Technical document

API – Portal

Version 2.4 – Apr 2022

Prepared by TIB Inc.
909 Sainte Thérèse street,
Drummondville (Québec) J2B 4V5
T 819 850-1788
https://tib.finance/

➢ Content

- ➢ Content 2
- ➢ Environments 3
- ➢ API Objects Overview 3
  - Structure related objects overview. 3
  - Customer related objects overview 7
  - Transaction related objects overview. 11
  - Process related objects overview 13
- ➢ Combined operations 17
- ➢ Calling the API 18
  - High level concept 18
  - TIB Implementation 19
  - HTML call headers 20
- ➢ Encryption process details 21
  - Step 1: Request asymmetric key. 21
  - Step 2: Generate the client-side symmetric key. 21
  - Step 3: Generate the client-side RSA asymmetric key. 21
  - Step 4: Combines client-side symmetric key and asymmetric key. 22
  - Step 5: Encrypt the combined keys. 22
  - Step 6: Transmit the key to the server. 23
  - Step 7: Decrypt the server-side received key. 24
  - Step 8: Combine symmetric keys. 25
  - Step 9: Perform the desired call. 25
  - Step 10: Decrypt the returned result from the server. 26
- ➢ Error handling 28
- ➢ Call Details 29
  - Calls URL 29
  - Call list 29
  - Sessions 31
  - Customers 32
  - Payment methods 39
  - Bills / Payments / Transfers 54
  - Reporting of operations 76
  - Merchants 80
  - General objects and enumerations 91

➢ Environments

Calls to the service are done via a WEB service. There are two URLs for the service:

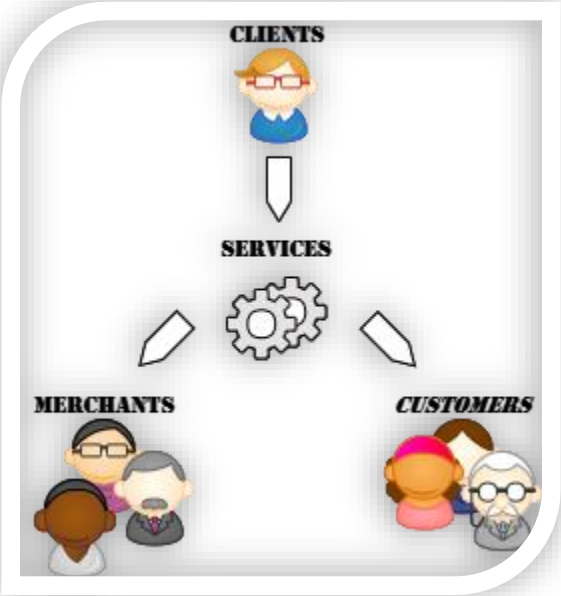
- Production: https://portal.tib.finance
- Development: http://sandboxportal.tib.finance

➢ API Objects Overview

This section explains the objects role in the API. All interaction is later described in the document in "detailed" section.

Structure-related objects overview.

To understand how to use the API, you must understand the main objects of the application.



Clients

TIB account is called a "Client" into the API. The client ID is required for the session creation call. This identification is provided by TIB during the account opening. This ID is a "Guid" formatted hexadecimal.

Example of Client ID
136d30a0-7ab0-4ebe-be27-75aaaa944c1b

Services

The service layer allows a client to have multiple different contracts with TIB Finance. It is used only when the client act for more than one company.

The service determines the limits and fees according to the contract.

The service ID is required for multiple calls. This identification is provided by TIB during the account opening. This ID is a “Guid” formatted hexadecimal.

Example of Service ID
798468f9-e87a-4c52-ace8-638a53bf4bea

Merchants

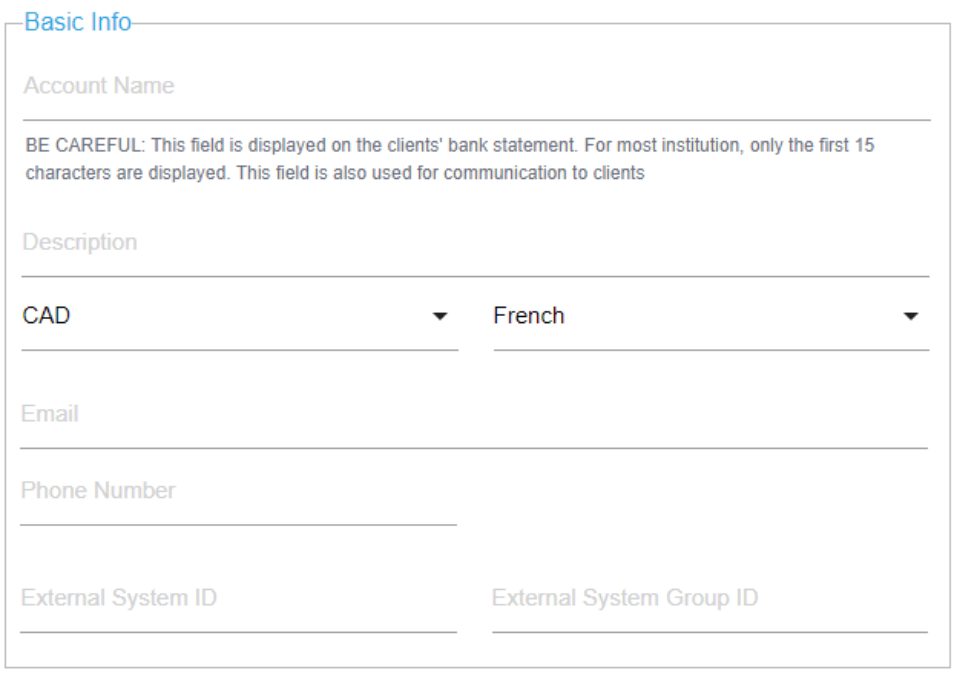
The merchant can be understood as the bank account of the client. The client may have multiple accounts to perform transactions. The merchant has two concepts: Basic Information and Account Information.

The primary merchant account is created by TIB Finance at the client account opening. Most of the call required the related merchant ID to define the transaction bank account. When transaction is a collection of a customer’s account, it defines the money destination account. When the transaction is a deposit to a customer account, it defines the money origin account. This ID is a “Guid” formatted hexadecimal.

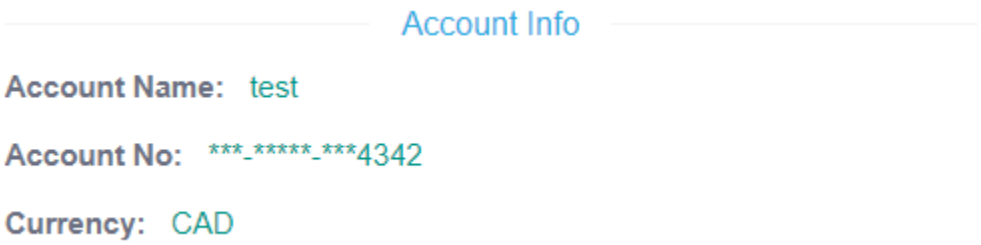
Other merchant can be created through the API, but it requires a validation process.

Example of Merchant ID
3f9aae04-c58b-4a5e-939e-7111a9a1057f

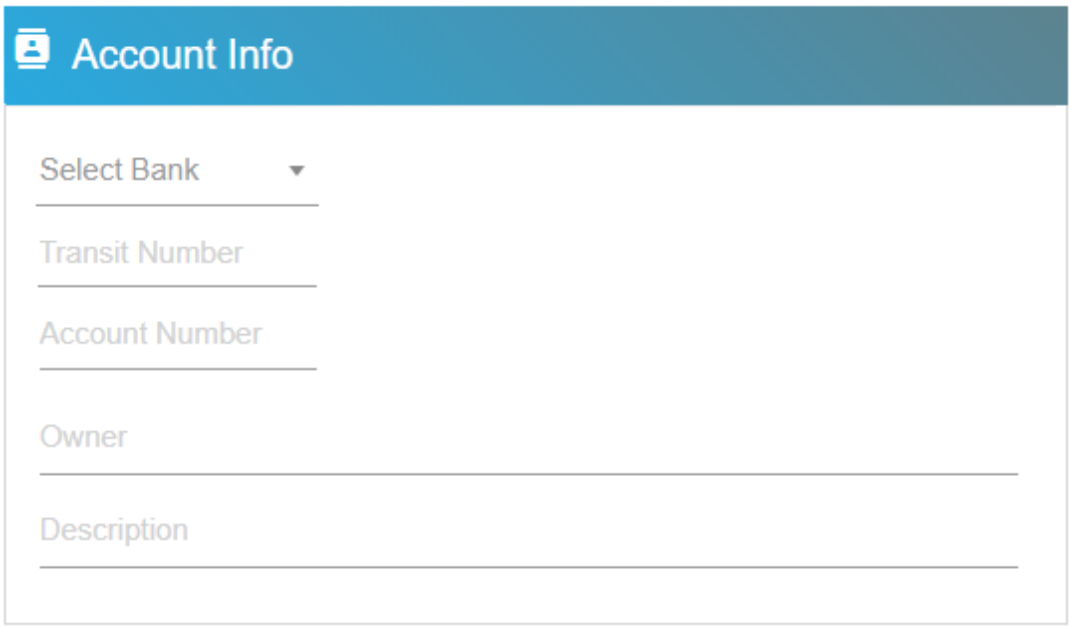
The following screens demonstrate the merchant Basic Information.



The merchant Account Information is bank account information. It cannot be fully extracted as only a preview can be obtained once the account is created.



The account number can be created or updated by the API, but it requires a validation process. This screen demonstrates the fields of an account.


Account Info
Select Bank
Transit Number
Account Number
Owner
Description

Customer related objects overview

Customers

Customers are the clients of the merchants. They are the one the merchant collect money from, or the one the merchant deposit money to.

The customer is only a container object that identify the entity of a person. This object will then have payment methods attached to it for the account information. The customer ID needs to be used when transmitting payment on the API. This ID is a “Guid” formatted hexadecimal.

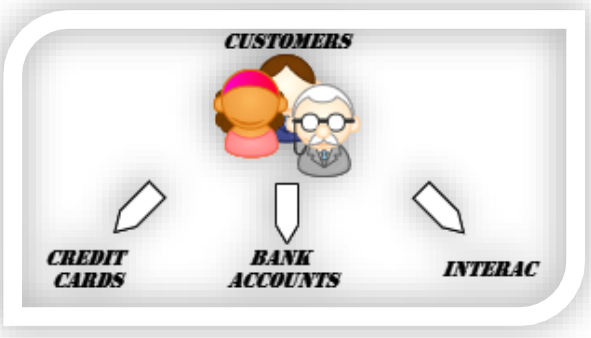
Example of Customer ID
05880372-5c30-4f17-8796-c353bfaece3f

The following screen demonstrates the customer information,

It is also possible to create payment with mode “Anonymous”. This mode required no customer but requires only the email address of the customer.

Payment methods

The payment methods are financial accounts attached to a customer. A customer can have multiple payment methods.



All payment methods have a unique identifier. This ID is a “Guid” formatted hexadecimal.

> ***Example of Payment method ID***
> *1aef40d7-8e77-4e01-9408-b985768acf28*

There are three payment method types supported by TIB Finance:

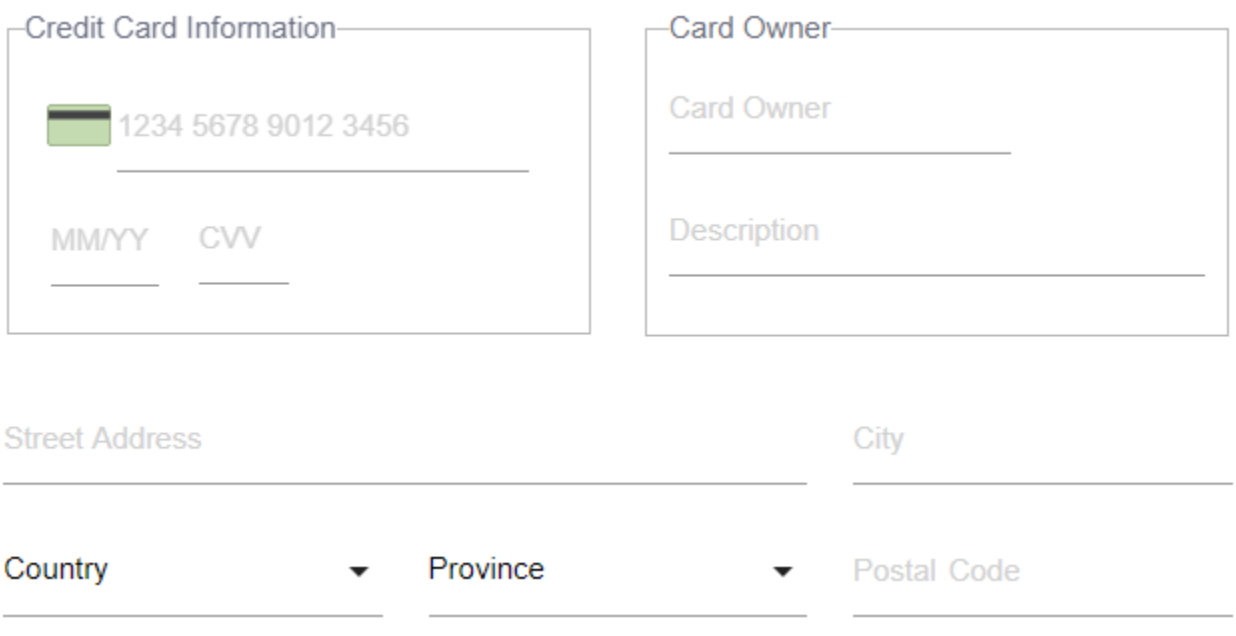
- Credit card
- Bank account
- Interac

Credit card

Credit card payment method allow the merchant to collect money from the customer’s credit card.

> *The credit card payment method cannot be used during deposit.*

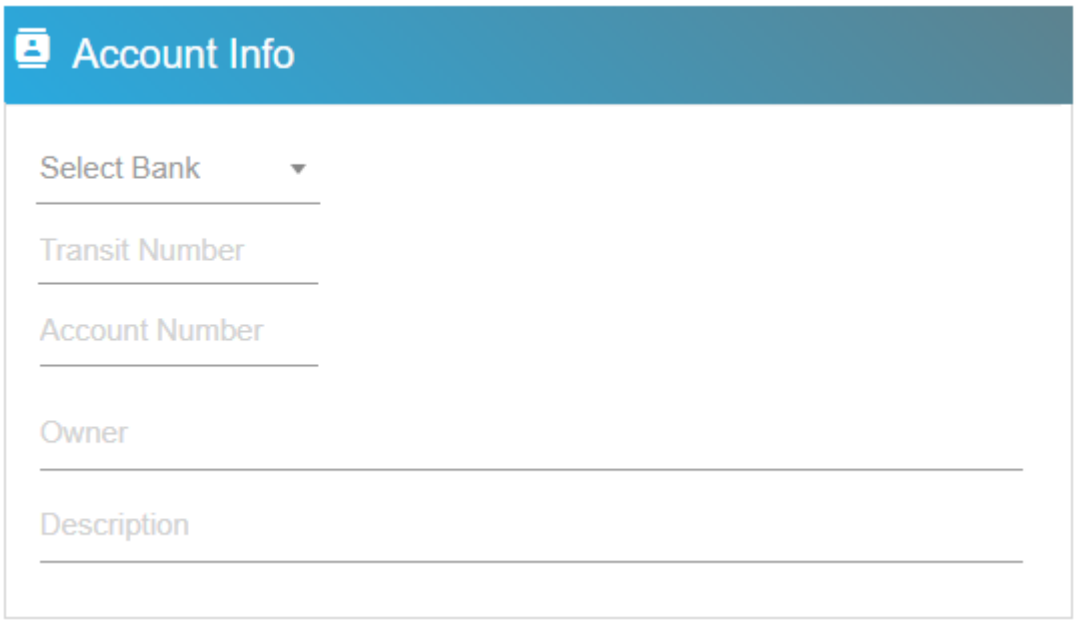
The following screen demonstrates the credit card information:



Bank account

Bank account payment method type allow to perform direct deposit and process pre-authorised payment.

