibleOn": "2025-06-23T01:56:22.770927-04:00"
  }
```

3 When the payment completes an `international.Payment.Sent` webhook event fires.

The payment ID (**f710a42a-e03a-47b8-a415-b3050061085e**) provided in the response body of the payment origination request (`id`) appears in the `details` object of the `international.Payment.Sent event` .

Sample International.Payment.Sent event

```json
{
  "id": "965a22ca-7005-4c88-b7c0-b0f1018381da",
  "eventName": "International.Payment.Sent",
  "status": "Pending",
  "partnerId": "19222b81-0e1e-452d-a842-b2f1011c16f3",
  "createdAt": "2024-01-08T18:30:52.247-05:00",
  "resources": [
    "international/v1/payments/f710a42a-e03a-47b8-a415-b3050061085e"
  ],
  "details": [
    {
      "paymentId": "f710a42a-e03a-47b8-a415-b3050061085e",
      "productId": "57146944-b145-4326-884d-b2f700ecf688",
      "quoteId": "f710a42a-e03a-47b8-a415-b3050061085e",
      "fromCurrency": "usd",
      "toCurrency": "gbp",
      "fromAmount": 500,
      "toAmount": 394,
      "feeAmount": "100",
      "accountNumber": "158560897007",
      "status": "Completed",
      "reason": null,
      "clientIdentifier": null,
      "priority": "True",
      "payerEntityType": "Individual",
      "companyName": null,
      "firstName": "Sara",
      "lastName": "Kim",
      "address": "250 Kuhn Highway",
      "city": "Grover",
      "stateProvince": "RR",
      "postalCode": "28073",
      "country": "GB",
      "birthDate": null
    }
  ]
}
```

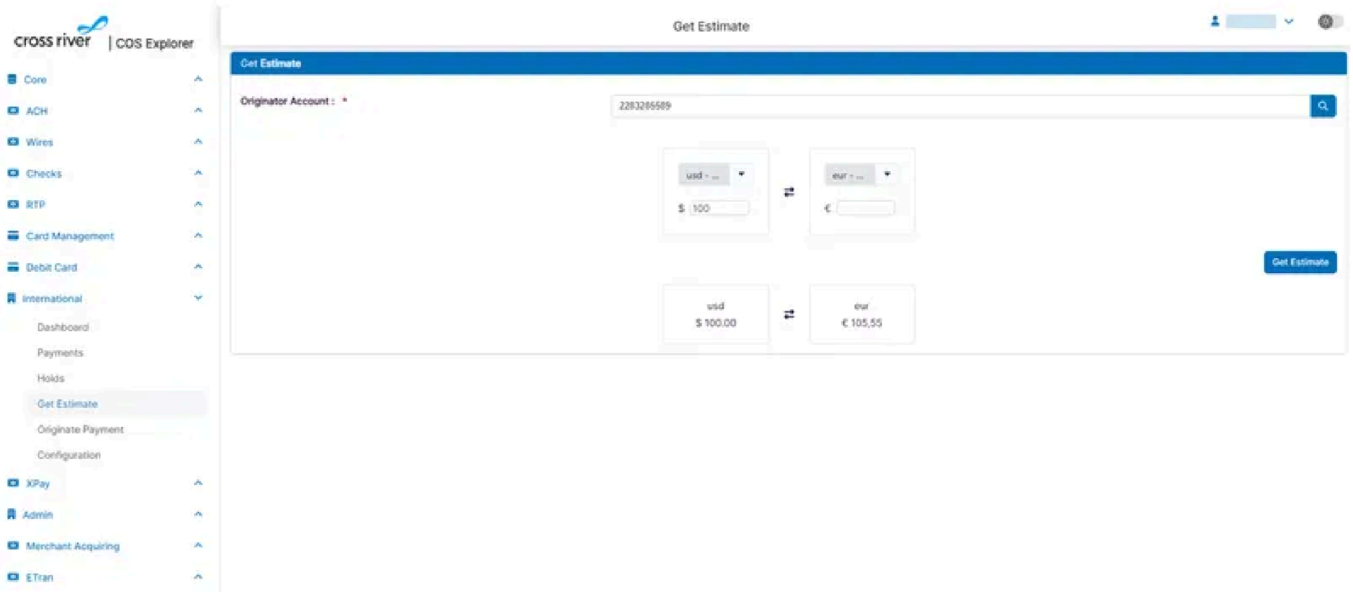
COS Explorer

Get an estimate

Before committing to sending an International Payment, you can get an estimated FX rate for your desired currency. This optional step includes Cross River's spread fee charge and does not require beneficiary information.

In the **International** tab:

1. Click **Get Estimate**. The **Get Estimate** page displays. Enter your account number (**Acct #**) in the search bar.
2. In the currency box enter *either* the dollar amount you want to send *or* the foreign currency amount you want the beneficiary receive. Do not enter both values.
3. Click **Get Estimate**. A currency estimate displays.



Originator Account : * 2283285589

usd - ... $ 100 ⇄ eur - ... €

Get Estimate

usd $ 100.00 ⇄ eur € 105,55

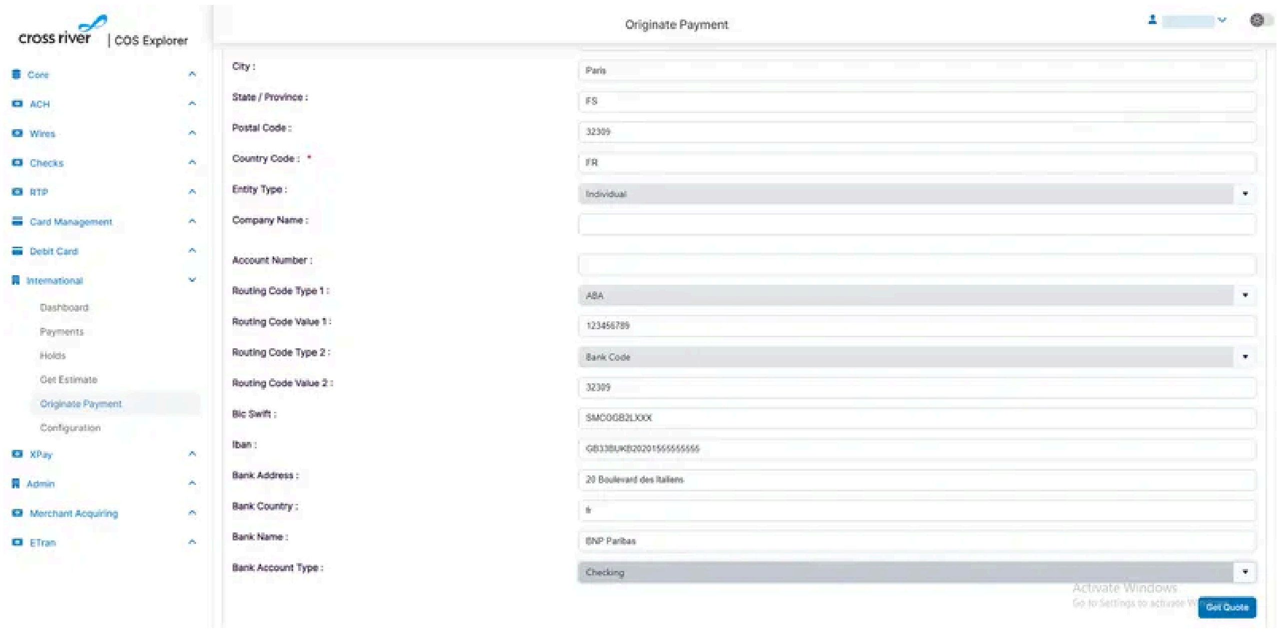
Originate an International Payment

1. Click **Originate Payment** and complete all required fields including beneficiary details and **Priority**, where **Yes** indicates **SWIFT** and **No** indicates local rail.

> The required information, such as, account number, IBAN, BIC/SWIFT, sort code, may vary depending on the destination country of the international payment.

2 Click **Get Quote**.

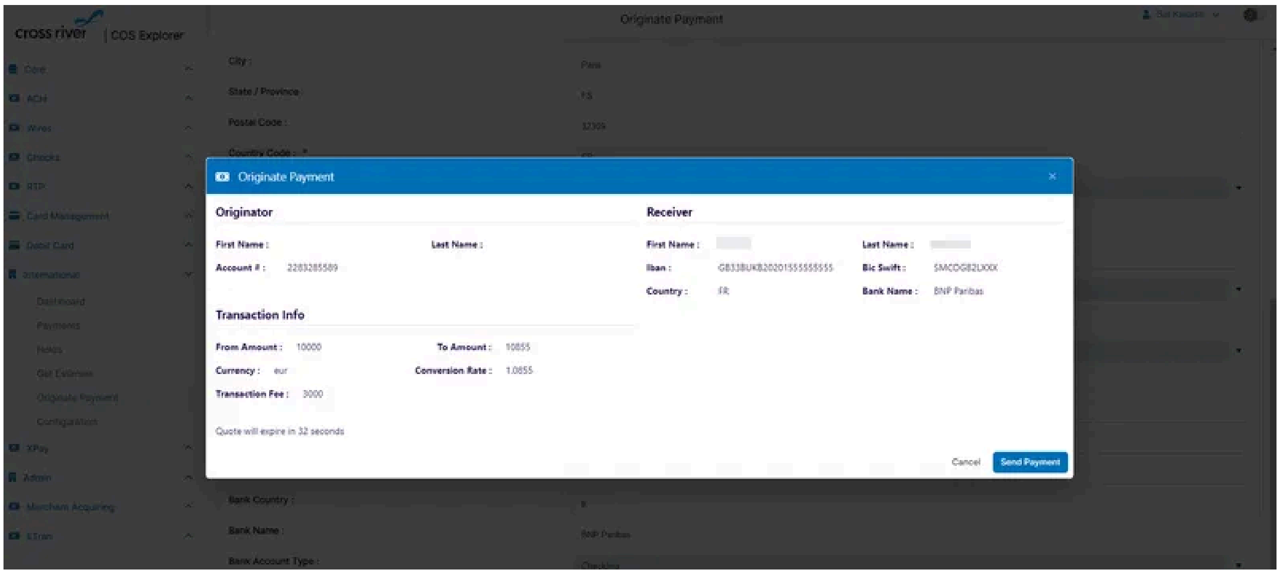
3 After clicking **Get Quote**, you will see the FX rate, spread fees, and transaction fees for the payment. The FX quote is valid for 1 minute before it updates. If you click **Send payment** after the quote expires you will need to go back to Step 2 and regenerate a quote.



Originate Payment

City :	Paris
State / Province :	FS
Postal Code :	32309
Country Code : *	FR
Entity Type :	Individual
Company Name :	
Account Number :	
Routing Code Type 1 :	ABA
Routing Code Value 1 :	123456789
Routing Code Type 2 :	Bank Code
Routing Code Value 2 :	32309
Bic Swift :	SMCOGB2LXXX
Iban :	GB33BUKB20201555555555
Bank Address :	20 Boulevard des Italiens
Bank Country :	fr
Bank Name :	BNP Paribas
Bank Account Type :	Checking

4 Click **Send Payment** to originate the payment.



Originate Payment

Originator

First Name :
Last Name :
Account # : 2283285589

Receiver

First Name : [redacted]
Last Name : [redacted]
Iban : GB33BUKB20201555555555
Bic Swift : SMCOGB2LXXX
Country : FR
Bank Name : BNP Paribas

Transaction Info

From Amount : 10000
To Amount : 10855
Currency : eur
Conversion Rate : 1.0855
Transaction Fee : 3000

Quote will expire in 32 seconds

5 To view your payment click **Payments**.

8. ACH

Our ACH tutorials explain how to use our APIs to:

- **Send an ACH payment**: Originate a standard or same-day ACH payment.
- **Send a client batch**: Send two or more payments in the same call.
- **Simulate inbound ACH payments**: Test the system with our simulation endpoints. You can also use this endpoint to fund a test account.

8.1. Send an ACH payment

In this tutorial, you'll learn how to

✅ Originate a push payment

✅ Monitor status of the payment

✅ Handle notifications of change to the outbound payment (rejected, returned, NOC)

> If you are new to APIs and how the work we recommend you visit **API basics**.

The tutorial uses these API endpoints:

API	Description
POST /ach/v1/payments	Originate the payment

> - Do not poll the APIs for status updates and reconciliation purposes

Before you begin

Make sure you have:

- **API credentials**
- Partner ID
- Account Number
- **Register** the following webhook events

Webhook	Description
`Ach.Payment.Sent`	Outbound payment has been transmitted to the Federal Reserve
`Ach.Return.Received`	A previously originated payment has been returned by the receiving bank
`Ach.Noc.Received`	A notification of change has been sent from the receiving bank regarding a previous origination
`Ach.Payment.Rejected`	The ACH payment was rejected

Originate the ACH payment

There are two types of ACH payments:

- **Push payment**
  When you transfer money to an account in another bank.
- **Pull payment**
  When you request funds from an account in another bank.

> Use a **validation** (prenote) if you want to verify and validate account details. A prenote simply means originating a payment of $0.00. If you don't receive an error response, the details are correct. This step is optional.

This tutorial covers push payments. A push payment can result in:

- The recipient receives the payment
- Cross River or ACH rejects the payment
- The recipient FI returns the payment
- The recipient receives the payment but ACH sends a Notification of Change.

Let's send $100 to Bob Smith who has an account at another bank. Since we are transferring the money to Bob's account the `transactionType` attribute must be **Push**.

To originate an ACH payment

1. Call `POST /ach/v1/payments`.

   The details of the push payment are defined by the **call attributes**.

   IMPORTANT
   We strongly recommend that you include an idempotency key in your request header to provide duplicate protection should the payment fail. Read more about **idempotency keys**.

   Money amounts in API calls and responses are written without a decimal point between the dollars and the cents.

   Curl

