Integration Guide

Version 1.25

Last update 20210911

Contents

Revision History			
Version	Date	Changed By	Comments / Reason
1.0	01/06/2013	MAS/AMP	Created
1.1	24/06/2013	MAS	- Updated sequence diagrams - Added additional error return codes
1.2	25/06/2013	MAS	- Renamed pur_auth and pur_capture calls to purPreAuth and purApprove respectively
1.3	26/06/2013	MAS	- Updated service codes - Updated formatting
1.4	11/07/2013	MAS	- Added new OTP service codes - Added notes for purCreate and purOTP calls
1.5	25/07/2013	MAS	- Updated purQuery response parameters - Added new response codes
1.6	01/10/2013	AMP	- Code samples amended
1.7	08/04/2014	MAS	- Added Javascript code samples - Added notes on verified and unverified accounts - Amended and added URL examples - Added notes column in response codes section
1.8	05/06/2014	MAS	- Updated redirect application URL formatting
1.9	21/07/2014	MAS	- Updated entire guide to cater for 1 Step transaction processing (purApprove & purCancel removed)
1.10	30/07/2014	MAS	- Amended URL in PHP sample code
1.11	22/08/2014	MAS	- Added Product Interest Rate API
1.12	04/09/2014	MAS	- Added notes to redirect URL section - Updated formatting
1.13	13/10/2014	MAS	- Added further notes to redirect URL section
1.14	12/06/2015	AMP	- Changed PHP sample code
1.15	16/07/2015	AMP	- Added Web Hook API basics - Changed Application Page Redirects URL
1.16	21/04/2016	AMP	- Added clearer OTP explanation
1.17	09/08/2016	AMP	- Added HTTPS POST parameter encryption explanation
1.18	11/01/2017	AMP	- Added response codes 003 & 004
1.19	06/02/2017	AMP	- Changed Application Page Redirects URL
1.20	14/05/2017	AMP	- Added more purQuery purchase state detail - Added response code 102 - Purchase Cancelled OK
1.21	2018/03/29	MOB	- Update URL to cgi
1.22	2019/07/17	AMP	- Added purQuery response status & code

1.23	2019/07/18	MOB	- Purcancel
1.24	20200928	MOB	- Refund Reversal
1.25	20210911	MOB	- URL Update

This document describes the detailed implementation requirements for mobicred Web Services, including how to configure the necessary web service requests and parsing the responses.

1. mobicred Usage & Integration Requirements

At time of integration, the merchant will be provided with 3 sets of credentials:

1. **An API username and password to be used to authenticate the merchant's API requests to mobicred (TEST and subsequently LIVE once signed off)**

2. **A Merchant ID and Key for each business unit under the merchant umbrella (TEST and subsequently LIVE once signed off)**

3. **A customer account username and password (TEST only)**

The merchant must also supply mobicred with their LIVE and if required, TEST, IPs which will be used to restrict access to the APIs.

CI and branding guidelines will be supplied by the mobicred sales team, and once the merchant has demonstrated that their integration is complete from both a technical and aesthetic/user experience perspective, will mobicred provide LIVE credentials for use in production.

The mobicred Test and Live API URLs are as follows:

TEST - https://test.mobicred.co.za/web_mcrtst/rest.w?

LIVE - https://live.mobicred.co.za/web_mcrliv/rest.w?

The offering of mobicred as a payment method involves incorporating three processes, namely:

Customer Applications – A merchant's customer will need to apply for the mobicred product before purchasing if they do not already have a mobicred account. The application process would be invoked either by the customer visiting the mobicred.co.za site or by the customer clicking on an 'apply now' link on the merchant checkout pages after selecting mobicred as a payment method. The merchant would redirect the customer to a secure mobicred application URL in order for the customer to apply. After a filling out a short questionnaire, the customer would either be approved or declined, and subsequently be redirected back to the specified URL appended by the merchant to the application page redirect to continue with the purchase.

