Unified API

M-BIRR Unified API

Note: The definitive version of this document is stored in GIT with the source code.

Version 2.1 Date 09/05/2017

Introduction

This API to allow 3rd party service providers to programmatically perform remittances or payments and receive notifications of transactions on accounts. This API merges both transaction notifications and payments and supercedes both the Payment API (V1.1) and the Transaction Notification API (V.3).

This API is the intellectual property of M-BIRR Limited (Ireland). Third party use, either directly or by proxy, may only be done so under a commercial licensing agreement with M-BIRR Limited or a subsidiary of M-BIRR Limited.

Unified API Features

The API features include:

- M-BIRR Account to M-BIRR Account Transaction Notifications
- M-BIRR Account to External Account Transaction Notifications
- M-BIRR Account to M-BIRR Account Payment Instructions
- External Account to M-BIRR Account Payments Instructions

The API is designed so as to be as symetrical as possible - i.e. a transaction notification from the M-BIRR system to a third party system is similar as a payment instruction in reverse.

M-BIRR to External accounts and vice/versa are used for inter-bank transactions.

The API is REST based. The API requests / responses are encoded in JSON. For simplicity, quotation marks are left out in the protocol description below.

Responses and errors

For both notification and payments the API responses will be one of the following:

- Payment/Notification Accepted with HTTP Status Code of 200
- Payment/Notification Rejected/Refunded with HTTP Status Code of 499
- Unexpected Error - Connection error / Timeout / Unexpected HTTP status code code (e.g. 404)

In the even of an unexpected error, the behaviour is implementation dependent. M-BIRR Notification allows for 2 policies

- Refund payment after attempts to communicate have been exhausted
- Retain payment after attempts to communicate have been exhaused

3rd Party Identification

The third party is identified by the `transaction_account`. This is the account that will be credited on transaction notifications or debited on payment instructions.

Simulations

Simulations can be performed for both notifications and payments using the `simualtion_only` setting. Simulations are **optional** however they have the following benefits:

- A transction can be 'prepped' to see if all things work (e.g. sufficient funds on account to cover amounts and/or fees)
- Can preview a beneficiaries details before a transaction has completed
- Can validate external data (e.g. reference numbers in buy gooods)

Replay

Each request has a `client_transaction_id` that uniquely identifies a transaction from a client system. These IDs are used to uniquely identify a transaction that have been retried / replayed. A transaction may be retried / replayed for example if a communications was interrupted before a response was received.

Replayed transactions will not result in an account being debited/credited more than once. However the response to a replayed transaction should be the same each time.

The server system (called) will respond with a `server_transaction_id` on a successful acceptance of the notification/payment. This can be used as part of an audit trail to reconsile transactions.

Operation Types

Each transaction on the M-BIRR system has a number of transaction `operation_types`. This indicates the type of transaction can take the values:

- BuyGoods
- PayBill
- Transfer
- Remittance (International transfers)

Customer Care Code

Each transaction on the M-BIRR system has a semi-unique 6 alpha-numeric transaction `customer_care_code` that an M-BIRR customer can use to identify a transaction. External 3rd parties can optionally use this to identify payments made in their systems.

End User Messages

The `end_user_message` field can be used by customers to send additional information (e.g. Bill Numbers / Invoice numbers when paying a bill).

It can also be used to transmit information to end users in the event of successful purchases OR payments from external systems.

Account Identification

An M-BIRR beneficiary or M-BIRR sender is uniquely identified by the M-BIRR `account_number` OR `mobile_number`. External (non-MBIRR) accounts are identified in the same way.

An error (499) will occur if both are present in a payment instruction and are inconsistent on M-BIRR.

Also on the M-BIRR system mobile users have the ability to 'hide' their mobile numbers. In such cases the account number will be used to identify a sender of funds.

Protocol

For simplicity, quotation marks (") have not been included in the JSON.

M-BIRR to M-BIRR a/c Notification

A notification is used to inform an external 3rd party system that an M-BIRR customer - the **sender** - has transferred money to another M-BIRR account - the **beneficiary**. The **beneficiary** account is typically an account of the 3rd party.