```
POST /ach/v1/payments
{
  "accountNumber": "2714035231",
  "receiver": {
    "routingNumber": "021000021",
    "accountNumber": "456789000",
    "accountType": "Checking",
    "name": "Bob Smith",
    "identification": "XYZ123",
  },
  "secCode": "WEB",
  "description": "Payment",
  "transactionType": "Push",
  "amount": 10000,
  "serviceType": "SameDay",
  "clientIdentifier": "21fe77da-e2f8-4475-9397-81a293d63b8x"
}
```

2. A successful API call returns a JSON response with the details of your originated payment.

JSON

```json
{
    "id": "b96b935a-4713-4aae-973b-aeee00f1a749",
    "accountNumber": "2714035231",
    "referenceId": "A2236S8S20FH",
    "paymentType": "Origination",
    "direction": "Outbound",
    "status": "Created",
    "source": "Api",
    "postingType": "Individual",
    "postingCode": "OK",
    "posting": "Pending",
    "originator": {
        "routingNumber": "021214891",
        "accountNumber": "2714035231",
        "accountType": "Checking",
        "name": "Cross River Bank",
        "identification": "021214891"
    },
    "receiver": {
        "routingNumber": "021000021",
        "accountNumber": "456789000",
        "accountType": "Checking",
        "name": "Bob Smith",
        "identification": "XYZ123"
    },
    "secCode": "WEB",
    "description": "Payment",
    "transactionType": "Push",
    "amount": 10000,
    "serviceType": "SameDay",
    "effectiveDate": "220811",
    "traceNumber": "021214898943562",
    "wasReturned": false,
    "wasCorrected": false,
    "holdDays": 0,
    "original": {
        "paymentId": "b96b935a-4713-4aae-973b-aeee00f1a749"
    },
    "createdAt": "2022-08-11T10:39:49.9996701-04:00",
    "partnerId": "1e5d3f04-ae24-4af6-9e30-aecf012b99dd",
    "productId": "cc62e17f-5912-483e-9e42-aed30112fbb6",
    "lastModifiedAt": "2022-08-11T10:39:49.9996701-04:00",
    "clientIdentifier": "21fe77da-e2f8-4475-9397-81a293d63b8x"
}
```

> The `status` attribute in the response indicates that your payment was created. It does not indicate that your payment was successful.

Payment confirmation

After originating the payment, its status changes to **Pending** or **Hold** for up to several hours.

- A **Pending** status lets you know that the payment request is created but has not been batched for release to the Federal Reserve. The batching process occurs several times a day.
- A **Hold** status indicates that the payment request is in review and has not yet been approved for release to the Federal Reserve.

Next, the payment moves to **Batched** status. If the payment is originated close to a weekend or bank holiday it may take a few days for the status to transition to **Batched**.

When the Federal Reserve accepts the payment the status changes to **processing**, which triggers the `Ach.Payment.Sent` event.

The payment ID provided in the response body of the payment origination request (`id`) appears in the `details` object of the `Ach.Payment.Sent` event. In this case: **b96b935a-4713-4aae-973b-aeee00f1a749**.

JSON

```json
{
  "id": "7c135bfa-234f-48d2-9d70-adc70135d15f",
  "eventName": "Ach.Payment.Sent",
  "status": "Pending",
  "partnerId": "30dee145-b6a2-4058-8dc3-ac4000dee91f",
  "createdAt": "2021-10-20T14:48:01.12-04:00",
  "resources": [
    "ach/v1/payments/b96b935a-4713-4aae-973b-aeee00f1a749"
  ],
  "details": [
    {
      "paymentId": "b96b935a-4713-4aae-973b-aeee00f1a749",
      "coreTransactionId": null,
      "memoPostId": "85b509f9-61f0-4af3-b64c-b04900de43da",
      "clientBatchId": null,
      "clientBatchSequence": null,
      "accountNumber": "2714035231",
      "postingCode": null,
      "clientIdentifier": null,
      "purpose": "ENTERED BY #60C3C9C8FD17BA0070A7FB4F#"
    }
    ]
}
```

Payment status webhook events

Rejected payment

There may be situations where a payment request is rejected. A payment request rejected after initially receiving a `success` response from `POST /ach/v1/payments` could be due to:

- **Technical Rejection**. The transaction was unable to post from the account being used for your request, and is covered in this tutorial.
  Example
  - You originate a push payment for an amount greater than the balance of the originator account
  - You originate a payment from an account with an active restriction

- **Manual Rejection**. The payment was rejected by the Cross River Operations or BSA/AML Teams. Contact Cross River directly for information on why the payment was rejected.

To simulate a technical rejection, originate a push payment for an amount that exceeds the originating account balance. Learn more about **simulations**.

Again, let's send $100 to Bob Smith who has an account at another bank. This time, the amount you are sending exceeds the amount in the originator account.

1 Call `POST /ach/v1/payments`.

2 Define the call attributes as above in To originate an ACH payment

Curl

```
{
  "accountNumber": "2674958042",
  "receiver": {
    "routingNumber": "021000021",
    "accountNumber": "456789000",
    "accountType": "Checking",
    "name": "Bob Smith",
    "identification": "XYZ123",
  },
  "secCode": "WEB",
  "description": "Payment",
  "transactionType": "Push",
  "amount": 10000,
  "serviceType": "SameDay",
  "clientIdentifier": "bd3e0315-5f29-4b1c-a004-1fcdd0b8bee1"
}
```

The following JSON response is returned:

JSON

```
{
    "id": "f868a125-22b8-4c7e-a8dd-aeee00f76ce7",
    "accountNumber": "2674958042",
    "referenceId": "A223084JDV79",
    "paymentType": "Origination",
    "direction": "Outbound",
    "status": "Created",
    "source": "Api",
    "postingType": "Individual",
    "postingCode": "OK",
    "posting": "Pending",
    "originator": {
        "routingNumber": "021214891",
        "accountNumber": "2674958042",
        "accountType": "Checking",
        "name": "Cross River Bank",
        "identification": "021214891"
    },
    "receiver": {
        "routingNumber": "021000021",
        "accountNumber": "456789000",
        "accountType": "Checking",
        "name": "Bob Smith",
        "identification": "XYZ123"
    },
    "secCode": "WEB",
    "description": "Payment",
    "transactionType": "Push",
    "amount": 10000,
    "serviceType": "SameDay",
    "effectiveDate": "220811",
    "traceNumber": "021214896697194",
    "wasReturned": false,
    "wasCorrected": false,
    "holdDays": 0,
    "original": {
        "paymentId": "f868a125-22b8-4c7e-a8dd-aeee00f76ce7"
    },
    "createdAt": "2022-08-11T11:00:50.8991331-04:00",
    "partnerId": "1e5d3f04-ae24-4af6-9e30-aecf012b99dd",
    "productId": "d5f3ce06-8550-413e-a2e1-aed1016d6eea",
    "lastModifiedAt": "2022-08-11T11:00:50.8991331-04:00",
    "clientIdentifier": "bd3e0315-5f29-4b1c-a004-1fcdd0b8bee1"
}
```

The `Ach.Payment.Rejected` event is triggered.

The `details` object in the `Ach.Payment.Rejected` event contains the payment ID from the response body (`id`). In this case: **f868a125-22b8-4c7e-a8dd-aeee00f76ce7**.

In the details object of the event, the `postingCode` tells you the **rejection code**, in this case **RES** (Account Restriction). If the payment needs to be reviewed manually, the `rejectionReason` gives you more information about the rejection.

Sample Ach.Payment.Rejected event