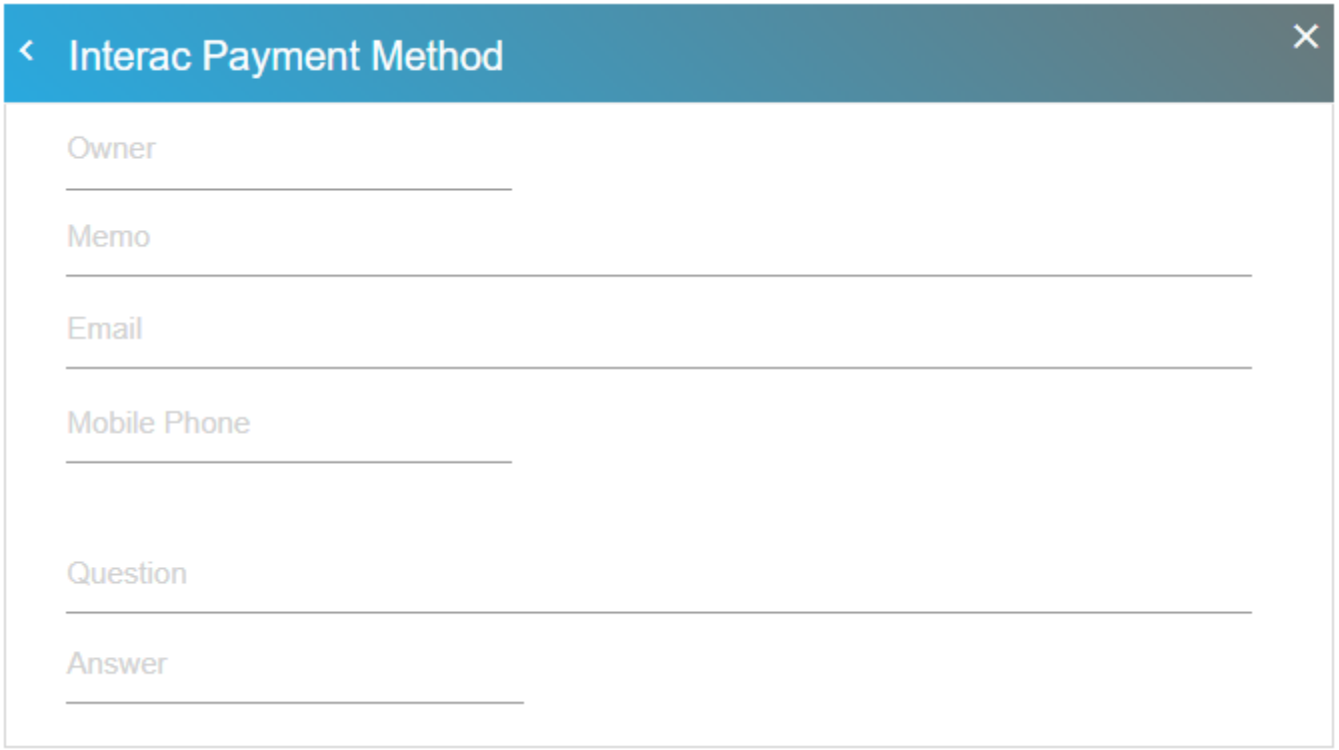
The following screen demonstrates the bank account information:



Interac

This payment method type allows to use Interac to collect or deposit money to a customer account.

The following screen demonstrates the interact information:

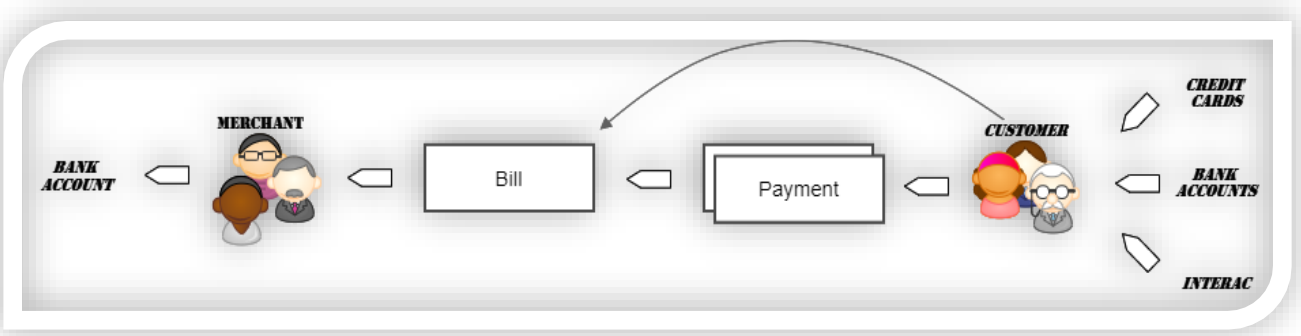

Interac Payment Method
Owner
Memo
Email
Mobile Phone
Question
Answer

Transaction related objects overview.

This section explains the payment related object overview to help understanding the way to process payment within the API.

Bills and payments.

It is possible with the API to create a bill and add payments for the bill. This allows the merchant to collect the customer based on bill information.



Bill

When creating a bill, it will return the created bill ID for further operation on the bill. This ID is a “Guid” formatted hexadecimal.

Example of Bill ID
288dde10-082f-4bc4-9a0d-4f61e11a32ab

Payment

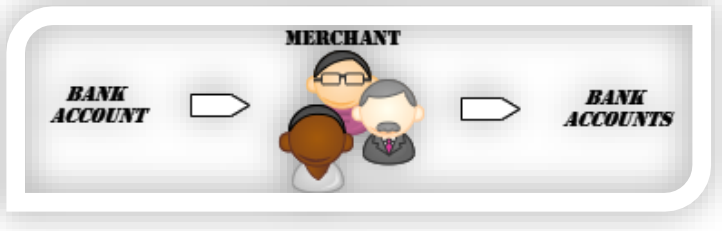
When creating a payment, it will return the created payment ID for further operation on the payment. This ID is a “Guid” formatted hexadecimal.

Example of Payment ID
8da73801-e4ee-498a-b5cd-359a10c44cd8

THERE IS MULTIPLE WAY FOR THE SYSTEM TO PROCESS THE PAYMENT. THE MOST COMMON VALUES USED ARE “AUTO SELECT EASIER” AND “ANONYMOUS”. THE FIRST MODE WILL PROCESS THE PAYMENT USING THE INFORMATION PROVIDED. THE SECOND WILL TRANSMIT THE PAYMENT BY EMAIL TO AN UNKNOWN CUSTOMER. ALL MODES ARE DESCRIBED IN DETAIL IN THE “DETAILS” SECTION OF THIS DOCUMENT.

Direct deposit

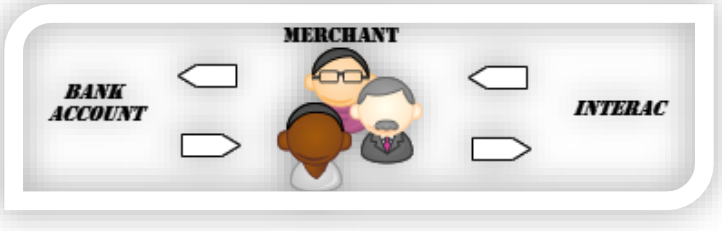
The API allow to create a direct deposit from the merchant account to a destination account without having to use customer and payment method objects.



This call only needs a merchant identification and the destination bank account information. When using “direct deposit”, TIB Finance will internally create a customer and a payment method based on the account information. Using methods such as “ListCustomers” will return the customers having been created using “direct deposit” method.

Direct Interac

This call needs a merchant identification and the destination email or mobile phone number.



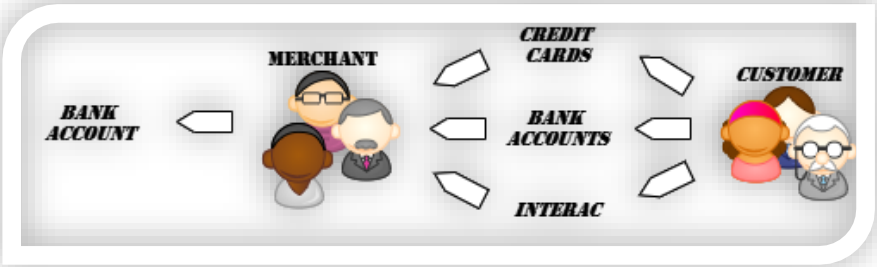
This API functionality allow to send money or collect money using a mobile phone number or email address only. The process will use Interac process to ask the person to specify his account.

ACP file

TIB Finance API support the ACP 005 file format. An organization number need to be assigned during the TIB Finance account creation to enable this format.

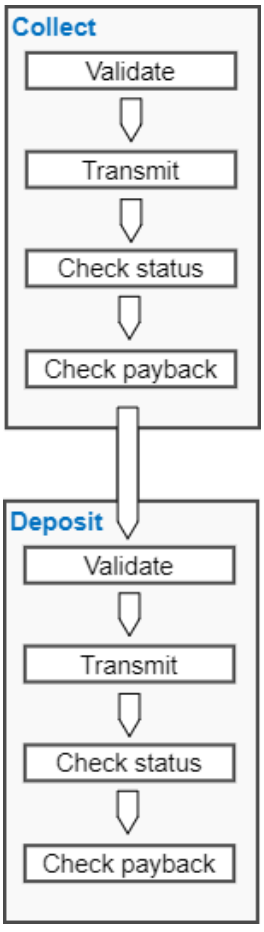
Free operations

The API action called “Free Operation” means to create a transaction not related to a Bill. A “free operation” can be created to collect a customer payment method or to deposit to a customer payment method (deposit is not allow for credit card payment method).



Process related objects overview.

When processing money transfer, TIB Finance regroup processing into operations. Each operation can execute four transactions.



Operations

The operations are multiple money movement in relation with the same logical process. Every transfer requested will generate a minimum of 3 operations.

Deposit flow operations:

1. Collect the fund from the merchant related account.
2. Deposit the fund into customer payment method.
3. Collect fees (all regrouped and collected on the 5th of the next month).

Collection flow operations:

1. Collect the fund from customer payment method.
2. Deposit the fund into the merchant related account.
3. Collect fees (all regrouped and collected on the 5th of the next month).

Transactions

All operations have multiple execution transactions. The transactions correspond to execution sequence of the operation.

1. Validation
2. Transmit: Collection/Deposit
3. Check status: Waiting and verification.
4. Check payback: NSF and Opposition Verification.

To get the transfer operation status, it is required to verify all sub transaction status.

Status

All the transactions have status and detailed description of the execution. This are required to verify the process status.

Target

All process steps can be extracted and interpreted, but the easiest way to implement a code logic to handle the execution status is to use the target. The target determines if the operation was for the merchant or the customer. When a problem occurred within the merchant account, the TIB Finance staff will get alert and will proceed to communication for manual resolution. Therefore, for most common API usage, the call only needs to check the transactions related to the customer.

When the collecting money, the collect operation is the one related to the customer. When deposit money, the deposit operation is the one related to the customer. Other operations are money transfer with the merchant account or system fees automatic collection. Those are not required for normal error handling.

Operation direction

All operations and transaction have the direction: Collect or Deposit. The direction combined with the target well identify the transfer type.

Transaction type

As specified previously, there are 4 transactions for every operation. To perform a good error handling, the status for all the transaction steps needs to get verified.

The transaction step exists only if the previous step is successful. Example, if the status check failed, the payback check transaction will not exist.

Statuses

Depending on the payment method type, transaction can be skip or can be in waiting status. The status is the base information to put in place a good error handling.

Transaction has the following status:

In progress = 0
Success = 1
Success because no result = 2
Success because voluntary skip = 3
Wait manual = 4
Transaction error = 10
Temporary error = 11
Fatal error = 12
Aborted = 13

NOTE: THE REAL ENUMERATION IS LATER DESCRIBED IN THIS DOCUMENT.

An easy way to perform error handling, is to process this way:

- Status 0: Still waiting process execution. (note that payback check can stay in this status for 3 months.)
- Status 1 to 4: Considered Success.
- Status 10: Banking error (Account error, NSF, opposition, etc.)
- Status 11 and 12: System error (normally resolved automatically by TIB finance).
- Status 13: Aborted.

Bank result

There is another enumeration providing general bank result status. This enumeration can be used to determine the general reason when the transaction status is 10 (Transaction error).

No result = 0
Confirmed = 1
Other errors = 2
NSF = 3
Account error = 4
Opposition = 5
Interac Refused = 6
Interac Failed = 7

Transaction description

The transaction description is text representing the error. For bank account payment method, the value contains the standard error code. This error code can be used for more detailed reason than "bank result".

Example: 901:Insuffi. funds

Other standard bank account error codes example:

- 901: Insufficient funds.
- 908: Funds not free.
- 902: Cannot locate account.
- 905: Account closed.
- 911: Frozen account.
- 912: Invalid/Error Account Number
- 903: Stop payment.
- 915: Refused - No agreement.
- 922: CT returned by payee.
- 916: Not in acc./agreement (Personal).
- 917: Agreement revoked (Personal).
- 918: No prenotification (Personal).
- 919: Not in acc./ agreement (Enterprise).
- 920: Agreement revoked (Enterprise).
- 921: No prenotification (Enterprise).
- 900: Validation rejection.
- 907: No debit allowed.
- 909: Curr/Acct Mismatch
- 910: Payor/Payee deceased.
- 914: Err. Payor/Payee name.
- 990: Default by a financial institution.

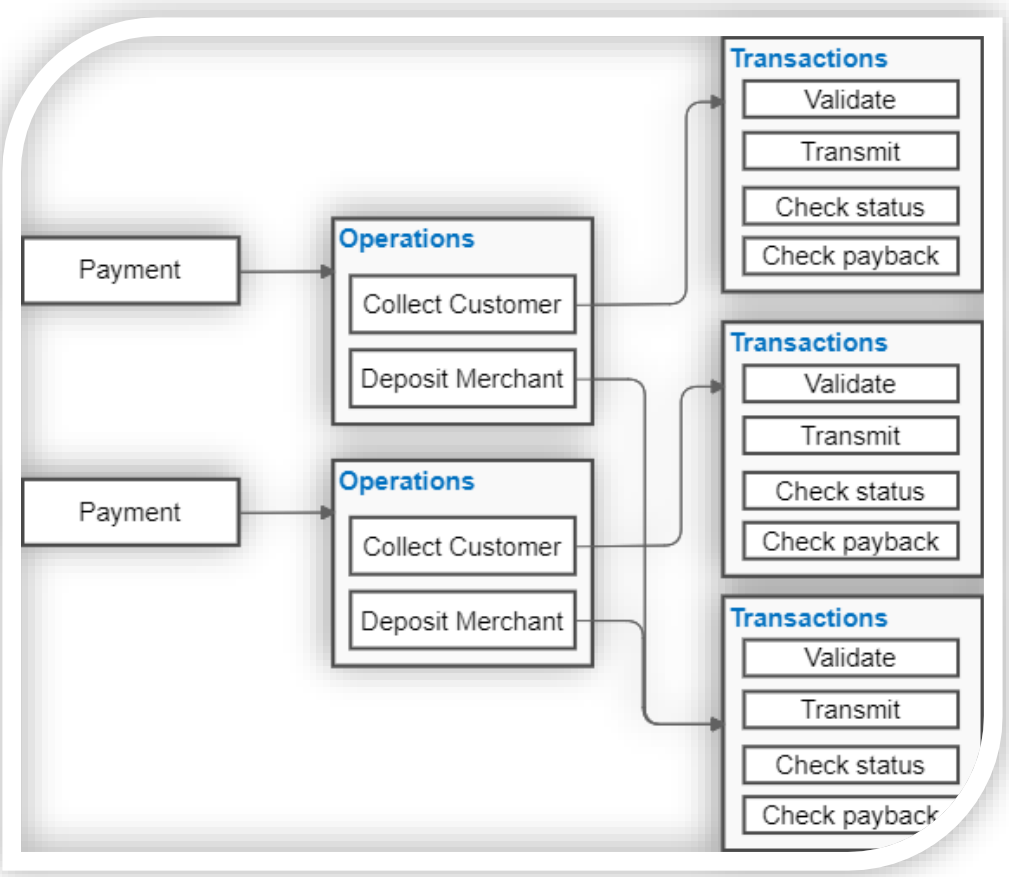
➢ Combined operations

Every operation targeting the merchant account are merged togethers.