On the M-BIRR system the sender's account is debited and the beneficiary account is credited.

Notification Request

JSON	JSON	JSON	Comment
{			
`sender`:	{		M-BIRR customer sending funds
	`name`:	Joe Bloggs	Mandatory

	account_number:	11223344	Mandatory
	mobile_number:	0912112233	Optional
	language:	en	Mandatory
	}		
beneficiary:	{		M-BIRR customer receiving funds
	name:	DSTV	Optional
	account_number:	22334455	Mandatory
	mobile_number:	0912223344	Optional
	language:	en	Optional
	}		
transaction:	{		
	simulation_only:	false	Mandatory
	client_transaction_id:	e9247e94-7d7c-4731-a802-cb42474bd25a	Mandatory when simulation is false
	timestamp:	2016-09-08T08:40:40.181Z	Mandatory
	amount_in_cents:	10000	Mandatory
	operation_type:	PayBill	Mandatory
	customer_care_code:	ZYXABC	Mandatory
	}		
transaction_account:	{		M-BIRR account credited
	account_number:	11223344	Mandatory
	balance_after_in_cents:	10010000	Mandatory
	}		
end_user_message:	Inv10392444		Optional
}			

Positive Acknowledgement to Notification (ACK)

HTTP Status Code: 200 - Success, Accept Payment

JSON	JSON	JSON	Comment
{			
end_user_message:	Thank you for paying your bill		Optional
server_transaction_id:	e123e123-e1e1-e1e1-1e1e-abcdefghijkl		Optional
}			

Negative Acknowledgement to Notification (NACK)

HTTP Status Code: 499 – Reject, Refund Payment

JSON	JSON	JSON	Comment
{			
end_user_message:	You are not registered with us. Transaction failed		Optional
}			

M-BIRR to M-BIRR a/c Payment

A payment is used by an external 3rd party system to transfer money from their M-BIRR account - the **sender** - to another M-BIRR account - the **beneficiary**.

On the M-BIRR system the sender's account is debited and the beneficiary account is credited.

Payment Request

JSON	JSON	JSON	Comment
{			
sender:	{		M-BIRR customer sending funds
	name:	Jane Banks	Optional
	account_number:	99334455	Mandatory
	mobile_number:	0912223344	Optional
	language:	en	Optional
	}		
beneficiary:	{		M-BIRR customer receiving funds
	name:	Joe Bloggs	Optional
	account_number:	11223344	Mandatory if mobile number not present
	mobile_number:	0912112233	Mandatory if account number not present
	language:	en	Optional
	}		
transaction:	{		
	simulation_only:	false	Mandatory
	client_transaction_id:	e9247e94-7d7c-4731-a802-cb42474bd25a	Mandatory when simulation is false
	timestamp	2016-09-08T08:40:40.181Z	Mandatory
	amount_in_cents:	10000	Mandatory
	operation_type:	Transfer	Mandatory
	}		
end_user_message:	Here comes the mula!		Optional
}			

Positive Acknowledgement to Payment (ACK)

HTTP Status Code: 200 - Success, Accept Payment

JSON	JSON	JSON	Comment
{			
server_transaction_id:	e123e123-e1e1-e1e1-1e1e-abcdefghijkl		Mandatory
beneficiary:	{		M-BIRR Customer receiving funds
	name:	Joe Bloggs	Mandatory
	account_number:	11223344	Mandatory if mobile number not present
	mobile_number:	0912223344	Mandatory if account number not present
	language:	en	Mandatory
	}		
transaction_account:	{		
	account_number:	99334455	Mandatory

	balance_after_in_cents:	990000	Mandatory
	}		
fee_in_cents:	490		Optional
end_user_message:	Payment Complete.		Optional
}			

Negative Acknowledgement to Payment (NACK)

HTTP Status Code: 499 – Reject, Refund Payment

JSON	JSON	JSON	Comment
{			
end_user_message:	Sorry but don't know this account		
}			

M-BIRR to External a/c Notification

Typically used for inter-Bank transfers this notification is used to inform an external 3rd party system that an M-BIRR customer - the **sender** - has transferred money to an external **non-M-BIRR account** - the **beneficiary**. The **beneficiary** account resides on the 3rd party system.