```
{
  "id": "496659ff-6034-480a-84af-aeee00f78ade",
  "eventName": "Ach.Payment.Rejected",
  "status": "Pending",
  "partnerId": "1e5d3f04-ae24-4af6-9e30-aecf012b99dd",
  "createdAt": "2022-08-11T11:01:16.483-04:00",
  "resources": [
    "ach/v1/payments/f868a125-22b8-4c7e-a8dd-aeee00f76ce7"
  ],
  "details": [
  {
      "paymentId": "f868a125-22b8-4c7e-a8dd-aeee00f76ce7",
      "coreTransactionId": null,
      "memoPostId": "517f24c5-3cf1-4168-b721-b04900e36f00",
      "clientBatchId": null,
      "clientBatchSequence": null,
      "accountNumber": "2674958042",
      "postingCode": "RES",
      "clientIdentifier": null,
      "purpose": "ENTERED BY #60C3C9C8FD17BA0070A7FB4F#",
      "rejectionReason": "Other"
    }
  ]
}
```

Returned payment

Sometimes an ACH payment is returned by the receiving bank. See **return codes** for a complete list of why a payment might be returned.

> **Example**
>
> The receiving bank can't find the specified receiver account number.

Most returns occur within 2 business days, but may take longer.

A payment return creates a new payment record with the `paymentType` set as **Return**. The new payment record is connected to the original payment record with the `previous.paymentId` field.

> **IMPORTANT**
> Once payment status transitions to **Complete**, the payment record is no longer updated, even if the payment is returned.

A payment returned from a receiving bank triggers the `ACH.Return.Received` webhook event.

Here's an example of an ACH return webhook event:

Sample Ach.Return.Received event

```
{
  "id": "d3ad6473-6690-44f1-a4a6-b04900d95419",
  "eventName": "Ach.Return.Received",
  "status": "Pending",
  "partnerId": "1e5d3f04-ae24-4af6-9e30-aecf012b99dd",
  "createdAt": "2023-07-24T09:11:16.133-04:00",
  "resources": [
    "ach/v1/payments/fb05013b-eaba-439e-8c6b-b04900d8f805"
  ],
  "details": [
    {
      "paymentId": "fb05013b-eaba-439e-8c6b-b04900d8f805",
      "coreTransactionId": "28633d37-f441-496a-ab07-b04900d93e75",
      "originalPaymentId": "1a1a9919-33b5-4933-9b9f-b034011462df",
      "accountNumber": "2151546989",
      "traceNumber": "021000021754553",
      "reasonCode": "R01",
      "reasonData": ""
    }
  ]
}
```

The event shows reasonCode **R01** (Insufficient Funds). The `originalPaymentId` field shows the ID of the original outbound payment.

Notification of Change (NOC)

There are times when Cross River receives an ACH notification of change (NOC) related to an outbound payment. When you create an API call, a NOC is referred to as a **Correction** in the `paymentType` attribute. Unlike an ACH return, a NOC indicates that the payment you previously originated:

- Posted to the receiver account
- Contained an error (such as an incorrect routing number)
  See **ACH correction codes** for a complete list

Register for the `Ach.Noc.Received` webhook event to be notified of an inbound NOC.

Here's an example of a NOC webhook event:

Sample Ach.Noc.Received event

```
{
  "id": "43d767a8-6b3a-4b5e-9fe6-b05401215333",
  "eventName": "Ach.Noc.Received",
  "status": "Pending",
  "partnerId": "1e5d3f04-ae24-4af6-9e30-aecf012b99dd",
  "createdAt": "2023-08-04T13:33:24.01-04:00",
  "resources": [
    "ach/v1/payments/197021dd-4b47-47b3-a75e-b0540120f5f2"
  ],
  "details": [
    {
      "paymentId": "197021dd-4b47-47b3-a75e-b0540120f5f2",
      "coreTransactionId": "578f36cd-f8e8-47ba-afa8-b05401213c5f",
      "originalPaymentId": "2cc0a0dd-6d7e-4ad9-9d3e-b0540105ba19",
      "accountNumber": "2696592019",
      "traceNumber": "021200333650075",
      "reasonCode": "C02",
      "reasonData": "026009593"
    }
  ]
}
```

The `reasonCode` attribute in the response is **C02**. This indicates an Incorrect Routing Number. The `reasonData` attribute is the correct receiver routing number.

If you receive a NOC notification, update your internal records with the information you receive in the `reasonData` attribute.

8.2. Send a client batch

In this tutorial, you'll learn how to

✅ Register the relevant webhooks

✅ Submit a client batch of payments

✅ Cancel the batch

> The tutorial assumes you have a knowledge of APIs and how they work. For more information on sending API see **API basics** for more details.

The tutorial uses these API endpoints:

API	Description
POST /ach/v1/client-batches	Submits 2 or more payments from the same Cross River account number as a batch
POST /ach/v1/client-batches/{id}/cancel	Cancels all the payments in the batch that have not yet been completely processed

> **IMPORTANT**
>
> - Do not poll the APIs for status updates and reconciliation purposes.
> - Incorporate webhooks into the payment reconciliation process.

The tutorial uses these webhooks. See all ACH-related **webhook events**.

Webhook	Description
`Ach.Batch.Imported`	Notifies you that the client batch has been imported
`Ach.Batch.Canceled`	Notifies you that all pending or on-hold payments in the client batch are canceled

Before you begin

Make sure you have:

- **API credentials**
- Partner ID
- Account Number

Register relevant webhook events

To receive the webhook events for this tutorial both partner accounts need to **register** each specific webhook event type. Once you are registered, the event objects are sent to the registered URLs.

The event object contains a list of resource identifiers used to download details on each event.

Submit a client batch

Use a client batch when you have several transfers to or from the same account. For example, for payroll. In this tutorial the example account number is **2207975570**. The following 5 different actions appear in the sample code.

Name	Account number	Transaction type	Service type	Amount	Routing number	Account type
John Wick	12345678	Push	SameDay	113	021000021	Checking
Cleveland Brown	2342123458	Pull	Standard	243591	021000021	Checking
Matt Cauthon	2342123458	Pull	Standard	50	021000021	Checking
Perrin Aybara	2342123458	Push	Standard	2750	021000021	Checking
Elayne Trakand	2342123458	Pull	SameDay	15000	021000021	Checking

1 Call `POST ach/v1/client-batches`. The details of each payment are defined by the call attributes. Values for these attributes must be provided for *each* payment in the batch.

> **IMPORTANT**
> We strongly recommend that you include an idempotency key in your request header to provide duplicate protection should the payment fail. Read more about **idempotency keys**.

> Money amounts in API calls and responses are written without a decimal point between the dollars and the cents.

Curl

```
POST ach/v1/client-batches
{
{
  "payments": [
    {
      "accountNumber": "2207975570",
      "receiver": {
        "routingNumber": "021000021",
        "accountNumber": "12345678",
        "accountType": "Checking",
        "name": "John Wick"
      },
      "secCode": "PPD",
      "description": "string",
      "transactionType": "Push",
      "amount": 113,
      "serviceType": "SameDay"
      },
{
      "accountNumber": "2207975570",
      "receiver": {
        "routingNumber": "021000021",
        "accountNumber": "2342123458",
        "accountType": "Checking",
        "name": "Cleveland Brown"
      },
      "secCode": "PPD",
      "description": "string",
      "transactionType": "Pull",
      "amount": 243591,
      "serviceType": "Standard"
    },
{
      "accountNumber": "2207975570",
      "receiver": {
        "routingNumber": "021000021",
        "accountNumber": "2342123458",
        "accountType": "Checking",
        "name": "Matt Cauthon"
      },
      "secCode": "PPD",
      "description": "string",
      "transactionType": "Pull",
      "amount": 50,
      "serviceType": "Standard"
    },
  {
        "accountNumber": "2207975570",
        "receiver": {
          "routingNumber": "021000021",
          "accountNumber": "2342123458",
          "accountType": "Checking",
          "name": "Perrin Aybara"
        },
        "secCode": "PPD",
        "description": "string",
        "transactionType": "Push",
        "amount": 2750,
        "serviceType": "Standard"
    },
  {
        "accountNumber": "2207975570",
        "receiver": {
          "routingNumber": "021000021",
          "accountNumber": "2342123458",
          "accountType": "Checking",
          "name": "Elayne Trakand"
        },
        "secCode": "PPD",
        "description": "string",
        "transactionType": "Pull",
        "amount": 15000,
        "serviceType": "SameDay"
    }
  ]
}
```

2 A successful API call returns a JSON response with the details of the client batch record that contains the client batch ID (in our case, **fcb7188f-e051-4f5b-addd-afaa00eadb3a**) and individual `paymentIdentifiers`. The identifiers appear in the response in the order that the payments were listed in the request. So in our case, the payment to **John Wick** is assigned `paymentIdentifier` **04793f3-13d9-4d52-b57e-afaa00eb6c74**, the payment from **Cleveland Brown** corresponds to **dbe36253-06d1-462a-9117-afaa00eb6c74**, and so on.

Sample client batch response