When collecting multiple client account, the money will be deposit only once in the merchant destination bank account. Example, the merchant collect 50$ to 10 customers, only 1 deposit transaction of 500$ will be performed in the merchant.

In the same idea, when deposit to multiple accounts, TIB will create only a single transaction to collect the merchant account.

That means the transaction may include multiple operations. That concept is especially important to understand the way the data extraction call will return execution information.



FOLLOWING THE DIAGRAM, YOU CAN NOTICE THE DEPOSIT OPERATION TO MERCHANT ACCOUNT HAVE BEEN COMBINED INTO THE SAME TRANSACTION FLOW LOGIC.

➢ Calling the API

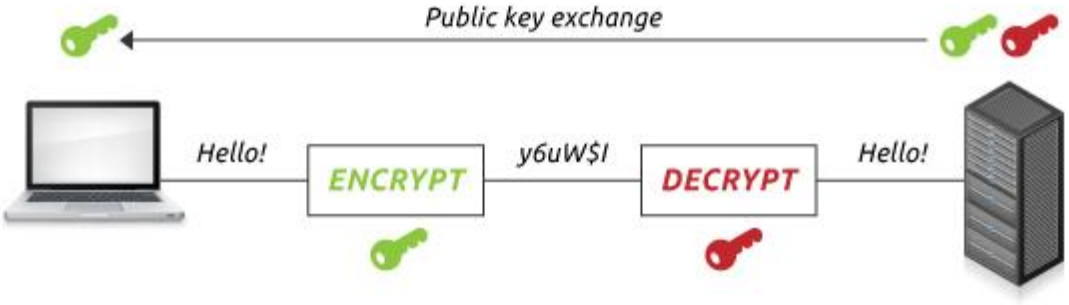
The TIB Finance web service API request the client to get involved in the encryption process. To perform every call, the client must perform three calls:

1. Request public key.
2. Perform key exchanges.
3. Perform the call.

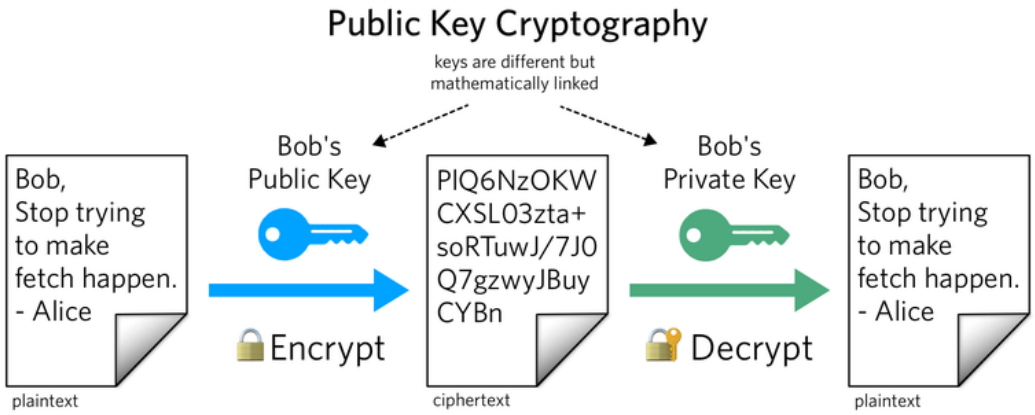
High level concept

To understand better the concept, you need to first understand the asymmetric key encryption process.

Base of asymmetric encryption



This concept allows to share a “public” key that can encrypt the data, but this key cannot decrypt the data. Therefore, the one who send the public key keeps the related “private” key to be able to decrypt the crypted data received.



Base of implementation

The asymmetric key has the strength to be able to send the key safely over the network, because it cannot decrypt the data, but has the weakness of not being able to encrypt large amount of data. Because of this, asymmetric transfer is used only to transmit the standard symmetric AES encryption key.

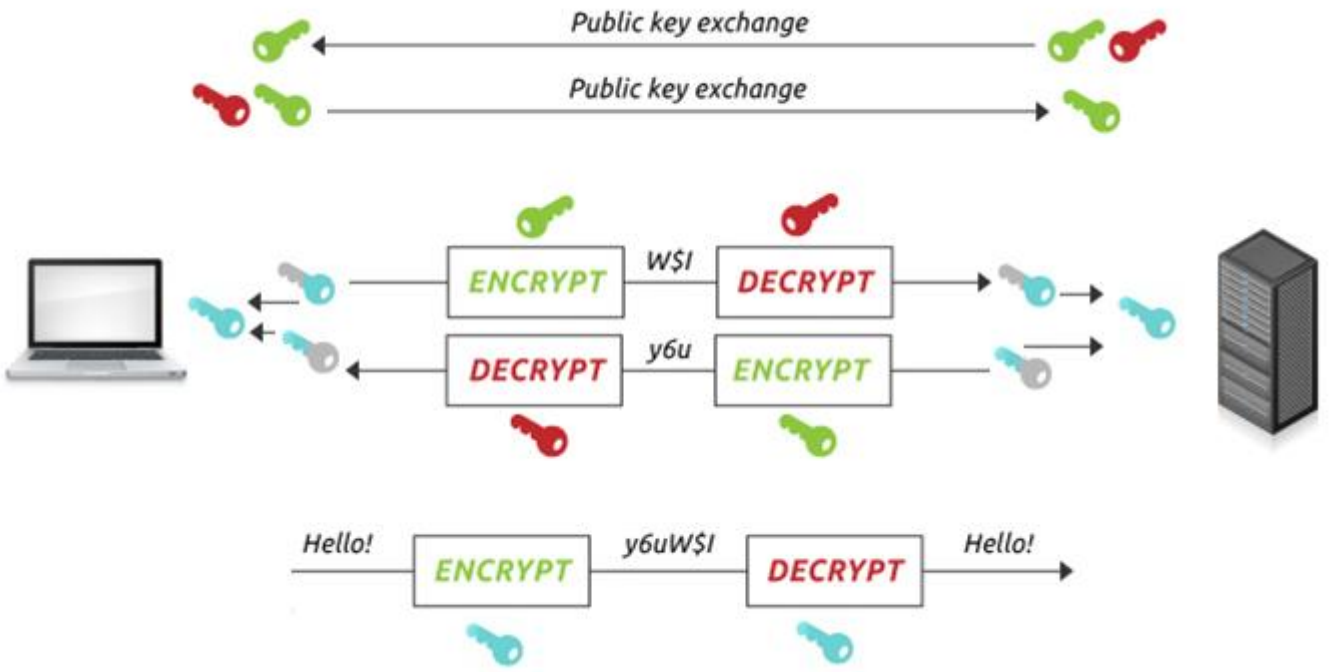
Within this concept, the client first requests a “public” key from the server. The client can create the symmetric key, encrypt data using it, encrypt the key with the public key received and finally transmit the encrypted data and the encrypted key. Using the private key, the server will be able to decrypt the symmetric key and then decrypt the data.

TIB Implementation

Two sides responsibility

Now, using this concept, it gives the responsibility to the client to generate the symmetric key. To ensure good protection, TIB’s API enforce the share of the encryption key creation responsibility.

That means the server send a public key to the client and the client send a public key to the server. Both sides generate a portion of the symmetric key and encrypt it using the other side provided public key. The encrypted key is transmitted back to both sides to create, decrypted using private key and keys are combined to obtains the full key.



Single usage

Once both sides possess the encryption key, it is also related to an identification token (to identify the key). It remains to the client to encrypt the data using the key and transmit the encrypted data and the Identification of the key to the server.

As soon as the server uses the full key to decrypt the received data, it “consumes” the key so it cannot be used again.

Real call structure

The previous explanation explains the base concepts, here are the real call flow that needs to be implemented.

1. **Request public key.**

The client needs to perform a parameter less call to obtain the server-side public key.

2. **Perform the key exchange.**

The client needs to generate its half symmetric key and its asymmetric key (public and private). Client needs then to combine the public portion and the symmetric key, encrypt the result with the server-side public key and transmit the encryption result to the server.

When the server receives the call, the server decrypts the data, use the public key to encrypt the second half of the symmetric key and return the result to the client (with the token of identifying the key)

3. **Perform the call.**

The client can decrypt the server-side portion of the key, combine the keys, encrypt the data and perform the call.

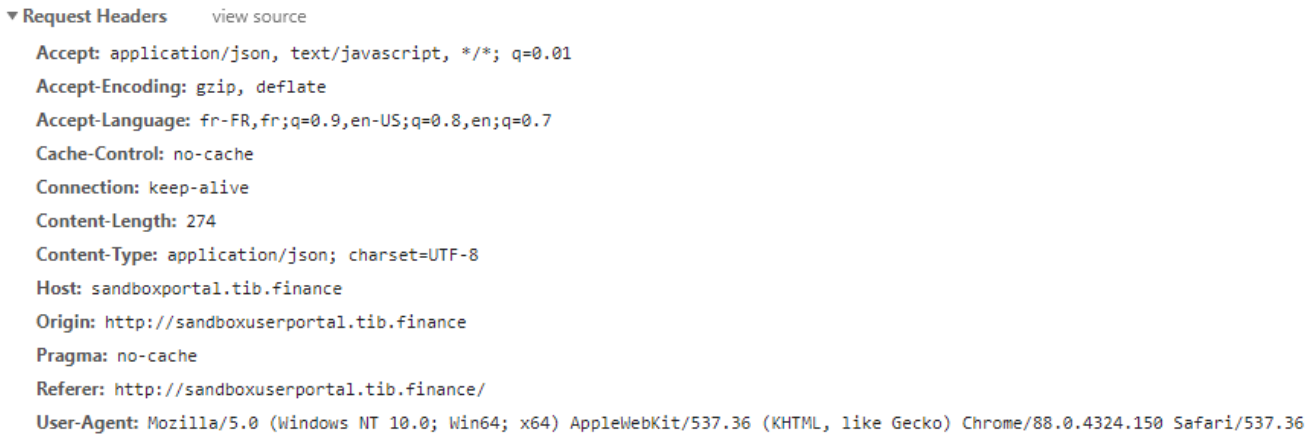
> *The details of the encryption process in the next section "Encryption process details".*

HTML call headers

For web call implementation, all the call to the API needs to:

- Use the content type: `"application/json"`.
- Get encoded with UTF8.

Call header example.



Note that the user-agent, referrer, cache-control, accept-encoding, accept-language are not required, but this example described a valid example when calling from Chrome's browser.

➢ Encryption process details

This section describes the steps for the encryption required on each call.

Step 1: Request asymmetric key.

The first step consists to ask the server the “public” portion of RSA asymmetric key.

To request the key, it requires a POST call on the following URL:

[BaseURL]/Data/GetPublicKey

This call requests no payload. It returned three JSON properties.

```
{
  "KeyToken": "f24d7e12-2d9c-4806-acd6-40e9cf6f8168",
  "PublicKeyXmlString": "<RSAKeyValue><Modulus>a</Modulus><Exponent>a</Exponent></RSAKeyValue>",
  "NodeAnswered": "PortalHost1"
}
```

KeyToken

Identification of the key returned by the server.

PublicKeyXMLString

XML representation of the public key. Note that the RSA key is 8192 bits.

NodeAnswer

The API web service is load balanced. Once call is initiated, all subsequent calls need to be performed on the same node.

Step 2: Generate the client-side symmetric key.

The full AES symmetric key is composed by a portion generated by the client-side and another half generated by the server side. Because in further step the client-side part needs to be transferred to the server (will be crypted first), we first generate that portion in this step.

This symmetric half key is composed of 16 bytes randomly generated.

Suggestion: You can use a Guid byte array.

½ symmetric key example

Bytes: 191,204,10,109,135,63,93,79,140,0,249,103,250,188,212,210
(Hex: 6d0accbf3f874f5d8c00f967fabcd4d2)

Step 3: Generate the client-side RSA asymmetric key.

To allow server to safely return the server-side portion of the key, the client needs to generate a RSA key to allow the server to crypt the transmission.

The client needs to generate 1024 bits RSA key.

The RSA public key needs to be XML formatted to be transmitted to the server.

Keep the “private” portion of the key for step #7.

Client-Side Public key

<RSAKeyValue><Modulus>x/zU3Pv4ji9jzelcHeAb77y5h3jaqjlcdBHk0BCKaQhRswdJvMUP9wEtkutnAjOSjGggR75L7t+4J5+Z78Rdy0SP/N/bMVpzR3MlCiKfxlADN9LhU/b8269YLpJm7nbbDkqMu7e3A6ms09f//FoROLJsY9LrTnQMC64gzP+GORU=</Modulus><Exponent>AQAB</Exponent></RSAKeyValue>

The XML need to be converted into a list of bytes based on UTF8 encoding.

Client-Side public key byte array

60,82,83,65,75,101,121,86,97,108,117,101,62,60,77,111,100,117,108,117,115,62,120,47,122,85,51,80,118,52,106,105,57,106,122,101,108,99,72,101,65,98,55,55,121,53,104,51,106,97,113,106,73,99,100,66,72,107,48,66,67,75,97,81,104,82,115,119,100,74,118,77,85,80,57,119,69,116,107,117,116,110,65,106,79,83,106,71,103,103,82,55,53,76,55,116,43,52,74,53,43,90,55,56,82,100,121,48,83,80,47,78,47,98,77,86,112,122,82,51,77,108,67,105,75,102,120,108,65,68,78,57,76,104,85,47,98,56,50,54,57,89,76,112,74,109,55,110,98,98,68,107,113,77,117,55,101,51,65,54,109,115,48,57,102,47,47,70,111,82,79,76,74,115,89,57,76,114,84,110,81,77,67,54,52,103,122,80,43,71,79,82,85,61,60,47,77,111,100,117,108,117,115,62,60,69,120,112,111,110,101,110,116,62,65,81,65,66,60,47,69,120,112,111,110,101,110,116,62,60,47,82,83,65,75,101,121,86,97,108,117,101,62

Step 4: Combines client-side symmetric key and asymmetric key.

The symmetric key of step 2 and the public key of step 3 needs to get combined. That way, everything will be encrypted once.