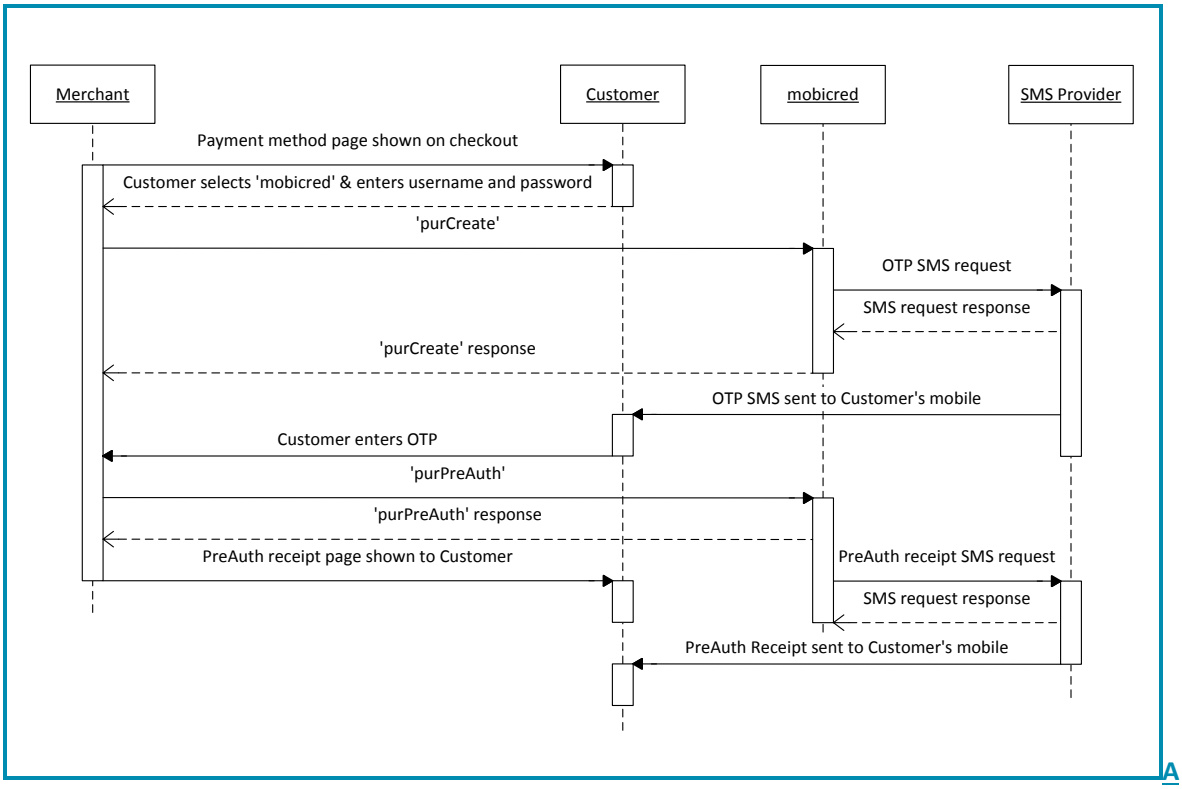
Customer Purchase Pre-Authorisations – Only customers who carry valid mobicred account credentials and where their account has been completely verified by mobicred may use the product to purchase online.

There are 3 possible reasons why an account may not be verified:

1. The customer is new and has just completed the application process whereby their banking details still need to be verified. This is not a real-time process and can take anywhere between 2 minutes and 24 hours.
2. The customer is an existing mobicred customer and has changed their banking details. This will re-initiate the banking details verification process.
3. The customer's purchasing behaviour has triggered mobicred fraud checks and the account is being reviewed.

After selecting mobicred as a payment option on the merchant's secure checkout pages, the customer must be asked to enter a mobicred username (email format) and password. The password must be hashed on screen entry by the merchant and both the username and password may not be stored. The merchant must then pass these credentials along with the details of the transaction to the mobicred API using the purCreate call. If provided with a successful response, the merchant must capture a One Time PIN (OTP) from the customer and submit this to the API referencing the original purchase request (purPreAuth).