```json
{
  "id": "fcb7188f-e051-4f5b-addd-afaa00eadb3a",
  "referenceId": "CB0461724AAWT",
  "status": "Processing",
  "accountNumber": "2207975570",
  "paymentCount": 5,
  "debitTotal": 258641,
  "creditTotal": 2863,
  "importCount": 0,
  "productId": "cc62e17f-5912-483e-9e42-aed30112fbb6",
  "partnerId": "1e5d3f04-ae24-4af6-9e30-aecf012b99dd",
  "createdAt": "2023-02-15T09:15:05.1486312-05:00",
  "lastModifiedAt": "2023-02-15T09:15:05.1486312-05:00",
  "paymentIdentifiers": [
    "f04793f3-13d9-4d52-b57e-afaa00eb6c74",
    "dbe36253-06d1-462a-9117-afaa00eb6c74",
    "1e1a08c7-1e60-452b-b22b-afaa00eb6c74",
    "a0bc6816-e85e-4ba6-bf9a-afaa00eb6c74",
    "6ed9ea59-2522-4550-8e5d-afaa00eb6c74"
    ]
}
```

The `status` attribute in the response indicates that your batch is **processing**. It does not indicate that the batch was imported successfully.

The `importCount` attribute in the response indicates how many payments were imported (meaning originated) out of the total at the time of the response.

When the `status` of the client batch changes to **imported** the `Ach.Batch.Imported` webhook event fires.

> The `status` in the webhook refers to the webhook status, not the client batch status.

Ach.Batch.Imported webhook

```
{
  "id": "b060d47c-7ab5-4a34-8803-afaa00ec617e",
  "eventName": "Ach.Batch.Imported",
  "status": "Pending",
  "partnerId": "1e5d3f04-ae24-4af6-9e30-aecf012b99dd",
  "createdAt": "2023-02-15T09:20:38.18-05:00",
  "resources": [
    "ach/v1/client-batches/fcb7188f-e051-4f5b-addd-afaa00eadb3a"
  ],
  "details": [
    {
      "clientBatchId": "fcb7188f-e051-4f5b-addd-afaa00eadb3a",
      "accountNumber": "2207975570"
    }
  ]
}
```

For example, when the Federal Reserve accepts the payment the status changes to **processing**, which triggers the `Ach.Payment.Sent` event. Every status change from this point on triggers the event until the payment posts to the receiving account (status: **complete**).

Cancel a payment in the batch

Sometimes it's most efficient to cancel a client batch before all the payments have imported rather than try to cancel a large number of payments individually. Call the cancel batch API to cancel any pending or on-hold payments within a batch.

Call `POST /v1/client-batches/{id}/cancel` to cancel the batch. Use the client batch ID: the id value (in our case, **fcb7188f-e051-4f5b-addd-afaa00eadb3a**) received in the response to the `POST ach/v1/client-batches` call.

Curl

```
curl -X POST
--header 'Accept: application/json'
--header 'Authorization: Bearer '<token>'

https://sandbox.crbcos.com/ACH/v1/client-batches/fcb7188f-e051-4f5b-addd-afaa
```

A successful API call returns a JSON response with the details of the client batch record that contains the individual payment identifiers of all payments that were canceled. (Payments with a **Completed** status cannot be canceled.) The status in the response is **Canceling**. As with the client batch request above, each identifier returned corresponds to the order in which the payments were presented in the request.

JSON

```
{
  "id": "fc0d501d-c36b-45b4-a0d5-afaa00eb6c74",
  "referenceId": "CB0460UJR6DNA",
  "status": "Canceling",
  "accountNumber": "2207975570",
  "paymentCount": 5,
  "debitTotal": 258641,
  "creditTotal": 2863,
  "importCount": 5,
  "productId": "cc62e17f-5912-483e-9e42-aed30112fbb6",
  "partnerId": "1e5d3f04-ae24-4af6-9e30-aecf012b99dd",
  "createdAt": "2023-02-15T09:17:09.077-05:00",
  "importedAt": "2023-02-15T09:17:19.123-05:00",
  "lastModifiedAt": "2023-02-15T09:17:26.8406542-05:00",
  "paymentIdentifiers": [
    "6ed9ea59-2522-4550-8e5d-afaa00eb6c74",
    "dbe36253-06d1-462a-9117-afaa00eb6c74",
    "1e1a08c7-1e60-452b-b22b-afaa00eb6c74",
    "f04793f3-13d9-4d52-b57e-afaa00eb6c74",
    "a0bc6816-e85e-4ba6-bf9a-afaa00eb6c74"
  ]
}
```

When the cancellation is complete, it triggers the `Ach.Batch.Canceled` webhook event. Note the client batch ID (in this case , **fcb7188f-e051-4f5b-addd-afaa00eadb3a**) corresponds to the client batch ID used in the cancellation request.

JSON

```json
{
  "id": "a6593026-ea2c-42b1-ae57-afaa00ec84b2",
  "eventName": "Ach.Batch.Canceled",
  "status": "Pending",
  "partnerId": "1e5d3f04-ae24-4af6-9e30-aecf012b99dd",
  "createdAt": "2023-02-15T09:21:08.21-05:00",
  "resources": [
    "ach/v1/client-batches/0d4cf4dd-bcbd-4eb7-878d-afaa00ec59c0"
  ],
  "details": [
    {
      "clientBatchId": "fcb7188f-e051-4f5b-addd-afaa00eadb3a",
      "accountNumber": "2207975570"
    }
  ]
}
```

8.3. Simulate inbound ACH payments

Simulations allow testing certain inbound payment flows in our sandbox environment. They can either be triggered explicitly using the API endpoints outlined below, or in the case of returns and corrections, automatically once an outbound payment completes. Automatic simulations are triggered by convention using the payment `purpose` field.

> **IMPORTANT**
> You can only simulate a return or NOC for a payment once it has updated to a status of **Complete**.

You can test outbound payments using the payment origination API endpoint. The sandbox environment automatically takes the payment through the various statuses until it is **Complete**. Typically this process takes up to several minutes from start to finish.

Where's my simulated payment?

It is important to note that simulation requests are queued and processed asynchronously on a schedule. It typically takes a few minutes before they show up as new payment records.

Inbound Originations

Inbound originations are payments that are originated at another financial institution and sent to your Cross River account.

These payments can have a `transactionType` of either *Push* (funds are being sent to your Cross River account) or *Pull* (funds are being taken from your Cross River account).

To simulate an inbound origination, you would manually trigger it using the simulated inbound originations endpoint. A sample of this request is displayed below.

Curl

```
POST /v1/payments/simulated-inbound-originations
{
  "originatorRoutingNumber": "021000021",
  "originatorName": "Tom Smith",
  "originatorIdentification": "99999999",
  "receiverAccountNumber": "1234567890",
  "receiverAccountType": "Checking",
  "receiverName": "John Smith",
  "receiverIdentification": "INV123",
  "secCode": "PPD",
  "description": "Testing",
  "transactionType": "Push",
  "amount": 1000,
  "serviceType": "SameDay",
}
```

The `receiverAccountNumber` is the number of the Cross River account receiving the payment. See **Send an ACH payment** for additional information.

Returns

You can simulate ACH returns manually using the simulation endpoint `POST /v1/payments/simulated-inbound-returns`. Populate the desired **return code** along with the ID of the outbound payment that you're trying to simulate a return for. A sample of this request is displayed below.

Curl

```
POST /v1/payments/simulated-inbound-returns
{
  "returnCode": "R01",
  "previousPaymentId": "00000000-0000-0000-0000-000000000000"
}
```

Alternatively, simulate a return automatically when originating an outbound payment. To do this, you'll need to use the `purpose` field of the payment request. Populate the `purpose` field with **RETURN_RXX**, where **XX** is the return code you wish to test. A sample of this request is displayed below.

Curl

```
POST /v1/payments
{
  "accountNumber": "2342123458",
  "receiver": {
    "routingNumber": "021200339",
    "accountNumber": "654321987",
    "accountType": "Checking",
    "name": "Glenn Quagmire",
    "identification": "XYZ123"
  },
  "secCode": "PPD",
  "description": "MembFee",
  "transactionType": "Pull",
  "amount": 4999,
  "serviceType": "Standard",
  "purpose": "RETURN_R01"
}
```

Contested returns have to be anticipated or approved for acceptance. In that case, only the initial return event is listed as returned, and a subsequent webhook is applied. The dishonoring of the return relays the funds back to the returning source and no webhook launches for those or any other rejecting/dishonoring attempts.

Without your direct consent, no returns of any kind occur while outside of the 24-hour (Corporate return) or 60-day (Consumer return) time frame . Transactions received outside of both scenarios are subject to review and processing. For any claims issued to the Cross River ACH Operations team, a notice is provided to the Originating party for Proof of Authorization or any other corroborating documents that warrant the authorization of the debit entry. Your RM will also be advised of any matters for awareness.

Corrections

Corrections (notifications of change) can be simulated manually using the simulation endpoint. You'll need to populate the desired correction code along with the ID of the outbound payment that you're truing to simulate a correction for. A sample of this request is displayed below.

Curl

```
POST /v1/payments/simulated-inbound-corrections
{
  "changeCode": "C01",
  "correctedData": "12345",
  "previousPaymentId": "00000000-0000-0000-0000-000000000000"
}
```

Alternatively, corrections can be simulated automatically when originating an outbound *payment*. To do this, you'll need to use the purpose field of the payment request. Populate the `purpose` field with **NOC_CXX**, where **XX** is the correction code you wish to test. A sample of this request is displayed below.

Curl