Combine order

[Symmetric][Asymmetric]

191,204,10,109,135,63,93,79,140,0,249,103,250,188,212,210,60,82,83,65,75,101,121,86,97,108,117,101,62,60,77,111,100,117,108,117,115,62,120,47,122,85,51,80,118,52,106,105,57,106,122,101,108,99,72,101,65,98,55,55,121,53,104,51,106,97,113,106,73,99,100,66,72,107,48,66,67,75,97,81,104,82,115,119,100,74,118,77,85,80,57,119,69,116,107,117,116,110,65,106,79,83,106,71,103,103,82,55,53,76,55,116,43,52,74,53,43,90,55,56,82,100,121,48,83,80,47,78,47,98,77,86,112,122,82,51,77,108,67,105,75,102,120,108,65,68,78,57,76,104,85,47,98,56,50,54,57,89,76,112,74,109,55,110,98,98,68,107,113,77,117,55,101,51,65,54,109,115,48,57,102,47,47,70,111,82,79,76,74,115,89,57,76,114,84,110,81,77,67,54,52,103,122,80,43,71,79,82,85,61,60,47,77,111,100,117,108,117,115,62,60,69,120,112,111,110,101,110,116,62,65,81,65,66,60,47,69,120,112,111,110,101,110,116,62,60,47,82,83,65,75,101,121,86,97,108,117,101,62

Step 5: Encrypt the combined keys.

The combined key of the step #4 needs to get encrypted with the server-side public key received at step #1. So, the RSA key needs to be used by an RSA algorithm to transform the byte array into a crypted byte array.

Combined key encrypted byte array

23,204,182,141,81,208,38,170,213,213,82,35,23,159,172,2,167,209,101,121,230,86,57,3,192,217,191,178,12,16,107,140,5,231,171,233,91,108,186,181,96,51,204,253,98,169,110,158,186,135,219,46,138,252,30,7,158,162,114,0,134,32,50,236,12,232,169,222,16,241,71,222,94,234,24,132,237,39,65,50,18,42,68,232,225,83,4,34,100,159,116,61,197,148,107,251,214,62,66,185,217,150,125,99,234,84,152,44,241,77,69,248,220,18,26,34,131,56,194,120,5,45,147,118,21,115,85,223,250,223,214,160,33,181,118,33,177,227,58,91,51,246,103,79,133,144,84,17,25,230,5,201,136,18,49,236,110,48,24,131,15,235,250,36,80,36,208,246,63,206,216,45,4,237,60,39,118,185,104,71,46,253,25,246,215,11,87,188,222,237,170,29,94,142,255,72,229,66,182,128,214,29,171,167,77,206,250,21,209,83,36,16,216,105,164,72,207,174,101,72,15,131,45,117,229,43,102,72,14,185,169,58,56,108,49,181,82,113,80,206,84,82,143,122,223,244,120,29,240,137,118,25,75,111,201,131,230,194,125,1,139,52,254,240,100,62,149,209,121,209,27,35,171,226,158,3,29,55,151,40,253,130,135,10,94,215,229,142,207,64,127,21,18,198,248,11,211,156,179,197,37,15,139,38,150,53,51,238,175,52,117,4,68,132,167,19,208,126,194,31,206,107,174,213,248,232,67,16,32,176,46,45,131,10,111,190,0,67,52,117,38,125,131,127,91,108,142,94,161,60,50,25,133,195,80,2,73,128,91,22,233,6,35,184,234,224,60,136,117,206,11,153,73,18,20,41,196,251,178,13,47,216,168,252

The last operation before transmitting the key to the server, it to convert the byte array into a Base 64 representation if the bytes.

Combined key encrypted base 64

F8y2jVHQJqrV1VIjF5+sAqfRZXnmVjkDwNm/sgwQa4wF56vpW2y6tWAzzP1iqW6euofbLor8HgeeonIAhiAy7Azoqd4Q8UfeXuoYhO0nQTISKkTo4VMEImSfdD3FlGv71j5CudmWfWPqVJgs8U1F+NwSGiKDOMJ4BS2TdhVzVd/639agIbV2IbHjOlsz9mdPhZBUERnmBcmIEjHsbjAYgw/r+iRQJND2P87YLQTtPCd2uWhHLv0Z9tcLV7ze7aodXo7/SOVCtoDWHaunTc76FdFTJBDYaaRIz65lSA+DLXXlK2ZIDrmpOjhsMbVScVDOVFKPet/0eB3wiXYZS2/Jg+bCfQGLNP7wZD6V0XnRGyOr4p4DHTeXKP2Chwpe1+WOz0B/FRLG+AvTnLPFJQ+LJpY1M+6vNHUERISnE9B+wh/Oa67V+OhDECCwLi2DCm++AEM0dSZ9g39bbI5eoTwyGYXDUAJJgFsW6QYjuOrgPIh1zguZSRIUKcT7sg0v2Kj8

Step 6: Transmit the key to the server.

It is time to transmit the combined encrypted base 64 key to the server.

To request the key, use the API to do a POST call on the following URL:

- **[BaseURL]/Data/ExecuteKeyExchange.**

The payload (call body) is a JSON data and need to have the following structure:

```
{
  "key":
  {
    "CallNode": "PortalHost1",
    "KeyToken": "f24d7e12-2d9c-4806-acd6-40e9cf6f8168",
    "AsymetricClientPublicKeyAndClientSymetricXmlBase64": "[Combined key encrypted base 64]”
  }
}
```

KeyToken

Identification of the key returned by the server at step #1.

CallNode

The node having answered on the step #1.

AsymetricClientPublicKeyAndClientSymetricXmlBase64

The symmetric key and asymmetric keys combined, crypted and converted to base 64 obtained on step #5.

Server will answer to this call with the server-side generated symmetric key. The key will be crypted with the client-side public key.

```
{
  "FullSymetricKeyToken": "69a8a2e6-97d7-4e93-b71b-8a0f05739376",
  "SymetricHostHalfKey": "[Base64 crypted key]"
}
```

FullSymetricKeyToken

The “Token” that represent the identity of the encryption key.

SymetricHostHalfKey

The server-side portion of the complete key, crypted with the client-side public key transmitted on step #6.

Example of returned encrypted ½ symmetric key

tKulb9SRnl5WlpBPv0CQvfazA2HtVjYmpyomeUtVIVFwLPx2uxjtlOSRKPgwt3FZd1+XWC7nZVG90wssVcwqcUtnyxnlltxUSgr4lraWhwXbl0teEUozzwzCwGcGPs0Djwl2VKEiaXLGthS/oGHp/hjOzM2iPZBJjdDCsQYePRs=

Step 7: Decrypt the server-side received key.

The server-side returned key is a 16 bytes randomly generated array.

First, the base 64 string needs to be converted back to byte array.

Returned encrypted ½ symmetric key byte array

180,171,165,111,212,145,158,94,86,150,144,79,191,64,144,189,246,179,3,97,237,86,54,38,167,42,38,121,75,85,33,81,112,44,252,118,187,24,237,148,228,145,40,248,48,183,113,89,119,95,151,88,46,231,101,81,189,211,11,44,85,204,42,113,75,103,203,25,200,150,220,84,74,10,248,150,182,150,135,5,219,151,75,94,17,74,51,207,12,194,192,103,6,62,205,3,143,9,118,84,161,34,105,114,198,182,20,191,160,97,233,254,24,206,204,205,162,61,144,73,141,208,194,177,6,30,61,27

Use the client-side “private” key to decrypt the byte array.

Decrypted ½ server-side generated symmetric key
172,166,216,60,166,217,100,78,142,146,79,229,192,105,139,143

Step 8: Combine symmetric keys.

As the symmetric key is the combination of the client-side generated key and the server-side generated key, both keys need to get combined. The client-side is the one generated at step #2. The server-side is the one received and decrypted on step #7.

Combine order
[Client-Side][Server-Side]
191,204,10,109,135,63,93,79,140,0,249,103,250,188,212,210,172,166,216,60,166,217,100,78,142,146,79,229,192,105,139,143

The final encryption key used to transmit data is 32 bytes (512 bits).

Step 9: Perform the desired call.

The key obtained at step 8 needs to be used to perform a call. The key will be valid only for a single call.

To perform the call, the normal call JSON payload need to be generated first.

Call Payload example:

```
{
  "ClientID": "23555c85-0662-4ec1-9deb-e79a7e343503",
  "Username": "MyUser",
  "Password": "MyPassword",
}
```

The IV of 16 bytes is required for the encryption. You need to randomly generate the IV on each call.

Encryption IV example
224,161,44,181,77,1,40,66,177,216,61,253,231,43,104,139

Using Rijndael algorithm, the key and the IV, you need to encrypt the JSON “string” data required by the desired POST call. Note that the JSON encoding required is UTF8. Finally, convert the encryption result into Base 64 string.

Encrypted payload base 64
50jNrNz8i6jJhWbWNaWYjCLU+T0DTePEr0FM1Cn13JXey9adH5uVIaT8US6qWsm43iYN4IX4hoi0NZiVJN36

RSmtvINUvBDPHBiFm4jfcmhQFgKj5iowStiaX1rzdzu1VMJLf4b8Llg8X87SPXTqPBxPeR42sI0rxAQSJA677E C+fZ68uwbpUFkTwVGtBvpuq4BA5DOfnaAvQIyIL3l++g==

The IV also need to get converted into Base 64.

IV Base 64 representation
TE5PNQlDgUqyLGktbQtdwA==

Every call to the API needs to have the following format:

```
{
  "data": {
    "CallNode": "PortalHost1",
    "KeyToken": "69a8a2e6-97d7-4e93-b71b-8a0f05739376",
    "Base64IV": TE5PNQlDgUqyLGktbQtdwA==",
    "Base64CryptedData": "[Encrypted payload Base 64]"
  }
}
```

Base64CryptedData
The base 64 payload string, resulting from encryption of the JSON UTF8.

Base64IV
16 bytes IV used for the Rijndael encryption.

CallNode
The node having answered on the step #1.

KeyToken
Must be filled with the key token received on step #6 inside `FullSymetricKeyToken` property.

Step 10: Decrypt the returned result from the server.

The server will also encrypt the return payload using the same key.

Every return call has the following structure.

```
{
  "CryptedBase64Data": "[Encrypted JSON of the response return by the server]",
  "IV": [ 207, 200, 20, 179, 136, 105, 81, 68, 130, 3, 193, 21, 182, 134, 174, 252 ]
}
```

Base64CryptedData
Encrypted JSON of the response return by the server. It is Base 64.

IV
IV de 16 octets utilisé par le serveur lors de l'encryptions.

The same Rijndael algorithm with the same key needs to be used to decrypt the information returned.

➢ Error handling

Once decrypted, every call returns different data depending on the call. However, every call also returns the following properties at the root:

```
{
  "Errors": [{"ErrorMessage":"Bad Credentials","ErrorCode":403}],
  "HasError": false,
  "Messages": ""
}
```

Errors

Array or errors in case call returned error.

HasError

True if the Error property is not empty.

Messages

Contains a concatenation if all message in the error list property.

➢ Call Details

This section explains all the call you can perform on the TIB API.

Calls URL

All the calls follow this URL structure:

[BaseURL]/Data/[Method name]

Example

http://sandboxportal.tib.finance/Data/CreateSession

Call list

Customers

- Create a customer.
- List all service customers.
- Get a customer detail.
- List the customers based on external identification.
- Modify an existing customer.
- Delete a customer.

Payment methods

- Create bank account payment method.
- Create credit card payment method.
- Create Interac payment method.
- Change Interac Payment Method Question and Answer
- Get a specific payment method.
- List payment methods
- Change the default payment method of a customer.
- Delete payment method.

Payments / Transfers

- Create Bill.
- List Bill.
- Get Bill.
- Delete Bill.
- Create Payment.
- Create Direct Deposit.
- Create Interac Transfer.
- Create from ACP File.
- Create Free Operation.
- Delete Transfer.
- Revert Transfer.
- List Recuring.
- Delete Recuring process.
- Reporting of Operation

- List Executed Operations.
- Other data extraction methods.

Merchants

- Merchant basic information object.

Whitelabeling (UI Looks)

- Set WhiteLabeling
- Delete WhiteLabeling
- Get WhiteLabeling
- Update WhiteLabeling Values
- Get List of WhiteLabeling (related Services/Merchants)

Clients

- Create sub-client
- Set client default service fee settings
- Set client settings
- Get client settings

General

- Address global object.
- Languages enumeration.
- Currencies enumeration.
- Countries enumeration.
- Payment method type enumeration.
- Authorized Payment method type enumeration.
- Provinces / States enumeration.
- Transfer direction enumeration.
- Transfer type enumeration.
- Transfer frequency enumeration.
- Transaction Transfer type enumeration.
- Date type enumeration.
- Operation target enumeration.
- Operation type enumeration.
- Operation status enumeration.
- WhiteLabeling Level enumeration
- Bank operation result enumeration.
- Providers enumeration.

Sessions

Create a new session.

Method for creating a session ID.

Method name

CreateSession

Argument

Property	Description
ClientId	ID provided when creating the account.
Username	Username having access to the application.
Password	User's password.

Response

Property	Description
SessionId	Session ID to be used for every subsequent call.

Call Example

```
{
  ClientId: "0651af61-ae9e-41a7-898e-8ec775c6f8a1",
  Username: "User",
  Password: "Password"
}
```

Response example

```
{
  SessionId: "46a84f3f-c4fb-4d00-b3ff-caf6c85d06d4"
}
```

Customers

All methods and objects related to customer action.

Create a customer

Method to create a new customer.

Method name

CreateCustomer

Argument

Property	Description
SessionToken	Session ID obtained by calling the CreateSession method.
ServiceId	Service ID in which the customer will be added.
Customer	Customer information to create. See customer object.

Response

Property	Description
SessionId	Session ID to be used for every subsequent call.

Call Example

```
{
  SessionToken: "0651af61-ae9e-41a7-898e-8ec775c6f8a1",
  ServiceId: "862b96b1-85d4-406f-b55c-8f48f895f68e",
  Customer: {
    CustomerName: "Jackie Tester",
    CustomerExternalId: "c123-55",
    Language: 1,
    CustomerDescription: "VIP Customer"}
}
```

Response example

```
{
  CustomerId: "46a84f3f-c4fb-4d00-b3ff-caf6c85d06d4"
}
```

List all service customers

Method to extract all customers of a specific service.

Method name

ListCustomers

Argument

Property	Description
SessionToken	Session ID obtained by calling the CreateSession method.
ServiceId	Service ID in which the customer will be added.

Response

Property	Description
Customers	List of all customers of the service. See customer object.

Call example

```
{
  SessionToken: "0651af61-ae9e-41a7-898e-8ec775c6f8a1",
  ServiceId: "862b96b1-85d4-406f-b55c-8f48f895f68e"
}
```

Response example

```
Customers: [{
  CustomerId: "05880372-5c30-4f17-8796-c353bfaece3f",
  CustomerName: "Jackie Tester",
  CustomerExternalId: "c123-55",
  Language: 1,
  CustomerDescription: "VIP Customer",
  PaymentMethods: []
},{
  CustomerId: "0d38bce8-b4d1-445b-acf1-a921ab0eee4c",
  CustomerName: "Jackie Tester2",
  CustomerExternalId: "c123-56",
  Language: 1,
  CustomerDescription: "VIP Customer",
  PaymentMethods: []
}]
```

Get a customer details

Method to get a single customer with its payment methods based on the customer ID.

Method name

GetCustomer

Argument

Property	Description
SessionToken	Session ID obtained by calling the CreateSession method.
CustomerId	Desired customer identification.

Response

Property	Description
Customer	The desired customer information. See customer object.