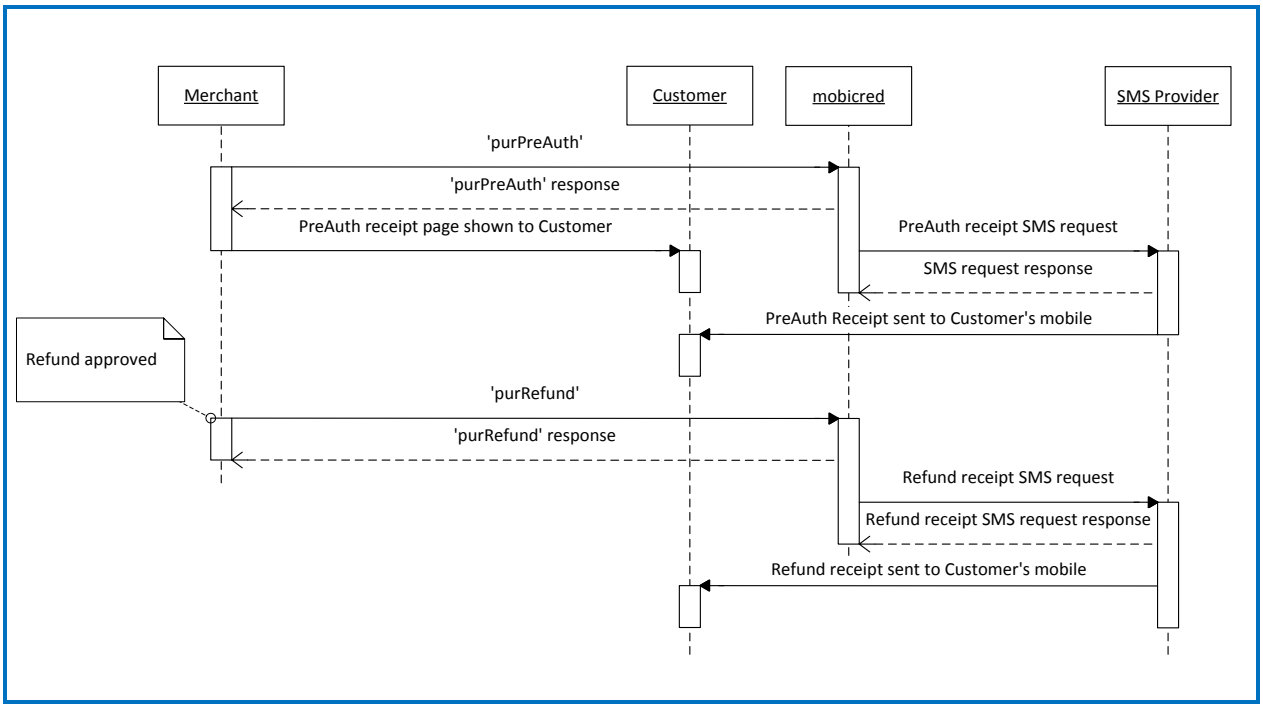
A successful purchase pre-authorisation (approved OK) means that the merchant may confirm the order as the funds have been reserved on the customer's account. All approved transactions will be settled weekly with the merchant.



Purchase Pre-Authorisation

Customer Purchase Updates – Should the customer require a refund on an approved pre-authorised purchase transaction, the merchant may submit a 'purRefund' call to the mobicred API and the transaction may be reversed. Both partial and full refunds are supported by mobicred.

In some extreme cases, mobicred may identify fraudulent purchasing activity on an account and may ask the merchant to refund the transaction. This will be coordinated between the Operational teams of mobicred and the participating merchant.



Asynchronous Refund of an Approved Pre-Authorisation

2. Application Page Redirects

On full-page redirect of the customer from the merchant's site to the mobicred product application page, 2 elements must be incorporated into the URL by the merchant, namely the merchant ID supplied by mobicred and the URL to which the merchant wished the customer to be redirected back to on completion of the application process.

For example:
https://live.mobicred.co.za/web_mcrliv/run.w?run=application&merchantId=xxxxx&returnUrl=xxxxxxx

Where **NNNN** must be replaced by the **merchant ID** assigned by mobicred to the merchant (provided on signing up), and the bold URL is the URL to which the merchant would like the customer come back to e.g. checkout page or homepage

It is strongly advised that the return URL is encoded by the merchant.

3. Product Interest Rate

To ensure that all references to the mobicred interest rate are used and displayed correctly on the Merchant's website, mobicred expose a getCurrentIntRate API. It is advised that the merchant call this API once a day, preferably in the early morning, to ensure that their stored interest rate is up to date. Please see the API section for more details.

4. Web Service Overview

4.1 What is a Web Service?

The mobicred API provides these web services as an interface for requesting mobicred API data from external services and using them within your Website to facilitate online purchases.