```
POST /v1/payments
{
  "accountNumber": "2342123458",
  "receiver": {
    "routingNumber": "021200339",
    "accountNumber": "654321987",
    "accountType": "Checking",
    "name": "Glenn Quagmire",
    "identification": "XYZ123"
  },
  "secCode": "PPD",
  "description": "MembFee",
  "transactionType": "Pull",
  "amount": 4999,
  "serviceType": "Standard"
  },
  "purpose": "NOC_C01"
}
```

The example above illustrates an outbound pull payment where an incorrect DFI account number correction (C01) will be automatically generated by the Cross River system after the outbound payment is complete.

Additional Information

Since returns and NOCs can only be simulated after an outbound payment reaches a status of **Complete**, the outbound payment's service type will affect the timing of your simulations. An outbound payment with a *standard* service type would only allow you to simulate a return the following day via the simulation endpoint. If the simulation was done using the `purpose` field of the outbound payment, then you'd automatically receive the simulated inbound payment two business days after the payment is completed. Some SEC codes will also encounter this timing scenario, such as IATs which are restricted to a *standard* service type.

This is generally why you would encounter any scenarios where you’ve originated a payment but the status hasn’t changed to Complete in over 24 business hours. If you originated any payments yesterday, you can use the simulation endpoints to simulate any return code you want.

Our ACH domain in Sandbox is configured to simulate the bank’s processes around ACH origination, which means that generally no human intervention is needed to action a payment in order for it to be processed and moved to a status of Complete. We do not mark any payments as paid; once you originate a payment, assuming all systemic validations pass then Sandbox will simulate the payment being sent to the Fed and also simulate receiving an acknowledgment file from the Fed.

When a payment request is submitted to COS, the bank can reject the payment for various reasons. For example, a payment which was on hold for compliance reasons was rejected because the partner was unable to provide additional payment-related information.

Payments which are manually rejected by someone in Cross River do not include any details for the rejection. Payments which are systemically rejected will contain the reason within the payment details. For example, if a payment was rejected because the originator Cross River account had insufficient funds then you would see **NSF** as the `postingCode` within the payment details.

9. Wires

Our ACH tutorials explain how to use our APIs to:

- **Send a wire payment**: Originate a wire payment.
- **Send a drawdown request**: Request a wire payment from an external account.
- **Respond to a drawdown request**: Respond to a request for payment sent from an external account.
- **Simulate an inbound wire**: Test the system with our simulation endpoints. You can also use this endpoint to fund a test account.

9.1. Send a wire payment

In this tutorial, you'll learn how to:

✅ Originate an outbound wire transfer

✅ Cancel an outbound wire transfer

✅ Originate an outbound international wire transfer

> If you are new to wires we recommend you read the **wires overview** before starting this tutorial.
>
> The tutorial assumes you have a knowledge of APIs and how they work. For more information on sending API calls, see the **API basics**.

The tutorial uses these API endpoints:

API	Description
POST /wires/v1/payments	Sends an outbound wire transfer
POST /wires/v1/payments/{id}/cancel	Cancels a wire transfe

> **IMPORTANT**
>
> - Do not poll the APIs for status updates and reconciliation purposes.
> - Incorporate webhooks into the payment reconciliation process.

The tutorial uses these webhooks:

Webhook	Description
`Wire.Payment.Sent`	Outbound wire has been transmitted to the Federal Reserve and has been successfully acknowledged. IMAD number is now available.
`Wire.Payment.Received`	Inbound wire payment received successfully.
`Wire.Payment.Rejected`	Outbound wire could not be processed due to compliance reasons or was rejected by the Federal Reserve.
`Wire.Payment.Canceled`	Outbound wire transfer was canceled

Before you begin

Make sure you have

- **API credentials**
- Partner ID
- Master account number

Register the relevant webhook events

To receive the webhook events for this tutorial you need to **register** each specific webhook event type. Once you are registered, the event objects are sent to the registered URLs.

The event object contains a list of resource identifiers used to download details of each event.

Originate a outbound payment

Call `POST /wires/v1/payments` to send the $100 wire transfer. In this example, we supply values for the following required attributes:

- Account number: **2342123458** (originator's account number)
- **Business function code**: **CTR**

- Receiver routing number: **026009593** (Fedwire routing number)
- Beneficiary information:
  - ID: **D**
  - Identifier: **4289341024** (beneficiary's account number)
  - Name: **Miguel Nelson**
- Amount: **10000** (amount in USD with no decimal point between the dollars and cents)
- Purpose: **Consulting Fee**

IMPORTANT
We strongly recommend that you include an **idempotency key** in your request header to provide duplicate protection should the payment fail.

Sample request

```
POST /wires/v1/payments
{
  "accountNumber": "2342123458",
  "businessFunctionCode": "CTR",
  "receiverRoutingNumber": "026009593",
  "beneficiary": {
    "idCode": "D",
    "identifier": "4289341024",
    "name": "Miguel Nelson",
    "address1": "250 Kuhn Highway",
    "address2": "Grover, NC 28073"
  },
  "beneficiaryReference": "Invoice A523",
  "amount": 10000,
  "purpose": "Consulting Fee"
}
```

A successful API call returns a JSON response with the details of your originated payment: Acme Co, as the `originator`, sent a wire transfer `direction: Outbound`, in the amount of $100 reflected in `amount` as `10000`, to Miguel Nelson, the `receiver`, whose account is at `BK AMER NYC`.

Sample response

```
{
  "id": "30423521-52e2-4329-b8f6-ada3011806f1",
  "accountNumber": "2342123458",
  "referenceId": "W21025MV1WA",
  "direction": "Outbound",
  "paymentType": "Transfer",
  "source": "Api",
  "status": "Created",
  "posting": "Pending",
  "amount": 10000,
  "currency": "usd",
  "purpose": "Consulting Fee",
  "businessFunctionCode": "CTR",
  "typeCode": "1000",
  "senderRoutingNumber": "021214891",
  "senderName": "Cross River Bank",
  "senderReference": "W21025MV1WA",
  "receiverRoutingNumber": "026009593",
  "receiverName": "BK AMER NYC       ",
  "originatingFi": {
    "idCode": "F",
    "identifier": "021214891",
    "name": "Cross River Bank",
    "address1": "885 Teaneck Rd",
    "address2": "Teaneck NJ 07666",
    "address3": "US"
  },
  "originator": {
    "idCode": "D",
    "identifier": "2342123458",
    "name": "Acme Co",
    "address1": "400 Business Street",
    "address2": "New York NY 10025"
  },
  "beneficiary": {
    "idCode": "D",
    "identifier": "4289341024",
    "name": "Miguel Nelson",
    "address1": "250 Kuhn Highway",
    "address2": "Grover, NC 28073"
  },
  "beneficiaryReference": "Invoice A523",
  "wasReversed": false,
  "isInternational": false,
  "createdAt": "2021-01-25T17:13:27.6503996-05:00",
    "effectiveDate": "2021-01-25T00:00:00-05:00",
    "originalPaymentId": "30423521-52e2-4329-b8f6-ada3011806f1",
    "partnerId": "bf1baac0-1ae9-45ed-ade6-55baf1ae19a6",
    "productId": "3941dec4-bd4b-4f3e-af7a-bc6d86b8a0dc",
    "lastModifiedAt": "2021-01-25T17:13:27.6573998-05:00",
    "postingCode": "OK"
  }
```

Confirm payment

After originating the payment, its status changes to **Pending** or **Hold** for up to several hours.

- A **Pending** status lets you know that the payment request is created but has not been batched for release to the Federal Reserve. The batching process occurs several times a day.
- A **Hold** status indicates that the payment request is in review and has not yet been approved for release to the Federal Reserve.

Acknowledgement of the payment by the Fed triggers the `Wire.Payment.Sent` event. This indicates successful acceptance of the payment by the Federal Reserve and the availability of an IMAD.

The payment ID provided in the response body of the payment origination request (`id`) appears in the `resources` object of the `wire.Payment.Sent` event for ease of identification. In this case the payment ID is **30423521-52e2-4329-b8f6-ada3011806f1**.

Sample Wire.Payment.Sent event

```
{
  "id": "dab167cf-3d77-45ed-ad85-ada30118fc11",
  "eventName": "Wire.Payment.Sent",
  "status": "Pending",
  "partnerId": "30dee145-b6a2-4058-8dc3-ac4000dee91f",
  "createdAt": "2021-01-25T18:03:01.887-04:00",
  "resources": [
    "wires/v1/payments/30423521-52e2-4329-b8f6-ada3011806f1"
  ],
  "details": []
}
```

Cancel an outbound payment

You can cancel a wire transfer if it has not reached batch status, which tells you that the payment is in the process of being released to the Fed. Specifically, you can cancel a wire transfer with a **pending** or **hold** status.

Call `POST /wires/v1/payments/{id}/cancel` . For the `id` attribute use the payment ID returned in the response to `POST /wires/v1/payments`. In our case, we'll use **30423521-52e2-4329-b8f6-ada3011806f1**.

Sample request

```
curl -X POST
--header 'Accept: application/json'
--header 'Authorization: Bearer <token>'
'https://sandbox.crbcos.com/Wires/v1/payments/<p>Sample request</p>curl -X GE
--header 'Accept: application/json'
--header 'Authorization: Bearer <token>'
'https://sandbox.crbcos.com/Wires/v1/payments/4d4c60b2-e073-409d-bc94-aff0000
```

A successful API call returns a JSON response with the details of your canceled payment and shows a status value of **Canceled**.

Sample response

```
{
  "id": "30423521-52e2-4329-b8f6-ada3011806f1",
  "accountNumber": "2342123458",
  "referenceId": "W21025MV1WA",
  "direction": "Outbound",
  "paymentType": "Transfer",
  "source": "Api",
  "status": "Canceled",
  "posting": "Posted",
  "amount": 10000,
  "currency": "usd",
  "purpose": "Consulting Fee",
  "businessFunctionCode": "CTR",
  "typeCode": "1000",
  "senderRoutingNumber": "021214891",
  "senderName": "Cross River Bank",
  "senderReference": "W21025MV1WA",
  "receiverRoutingNumber": "026009593",
  "receiverName": "BK AMER NYC       ",
  "originatingFi": {
    "idCode": "F",
    "identifier": "021214891",
    "name": "Cross River Bank",
    "address1": "885 Teaneck Rd",
    "address2": "Teaneck NJ 07666",
    "address3": "US"
  },
  "originator": {
    "idCode": "D",
    "identifier": "2342123458",
    "name": "Acme Co",
    "address1": "400 Business Street",
    "address2": "New York NY 10025"
  },
  "beneficiary": {
    "idCode": "D",
    "identifier": "4289341024",
    "name": "Miguel Nelson",
    "address1": "250 Kuhn Highway",
    "address2": "Grover, NC 28073"
  },
  "beneficiaryReference": "Invoice A523",
  "wasReversed": false,
  "isInternational": false,
  "createdAt": "2021-01-25T17:13:27.6503996-05:00",
    "canceledAt":"2021-01-25T19:04:10.7921285-05:00",
    "effectiveDate": "2021-01-25T00:00:00-05:00",
    "originalPaymentId": "30423521-52e2-4329-b8f6-ada3011806f1",
    "partnerId": "bf1baac0-1ae9-45ed-ade6-55baf1ae19a6",
    "productId": "3941dec4-bd4b-4f3e-af7a-bc6d86b8a0dc",
    "lastModifiedAt": "2021-01-25T17:13:27.6573998-05:00",
    "postingCode": "OK"
  }
```

The `Wire.Payment.Canceled` event also communicates the payment cancellation, just like the response to calling `POST /wires/v1/payments/{id}/cancel`.

The payment ID provided in the response body of the payment origination request (`id`) appears in the `details` object of the `wire.Payment.Canceled` event for ease of identification, along with other transfer details. In this case the payment ID is **30423521-52e2-4329-b8f6-ada3011806f1**. Note that the **Pending** status refers to the webhook and not the payment.

Sample Wire.Payment.Canceled event