Call example

```
{
  SessionToken: "0651af61-ae9e-41a7-898e-8ec775c6f8a1",
  CustomerId: "05880372-5c30-4f17-8796-c353bfaece3f",
}
```

Response example

```
Customer: {CustomerId: "05880372-5c30-4f17-8796-c353bfaece3f",
  CustomerName: "Jackie Tester",
  CustomerExternalId: "c123-55",
  Language: 1,
  CustomerDescription: "VIP Customer",
  PaymentMethods: [{
    PaymentMethodId: "b96d8827-5e57-4698-ab57-5601a9b973a2",
    IsCustomerAutomaticPaymentMethod: false,
    PaymentMethodType: 3,
    PaymentMethodDescription: "Compte principal",
    AccountPreview: "***-*****-***1234"
    PreauthorizedMerchants: [{
        MerchantId : “b96d8827-5e57-4698-ab57-5601a9b973a2”,
        MerchantName: “Name”
    }]
  }]
}
```

List the customers based on external identification

Because the external identification is not forced by the API to be unique, the call returns a list of matching customers. A normal usage would always return only 1 element as a good practice would be to ensure to provide unique external number per customer.

Method name

GetCustomersByExternalId

Argument

Property	Description
SessionToken	Session ID obtained by calling the CreateSession method.
ExternalCustomerId	Desired customer external identification.

Response

Property	Description
Customers	List of all customers of the TIB client matching the external identification. See customer object.

Call example

```
{
  SessionToken: "0651af61-ae9e-41a7-898e-8ec775c6f8a1",
  ExternalCustomerId: "c123-55",
}
```

Response example

```
Customers: [{
  CustomerId: "05880372-5c30-4f17-8796-c353bfaece3f",
  CustomerName: "Jackie Tester",
  CustomerExternalId: "c123-55",
  Language: 1,
  CustomerDescription: "VIP Customer",
  PaymentMethods: [{
    PaymentMethodId: "b96d8827-5e57-4698-ab57-5601a9b973a2",
    IsCustomerAutomaticPaymentMethod: false,
    PaymentMethodType: 3,
    PaymentMethodDescription: "Compte principal",
    AccountPreview: "***-*****-***1234"
    PreauthorizedMerchants: [{
        MerchantId : “b96d8827-5e57-4698-ab57-5601a9b973a2”,
        MerchantName: “Name”
    }]

  }]
}]
```

Modify an existing customer

Method to modify the customer information. This has no impact on its payment methods.

Method name

SaveCustomer

Argument

Property	Description
SessionToken	Session ID obtained by calling the CreateSession method.
Customer	Same “customer” customer object as “CreateCustomer”, except the ID is required. See customer object.

Response

Nothing returned (only error handling properties).

Call example

```
{
  SessionToken: "0651af61-ae9e-41a7-898e-8ec775c6f8a1",
  Customer: {
    CustomerId: "05880372-5c30-4f17-8796-c353bfaece3f",
    CustomerName: "Jackie Tester",
    CustomerExternalId: "c123-55",
    Language: 1,
    CustomerDescription: "VIP Customer"
  }
}
```

Delete a customer

Method name

DeleteCustomer

Argument

Property	Description
SessionToken	Session ID obtained by calling the CreateSession method.
CustomerId	Desired customer identification.

Response

Nothing returned (only error handling properties).

Call example

```
{
  SessionToken: "0651af61-ae9e-41a7-898e-8ec775c6f8a1",
  CustomerId: "05880372-5c30-4f17-8796-c353bfaece3f",
}
```

Customer object

The object to create and modify a customer. It's also the object returned when extracting customer.

CustomerId (Customer identification)

Not required during creation. This ID is a "Guid" formatted hexadecimal.

Customer identification restriction
Required
Example format: 05880372-5c30-4f17-8796-c353bfaece3f

CustomerName

The full name of the customer.

Customer name restrictions
Required
Max length: 150
Letters and numbers: accepted
French characters: accepted
Special characters accepted are: _\-@.,'!?;:&$%() and space*

CustomerExternalId

External identity that represents the end user inside another client's system.

Customer external identification restrictions
Not required
Max length: 150
Letters and numbers: accepted
French characters: accepted
Special characters accepted are: _\-@.,'!?;:&$%() and space*

Language

The language of the customer. If null at creation, the primary merchant default language will be used. See languages enumeration.

CustomerDescription

An optional description to help described the customer. Can be used to categorize the customers.

Customer description restrictions
Not required
Max length: 150
Letters and numbers: accepted
French characters: accepted
Special characters accepted are: _\-@.,'!?;:&$%() and space*

PaymentMethods

Array of payment method objects. Not used for customer creation or modification; only used in returned data. See payment method object for details.

Object example

```
{
  CustomerId: "05880372-5c30-4f17-8796-c353bfaece3f",
  CustomerName: "Jackie Tester",
  CustomerExternalId: "c123-55",
  Language: 1,
  CustomerDescription: "VIP Customer",
  PaymentMethods: [{
      PaymentMethodId: "b96d8827-5e57-4698-ab57-5601a9b973a2",
      IsCustomerAutomaticPaymentMethod: false,
      PaymentMethodType: 3,
      PaymentMethodDescription: "Compte principal",
      AccountPreview: "***-*****-***1234",
      PreauthorizedMerchants: [{
        MerchantId : “”,
        MerchantName: “”
    }]

  }]
}
```

Payment methods

The payment methods are different payment ways supported by the API and are related to customer.

Create bank account payment method

The method to add a bank account to a customer for collecting or deposit money.

Method name

CreateDirectAccountPaymentMethod

Argument

Property	Description
SessionToken	Session ID obtained by calling the CreateSession method.
CustomerId	ID of the customer to add the payment method.
IsCustomerAutomaticPaymentMethod	Determine if the payment method become the automatic method to use when customer is used, and payment method is not specified.
IscustomerwithdrawalAuthorized	Determines if the customer's Withdrawal is authorized.
Account	See bank account object.

Response

Property	Description
PaymentMethodId	The payment method ID of the created bank account.

Call Example

```
{
  SessionToken: "0651af61-ae9e-41a7-898e-8ec775c6f8a1",
  CustomerId: "986cec31-be7a-4d7c-a703-0f4c67791362",
  IsCustomerAutomaticPaymentMethod: "true",
  Account: {
    Owner: "Jeff Testing",
    AccountName: "Personal bank account",
    BankNumber: "003",
    InstitutionNumber: "12345",
    AccountNumber: "9876543"
  }
}
```

Response example

```
{
  PaymentMethodId: "06e45951-b19c-4001-8e00-d6257ef1ac1c"
}
```

Create credit card payment method

Method to add a credit card payment method to an existing customer. This payment method can only be used to collect money.

Method name

CreateCreditCardPaymentMethod

Argument

Property	Description
SessionToken	Session ID obtained by calling the CreateSession method.
CustomerId	ID of the customer to add the payment method.
IsCustomerAutomaticPaymentMethod	Determine if the payment method become the automatic method to use when customer is used, and payment method is not specified.
CreditCard	See credit card object.

Response

Property	Description
PaymentMethodId	The payment method ID of the created credit card.

Call Example

```
{
  SessionToken: "0651af61-ae9e-41a7-898e-8ec775c6f8a1",
  CustomerId: "986cec31-be7a-4d7c-a703-0f4c67791362",
  IsCustomerAutomaticPaymentMethod: "true",
  CreditCard: {
     Pan: 4242424242424242,
     Cvd: 123,
     ExpirationMonth: 12,
     ExpirationYear: 24,
     CreditCardDescription: "Test card",
     CardOwner: "Johny Cardholder",
     CreditCardRegisteredAddress: {
       StreetAddress: "1 Testing road",
       AddressCity: "Testcity",
       ProvinceStateId: 10,
       CountryId: 1,
       PostalZipCode: "H1H1H1"
     }
   }
}
```

Response example

```
{
  PaymentMethodId: "06e45951-b19c-4001-8e00-d6257ef1ac1c"
}
```

Create Interac payment method

Method to add Interac payment method to a customer. This method can be used to collect or deposit money.

Method name

CreateInteracPaymentMethod

Argument

Property	Description
SessionToken	Session ID obtained by calling the CreateSession method.
CustomerId	ID of the customer to add the payment method.
IsCustomerAutomaticPaymentMethod	Determine if the payment method become the automatic method to use when customer is used, and payment method is not specified.
InteracInformation	See Interac object.

Response

Property	Description
PaymentMethodId	The payment method ID of the created Interac information.

Call Example

```
{
  SessionToken: "0651af61-ae9e-41a7-898e-8ec775c6f8a1",
  CustomerId: "986cec31-be7a-4d7c-a703-0f4c67791362",
  IsCustomerAutomaticPaymentMethod: "true",
  InteracInformation: {
    Description: "Interac Test",
    Owner: "Kelly Interac",
    TargetEmailAddress: "kinterac@dummytest.com",
    TargetMobilePhoneNumber: "888-123-4567",
    InteracQuestion: "Remember the fruit",
    InteracAnswer: "Orange"
  }
}
```

Response example

```
{
  PaymentMethodId: "06e45951-b19c-4001-8e00-d6257ef1ac1c"
}
```

Change Interac Payment Method Question and Answer

It is not possible to change an existing payment method question and answer because payment maybe be in execution process with the actual payment method information. However, it is possible to perform a call with new question and answer that will create a new payment method and logically delete the old one.

Method name

ChangeInteracPaymentMethodQuestionAndAnswer

Argument

Property	Description
PaymentMethodId	Payment method ID to replace.
InteracQuestion	The new question. See Interac object for restriction.
InteracAnswer	The new answer. See Interac object for restriction.

Response

Property	Description
PaymentMethodId	The new payment method with all the previous information but with the new question and answer.

Call Example

```
{
  SessionToken: "0651af61-ae9e-41a7-898e-8ec775c6f8a1",
  PaymentMethodId: "06e45951-b19c-4001-8e00-d6257ef1ac1c",
  InteracQuestion: "Remember the fruit",
  InteracAnswer: "Orange"
}
```

Response example

```
{
  PaymentMethodId: "b96d8827-5e57-4698-ab57-5601a9b973a2"
}
```

Get a specific payment method

Get the payment method information based on a payment method unique identifier.

Method name

GetPaymentMethod

Argument

Property	Description
PaymentMethodId	Payment method ID to extract.

Response

Property	Description
PaymentMethod	The payment method information. See Payment method generic object.

Call Example

```
{
  SessionToken: "0651af61-ae9e-41a7-898e-8ec775c6f8a1",
  PaymentMethodId: "06e45951-b19c-4001-8e00-d6257ef1ac1c"
}
```

Response example

```
{
  PaymentMethod: {
    PaymentMethodId: "b96d8827-5e57-4698-ab57-5601a9b973a2",
    Owner: "Fanny Tester",
    PaymentMethodDescription: "Compte principal",
    IsCustomerAutomaticPaymentMethod: false,
    PaymentMethodType: 3,
    AccountPreview: "***-*****-***1234",

    PreauthorizedMerchants: [{

        MerchantId : "b96d8827-5e57-4698-ab57-5601a9b973a2",
        MerchantName: "Name merchant"
    }]

  }
}
```

List payment methods

Allow to list all payment methods of a customer.

Method name

ListPaymentMethods

Argument

Property	Description
CustomerId	ID of the customer to list its payment methods.

Response

Property	Description
PaymentMethods	The payment methods information of the customer. See Payment method generic object.

Call Example

```
{
  SessionToken: "0651af61-ae9e-41a7-898e-8ec775c6f8a1",
  CustomerId: "986cec31-be7a-4d7c-a703-0f4c67791362"
}
```

Response example

```
{
  PaymentMethods: [{
    PaymentMethodId: "b96d8827-5e57-4698-ab57-5601a9b973a2",
    Owner: "Fanny Tester",
    PaymentMethodDescription: "Compte principal",
    IsCustomerAutomaticPaymentMethod: false,
    PaymentMethodType: 3,
    AccountPreview: "***-*****-***1234",
    PreauthorizedMerchants: [{

        MerchantId : "b96d8827-5e57-4698-ab57-5601a9b973a2",
        MerchantName: "Name merchant"
    }]

  },{
    PaymentMethodId: "886fe591-bb09-4442-84d4-509293044d90",
    Owner: "Kelly Cardson",
    PaymentMethodDescription: "Test Credit Card",
    IsCustomerAutomaticPaymentMethod: true,
    PaymentMethodType: 1,
    AccountPreview: "**************4242",
    ExpirationDate: "2024-12-01T00:00:00",
    PreauthorizedMerchants: [{

        MerchantId : "b96d8827-5e57-4698-ab57-5601a9b973a2",
        MerchantName: "Name merchant"
    }]

  }]
}
```

Change the default payment method of a customer

Define the default payment method to be used for a customer when the payment method is not specified directly.

Method name

SetDefaultPaymentMethod

Argument

Property	Description
PaymentMethodId	Payment method ID to set default
CustomerId	The customer ID possessing the payment method

Response

Nothing returned (only error handling properties).

Call Example

```
{
  SessionToken: "0651af61-ae9e-41a7-898e-8ec775c6f8a1",
  PaymentMethodId: "06e45951-b19c-4001-8e00-d6257ef1ac1c",
  CustomerId: "986cec31-be7a-4d7c-a703-0f4c67791362"
}
```

Delete payment method

Method name

DeletePaymentMethod

Argument

Property	Description
PaymentMethodId	Payment method to delete

Response

Nothing returned (only error handling properties).

Call Example

```
{
  SessionToken: "0651af61-ae9e-41a7-898e-8ec775c6f8a1",
  PaymentMethodId: "06e45951-b19c-4001-8e00-d6257ef1ac1c"
}
```

Bank account object

The object representing the bank account of a customer inside the “account” payment method.

Owner

The account owner name.

Owner restrictions
Required
Max length: 150
Letters and numbers: accepted
French characters: accepted
Special characters accepted are: _\-@.,'!?;:&$%() and space*

AccountName

A small description of the account.

Account name restrictions
Required
Max length: 150
Letters and numbers: accepted
French characters: accepted
Special characters accepted are: _\-@.,'!?;:&$% and space*

BankNumber

The bank identification number. 3 digits for Canadian banks.

Bank number restrictions
Required
Max length: 3
Letters: not accepted
Numbers: accepted

InstitutionNumber

The account TRANSIT number. 5 digits for Canadian banks.

Institution number restrictions
Required
Max length: 5
Letters: not accepted
Numbers: accepted

AccountNumber

The bank account number. Including the check digit if there is one.

Account number restrictions
Required
Max length: 15
Letters: not accepted
Numbers: accepted

Object example

```
{
  Owner: "Jeff Testing",
  AccountName: "Personal bank account",
  BankNumber: "003",
  InstitutionNumber: "12345",
  AccountNumber: "9876543"
}
```

Credit card object

The object representing a credit card of a customer inside the credit card payment method.

CreditCardDescription

Description to identify the card.

Credit card description restrictions
Not required
Max length: 150
Letters and numbers: accepted
French characters: accepted
Special characters accepted are: _\-@.,'!?;:&$% and space*

CardOwner

The name on the credit card.

Card owner restrictions
Required
Max length: 150
Letters and numbers: accepted
French characters: not accepted
Special characters accepted are: - and space

Pan

The credit card number.

PAN restrictions
Required
Min length: 14
Max length: 16
Letters: not accepted
Numbers: accepted

ExpirationMonth

Card expiration month number.

Expiration month restrictions
Required
Range: 1 to 12
Letters: not accepted
Numbers: accepted

ExpirationYear

Card expiration year number (2 digits only).

Expiration year restrictions
Required
Range: 1 to 99
Letters: not accepted
Numbers: accepted

CVD

The card verification code. It is required for antifraud process.

CVD restrictions
Not required
Min length: 14
Max length: 16
Letters: not accepted
Numbers: accepted

CreditCardRegisteredAddress

The card related street address. Help for antifraud check. See Address global object.