On the M-BIRR system the sender's M-BIRR account is debited and the **transaction account** (a routing account) is credited.

Notification Request

JSON	JSON	JSON	Comment
{			
sender:	{		M-BIRR customer sending money
	name:	Joe Bloggs	Mandatory
	account_number:	11223344	Mandatory if mobile number not present
	mobile_number:	0912112233	Mandatory if account number not present
	language:	en	Mandatory
	}		
beneficiary:	{		External Bank A/C
	name:	Jane Banks	Optional
	account_number:	991111222233334444	Mandatory if mobile number not present
	mobile_number:	0912223344	Mandatory if account number not present
	language:	en	Optional
	}		
transaction:	{		
	simulation_only:	false	Mandatory
	client_transaction_id:	e9247e94-7d7c-4731-a802-cb42474bd25a	Mandatory when simulation is false
	timestamp	2016-09-08T08:40:40.181Z	Mandatory
	amount_in_cents:	10000	Mandatory
	operation_type:	Transfer	Mandatory
	customer_care_code:	ZYXABC	Mandatory
	}		
transaction_account:	{		M-BIRR routing account

	account_number:	99999999	Mandatory
	balance_after_in_cents:	100100000	Mandatory
	}		
end_user_message:	Inv10392444		Optional
}			

Positive Acknowledgement to Notification (ACK)

HTTP Status Code: 200 - Success, Accept Payment

JSON	JSON	JSON	Comment
{			
end_user_message:	Payment Complete		
server_transaction_id:	e123e123-e1e1-e1e1-1e1e-abcdefghijkl		
beneficiary:	{		Destination a/c details on external system
	name:	Jane Banks	Mandatory
	account_number:	991111222233334444	Mandatory if mobile number not present
	mobile_number:	0912223344	Mandatory if account number not present
	language:	en	Mandatory
	}		
}			

Negative Acknowledgement to Notification (NACK)

HTTP Status Code: 499 – Reject, Refund Payment

JSON	JSON	JSON	Comment
{			
end_user_message:	Sorry but don't know this account		
}			

External to M-BIRR a/c Payment

Typically used for inter-Bank transfers this payment is used by an external 3rd party system to transfer money from an external **non-M-BIRR account** - the **sender** - to a **beneficiary** M-BIRR account. The **beneficiary** account resides on the M-BIRR system.

On the M-BIRR system the **transaction account** (a routing account) is debited and the beneficiary M-BIRR account is credited.

Payment Request

JSON	JSON	JSON	Comment	
{				
sender:	{		External 3rd party customer sending money	
	name:	Jane Banks	Mandatory	
	account_number:	991111222233334444	Mandatory if mobile number not present	
	mobile_number:	0912223344	Mandatory if account number not present	
	language:	en	Mandatory	
	}			
beneficiary:	{		M-BIRR Customer receiving money	

	name:	Joe Bloggs	Optional	
	account_number:	11223344	Mandatory if mobile number not present	
	mobile_number:	0912112233	Mandatory if account number not present	
	language:	en	Optional	
	}			
transaction:	{			
	simulation_only:	false	Mandatory	
	client_transaction_id:	e9247e94-7d7c-4731-a802-cb42474bd25a	Mandatory when simulation is false	
	timestamp:	2016-09-08T08:40:40.181Z	Mandatory	
	amount_in_cents:	10000	Mandatory	
	operation_type:	Transfer	Mandatory	
	customer_care_code:	ZYXABC	Optional	
	}			
transaction_account:	{			
	account_number:	99999999	M-BIRR routing account	Mandatory
	}			
end_user_message:	Here comes the mula!		Optional	
}				

Positive Payment Response (ACK)