```json
{
  "id": "ff1b5fa5-ae3d-48eb-b389-b12901195195",
  "eventName": "Wire.Payment.Canceled",
  "status": "Pending",
  "partnerId": "1e5d3f04-ae24-4af6-9e30-aecf012b99dd",
  "createdAt": "2021-03-04T12:04:15.003-05:00",
  "resources": [
    "wires/v1/payments/30423521-52e2-4329-b8f6-ada3011806f1"
  ],
  "details": [
    {
      "paymentId": "30423521-52e2-4329-b8f6-ada3011806f1",
      "accountNumber": "2342123458",
      "direction": "Outbound",
      "imad": null,
      "omad": null,
      "paymentType": "Transfer",
      "purpose": "Consulting Fee",
      "amount": "10000",
      "clientIdentifier": null,
      "originatingFiName": "34115033",
      "originatingFiIdentifier": "021214891",
      "originatorName": "Acme Co",
      "originatorIdentifier": "2342123458",
      "originatorAddress1": "400 Business Street",
      "originatorAddress2": "",
      "originatorAddress3": "New York NY 10025",
      "beneficiaryFiName": null,
      "beneficiaryFiIdentifier": null,
      "beneficiaryName": "Miguel Nelson",
      "beneficiaryIdentifier": "4289341024",
      "beneficiaryAddress1": "250 Kuhn Highway",
      "beneficiaryAddress2": "Grover, NC 28073"
      "beneficiaryAddress3": null,
      "beneficiaryReference": "Invoice A523",
      "senderReference": "W21025MV1WA",
      "originatorToBeneficiary1": null,
      "originatorToBeneficiary2": null,
      "originatorToBeneficiary3": null,
      "originatorToBeneficiary4": null,
      "coreTransactionId": "fa7ae94e-0fc0-4c4f-9680-b129011945d8"
    }
  ]
}
```

9.2. Send a drawdown request

An outbound wire **drawdown request**, also called a reverse wire, asks a someone to send funds back to you as the requester.

In this tutorial, you'll learn how to:

✅ Send a drawdown request

> If you are new to wires we recommend you read the **wires overview** before starting this tutorial.
>
> The tutorial assumes you have a knowledge of APIs and how they work. For more information on sending API calls, see the **API basics**.

For this tutorial you'll need to understand these terms. These definitions are specific to this tutorial. The terms might be used slightly differently for a drawdown response.

Term	Description
Originator	The entity requesting the payment
Beneficiary	The entity receiving the payment
Originating financial institution	The financial institution wherethe payment request comes from
Intermediary financial institution	A financial institution that a payment is routed through (required for international wires, otherwise not always needed)
Beneficiary financial institution	The financial institution that receives the payment request
Drawdown Credit Account NumberSending financial institution	The financial institution that sends the payment request 9 digit routing/ABA number of the originating financial institution
Receiveringfinancial institutionRoutingNumber	The routing number of the financial institution that receives the payment request
Drawdown request	A request for someone to send a payment by wire

The tutorial uses these API endpoints:

API	Description
POST /wires/v1/payments/corporate-drawdown-requests	Requests an inbound wire transfer

The tutorial uses these webhooks.

Webhook	Description
`Wire.Payment.Sent`	Outbound wire has been transmitted to the Federal Reserve and has been successfully acknowledged. IMAD number is now available.
`Wire.Payment.Received`	Inbound wire payment received successfully.
`Wire.Payment.Rejected`	Outbound wire could not be processed due to compliance reasons or was rejected by the Federal Reserve.

Before you begin

Make sure you have:

- **API credentials**
- Account number of the account requesting the payment
- Receiver routing number

Register the relevant webhook events

To receive the webhook events for this tutorial both partner accounts need to **register** each specific webhook event type. Once you are registered, the event objects are sent to the registered URLs.

The event object contains a list of resource identifiers used to download details on each event.

Request a wire payment

When you initiate an outbound wire drawdown request (DRC 1031), the request is sent to the specified bank where the account responsible for the payout is held. The bank's response includes the payment status and other relevant information about the request. Webhook events are triggered when specific payment statuses are set.

In this tutorial, the PMR Corporation requests a $500,000 wire payment from Lindberg Incorporated.

> If you want to enter originator values yourself (instead of the system populating those attributes based on your account number), ensure that **Allow Custom Originator** is enabled on the product or account .

1. Call `POST /v1/payments/corporate-drawdown-requests`.

2

your tutorial request example

```
API call
{
  "accountNumber": "2714035231",
  "originator":
  {
      "idCode": "D",
      "identifier": "2714035231",
      "name": "PMR Corp",
      "address1": "10431 Howe Drive",
      "address2": "Shawnee Mission KS 66206",
      "address3": "US"
  },
  "receiverRoutingNumber": "026009593",
  "drawdownCreditAccountNumber": "021214891",
  "beneficiaryFi":
  {
      "idCode": "F",
      "identifier": "021214891",
      "name": "Cross River Bank",
      "address1": "400 KELBY STREET",
      "address2": "Fort Lee NJ 07024",
      "address3": "US"
  },
  "drawdownDebitAccount":
  {
      "idCode": "D",
      "identifier": "777555333",
      "name": "Lindberg Incorporated",
      "address1": "5 Stop St",
      "address2": "New York NY 52555",
      "address3": "US"
  },
  "beneficiary":
  {
      "idCode": "D",
      "identifier": "2714035231",
      "name": "PMR Corp",
      "address1": "10431 Howe Drive",
      "address2": "Shawnee Mission KS 66206",
      "address3": "US"
  },
  "beneficiaryReference": "testref",
  "originatorToBeneficiary1": "1string",
  "originatorToBeneficiary2": "2string",
  "originatorToBeneficiary3": "3string",
    "originatorToBeneficiary4": "4string",
    "debitDrawdownAdviceCode": "WRE",
    "debitDrawdownInformation1": "string1",
    "debitDrawdownInformation2": "string2",
    "debitDrawdownInformation3": "string3",
    "debitDrawdownInformation4": "string4",
    "debitDrawdownInformation5": "string5",
    "receiverFiInformation1": "Astring",
    "receiverFiInformation2": "Bstring",
    "receiverFiInformation3": "Cstring",
    "receiverFiInformation4": "Dstring",
    "receiverFiInformation5": "Estring",
    "fiToFiInformation1": "Hstring",
    "fiToFiInformation2": "Istring",
    "fiToFiInformation3": "Jstring",
    "fiToFiInformation4": "Kstring",
    "fiToFiInformation5": "Lstring",
    "fiToFiInformation6": "Mstring",
    "amount": 50000000,
    "purpose": "drawdown testing",
    "clientIdentifier": "40708c24-7dee-49c3-b4d2-c48d876f1f5e"
  }
```

> We recommend that you save the `request-id`, you can see in the response header to use if you need to contact the Cross River Support Team about that specific call.

3. A successful API call returns a JSON response with the details of the drawdown request.

Sample response

```
{
  "id": "e201c363-1039-45cc-99c3-af8500eebb92",
  "accountNumber": "2714035231",
  "referenceId": "W230093GG8S",
  "direction": "Outbound",
  "paymentType": "Drawdown",
  "source": "OpsPortal",
  "status": "Created",
  "posting": "Pending",
  "partnerAuthorization": "NotRequired",
  "partnerAuthorization2": "NotRequired",
  "amount": 50000000,
  "currency": "usd",
  "purpose": "ENTERED BY #6384FF582599E9D9AAADC7BE# - sample drawdown ca
  "businessFunctionCode": "DRC",
  "typeCode": "1031",
  "senderRoutingNumber": "021214891",
  "senderName": "CROSS RIVER BANK",
  "senderReference": "W230093GG8S",
  "receiverRoutingNumber": "026009593",
  "receiverName": "BK AMER NYC       ",
  "originatingFi": {
    "idCode": "F",
    "identifier": "021214891",
    "name": "CROSS RIVER BANK",
    "address1": "400 KELBY STREET",
    "address2": "FORT LEE NJ 07024",
    "address3": "US"
  },
  "originator": {
    "idCode": "D",
    "identifier": "2662824164",
    "name": "PMR CORPORATION",
    "address1": "10431 Howe Drive",
    "address2": "Shawnee Mission KS 66206",
    "address3": "US"
  },
  "beneficiaryFi": {
    "idCode": "F",
    "identifier": "021214891",
    "name": "CROSS RIVER BANK",
    "address1": "400 KELBY STREET",
    "address2": "FORT LEE NJ 07024",
    "address3": "US"
  },
    "beneficiary": {
      "idCode": "D",
      "identifier": "2662824164",
      "name": "PMR Corporation",
      "address1": "10431 Howe Drive",
      "address2": "Shawnee Mission KS 66206",
      "address3": "US"
    },
    "originatorToBeneficiary1": "1string",
    "originatorToBeneficiary2": "2string",
    "originatorToBeneficiary3": "3string",
    "originatorToBeneficiary4": "4string",
    "beneficiaryReference": "testref",
    "wasReversed": false,
    "isInternational": false,
    "clientIdentifier": "40708c24-7dee-49c3-b4d2-c48d876f1f5e",
    "createdAt": "2023-01-09T09:29:11.9495462-05:00",
    "effectiveDate": "2023-01-09T00:00:00-05:00",
    "originalPaymentId": "e201c363-1039-45cc-99c3-af8500eebb92",
    "partnerId": "cd9c12f4-7691-424a-b38b-af5b0134c611",
    "productId": "a68a0fa5-98fe-4cd5-b626-af5b0137e0cb",
    "lastModifiedAt": "2023-01-09T09:29:11.9651496-05:00",
    "postingCode": "OK"
  }
```

4 Successful posting of the payment triggers the `wire.payment.received` webhook event.

9.3. Respond to a drawdown request

When a bank receives a **drawdown request**, it *responds* by wiring the money that was asked for to the specified bank account.

In this tutorial, you'll learn how to:

✅ Respond to a drawdown request

> If you are new to wires we recommend you read the **wires overview** before starting this tutorial.
>
> The tutorial assumes you have a knowledge of APIs and how they work. For more information on sending API calls, see the **API basics**.

For this tutorial you'll need to understand these terms. These definitions are specific to this tutorial. The terms might be used slightly differently for a drawdown request.

Term	Description
Originator	The entity sending the payment
Beneficiary	The entity receiving the payment
Sending financial institution	The financial institution that initiates the drawdown response
Receiving financial institution	The financial institution that receives the wire
Drawdown request	A request for someone to send a payment by wire

The tutorial uses this API endpoint:

API	Description
POST /wires/v1/payments/{id}/drawdown-responses	Respond to a drawdown request by sending payment

The tutorial uses these webhooks:

Webhook	Description
`Wire.Payment.Sent`	Outbound wire has been transmitted to the Federal Reserve and has been successfully acknowledged. IMAD number is now available.
`Wire.Payment.Received`	Inbound wire payment received successfully.
`Wire.Payment.Rejected`	Outbound wire could not be processed due to compliance reasons or was rejected by the Federal Reserve.

Before you begin

Make sure you have:

- **API credentials**
- Beneficiary reference ID
- Inbound drawdown request

Register the relevant webhook events

To receive the webhook events for this tutorial both partner accounts need to **register** each specific webhook event type. Once you are registered, the event objects are sent to the registered URLs.

The event object contains a list of resource identifiers used to download details on each event.

Respond to a drawdown request

In this tutorial you will respond to a drawdown request from Mastercard for **12,480,981.47** USD. An incoming drawdown request looks like the code sample below.

Sample inbound drawdown request