These web services use HTTP requests to specific URLs, passing URL parameters as arguments to the services. Generally, these services return data in the HTTP request as either JSON or XML for parsing and/or processing by your Website.

A typical Web Service request is generally of the following form:

```
http://live.mobicred.co.za/web_mcrliv/rest.w?rqDataMode=VAR/output&rqService=service&param1=...
```

Where service indicates the particular service requested and output indicates the response format (usually json or xml).

The following information describes some common practices useful for setting up your web service requests and processing your web service responses.

4.2 SSL Access

You are required to access the mobicred API Web Services over **HTTPS** as these applications include sensitive user data. To do so, change the protocol in your request URL to https as shown below:

```
https://live.mobicred.co.za/web_mcrliv/rest.w?rqDataMode=VAR/output&rqService=service&param1=...
```

4.3 Building a valid URL

You may think that a "valid" URL is self-evident, but that's not quite the case. A URL entered within an address bar in a browser, for example, may contain special characters (e.g."上海+中國"); the browser needs to internally translate those characters into a different encoding before transmission. By the same token, any code that generates or accepts UTF-8 input might treat URLs with UTF-8 characters as "valid", but would also need to translate those characters before sending them out to a web server. This process is called URL-encoding.

We need to translate special characters because all URLs need to conform to the syntax specified by the W3 Uniform Resource Identifier specification. In effect, this means that URLs must contain only a special subset of ASCII characters: the familiar alphanumeric symbols, and some reserved characters for use as control characters within URLs. The table below summarizes these characters:

Set	Characters	URL usage
Alphanumeric	a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9	Text strings, scheme usage (http), port (8080), etc.
Unreserved	- _ . ~	Text strings
Reserved	! * ' ( ) ; : @ & = + $, / ? % # [ ]	Control characters and/or Text Strings

When building a valid URL, you must ensure that it contains only those characters shown above. Conforming a URL to use this set of characters generally leads to two issues, one of omission and one of substitution:

Characters that you wish to handle exist outside of the above set. For example, characters in foreign languages such as 上海+中國 need to be encoded using the above characters. By popular convention, spaces (which are not allowed within URLs) are often represented using the plus '+' character as well.

Characters exist within the above set as reserved characters, but need to be used literally. For example, ? is used within URLs to indicate the beginning of the query string; if you wish to use the string "? and the Mysterions," you'd need to encode the '?' character.

All characters to be URL-encoded are encoded using a '%' character and a two-character hex value corresponding to their UTF-8 character. For example, 上海+中國 in UTF-8 would be URL-encoded as %E4%B8%8A%E6%B5%B7%2B%E4%B8%AD%E5%9C%8B. The string ? and the Mysterians would be URL-encoded as %3F+and+the+Mysterians.

Converting a URL that you receive from user input is sometimes tricky. For example, a user may enter an address as "5th&Main St." Generally, you should construct your URL from its parts, treating any user input as literal characters.

Additionally, URLs are limited to 2048 characters for all web services. For most services, this character limit will seldom be approached. However, note that certain services have several parameters that may result in long URLs.

All supported methods are URL request, HTTP verbs: GET, POST. The mobicred system makes no distinction between GET and POST – they can be used interchangeably (no need to use GET to fetch data and POST to create data).

NB! Using GET or URL requests is great for testing and getting things working, however for security reasons production servers should use the POST method which will ensure that all the parameter's data is encrypted over the HTTPS connection.

4.4 Processing Responses

As the exact format of individual responses with a web service request is not guaranteed (some elements may be missing or in multiple locations), you should never assume that the format returned for any given response will be the same for different queries. Instead, you should **process** the response and select appropriate values via **expressions**. This section discusses how to extract these values dynamically from web service responses.

The mobicred Web Services provide responses that are easy to understand and provide a number of specific values. Generally, you will want to parse responses from the web service and extract only those values that you require.

The parsing scheme you use depends on whether you are returning output in XML or JSON. JSON responses, being already in the form of Javascript objects, may be processed within Javascript itself on the client; XML responses should be processed using an XML processor and an XML query language to address elements within the XML format. We use XPath in the following examples, as it is commonly supported in XML processing libraries.