Credit card address restrictions
Not required
(See Address global object)

Object example


```
{
  Pan: 4242424242424242,
  Cvd: 123,
  ExpirationMonth: 12,
  ExpirationYear: 24,
  CreditCardDescription: "Test card",
  CardOwner: "Johny Cardholder",
  CreditCardRegisteredAddress: {
    StreetAddress: "1 Testing road",
    AddressCity: "Testcity",
    ProvinceStateId: 10,
    CountryId: 1,
    PostalZipCode: "H1H1H1"
  }
}
```

Interac object

The object representing a credit card of a customer inside the credit card payment method.

Description

This correspond to a memo to be displayed to the user.

Description restrictions
Required
Max length: 140
Letters and numbers: accepted
French characters: accepted
Special characters accepted are: _\-@.,'!?;:&$% and space*

Owner

The name of the person having the account.

Owner restrictions
Required
Max length: 80
Letters and numbers: accepted
French characters: accepted
Special characters accepted are: _@.,' and space

TargetEmailAddress

The question that will be send to the target to accept Deposit.

Target email address restrictions
*Required**
Max length: 80
Forced format: Valid email address

*MOBILE PHONE OR EMAIL IS REQUIRED, NOT BOTH.

TargetMobilePhoneNumber

Target mobile phone number restrictions
*Required**
Forced format: ##########

*MOBILE PHONE OR EMAIL IS REQUIRED, NOT BOTH.

InteracQuestion

Interac question restrictions
*Required**
Max length: 40
Letters and numbers: accepted
French characters: accepted
Special characters accepted are: _@.,' and space

* NOT REQUIRED FOR COLLECTION, ONLY FOR DEPOSIT

InteracAnswer

Interac answer restrictions
*Required**
Max length: 40
Letters and numbers: accepted
French characters: accepted
Special characters accepted is: -

* NOT REQUIRED FOR COLLECTION, ONLY FOR DEPOSIT

Object example

```
{
  "Description": "Interac Test",
  "Owner": "Kelly Interac",
  "TargetEmailAddress": "kinterac@dummytest.com",
  "TargetMobilePhoneNumber": "888-123-4567",
  "InteracQuestion": "Remember the fruit",
  "InteracAnswer": "Orange"
}
```

Payment method generic object

This object is a generic object used to list different types of payment method together or return a payment method summary.

PaymentMethodId

The payment method unique identification.

Owner

The payment method owner full name. Name on the card for credit card type.

PaymentMethodDescription

Description to recognize the card.

IsCustomerAutomaticPaymentMethod

Determine if the payment method become the automatic method to use when customer is used, and payment method is not specified.

PaymentMethodType

Bank account, Credit card or Interac. See payment method type enumeration.

Account preview

Visual information of the payment method number.

ExpirationDate

Credit card expiration date.

PreauthorizedMerchants

List of merchants that are Authorized for PPA on the paymentMethod

Account example

```
{
  PaymentMethodId: "b96d8827-5e57-4698-ab57-5601a9b973a2",
  Owner: "Fanny Tester",
  PaymentMethodDescription: "Compte principal",
  IsCustomerAutomaticPaymentMethod: false,
  PaymentMethodType: 3,
  AccountPreview: "***-*****-***1234",

  PreauthorizedMerchants: [{

        MerchantId : "b96d8827-5e57-4698-ab57-5601a9b973a2",
        MerchantName: "Name merchant"
    }]

}
```

Credit card example

```
{
  PaymentMethodId: "886fe591-bb09-4442-84d4-509293044d90",
  Owner: "Kelly Cardson",
  PaymentMethodDescription: "Test Credit Card",
  IsCustomerAutomaticPaymentMethod: false,
  PaymentMethodType: 1,
```

```
  AccountPreview: "***************4242",
  ExpirationDate: "2024-12-01T00:00:00",
  PreauthorizedMerchants: [{
        MerchantId : "b96d8827-5e57-4698-ab57-5601a9b973a2",
        MerchantName: "Name merchant"
    }]
}
```

MerchantIdName Generic object

Account Information

MerchantId

The Merchant Unique Id

Merchant Name

The Merchant Name

Bills / Payments / Transfers

Create Bill

Method to create an invoice one which payment will be added thereafter.

Method name

CreateBill

Argument

Property	Description
BreakIfMerchantNeverBeenAuthorized	If the specified merchant is not already authorized, the bill can be created. It will eventually break when the payment related to the bill tried to execute. This property allows to specify the desire of breaking the bill creation if the merchant is not already authorized.
BillData	The bill information to create. See bill object.

Response

Property	Description
BillId	The unique identification of the created bill.

Call Example

```
{
  SessionToken: "0651af61-ae9e-41a7-898e-8ec775c6f8a1",
  BreakIfMerchantNeverBeenAuthorized: true,
```

```
  BillData: {
    MerchantId: "122c2650-6418-469a-a2ce-4fdc02c601ac",
    BillTitle: "test interac",
    BillDescription: "test interac",
    BillAmount: 1,
    ExternalSystemBillNumber1: "",
    ExternalSystemBillNumber2: "",
    ExternalSystemBillNumber3: "",
    BillCurrency: 2,
    Language: 1,
    RelatedCustomerId: "986cec31-be7a-4d7c-a703-0f4c67791362"
  }
}
```

Response example

```
{
  BillId: "45c35985-2b94-4abd-a608-8685aeb75226"
}
```

List Bills

List all the bills of a service or a specific merchant of the service.

Method name

ListBills

Argument

Property	Description
ServiceId	Required: The service ID need to be specified to list only the bills of a specific service.
MerchantId	Optional: The merchant identification can be specified to return only bills of a specific merchant.
FromDateTime	Extract bills after this date. It uses the creation date and time of the bills.
ToDateTime	Extract bills before this date. It uses the creation date and time of the bills.

Response

Property	Description
Bills	Array of bills. See Bill object.

Call Example

```
{
  SessionToken: "86ee144e-9c27-4039-aa1f-43be0042aecf",
  ServiceId: "ae18de50-74c0-4fac-bee1-fbda4c6b8355",
  MerchantId: "122c2650-6418-469a-a2ce-4fdc02c601ac ",
  FromDateTime: "2021-02-16T13:45:00.000Z",
  ToDateTime: "2021-02-16T21:00:00.000Z"
}
```

Response example

```
{
  Bills: [{
    BillId: "45c35985-2b94-4abd-a608-8685aeb75226",
    CreatedDate: "2021-01-28T13:47:27.443-05:00",
    MerchantId: "122c2650-6418-469a-a2ce-4fdc02c601ac",
    BillTitle: "test interac",
    BillDescription: "test interac",
    BillAmount: 1,
    ExternalSystemBillNumber1: "",
    ExternalSystemBillNumber2: "",
    ExternalSystemBillNumber3: "",
    BillCurrency: 2,
    Language: 1,
    RelatedCustomerId: "986cec31-be7a-4d7c-a703-0f4c67791362"
  }]
}
```

Get Bill

Extract a single bill information using the bill unique identification.

Method name

GetBill

Argument

Property	Description
BillId	The identification of the bill to extract.

Response

Property	Description
Bill	See Bill object.

Call Example

```
{
  SessionToken: "86ee144e-9c27-4039-aa1f-43be0042aecf",
  BillId: "45c35985-2b94-4abd-a608-8685aeb75226"
}
```

Response example

```
{
  Bill: {
    BillId: "45c35985-2b94-4abd-a608-8685aeb75226",
    CreatedDate: "2021-01-28T13:47:27.443-05:00",
    MerchantId: "122c2650-6418-469a-a2ce-4fdc02c601ac",
    BillTitle: "test interac",
    BillDescription: "test interac",
    BillAmount: 1,
    ExternalSystemBillNumber1: "",
    ExternalSystemBillNumber2: "",
    ExternalSystemBillNumber3: "",
    BillCurrency: 2,
    Language: 1,
    RelatedCustomerId: "986cec31-be7a-4d7c-a703-0f4c67791362"
  }
}
```

Delete Bill

Remove a bill from the API. Note that this is only a logical delete and this action will not stop in progress transaction.

Method name

DeleteBill

Argument

Property	Description
BillId	The identification of the bill to delete.

Response

Nothing returned (only error handling properties).

Call Example

```
{
  SessionToken: "0651af61-ae9e-41a7-898e-8ec775c6f8a1",
  PaymentMethodId: "06e45951-b19c-4001-8e00-d6257ef1ac1c"
}
```

Create Payment

Add a new payment to apply on an existing bill.

Method name

CreatePayment

Argument

Property	Description
BillId	A payment needs to be related to the bill ID of a previously created bill.
SetPaymentCustomerFromBill	Determines if the payment use the customer ID or not.
CustomerEmail	Set the customer email to send the request by email to the customer. It allows the customer to fill its payment method information by himself. This requires the Payment Flow to be set to Anonymous.
ExternalReferenceId	An external reference information for the payment.
SafetyToBreakIfOverRemainingBillAmount	When doing multiple payment on the same bill, this allows to force the system to break if the new payment push the sum of collected amounts over the bill amount.
AutorizedPaymentMethod	Allow to protect the payment from being processed with any payment method type. It forces the desired payment method type. See Authorized Payment Method enumeration.
PaymentInfo	See next table for payment information
StatementDescription	Statement Description
AskForCustomerConsent	Ask for customer's Consent .

Argument sub object: Payment information

Property	Description
PaymentFlow	This is the only "required" properties inside the "PaymentInfo". This controls the way the API will process the payment. See next table for details.
RelatedCustomerId	Can set the identity of the customer that will pay the payment. This parameter is optional because the system will use the customer of the bill by default.
DueDate	Can tell the system when the payment needs to be process. This information is optional because it is not specified, the system will process the payment as soon as possible (based on the payment method restrictions).

Language	“FR” or “EN”. This is optional because it is automatically set to bill default language if omitted. See language enumeration.
Amount	The amount of the payment to be process. This is optional because it will be equal to the bill amount by default.
ForcedCustomerPaymentMethodId	It is possible to specify the payment method ID desired. This is required if the customer related has no “default payment method” set on the customer profile. If specified, the payment method ID needs to be owned by the customer of the payment.
TransferFrequency	It is possible when creating a payment to specify the payment is a recurring payment. The recurring date will be based on the first due date. See Transfer frequency enumeration.
GroupId	When creating multiple payment, this property can be used to specify a “batch” identification to recognize all payments together.
AskForCustomerConsent	Ask for customer’s consent.

Payment Flow Enumeration

Value	Title	Description
-1	Unknown	
0	Not set	
1	Anonymous	Need customer email property set. The customer will receive an email with a link to create his payment method and proceed to the payment.
2	Known customer must use pre-saved payment method	Need customer email property set. The customer will receive an email with a link to select his payment method from the payment method already related to this customer.
3	Known customer can manage payment method.	Need customer email property set. The customer will receive an email with a link to select payment method or create a new one.
4	Known customer can fully manage payment method.	Same as type “3” but the customer can set an automatic payment method. This can therefore allow the system to process further payment directly without sending email.
5	Known customer automatic process.	Automatically process the payment based on the customer default payment method. If no default payment method is available, the API will return an error.

6	Known customer automatic process with forced payment method.	Automatically process the payment based on the customer Forced customer payment method defined.
7	Auto select easier	Tell the API to determined based on the context what is the best way to process the payment. When this option is selected, the system will return the chosen type in the returned data. When this mode is selected, the caller can use the returned execution type for further action.
8	Auto select except automatic	Same as number "7", but the type 5 and 6 will never be selected.
9	AnonymousOnlinePaymentWithConsent	Payment must be send to user for online entry

Response

Property	Description
PaymentId	The created payment unique identifier
AutoSelectPaymentFlowResult	Enumeration describing the payment flow having been automatically selected when the Payment Flow was set to an "Auto select" mode. See the Payment flow enumeration. It's the same enumeration except the "auto" modes are not possible.
PaymentFlowParsingResult	Enumeration of the result of the interpretation of the context in relation with the payment flow selected. This should be always "success" then mode is "auto-select" because the flow automatically adapted to the context.

Payment Flow Parsing Result Enumeration

Value	Title	Description
-1	Unknown	
0	Not set	
1	Success	This value means the selected payment flow can be executed.
2	Invalid Payment Flow	Means an invalid payment flow number.
3	Bill is not related to known customer	Occurred if the bill is not found within the specified customer.
4	Bill customer has no payment method	Occurred when trying the create a payment and the related customer has no payment method.

5	Bill customer has no default payment method	Occurred when trying the create a payment and the related customer has payment method, but none are set to be the default one.
6	Forced payment method Id needed	Occurred if the mode is set to “6” and the forced payment method is not set in the create payment call.
7	Bill customer does not have specified payment method Id	Occurred if specified the forced payment method but the specified one is not related to the customer.
8	Anonymous must have Email	Occurred then flow required to send an email to the customer but the customer email address is not specified.
9	CustomerPPAConsentIsNeeded	this is needed for when a payment method is not Authorized for PPA

Call example #1
This call will create a payment of amount equal to the related bill amount, to be processed as soon as possible. The mode “7” is set to it will be executed the easier mode.

```
{
  SessionToken: "86ee144e-9c27-4039-aa1f-43be0042aecf",
  BillId: "45c35985-2b94-4abd-a608-8685aeb75226"
  SetPaymentCustomerFromBill: true,
  PaymentInfo: {
    PaymentFlow: 7
  }
}
```

Response example #1

```
{
  PaymentId: "c9a521d5-60a1-4398-8f6c-7462797d584c",
  AutoSelectPaymentFlowResult: 5,
  PaymentFlowParsingResult: 1
}
```

The call example return means the context has been parse with success and the selected execution mode is “Known customer automatic process.”

Call example #2

Create payment forcing the customer identification, specifying the amount and with automatic process execution mode.

```
{
  SessionToken: "4840fe52-5ab2-43be-8a07-4354f263431b",
  BillId: "45c35985-2b94-4abd-a608-8685aeb75226",
  SetPaymentCustomerFromBill: "false",
  PaymentInfo: {
    PaymentFlow: 6,
    RelatedCustomerId: "986cec31-be7a-4d7c-a703-0f4c67791362",
    DueDate: "2021-02-16T16:10:19.000Z",
    PaymentAmount: 1.22
  }
}
```

Create Direct Deposit

Create a deposit to account.

This Method is obsolete, Use `CreateFreeOperation`.

Method name

CreateDirectDeposit

Argument

Property	Description
OriginMerchantId	The merchant ID from which the money will be taken.
DestinationAccount	The bank account where the money will be deposit. See Bank Account Object.
DepositDueDate	Optional. Determine when the money has to get deposit. Empty means as soon as possible.
Amount	The amount to deposit.
Currency	"CAD" or "USD". This is optional because it is automatically set to merchant default currency if omitted. See currency enumeration.
Language	"FR" or "EN". This is optional because it is automatically set to merchant default language if omitted. See language enumeration.
ReferenceNumber	External number to recognize the transaction.
StatementDescription	Statement Description

Response

Nothing returned (only error handling properties).

Call Example

```
{
  SessionToken: "86ee144e-9c27-4039-aa1f-43be0042aecf",
  OriginMerchantId: "122c2650-6418-469a-a2ce-4fdc02c601ac ",
  DestinationAccount: {
    Owner: "Jeff Testing",
    AccountName: "Personal bank account",
    BankNumber: "003",
    InstitutionNumber: "12345",
    AccountNumber: "9876543"
  },
  DepositDueDate: "2021-02-16T16:10:19.000Z",
  Currency: 1
  Language: 1,
  ReferenceNumber: "C12343-324",
}
```

Create Interac Transfer

Create an Interac transfer to collect or deposit via Interac system.