```
{
  "id": "5beea7ed-e9a1-4914-9bdc-af7100eb4dbd",
  "accountNumber": "2805121064",
  "referenceId": "W22354K5ET6",
  "fedBatchId": "9da4f836-7dae-4c54-8dee-af7100eb4dc2",
  "direction": "Inbound",
  "paymentType": "Drawdown",
  "source": "File",
  "status": "Completed",
  "posting": "Posted",
  "partnerAuthorization": "NotRequired",
  "partnerAuthorization2": "NotRequired",
  "amount": 1248098147,
  "currency": "usd",
  "purpose": "",
  "imad": "20221220B1QGC05C001017",
  "omad": "20221220QMGFNP7200015412200510FT01",
  "businessFunctionCode": "DRC",
  "typeCode": "1031",
  "senderRoutingNumber": "021000021",
  "senderName": "JPMORGAN CHASE",
  "senderReference": "1097200354JO",
  "receiverRoutingNumber": "021214891",
  "receiverName": "CROSS RIVER BK",
  "originator": {
    "idCode": "D",
    "identifier": "014053007",
    "name": "MASTERCARD INTERNATIONAL",
    "address1": "INCORPORATE/SERVICES",
    "address2": "2200 MASTERCARD BOULEVARD",
    "address3": "O'FALLON MO 63366- US"
  },
  "beneficiary": {
    "idCode": "D",
    "identifier": "014053007",
    "name": "MASTERCARD INTERNATIONAL",
    "address1": "INCORPORATE/SERVICES",
    "address2": "2200 MASTERCARD BOULEVARD",
    "address3": "O'FALLON MO 63366- US"
  },
  "originatorToBeneficiary1": "ICA 17128 CC SETTLEMENT",
  "beneficiaryReference": "2022122058465489",
  "receiptDate": "1220",
  "receiptTime": "0510",
  "wasReversed": false,
    "isInternational": false,
    "createdAt": "2022-12-20T09:17:27.307-05:00",
    "processedAt": "2022-12-20T09:27:07.98-05:00",
    "effectiveDate": "2022-12-20T00:00:00-05:00",
    "completedAt": "2022-12-20T09:27:07.98-05:00",
    "postedAt": "2022-12-20T09:27:07.98-05:00",
    "originalPaymentId": "5beea7ed-e9a1-4914-9bdc-af7100eb4dbd",
    "partnerId": "6ee82269-9e89-450e-9065-a9e5012c2645",
    "productId": "701d116d-ceec-4868-94f0-ab3c00e82bde",
    "lastModifiedAt": "2022-12-20T09:27:07.9803522-05:00",
    "postingCode": "OK"
  }
```

1 Call `POST /v1/payments/{id}/drawdown-responses`. For this call, some/all of the attributes are required.

The id attribute must be set to the payment ID of the request that came in. In this case, **5beea7ed-e9a1-4914-9bdc-af7100eb4dbd**.

Sample "response to drawdown" request

```
{
"originator":
    {
    "idCode":"D",
    "identifier":"2805121064",
    "name":"CRB FBO RPPS Settlement",
    "address1":"885 Teaneck Rd",
    "address2":"Teaneck, NJ 07666 US"
    },
"beneficiaryReference":"1097200354JO"
}
https://sandbox.crbcos.com/Wires/v1/payments/5beea7ed-e9a1-4914-9bdc-af7
```

2 A successful API call returns a JSON response with the details of the payment sent. From this point, the payment behaves like any **outbound wire payment**.

JSON

```
{
  "id": "cb9bd5f3-e720-47dc-ab9b-af7100fe54d4",
  "accountNumber": "2805121064",
  "coreTransactionId": "e3e7a91e-fe43-40c4-a264-af710135d35d",
  "referenceId": "W2235435X38",
  "fedBatchId": "0259951e-e377-4940-b312-af71013af155",
  "fedBatchSequence": 1,
  "direction": "Outbound",
  "paymentType": "DrawdownResponse",
  "source": "OpsPortal",
  "status": "Completed",
  "posting": "Posted",
  "partnerAuthorization": "NotRequired",
  "partnerAuthorization2": "NotRequired",
  "amount": 1248098147,
  "currency": "usd",
  "purpose": "ENTERED BY #61AA4FE6014AFB01143EBB35#",
  "imad": "20221220QMGFT009000933",
  "omad": "20221220B1QGC01R06123612201412",
  "businessFunctionCode": "DRW",
  "typeCode": "1032",
  "senderRoutingNumber": "021214891",
  "senderName": "CROSS RIVER BK",
  "senderReference": "W2235435X38",
  "receiverRoutingNumber": "021000021",
  "receiverName": "JPMORGAN CHASE",
  "originator": {
    "idCode": "D",
    "identifier": "2805121064",
    "name": "CRB FBO RPPS Settlement",
    "address1": "885 Teaneck Rd",
    "address2": "Teaneck, NJ 07666 US"
  },
  "beneficiary": {
    "idCode": "D",
    "identifier": "014053007",
    "name": "MASTERCARD INTERNATIONAL",
    "address1": "INCORPORATE/SERVICES",
    "address2": "2200 MASTERCARD BOULEVARD",
    "address3": "O'FALLON MO 63366- US"
  },
  "beneficiaryReference": "1097200354JO",
  "receiptDate": "1220",
  "receiptTime": "1412",
  "wasReversed": false,
    "isInternational": false,
    "createdAt": "2022-12-20T10:25:59.53-05:00",
    "processedAt": "2022-12-20T14:07:01.817-05:00",
    "effectiveDate": "2022-12-20T00:00:00-05:00",
    "completedAt": "2022-12-20T14:46:58.82-05:00",
    "postedAt": "2022-12-20T13:48:02.473-05:00",
    "originalPaymentId": "5beea7ed-e9a1-4914-9bdc-af7100eb4dbd",
    "partnerId": "6ee82269-9e89-450e-9065-a9e5012c2645",
    "productId": "701d116d-ceec-4868-94f0-ab3c00e82bde",
    "lastModifiedAt": "2022-12-20T14:46:58.8208426-05:00",
    "postingCode": "OK"
  }
```

9.4. Simulate an inbound wire

Simulations allow testing certain inbound payment flows in our sandbox environment. Simulating an inbound wire transfer is triggered explicitly using the API endpoint outlined below as well as the webhook employed.

> If you are new to wires we recommend you read the **wires overview** before starting this tutorial.
>
> The tutorial assumes you have a knowledge of APIs and how they work. For more information on sending API calls, see the **API basics**.

The tutorial uses this API endpoint

API	Description
POST /Wires/v1/payments/simulations	Simulate a post of an inbound wire transfer to an account

Before you begin

Make sure you have

- **API credentials**
- Partner ID
- **Register** the following webhook:

Webhook	Description
`Wire.Payment.Received`	Inbound wire has been received from another bank

Request a refund by wire transfer

When you call `POST /Wires/v1/payments/simulations`, the Cross River sandbox creates a simulation of an inbound wire transfer to a deposit or subledger account so you can see what the inbound transfer looks like and test any automations you are designing. In this example, Stream247 is refunding $19.99 to their customer, Jana Parker, by sending a wire transfer from their JP Morgan Chase account.

POST /Wires/v1/payments/simulations request

```
{
  "accountNumber": "2846838676",
  "originator": {
    "idCode": "D",
    "identifier": "4236598541",
    "name": "Stream247",
    "address1": "257 Dalton Groves",
    "address2": "Barton City, MI 48075"
  },
  "businessFunctionCode": "CTR",
  "senderRoutingNumber": "021000021",
  "senderDiName": "JP Morgan Chase",
  "originatorFi": {
    "idCode": "F",
    "identifier": "021000021",
    "name": "JP Morgan Chase"
  },
  "beneficiaryReference": "Refund Sep2022",
  "amount": 1999
}
```

These details show in the API response:

The Stream247 account at `JP Morgan Chase` as the `originator` sent an inbound wire transfer `direction: Inbound`, to Jana Parker, the `beneficiary`, via `Cross River Bank2` in the `amount` of `$19.99` reflected in amount as **1999**. The payment ID is **e68e32c3-a475-4d0b-a6d8-ae3c01186ff4**, as seen in the `id` attribute.

POST /v1/payments/simulations response