4.5 Processing XML with XPath

XML is a relatively mature structured information format used for data interchange. Although it is not as lightweight as JSON, XML does provide more language support and more robust tools. Code for processing XML in Java, for example, is built into the **javax.xml** packages.

When processing XML responses, you should use an appropriate query language for selecting nodes within the XML document, rather than assume the elements reside at absolute positions within the XML markup. XPath is a language syntax for uniquely describing nodes and elements within an XML document. XPath expressions allow you to identify specific content within the XML response document.

XPath Expressions

Some familiarity with XPath goes a long way towards developing a robust parsing scheme. This section will focus on how elements within an XML document are addressed with XPath, allowing you to address multiple elements and construct complex queries.

XPath uses *expressions* to select elements within an XML document, using a syntax similar to that used for directory paths. These expressions identify elements within an XML document tree, which is a hierarchical tree similar to that of a DOM. Generally, XPath expressions are greedy, indicating that they will match all nodes which match the supplied criteria.

The following is an abstract XML to illustrate our examples:

```
<rqResponse>
 <rqAuthentication></rqAuthentication>
 <pcMCReference>mcr00009101<pcMCReference>
 <pcMerchantRequestID>000005<pcMerchantRequestID>
 <pdtDateTime>04/06/2013 23:58:03.161+02:00<pdtDateTime>
 <piResponseCode>1<piResponseCode>
 <pcStatus>Pending<pcStatus>
 <pcReason><pcReason>
 <pcRecCustomerMsg><pcRecCustomerMsg>
 <pcCustomField><pcCustomField>
 <rqErrorMessage></rqErrorMessage>
</rqResponse>
```

Node Selection in Expressions

XPath selections select *nodes*. The root node encompasses the entire document. You select this node using the special expression "/". Note that the root node is not the top-level node of your XML document; actually, it resides one level above this top-level element and includes it.

Element nodes represent the various elements within the XML document tree.

A <rqResponse> element, for example, represents the top-level element returned in our sample service above. You select individual nodes either via absolute or relative paths, indicated by the presence or absence of a leading "/" character.

Absolute path: the "/rqResponse/pcMCReference" expression selects all <pcMCReference> nodes that are children of the <rqResponse> node. (Note that both of these elements descend from the root node "/".)

Relative path from the current context: the expression "pcMCReference" would match any <pcMCReference> elements within the current context. Generally, you shouldn't have to worry about context, as you're usually processing web service results via a single expression.

Either of these expressions may be augmented through addition of a wildcard path, indicated with a double-slash ("//"). This wildcard indicates that zero or more elements may match in the intervening path. The XPath expression "//address," for example, will match all nodes of that name in the current document. The expression//viewport//lat would match all <lat> elements that can trace <viewport> as a parent.

By default, XPath expressions match all elements. You can restrict the expression to match a certain element by providing a *predicate*, which is enclosed in square brackets ([]). The XPath expression "/rqResponse/result[2] always returns the second result, for example.

XPath Expression	Type of Expression	Selection
"/"	Root node	<rqResponse> <rqAuthentication></rqAuthentication> <pcMCReference>mcr00009101<pcMCReference> <pcMerchantRequestID>000005<pcMerchantRequestID> <pdtDateTime>04/06/2013 23:58:03.161+02:00<pdtDateTime> <piResponseCode>1<piResponseCode> <pcStatus>Pending<pcStatus> <pcReason><pcReason> <pcRecCustomerMsg><pcRecCustomerMsg> <pcCustomField><pcCustomField> <rqErrorMessage></rqErrorMessage> </rqResponse>
"/rqResponse/pcMCReference"	Absolute Path	<pcMCReference>mcr00009101<pcMCReference>

It is important to note that when selecting elements, you select nodes, not just the text within those objects. Generally, you will want to iterate over all matched nodes and extract the text. You may also match text nodes directly; see Text Nodes below.

Note that XPath supports attribute nodes as well; however, all mobicred web services serve elements without attributes, so matching of attributes is not necessary.

Text Selection in Expressions

Text within an XML document is specified in XPath expressions via a *text node* operator. This operator "text()" indicates extraction of text from the indicated node. For example, the XPath expression "//formatted_address/text()" will return all text within <formatted_address> elements.