Method name

CreateDirectInteracTransaction

Argument

Property	Description
MerchantId	The merchant ID from which the money will be taken or the merchant that will receive the money if collection.
InteracInformation	The Interac information. See Interac Object.
TransferDirection	Collect or deposit. See Transfer direction enumeration.
DueDate	Optional. Determine the desired date for the transfer. Empty means as soon as possible.
Amount	The amount to collect or deposit.
Currency	“CAD” or “USD”. This is optional because it is automatically set to merchant default currency if omitted. See currency enumeration.
Language	“FR” or “EN”. This is optional because it is automatically set to merchant default language if omitted. See language enumeration.
ReferenceNumber	External number to recognize the transaction.
StatementDescription	Statement Description

Response

Nothing returned (only error handling properties).

Call Example

```
{
  SessionToken: "86ee144e-9c27-4039-aa1f-43be0042aecf",
  MerchantId: "122c2650-6418-469a-a2ce-4fdc02c601ac",
  InteracInformation: {
    "Description": "Interac Test",
    "Owner": "Kelly Interac",
    "TargetEmailAddress": "kinterac@dummytest.com",
    "TargetMobilePhoneNumber": "888-123-4567",
    "InteracQuestion": "Remember the fruit",
    "InteracAnswer": "Orange"
  },
  DueDate: "2021-02-16T16:10:19.000Z",
  Currency: 1
  Language: 1,
  ReferenceNumber: "C12343-324",
}
```

Create from ACP file

Create a batch of collection or deposit using the standard ACP file format.

Method name

CreateTransactionFromRaw

Argument

Property	Description
MerchantId	The merchant ID from which the money will be taken or the merchant that will receive the money if collection.
RawAcpFileContent	Text containing the ACP file format. Supported format are 1465 characters per line or 120 characters per line.

Response

Property	Description
TransactionsGroupId	An identification regrouping the batch processed.

Call Example

```
{
  SessionToken: "0651af61-ae9e-41a7-898e-8ec775c6f8a1",
  MerchantId: "122c2650-6418-469a-a2ce-4fdc02c601ac",
  RawAcpFileContent: "[THE ACP FILE CONTENT TEXT]"
}
```

Response example

```
{
  TransactionsGroupId: "TIBGID-132162625479516834"
}
```

NOTE: CREATED TRANSFERS RECEIVED BY ACP FILE WILL RESULT IN FREE OPERATION FOR STATUS COLLECTION.

Create Free Operation

Create free operation to deposit or collect a customer directly.

Method name

CreateFreeOperation

Argument

Property	Description
MerchantId	The merchant ID from which the money will be taken or the merchant that will receive the money if collection.
CustomerId	The customer identification related to the payment method
PaymentMethodId	The payment method that will receive the money or that will be collected.
TransferType	FreeCollection or FreeDeposit authorized. See Transfer type enumeration.
ReferenceNumber	External number to recognize the transaction.
Amount	The amount to transfer.
Language	"FR" or "EN". See language enumeration.
TransactionDueDate	Optional. Determine when the transaction is desired. Empty means as soon as possible.
GroupId	When creating multiple payment, this property can be used to specify a "batch" identification to recognize all payments together.
TransferFrequency	It is possible when creating a payment to specify the payment is a recurring payment. The recurring date will be based on the first due date. See Transfer frequency enumeration.
StopSameIdentifications	A nullable Boolean to insure Not To Create Duplicate Transfers (Requires "GroupId" and "ReferenceNumber")
`StatementDescription`	A statement Description.

Response

Property	Description
PaymentId	The identification of the created free operation.

Call Example

```
{
  SessionToken: "0651af61-ae9e-41a7-898e-8ec775c6f8a1",
  MerchantId: "122c2650-6418-469a-a2ce-4fdc02c601ac",
  PaymentMethodId: "b96d8827-5e57-4698-ab57-5601a9b973a2",
  TransferType: 1,
  ReferenceNumber: "C123-01312",
  Amount: 12.44,
  Language: 1,
  TransactionDueDate: "2021-02-16T16:10:19.000Z",
  GroupId: "HT123123",
  TransferFrequency: 0

}
```

Response example

```
{
  PaymentId: "c9a521d5-60a1-4398-8f6c-7462797d584c"
}
```

Delete Transfer

Remove a free operation or payment from the system.

Method name

DeletePayment

Argument

Property	Description
PaymentId	The identification number of the transfer or the payment.

Response

Nothing returned (only error handling properties).

Two different messages get returned in case of Error

`"Payment does not exist.":` When the Transfer does nor Exist

`"Payment is already executed. Cannot delete payment.":` When the Transfer is Executed.

Call Example

```
{
  SessionToken: "0651af61-ae9e-41a7-898e-8ec775c6f8a1",
  PaymentId: "c9a521d5-60a1-4398-8f6c-7462797d584c"
}
```

Revert Transfer

Allows to revert a payment or a free operation.

Method name

RevertTransfer

Argument

Property	Description
TransferId	The identification number of the transfer or the payment.

Response

Nothing returned (only error handling properties).

Call Example

```
{
  SessionToken: "0651af61-ae9e-41a7-898e-8ec775c6f8a1",
  TransferId: "c9a521d5-60a1-4398-8f6c-7462797d584c"
}
```

List Recuring

When payment or free operation is created using “TransferFrequency”, the API will automatically create next payment after the first is created. This method allows to list the transfer having “active” recuring activated.

Method name

GetRecuringTransfers

Argument

Property	Description
ServiceID	The identification of the service to list its active recuring transfers.

Response

Property	Description
RecuringTransfers	List of active recuring transfer for the service. See Recuring transfers enumeration.

Call Example

```
{
  SessionToken: "0651af61-ae9e-41a7-898e-8ec775c6f8a1",
  ServiceId: "862b96b1-85d4-406f-b55c-8f48f895f68e"
}
```

Response example

```
{
  RecuringTransfers: [{
    NextRecuringDate: “2021-01-28T13:47:27.443-05:00”,
    RecuringTransferId: "89d720f2-78ae-4816-8fda-0099aa867c38",
    RecuringMode: 2,
    RecuringRefDate: "2021-02-12T13:47:27.443-05:00",
    CreatedDate: "2021-01-28T13:47:27.443-05:00",
    RelatedMerchantId: "122c2650-6418-469a-a2ce-4fdc02c601ac",
    RelatedMerchantName: "Company merchant",
    CustomerName: "Client",
    RelatedCustomerId: "986cec31-be7a-4d7c-a703-0f4c67791362"
    Amount: 98.6
  }]
}
```

Delete Recuring Process

Delete recuring process added when using “TransferFrequency” while creating payment. The recuring identification can be known using the List Recuring method.

Method name

DeleteRecuringTransfer

Argument

Property	Description
RecuringTransferId	The identification of the recuring process to delete.

Response

Nothing returned (only error handling properties).

Call Example

```
{
  SessionToken: "0651af61-ae9e-41a7-898e-8ec775c6f8a1",
  RecuringTransferId: "89d720f2-78ae-4816-8fda-0099aa867c38"
}
```

Bill object

Object representing a bill to be used during bill creation or returned when querying the bill information from the API.

MerchantId

Bill related merchant identification.

Bill related merchant restriction
Required
Example format: 05880372-5c30-4f17-8796-c353bfaece3f

Title

The title of the bill to recognize it and give information to the destination customer.

Bill title restrictions
Required
Max length: 150
Letters and numbers: accepted
French characters: accepted
Special characters accepted are: _\-@.,'!?;:&$%()/ and space*

Description

Description to be displayed to the customer for communication.

Bill description restrictions
Required
Max length: 1000
Letters and numbers: accepted
French characters: accepted
Special characters accepted are: _\-@.,'!?;:&$%()/ and space*

Amount

The amount of the bill.

Bill amount restrictions
Required
Range: 0.01 to 50000

Letters: not accepted
Numbers: accepted including decimal

ExternalSystemBillNumber1, ExternalSystemBillNumber2, ExternalSystemBillNumber3

There are three number to put data to recognize the bill. The first number can be displayed to the customer during communication. So, it is mostly used to show the bill number from the system calling the API. There are two more field to store ID or other external system information.

Bill external number restrictions
Required
Max length: 1000
Letters and numbers: accepted
French characters: accepted
Special characters accepted are: _\-@.,'!?;:&$% and space*

Currency

CAD or USD.

Bill currency restrictions
*Not required**
(See currency enumeration)

* THIS IS OPTIONAL BECAUSE IT IS AUTOMATICALLY SET TO MERCHANT DEFAULT CURRENCY IF OMITTED.

Language

Language for communication related to the bill. French or English supported.

Bill language restrictions
Not required
(See language enumeration)

* THIS IS OPTIONAL BECAUSE IT IS AUTOMATICALLY SET TO MERCHANT DEFAULT LANGUAGE IF OMITTED.

Related customer ID

The bill can be related to a customer or not. If it is related to the customer, it tells the API each payment (typically one payment per bill) for the bill will be done by the same customer. That is the most common way to use the bill and payment process.

If the bill is not related to a specific customer, that means the customer will need to be set when creating the payment on the bill. This allows multiple different customers to pay on the same bill.

Bill related customer restriction
Not required
Example format: 05880372-5c30-4f17-8796-c353bfaece3f

Object example


```
{
  BillId: "45c35985-2b94-4abd-a608-8685aeb75226",
  CreatedDate: "2021-01-28T13:47:27.443-05:00",
  MerchantId: "122c2650-6418-469a-a2ce-4fdc02c601ac",
  BillTitle: "test interac",
  BillDescription: "test interac",
  BillAmount: 1,
  ExternalSystemBillNumber1: "",
  ExternalSystemBillNumber2: "",
  ExternalSystemBillNumber3: "",
  BillCurrency: 2,
  Language: 1,
  RelatedCustomerId: "986cec31-be7a-4d7c-a703-0f4c67791362"
}
```

Recuring transfer object

Object representing an active recuring transfer.

RecuringTransferId

The identification of the recuring process of a transfer. This is not the transfer identification, but the fact that transfers will be automatically generated.

RecuringMode

The frequency. See Transfer frequency enumeration.

RecuringRefDate

The starting date of the recuring process. This date is used to determine the first transfer and is used based on the recuring mode to determine the next transfer to be created.

CreatedDate

The date this recuring has been created.

RelatedMerchantId

The merchant identification for which transfers will be created.

RelatedMerchantName

The name of the merchant. This will appear on the customer statement.

CustomerName

The name of the customer for which the transfers will be created.

CustomerId

The customer identification for the created transfers.

Amount

The amount of the recuring transfer.

Object example

```
{
  RecuringTransferId: "89d720f2-78ae-4816-8fda-0099aa867c38",
  RecuringMode: 2,
  RecuringRefDate: "2021-02-12T13:47:27.443-05:00",
  CreatedDate: "2021-01-28T13:47:27.443-05:00",
  RelatedMerchantId: "122c2650-6418-469a-a2ce-4fdc02c601ac",
  RelatedMerchantName: "Company merchant",
  CustomerName: "Client",
  RelatedCustomerId: "986cec31-be7a-4d7c-a703-0f4c67791362"
  Amount: 98.6
}
```

Reporting of operations

List Executed Operations

Method name

ListExecutedOperations

Argument

Property	Description
FromDate	Extract the data having transaction date after this value.
ToDate	Extract the data having transaction date before this value.
TransferType	Type of transaction (payment, free collection, or free deposit) to extract. See Transfer type enumeration.
TransferGroupId	The “group identification” provided or auto generated that identifies the multiple transfer batch.
OnlyWithErrors	Return only transfer having transaction error.
MerchantId	Return only the transfer for this merchant identifier.
DateType	Determine if the FromDate and the ToDate is for the date of the creation of the transaction or the last modified date of the transaction. See date type enumeration.

Response

Property	Description
OperationList	List of executed operation. See the explanation for combined operation. The operation list is not what is called “operation” in this document. It is the logical combination of the operation having been executed. See Combined operation object.

Call Example

```
{
  SessionToken: "0651af61-ae9e-41a7-898e-8ec775c6f8a1",
  ServiceId: "862b96b1-85d4-406f-b55c-8f48f895f68e",
  OnlyWithErrors: true,
  FromDate: "2021-02-12T13:47:27.443-05:00",
  ToDate: "2021-02-12T13:47:27.443-05:00",
  TransferType: 2
  DateType: 2
}
```

Other data extraction methods

There are other methods for extracting payment and free operations.

- List Transfers.
- Get deposit operations.
- Get free collection operations.

THOSE METHODS ARE NOT DOCUMENTED INSIDE THIS DOCUMENT AS THEY ARE NOT RETURNING DATA ORGANIZED TO IDENTIFY GROUPED OPERATIONS.

Combined operation object

This object is returned by the List Executed Operation method and represent the process status of transfers.

OperationTypeRef

Determine if the process was Payment, Free Collection or Free Deposit.

Amount

The amount of the combined operation. (Customer operation is never combined, but merchant related operations are combined.).

OperationTarget

Determine if the process is targeting Merchant or Customer. See Operation target enumeration.

TransferDirection

Determine if process was for collection or deposit into the payment methods or the merchant account.

TargetSystemId

Refer to the payment identification or the free collection identification or the free deposit identification.

Transactions

List or transactions for the current process. See Transaction object.

FreeCollectionList

(The property name should be **CombinedOperations**, it is still FreeCollectionList for compatibility)

The list of operations included into the execution process. See Combined operation object.

Object example

```
{
  OperationTypeRef: "FreeCollections",
  Amount: 98.6,
  OperationTarget: 2,
  TransferDirection: "1",
  TargetSystemId: "45c35985-2b94-4abd-a608-8685aeb75226",
  Transactions: [],
  FreeCollectionList: []
}
```

Transaction object

Transaction data to consult process status.

OperationTarget

Determine if the process is targeting Merchant or Customer. See Operation target enumeration

OperationType

The transaction step (Validation, Transmission, StatusCheck or PaybackCheck). See Operation type enumeration.

OperationDirection

Determine if process was for collection or deposit into the payment methods or the merchant account.

Status

The step execution status. See Operation status enumeration.

Description

Description of the transaction.

BankingOperationResult

An enumeration regrouping the execution result. See Bank Operation Result enumeration.

BankDescription

A general description of the execution result.

AccountName

The account name related to the execution.

Accout Preview

(There is a typo error in the property name. Accout is really the property name. The name has not changed for compatibility reason)

The preview of the account number. Example: ***-*****-***1234.

Account type

The payment method type used. See payment method type enumeration.

TransactionDescription

A precise description of the execution result.

TransactionDueDate

The due date of the transmission.

LastModifiedDate

The modified date of the transaction process. Validation and Transmission usually arrived very close. Status check arrived usually within the same day. Payback check can occured maximum 3 months after the creation.

Operation object

FreeCollectionId

(The property name should be “RelatedTransferId”, because it is good for payment, free collection and free deposit. The name has not changed for compatibility reason).

MerchantId

Merchant related to the operation.

OperationDirection

Determine if process was for collection or deposit into the payment methods or the merchant account.

OperationKind

Payment, free collection or free deposit. See transfer type enumeration.

Amount

The amount of the operation. (Customer operation is never combined, but merchant related operations are combined.).

CreatedDate

The date the transfer has been created.

AccountName

Account related to the operation

ReferenceId

The external identifier provided when creating the payment, free collection or free deposit.

ExecutedDate

Date the banking operation has been executed.