```json
{
  "id": "e68e32c3-a475-4d0b-a6d8-ae3c01186ff4",
  "accountNumber": "2846838676",
  "referenceId": "W2204571AT5",
  "direction": "Inbound",
  "paymentType": "Transfer",
  "source": "Api",
  "status": "Hold",
  "posting": "Pending",
  "amount": 1999,
  "currency": "usd",
  "purpose": "simulation",
  "imad": "1563371856856662630811",
  "omad": "942955532798176168895701363106962410",
  "businessFunctionCode": "CTR",
  "typeCode": "1000",
  "senderRoutingNumber": "021000021",
  "senderName": "JP Morgan Chase",
  "senderReference": "test",
  "receiverRoutingNumber": "021214891",
  "receiverName": "Cross River Bank2",
  "originatingFi": {
    "idCode": "F",
    "identifier": "021000021",
    "name": "JP Morgan Chase"
  },
  "originator": {
    "idCode": "D",
    "identifier": "4236598541",
    "name": "Stream247",
    "address1": "257 Dalton Groves",
    "address2": "Barton City, MI 48705"
  },
  "beneficiaryFi": {
    "idCode": "F",
    "identifier": "021000021",
    "name": "Cross River Bank",
    "address1": "400 Kelby Street",
    "address2": "Fort Lee, NJ",
    "address3": "US"
  },
  "beneficiary": {
    "idCode": "D",
    "identifier": "2846838676",
    "name": "John Smith",
      "address1": "250 Kuhn Highway",
      "address2": "",
      "address3": "Grover, NC 28073"
    },
    "beneficiaryReference": "Refund Sep2022",
    "receiptDate": "0214",
    "receiptTime": "1201",
    "wasReversed": false,
    "isInternational": false,
    "createdAt": "2022-02-14T12:01:02.460748-05:00",
    "effectiveDate": "2022-02-14T00:00:00-05:00",
    "originalPaymentId": "e68e32c3-a475-4d0b-a6d8-ae3c01186ff4",
    "partnerId": "f03ff044-8883-4939-9d22-ade301661897",
    "productId": "9b8490c9-ccac-4f7d-8b49-ae3800e6b418",
    "lastModifiedAt": "2022-02-14T12:01:02.4967164-05:00",
    "postingCode": "OK"
  }
```

The `Wire.Payment.Received` webhook event fires. The payment ID (**e68e32c3-a475-4d0b-a6d8-ae3c01186ff4**) appears in the `resources` object. That is the same ID that you received in the response to your simulation request.

Wire.Payment.Received webhook

```
  {
    "id": "3809b97e-58dc-4082-bc64-ae3c01188e1c",
    "eventName": "Wire.Payment.Received",
    "status": "Pending",
    "partnerId": "f03ff044-8883-4939-9d22-ade301661897",
    "createdAt": "2022-02-14T12:01:28.197-05:00",
    "resources": [
      "wires/v1/payments/e68e32c3-a475-4d0b-a6d8-ae3c01186ff4"
    ],
    "details": []
  }
```

10. Checks

In this tutorial, you'll learn how to:

✅ Deposit a check

✅ Handle rejected checks

> This tutorial assumes you have a knowledge of APIs and how they work. For more information on sending API calls, see **API basics**.

The tutorial uses these API endpoints

API	Description
POST /checks/v1/payments	Provides the necessary information and images to deposit a check

The tutorial uses these webhooks.

Webhook	Description
`Check.Payment.Sent`	A check deposit was sent to the Federal Reserve for clearing
`Check.Payment.Rejected`	An outbound check wasn't processed due to compliance reasons or rejection by the Federal Reserve
`Check.Payment.Returned`	Receiving bank returned a check to the sender

Before you begin

Make sure you have:

- **API credentials**

- Cross River bank account number
- Images of the front and back of the check in Base64 (C9 check) format

Register relevant webhook events

To receive the webhook events for this tutorial both partner accounts need to **register** each specific webhook event type. Once you are registered, the event objects are sent to the registered URLs.

The event object contains a list of resource identifiers used to download details on each event.

Deposit a check

In this tutorial you're a Banking-as-a-Service (BaaS) partner that offers a deposit account product to consumers. Your app allows your customers to perform mobile deposits. One of your customers just made a $1 check deposit into their account using your app, which is calling the Cross River system to pass the deposit details.

To deposit a check

1. Call `POST /Checks/v1/payments`. For this call, you must supply the number of the account where the check is being deposited and images of both the front and back of the check as Base64-encoded values. See the full list of **attributes**.

   Sample request

   ```
   POST /checks/v1/payments
   {
     "accountNumber": "2193590144",
     "amount": 100,
     "frontImage": "image/jpg;base64,/9j/4REyRXhpZgAATU0AKgA...",
     "backImage": "image/jpg;base64,/9j/4QuqRXhpZgAATU0AKg...",
     "isRedeposit": false
   }
   ```

2 A successful API call returns a JSON response with the details of the check. The `status` attribute in the response only tells you that the payment was created. It's not an indicator of a successful payment. The `id` attribute provides you with the payment ID. In this case, it's **9a44fdbf-89e2-4300-8f05-ad9501439bb1**. The schedule attribute shows the schedule for paying the deposit into the account.

Sample response

```
{
  "id": "9a44fdbf-89e2-4300-8f05-ad9501439bb1",
  "accountNumber": "2193590144",
  "referenceId": "C2436F698K0D",
  "paymentType": "Forward",
  "checkType": "Standard",
  "direction": "Outbound",
  "status": "Created",
  "source": "OpsPortal",
  "posting": "Pending",
  "postingCode": "OK",
  "coreTransactionId": "21ba5fb0-e609-4560-a36f-ad9501439bb1",
  "memoPostId": "dc20e619-c5f7-4890-b3c8-ad9501439bb1",
  "originalPaymentId": "9a44fdbf-89e2-4300-8f05-ad9501439bb1",
  "customerId": "700f0d35-9940-4132-8608-ad89013927a9",
  "payerRoutingNumber": "",
  "payerAccountNumber": "",
  "payeeName": "",
  "checkNumber": "",
  "bofdRoutingNumber": "021214891",
  "sequenceNumber": "1353885824",
  "amount": 100,
  "currency": "usd",
  "recognizedAmount": 0,
  "iqaPassed": false,
  "hasFrontImage": true,
  "hasBackImage": true,
  "isRedeposit": false,
  "policy": "NewAccount",
  "schedule": [
    0,
    0,
    100
  ],
  "createdAt": "2021-08-31T15:38:13.2795042-04:00",
  "wasReturned": false,
  "purpose": "ENTERED BY #60C367E26DD36A0068580230#",
  "depositBusinessDate": "210831",
  "productId": "d5dc52bb-df80-4a5d-a5a8-ad89013844bd",
  "partnerId": "ede1a60d-3d51-47e8-9a9b-ad8901381f9e",
  "lastModifiedAt": "2021-08-31T15:38:13.2951299-04:00"
```

3 The Cross River system executes its OCR process to convert the check images into a format that complies with Federal Reserve Standards. Cross River sends the check deposit to the Federal Reserve for clearing, which triggers the `Check.Payment.Sent` webhook event. Note that the payment ID appears in the `resources` array.

> If you are registered for the `Core.Transaction.Completed` webhook event, you also receive that webhook at this point. When the check payment is released to the Federal Reserve, the Cross River system executes the core transaction related to the payment. This does not mean the funds are available. The `schedule` attribute in the response indicates the availability schedules of the deposited funds.

JSON

```json
{
  "id": "160e52d2-4355-4fa3-a421-ad8800f886a2",
  "eventName": "Check.Payment.Sent",
  "status": "Pending",
  "partnerId": "4c5b488d-711d-428a-bdae-ad800131970d",
  "createdAt": "2021-08-31T15:39:51.3-04:00",
  "resources": [
    "checks/v1/payments/9a44fdbf-89e2-4300-8f05-ad9501439bb1"
  ],
  "details": [
    {
      "paymentId": "9a44fdbf-89e2-4300-8f05-ad9501439bb1",
      "paymentType": "Forward",
      "coreTransactionId": "21ba5fb0-e609-4560-a36f-ad9501439bb1",
      "memoPostId": "dc20e619-c5f7-4890-b3c8-ad9501439bb1",
      "accountNumber": "2193590144",
      "depositBusinessDate": "230808",
      "postingCode": "OK",
      "amount": "10000",
      "recognizedAmount": "100",
      "payerRoutingNumber": "314074269",
      "payerAccountNumber": "28293886",
      "checkNumber": "1237",
      "checkType": "Standard",
      "sequenceNumber": "1353885824",
      "micr": "1237",
      "purpose": "SKIP_IQA",
      "clientIdentifier": null,
      "schedule": "0,0,10000",
      "policy": "Standard",
      "rejectionReason": null,
      "isRedeposit": "False",
      "originalPaymentId": "9a44fdbf-89e2-4300-8f05-ad9501439bb1"
    }
  ]
}
```

Check validation

A check is systemically validated for compliance when you deposit it, for example the system verifies that the check has a valid MICR number. Results of this validation appear in the body of the response to the deposit request. Non-compliant checks are rejected and the **rejection reason** appears in the `rejectionReason` field.

Rejected checks

Check rejection triggers the `Check.Payment.Rejected` webhook event. In this case the payment ID is **b9e53e1c-683e-469c-8f79-ad8800f1ccc**.

Check.Payment.Rejected webhook event

```
{
  "id": "96595e06-508f-4d5f-ba80-ad8800f1e69d",
  "eventName": "Check.Payment.Rejected",
  "status": "Pending",
  "partnerId": "4c5b488d-711d-428a-bdae-ad800131970d",
  "createdAt": "2021-08-18T10:40:44.033-04:00",
  "resources": [
    "checks/v1/payments/b9e53e1c-683e-469c-8f79-ad8800f1ccc3"
  ],
  "details": [
    {
      "paymentId": "b9e53e1c-683e-469c-8f79-ad8800f1ccc3",
      "paymentType": "Forward",
      "coreTransactionId": "2c9292b0-cf01-44c5-b416-b05800de64c7",
      "memoPostId": "ef1aee23-1546-438a-a2bd-b05800de64c7",
      "accountNumber": "2151546989",
      "depositBusinessDate": "230808",
      "postingCode": "OK",
      "amount": "70000",
      "recognizedAmount": "100",
      "payerRoutingNumber": "314074269",
      "payerAccountNumber": "28293886",
      "checkNumber": "1237",
      "checkType": "Standard",
      "sequenceNumber": "0283216503",
      "micr": "1237",
      "purpose": "SKIP_IQA",
      "clientIdentifier": null,
      "schedule": "0,0,70000",
      "policy": "Standard",
      "rejectionReason": "Duplicate",
      "isRedeposit": "False",
      "originalPaymentId": "b9e53e1c-683e-469c-8f79-ad8800f1ccc3"
    }
  ]
}
```

As already described, the **rejection reason** appears in the `rejectionReason` field of the event. In the example above, the check payment is rejected because of **Duplicate**. Address any check issues with the customer prior to resubmitting the check deposit to avoid subsequent rejections.