HTTP Status Code: 200 - Success, Accept Payment

JSON	JSON	JSON	Comment
{			
end_user_message:	Payment Complete.		
server_transaction_id:	e123e123-e1e1-e1e1-1e1e-abcdefghijkl		
beneficiary:	{		Destination a/c details
	name:	Joe Bloggs	Mandatory
	account_number:	11223344	Mandatory if mobile number not present
	mobile_number:	0912223344	Mandatory if account number not present
	language:	en	Mandatory
	}		
transaction_account:	{		
	account_number:	99999999	
	balance_after_in_cents:	100100000	
	}		
fee_in_cents:	490		
}			

Negative Payment Response (NACK)

HTTP Status Code: 499 – Reject, Refund Payment

JSON	JSON	JSON	Comment

{			
`end_user_message`:	Sorry but don't know this account		
}			

Protocol Security

The API will operate over HTTPS. The service provider MAY (optionally) provide M-BIRR with the public key / SSL certificate if the certificate is not certified by a public CA e.g. self signed.

In addition each transaction notification / payment instruction is digitally signed by an private key. A corresponding public key will be supplied to the other party to validate the transaction notification data.

RSA or DSA algorithms can be used to validate the authenticity and integrity of a request (notification or payment). The process below uses SSL to generate public and private keys. Note that the SSL API varies depending on the programming. In the examples below the command line **open ssl** is used.

It is recommended that **all transaction signatures are stored** (e.g. in a DB) with the payment/notification requests in the event of dispute over a particular transaction.

Header variables

- `Content-Signature` (mandatory - in request only, not required in response)
  - a hash of all of the parameters content constructed with a private M-BIRR key and validated by a public M-BIRR key (See http://docs.oracle.com/javase/tutorial/security/apisign/step3.html)
  - e.g. `Content-Signature: <base64 encoded signature>`
  - Base64 encoded

RSA Content Signature

Creating public/private keys

Private RSA keys can be generated with SSL

```
# openssl genrsa -out mykey.pem 1024
```

and would look something like:

```
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQDp0zVTEt+CROphB7mDKxFG4YVs117kgldBc5CquDOp4BnsoHqZ
oCctbEHy5on5n62mpPSWDkLkKaXQ2bmcuFhDdUm/idbmu9w+ihr7IUoVmU93xcaA
wmT0NcZ29hYwmXhF8xKOnXrOZ8+/cm0hjKuFOeIIv5wTdbjc2Jw9f1/FDwIDAQAB
AoGBALkCjAvKQB1kCAt0QD3gAtCRuq+vDX9C/mVkWlAqZX6BOiH3rGoHxVf0c44N
CZkA1jaoCHr+l8DOj0vcKjgtyG88gjeABhUf2sb2mjkPXUfA/J+seN7B81oIF5hh
FBB4ZrIMnaaKOuNm/w+V+qgQGQq5LjdbdN6+L/B10VcI9+ehAkEA+p8cxk/08muk
Ww0n2a2Vs/W3Wd9USI1+dza+kNa77saB8T2p+2wvIjYUtGSuEKlE4uZ6V+MChaGq
Udj/xRuD0QJBAO7X0Vd5/Ap/gKiQSHZyCsrr+4yyHwxyEnPQg5e3DEyzdjIvPdKY
E1eySf/vCTmaUR0zvqjUMXEWQtgIsX//Ut8CQEQjYlVgg74tGeBPcVgzY5Ir6g+b
MnUPUyTiU2lZmHrQqZ+HHmHnBeGNRo/NF64b90ihVP30EEl4rK/YCaHzWyECQC14
nx1VVvu0rIzq8zDBo3ZgmQJ/QtP+v66W6wauTcQogn8paU63ry8J/XahTpNN4tV5
bXn88/DAuTch6JEizecCQEcuEPhoxlRCvdWEmuay2oMRlxvPuAUlQs07TXr/7Gs3
3p9VCJcGW6uHBSiJ60P8bMkmREW8KuUBUg0WGYbs1hk=
-----END RSA PRIVATE KEY-----
```