Merchants

Merchant

CreateMerchant

Allows a Client to create a Merchant under a specific Service/SubClient

Method name:

CreateMerchant

Property	Description	Required
ServiceID	Service ID/Sub-Client ID in which the merchant will be added.	X
MerchantInfo	See Merchant basic information object	X
MerchantName	Merchant registered name	X
ExternalSystemId	External identity of the merchant	
ExternalSystemGroupId	External system Group identity of the merchant if exists	
MerchantCurrency	Determines the merchant transaction default currency. Note that only CAD is currently available. See **currencies enumeration**.	X
Language	Determines the default communication language for any transaction related to the merchant. See **languages enumeration**.	X
Email	The merchant email address for communications.	X
EmailCopyTo	Second merchant email address	
PhoneNumber	Phone number related to this specific merchant	
MerchantDescription	A small description of the merchant	
FavoriteProvider	The merchants favorite provider see **Providers Enum**	
Address	See **the Address Global Object**	X
Account	See **The bank account object**	X

Response

property	Description

MerchantId	The new MerchantId related with the appropriate Service

Call Example:

```
{
CreateMerchant: [{
    SessionToken: "86ee144e-9c27-4039-aa1f-43be0042aecf",
    MerchantInfo: {
          MerchantName: "1234567 Canada inc.",
          ExternalSystemId: "M3493LDO",
          ExternalSystemGroupId: "#PQSD23",
          MerchantCurrency: 1,
          Language: 2,
          Email: "perterparker@gmail.com",
          EmailCopyTo = "maryjane@gmail.com",
          PhoneNumber = "5145148888",
          MerchantDescription: "SpiderCo Halloween Shop",
          Address {
              StreetAddress: "111 Wellington Street",
              AddressCity: "Ottawa",
              ProvinceStateId: 8,
              CountryId: 1
              PostalZipCode = "K1A0A4",
          },
          Account: {
              AccountName: "SpiderCo Halloween Shop",
              Owner: "1029483 Canada inc.",
              BankNumber: "815",
              InstitutionNumber: "60003",
              AccountNumber: "0052698",
          }
    },
    ServiceId: "89d720f2-78ae-4816-8fda-0099aa867c38"
}]
```

Response:

```
{
    “MerchantId”: “b96d8827-5e57-4698-ab57-5601a9b973a2”
}
```

Merchant basic information object

Account Name (required)

This field is displayed on the clients' bank statement. This field is also used for communication when the description is not provided.

Account name restrictions
Required
*Max length: 150 (or 15)**
Letters and numbers: accepted
*French characters: accepted***
Special characters are: not accepted [except space]

* FOR MOST INSTITUTION, ONLY THE FIRST 15 CHARACTERS ARE DISPLAYED.

** FRENCH CHARACTERS ARE ACCEPTED BUT ARE NOR DISPLAYED ON THE CLIENT STATEMENT.

Description

Replaces the account name for communication.

Account name restrictions
Not required
Max length: 150
Letters and numbers: accepted
French characters: accepted
Special characters accepted are: _\-@.,'!?;:&$%() and space*

Currency

Determines the merchant transaction default currency. Note that only CAD is currently available. See currencies enumeration.

Language

Determines the default communication language for any transaction related to the merchant. See languages enumeration.

Email

The merchant email address for communications.

Merchant Email restrictions
Required
Max length: 150
Constraint: Valid email format only

Phone number

The merchant phone number.

Phone number restrictions
Not required
Max length: 150
Letters: not accepted
Numbers: accepted
Special characters accepted are: - and space

WhiteLabeling.

SetWhitelabeling.

Sets whitelabeling for a (client, Service, Merchant) based on the WhitelabelingLevel Property

Method Name :

SetWhiteLabeling

Property	Description
Id	The Id of the entity that Will have the white labeling set to
WhiteLabelingLevel	The type of entity that will have the whitelabeling See WhitelabelingLevelEnum
WhiteLabelingData	List Of WhiteLabelingData that need to be set See

Response

Nothing Returend (only error handling properties)

Call Example :

```
{
  "Id": "89d720f2-78ae-4816-8fda-0099aa867c38",
  "WhiteLabelingLevel": 1,
  "WhiteLabelingData": [
    {
      "CssProperty" :"background-color",
      "CssValue" : "black"
    }
  ]
}
```

UpdateWhitelabeling.

Update a whitelabeling info for an entity

Method Name :

UpdateWhiteLabelingData

Property	Description
Id	The Id of the entity that Will have the white labeling set to

WhiteLabelingLevel	The type of entity that will have the whitelabeling Qee WhitelabelingLevelEnum
WhiteLabelingData	List Of WhiteLabelingData that need to be to update (Id Required for each WhitelabelingData)

Response

Property	Description
UpdatedWhitelabelingData	The updated values

```
{
  "UpdatedWhitelabelingData": [
    {
      “Id”: “1z5e89r6-78ae-4816-8fda-0099aa867c38”,
      "CssProperty" :"background-color",
      "CssValue" : "black"
    }
  ]
}
```

Call Example :

```
{
  "Id": "20f24z45-78ae-4816-8fda-0099aa867c38",
  "WhiteLabelingLevel": 1,
  "WhiteLabelingData": [
    {
      “Id”: “”,
      "CssProperty" :"background-color",
      "CssValue" : "black"
    }
  ]
}
```

GetWhitelabeling.

Get whitelabeling info for an entity

Method Name :

GetWhiteLabelingData

Property	Description
Id	The Id of the entity that Will have the white labeling set to
WhiteLabelingLevel	The type of entity that will have the whitelabeling Qee WhitelabelingLevelEnum

Response

Property	Description
WhitelabelingData	The whitelabeling for the desired entity.

Response

```
{
  "WhitelabelingData": [
    {
      “Id”: “445z155e-78ae-4816-8fda-0099aa867c38”,
      "CssProperty" :"background-color",
      "CssValue" : "black"
    }
  ]
}
```

Call Example :

```
{
  "Id": "89d720f2-78ae-4816-8fda-0099aa867c38",
  "WhiteLabelingLevel": 1,
}
```

GetListWhitelabeling.

Get List Of whitelabeling info for a Client entity all the whitelabelings for (client, Merchants, Services)

Method Name :

GetListWhiteLabelingData

Property	Description
SessionId	Session Id Of the logged in Client

Response

Property	Description
Whitelabelings	A list of whitelabeling Data See whitelabelingModel

Response

```
{
  "Whitelabelings": [
      {
          “Id” : “89d720f2-78ae-4816-8fda-0099aa867c38”,
          “WhiteLabelinglevel” : “256720f2-78ae-4816-8fda-0099aa867c38”,
          ‘WhiteLabelingData” : [
               {
                     “Id”: “95sd60f2-78ae-4816-8fda-0099aa867c38”,
                    "CssProperty" :"background-color",
                    "CssValue" : "black"
```

```
                }
            ]

        }

    ]
}
```

Call Example :

```
{
  "SessionId": "89d720f2-78ae-4816-8fda-0099aa867c38",
}
```

DeleteWhitelabeling.

Delete whitelabeling info for an entity

Method Name :

DeleteWhiteLabeling

Property	Description
Id	The Id of the entity that Will have the white labeling Deleted
WhiteLabelingLevel	The type of entity that will have the whitelabeling Qee WhitelabelingLevelEnum

Response

Nothing returned (only error handling properties)

Call Example :

```
{
  "Id": "89d720f2-78ae-4816-8fda-0099aa867c38",
  "WhiteLabelingLevel": 1,
}
```

Whitelbeling object

Id

- The unique identifier of a whitelabeling.

Whitelabeling Level

- To determine which entity will get the current whitelabeled.

List of WhiteLabeling Data.

WhiteLabeling Data object:

- Id the white labeling Unique identifier.
- CssProperty: a string to determine what property will be whitelabeled.
- CssValue: a string to determine the value the WhiteLabeling data

WhiteLabelingData object

WhiteLabelingDataId

WhiteLabelingData Unique identifier

CssProperty

And internally defined CSS Property To be whitelabeled

CssValue

the value that CSS Property will have

Sub Clients

CreateSubClient

Allows a client to Create a new sub client

Method name:

CreateSubClient

Property	Description
Name	The new subclient's name
Language	The default language for the new sub client "Fr" or "En" are supported.

Response

property	Description
`ServiceId`	The new Sub Client ID

Response:

```
{
    “ServiceId”: “89d720f2-78ae-4816-8fda-0099aa867c38”
}
```

Call Example:

```
{
     “Name”: “new sub client”,
     “Language”: 1
}
```

SetClientDefaultServiceFeeSettings

Allows you to set a client default service fee settings

Methods Name

SetClientDefaultServiceFeeSettings

Property	Description
ClientId	the clients unique identifier
ServiceFeeSettings	The Service fee Settings

Response
Nothing Returned (only Error Handling properties)

Call Example:

```
{
       “ClientId” : “89d720f2-78ae-4816-8fda-0099aa867c38”,
       “ServiceFeeSettings” : {
              // See ServiceFeeSettingsModel Object.
       }
}
```

SetClientSettings

Allows you to set settings for a client.

Method Name:

SetClientSettings

Property	Description
ClientId	The client Unique identifier
ClientSettings	See ClientSettings model.

Response
Nothing Returned (only Error Handling properties)

Call Example:

```
{
	“ClientId” : “89d720f2-78ae-4816-8fda-0099aa867c38”,
	“ClientSettings” : {
		// See ClientSettings Object.
	}
}
```

GetClientSettings

Allows you to get settings for a client.

Method Name:

GetClientSettings

Property	Description
ClientId	The client Unique identifier

Response

Property	Description
ClientSettings	the desired client settings
ServiceSettings	The clients service settings
ServicefeeSettings	The client’s service fee settings

Response Example

```
{
	“ClientSettings”: {
		// See ClientSettings Object
	},
	“ServiceSettings”: {
		// See ServiceSettings Object
	},
	“ServicefeeSettings”: {
		// See ServicefeeSettings Object
	}
```

```
}
```

Call Example:

```
{
        “ClientId”: “89d720f2-78ae-4816-8fda-0099aa867c38”,
}
```

General objects and enumerations

Address global object

Street Address

The door number, road name and apartment.

Street address restrictions
Required
Max length: 250
Letters and numbers: accepted
French characters: accepted
Special characters accepted are: _\-@.,'!?;:&$% and space*

City

City restrictions
Required
Max length: 250
Letters and numbers: accepted
French characters: accepted
Special characters accepted are: -' and space

Province / State

The province or the state of the merchant. See Provinces / States enumeration.

Country

The country of the merchant office. See countries enumeration.

PostalZipCode

The postal code or zip code.

Object example

```
{
  StreetAddress: "1 Testing road",
  AddressCity: "Testcity",
  ProvinceStateId: 10,
  CountryId: 1,
  PostalZipCode: "H1H1H1"
}
```

Languages enumeration

Language enumeration
Unkown = -1
NotSet = 0
French = 1
English = 2

Currencies enumeration

Currency enumeration
Unkown = -1
NotSet = 0
CAD = 1
USD = 2

Countries enumeration

Countries enumeration
Unkown = -1
NotSet = 0
Canada = 1
USA = 2

Payment method type enumeration

Payment method type enumeration
Unkown = -1

NotSet = 0 *CreditCard = 1* *DirectAccount = 2* *Interac = 3*

Authorized Payment method type enumeration

Authorized payment method type enumeration
Unkown = -1
NotSet = 0
CreditCard = 1
DirectAccount = 2

Provinces / States enumeration

Ctry	City	#
----	Unkown	-1
----	NotSet	0
CA	Alberta	1
CA	British Columbia	2
CA	Manitoba	3
CA	New Brunswick	4
CA	Newfoundland	5
CA	Nova Scotia	6
CA	Nunavut	7
CA	Ontario	8
CA	Prince Edward Island	9
CA	Quebec	10
CA	Saskatchewan	11
CA	Northwest Territories	12
CA	Yukon Territory	13
US	Alaska	16
US	Alabama	17
US	Arkansas	19
US	Arizona	21
US	California	22
US	Colorado	23
US	Connecticut	24
US	District of Columbia	25
US	Delaware	26

Ctry	City	#
US	Florida	27
US	Georgia	29
US	Guam	30
US	Hawaii	31
US	Iowa	32
US	Idaho	33
US	Illinois	34
US	Indiana	35
US	Kansas	36
US	Kentucky	37
US	Louisiana	38
US	Massachusetts	39
US	Maryland	40
US	Maine	41
US	Marshall Islands	42
US	Michigan	43
US	Minnesota	44
US	Missouri	45
US	Mariana Islands	46
US	Mississippi	47
US	Montana	48
US	North Carolina	49
US	North Dakota	50
US	Nebraska	51

Ctry	City	#
US	New Hampshire	52
US	New Jersey	53
US	New Mexico	54
US	Nevada	55
US	New York	56
US	Ohio	57
US	Oklahoma	58
US	Oregon	59
US	Pennsylvania	61
US	Puerto Rico	62
US	Palau	63
US	Rhode Island	64
US	South Carolina	65
US	South Dakota	66
US	Tennessee	67
US	Texas	68
US	Utah	69
US	Virginia	70
US	Virgin Islands	71
US	Vermont	72
US	Washington	73
US	West Virginia	74
US	Wisconsin	75
US	Wyoming	76

Transfer direction enumeration

Payment method type enumeration
Unkown = -1
NotSet = 0
Collect = 1
Deposit = 2

Transfer type enumeration

Transfer type enumeration
Unkown = -1
NotSet = 0
Payment = 1
FreeDeposit = 2
FreeCollection = 3
Fee = 2
Revert = 3
PaymentAndFreeCollection = 3

Transfer frequency enumeration

Transfer frequency enumeration
Unkown = -1
Once = 0
Daily = 1
Weekly = 2
EveryTwoWeeks = 3
Monthly = 4
Trimester = 5
BiAnnually = 6
Annually = 7

Transaction Transfer type enumeration

Transfer type enumeration
Unkown = -1
NotSet = 0
Payment = 1
FreeCollection = 2
PaymentAndFreeCollection = 3
FreeDeposit = 4
PaymentAndFreeDeposit = 5
FreeCollectionAndFreeDeposit = 6
All = 7

Date type enumeration

Date type enumeration
Unkown = -1
NotSet = 0
CreatedDate = 1
LastModifiedDate = 2

Operation target enumeration

Operation target enumeration
Unkown = -1
NotSet = 0
Customer = 1
Merchant = 2

Operation type enumeration

Operation type enumeration
Unkown = -1
NotSet = 0
Validation = 1
Transmission = 2
StatusCheck = 3
PaybackCheck = 4

Operation status enumeration

Operation status enumeration
Unkown = -1
NotSet = 0
Success_Success = 1
Success_NoResultReturned = 2
Success_Skip = 3
Success_WaitManual = 4
Success_ Success_Error = 10
Error_Temporary = 11
Error_Fatal = 12
Abort = 4

Bank operation result enumeration

Bank operation result enumeration
No result = 0
Confirmed = 1
Other errors = 2
NSF = 3
Account error = 4
Opposition = 5
Interac Refused = 6
Interac Failed = 7

WhiteLabelingLevel enumeration

-

WhiteLabelingLevel Enum
Default = -1
NotSet = 0
Merchant = 1
Service = 2
Client = 3

Provider enumeration

Operation status enumeration

Unkown = -1
NotSet = 0
Sandbox_Account = 100
Sandbox_CreditCard = 200
Sandbox_Interac= 300
CA_CreditCard_Moneris =1000
CA_CreditCard_BankOfAmerica = 1001
CA_Account_Desjardins = 1100
CA_Account_RBC_1 = 1101
CA_Account_RBC_2 = 1102
CA_Account_RBC_3 = 1103
CA_Account_RBC_4 = 1104
CA_Account_RBC_5 = 1105
CA_Interac_RBC= 1200