A public key can be generated with

```
# openssl rsa -in mykey.pem -pubout > mykey.pub
```

And looks something like:

```
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDp0zVTEt+CROphB7mDKxFG4YVs
117kgldBc5CquDOp4BnsoHqZoCctbEHy5on5n62mpPSWDkLkKaXQ2bmcuFhDdUm/
idbmu9w+ihr7IUoVmU93xcaAwmT0NcZ29hYwmXhF8xKOnXrOZ8+/cm0hjKuFOeII
v5wTdbjc2Jw9f1/FDwIDAQAB
-----END PUBLIC KEY-----
```

The public key is provided to the other party receiving the request to validate the authenticity and integrity of the request.

Signing

The content of a request can be signed as follows:

```
# openssl dgst -sha1 -sign mykey.pem -out hashAndSign.sha1.rsa.mykey
content.txt
```

The example above produces the signature (`hashAndSign.sha1.rsa.mykey`) such as:

```
0000000 27 10 f2 9c 4e 74 73 9a b8 69 64 2a e5 fb 6d bb
0000010 43 87 d7 a7 55 8b e0 5a 87 ea 10 41 e2 55 36 c5
0000020 7f 99 4d b1 c8 31 48 68 c5 4b 4f 34 46 b5 72 3a
0000030 e6 97 09 72 05 43 93 99 d3 40 fc a9 36 c7 38 77
0000040 e5 e5 72 68 be f8 88 95 b4 a0 82 e4 71 86 ff e5
0000050 73 d2 6a 06 a0 f1 92 25 c8 4e 8f 3d 1f 0b f0 9c
0000060 7f 8b e4 e0 ce cc 6a a2 0b 1d ff 92 a2 62 69 b3
0000070 93 18 9a 95 f1 00 6d e9 8e 40 79 3f 9c cd 07 2e
```

the signature can be verified as follows:

```
# openssl dgst -sha1 -verify mykey.pub -signature hashAndSign.sha1.rsa.
mykey content.txt
Verified OK
```

The signature can also be produced using a single line of hex

```
```

```
# openssl dgst -sha1 -sign mykey.pem -hex content.txt

SHA1(content.txt)=
2710f29c4e74739ab869642ae5fb6dbb4387d7a7558be05a87ea1041e25536c57f994db1
c8314868c54b4f3446b5723ae697097205439399d340fca936c73877e5e57268bef88895
b4a082e47186ffe573d26a06a0f19225c84e8f3d1f0bf09c7f8be4e0cecc6aa20b1dff92
a26269b393189a95f1006de98e40793f9ccd072e
```

This is the same as the hexdump above (except on one line). This is the content that would be in the Content-Signature header.

Warning

During testing you might find that the signature produced does not match the one expected. If you are testing with a file like content.txt please make sure that there are no white spaces and that the last EOF character is removed from the file (`perl -pe 'chomp if eof' content.txt > content.chomp.txt`)

DSA Content Signature

This example shows the content signature using DSA signing.

First we create a private DSA key using Open SSL at command line

```
openssl dsaparam -out dsaparam.pem 1024
openssl gendsa -out dsaprivkey.pem dsaparam.pem
```

The private key file `dsaprivkey.pem` created and used in this example is

```
-----BEGIN DSA PRIVATE KEY-----
MIIBugIBAAKBgQCWIjoh9G0+mnfVVf1Iq+7dRxrbLu9tT2IEBOe7Krqb9Tf5PUkp
tAKzhi9cDme9Jj91IsCxZfOtY+KwnslKnm44YSm4OdbV5o8OA1asGnqJoc9MZB1u
fxtNYEW2O0ueyUsaLw8BbMeHhKWJlRrnpghYZiQCMst/pgXS2eG7zljaJwIVAJ+3
biOcgnQPoVS6GhsuESpwwRDLAoGAF+O5xfp45hNI/IXXR/51snx4kew3AJ+bQZxC
Tz0nIG5y5Pbc+FHUEKDnAaNSGpMFWBQKSuzKc8nxz1YXWsjrL5IP2z6jlGuOvuQL
tcbE6e+0tW8N6UcBHm3Ji8WazcxIaq3aYONZxfWiZ0jnJtYsYJT36U8eOADk6m8u
Xk28kEMCgYAwvpqY0Ffl0WTLPub+J1OlFVGYJTCKGgsNYb9OU0yFD7rHowFSFNmq
+dyHpsICwjDPNtfDipA1PjhCtbNNoZbtCLz9pvxCjlO2jTwMauHm9HQG7ehAxBZs
kdko3qW7iZA7TUJWGkzAg87v4qfwOWSmkZsGYaxkZr8kv1SfY0jDOAIUKcvfO/l0
N2GEPdGsiM0FW73Kwc0=
-----END DSA PRIVATE KEY-----
```

From this private DSA key we create a public key using the Open SSL command line tool

```
openssl dsa -in dsaprivkey.pem -outform PEM -pubout -out dsapubkey.pem
```

The public key file `dsapubkey.pem` created and used in this example is

```
-----BEGIN PUBLIC KEY-----
MIIBtjCCASsGByqGSM44BAEwggEeAoGBAJYiOiH0bT6ad9VV/Uir7t1HGtsu721P
YgQE57squpv1N/k9SSm0ArOGL1wOZ70mP3UiwLFl861j4rCeyUqebjhhKbg51tXm
jw4DVqwaeomhz0xkHW5/G01gRbY7S57JSxovDwFsx4eEpYmVGuemCFhmJAIyy3+m
BdLZ4bvOWNonAhUAn7duI5yCdA+hVLoaGy4RKnDBEMsCgYAX47nF+njmE0j8hddH
/nWyfHiR7DcAn5tBnEJPPScgbnLk9tz4UdQQoOcBo1IakwVYFApK7MpzyfHPVhda
yOsvkg/bPqOUa46+5Au1xsTp77S1bw3pRwEebcmLxZrNzEhqrdpg41nF9aJnSOcm
1ixglPfpTx44AOTqby5eTbyQQwOBhAACgYAwvpqY0Ffl0WTLPub+J1OlFVGYJTCK
GgsNYb9OU0yFD7rHowFSFNmq+dyHpsICwjDPNtfDipA1PjhCtbNNoZbtCLz9pvxC
jlO2jTwMauHm9HQG7ehAxBZskdko3qW7iZA7TUJWGkzAg87v4qfwOWSmkZsGYaxk
Zr8kv1SfY0jDOA==
-----END PUBLIC KEY-----
```

To sign the content file `content.txt` with content use the open ssl digest

```
openssl dgst -dss1 -sign dsaprivkey.pem  -out hashAndSign.dss1.dsa
content.txt
```

which produces the hex file `hashAndSign.dss1.dsa` with content like

```
0000000 30 2c 02 14 13 d3 61 2e 24 27 46 b6 62 46 65 30
0000010 ff 84 89 d0 79 49 65 d5 02 14 5d 5c 0d ec d2 73
0000020 e6 ff 4b 87 c2 a4 99 3c a2 e6 c4 14 dc b3
000002e
```

To verify the signature use

```
openssl dgst -dss1 -verify dsapubkey.pem -signature hashAndSign.dss1.
dsa content.txt
Verified OK
```

Java Library Key Encoding (PKCS8)

Note that the Java libraries expect the encoding of the keys to be in PKCS8 format. The keys generated by the above commands are in "SSLeay" or "traditional" format and must be converted. To do this with the keys generated above use the following to do the conversion

```
openssl pkcs8 -topk8 -nocrypt -in dsaprivkey.pem -out dsaprivkey.pkcs8.
pem
```

Change History

Version 2.1

- Corrected JSON - `balance_in_cents` should have been `balance_after_in_cents`
- Removed `sha1=` from the content signature (not needed and misleading)
- Content signature is not hex encoded (it is base64 encoded)
- Corrected JSON - `time_stamp` should have been `timestamp` (removed underscore)
- client_transaction_id - mandatory when simulation is false