XPath Expression	Type of Expression	Selection
"//text()"	All text nodes (including whitespace)	Mcr00009101 000005 04/06/2013 23:58:03.161+02:00 1 Pending

Alternatively, you may evaluate an expression and return a set of nodes and then iterate over that "node set," extracting the text from each node. We use this approach in the example below.

For more information on XPath, consult the XPath W3C Specification.

Evaluating XPath in Java

Java has wide support for parsing XML and using XPath expressions within the javax.xml.xpath.* package. For that reason, the sample code in this section uses Java to illustrate how to handle XML and parse data from XML service responses.

To use XPath in your Java code, you will first need to instantiate an instance of an **XPathFactory** and call newXPath() on that factory to create an **XPath** object. This object can then process passed XML and XPath expressions using the evaluate() method.

When evaluating XPath expressions, make sure that you iterate over any possible "node sets" which may be returned. Because these results are returned as DOM nodes in Java code, you should capture such multiple values within a NodeList object and iterate over that object to extract any text or values from those nodes.

The following code illustrates how to create an XPath object, assign it XML and an XPath expression, and evaluate the expression to print out the relevant content.

```
import org.xml.sax.InputSource;
import org.w3c.dom.*;
import javax.xml.xpath.*;
import java.io.*;

public class SimpleParser {

 public static void main(String[] args) throws IOException {

     XPathFactory factory = XPathFactory.newInstance();

  XPath xpath = factory.newXPath();

  try {
   System.out.print("Web Service Parser 1.0\n");

   // In practice, you'd retrieve your XML via an HTTP request.
   // Here we simply access an existing file.
   File xmlFile = new File("XML_FILE");

   // The xpath evaluator requires the XML be in the format of an InputSource
      InputSource inputXml = new InputSource(new FileInputStream(xmlFile));

   // Because the evaluator may return multiple entries, we specify that the expression
   // return a NODESET and place the result in a NodeList.
   NodeList nodes = (NodeList) xpath.evaluate("XPATH_EXPRESSION", inputXml, XPathConstants.NODESET);

   // We can then iterate over the NodeList and extract the content via getTextContent().
   // NOTE: this will only return text for element nodes at the returned context.
   for (int i = 0, n = nodes.getLength(); i < n; i++) {
    String nodeString = nodes.item(i).getTextContent();
    System.out.print(nodeString);
    System.out.print("\n");
   }
  } catch (XPathExpressionException ex) {
      System.out.print("XPath Error");
  } catch (FileNotFoundException ex) {
   System.out.print("File Error");
  }
 }
}
```

4.6 Processing JSON with Javascript

JSON (Javascript Object Notation) has an obvious advantage over XML in that the response is lightweight. Parsing such a result is trivial in JavaScript as the format is already a valid Javascript object. For example, to extract the value of the 'formatted_address' keys within a JSON result object, simply access them using the following code:

```
for (i = 0; i < myJSONResult.results.length; i++) {
  myAddress[i] = myJSONResult.results[i].formatted_address;
}
```

Note that because JSON may contain multiple values, it's wisest to iterate over the length of the results array if you want to capture all possible values. In practice, you may wish to only return the first result (results[0]), however.

Parsing JSON in other languages is only moderately more difficult. The following Python example initiates a Geocoding web service request and displays all resultingformatted_address values to the user within an array:

```
import simplejson, urllib

GEOCODE_BASE_URL = 'http://maps.googleapis.com/maps/api/geocode/json'

def geocode(address,sensor, **geo_args):
    geo_args.update({
        'address': address,
        'sensor': sensor
    })

    url = GEOCODE_BASE_URL + '?' + urllib.urlencode(geo_args)
    result = simplejson.load(urllib.urlopen(url))

    print simplejson.dumps([s['formatted_address'] for s in result['results']], indent=2)

if __name__ == '__main__':
    geocode(address="San+Francisco",sensor="false")

Output:

[
  "San Francisco, CA, USA"
]
```

5 mobicred API Calls

An example of the mobicred request URL format (purchase OTP) is shown below:

https://test.mobicred.co.za/web_mcrtst/rest.w?rqDataMode=VAR/JSON&rqAuthentication=user:test_merchant|pa55w0rd|GSMUS|%26login_company_obj%3d-1%26login_company_branch_obj%3d-1%26process_date%3d2014/03/15&rqservice=ilDataService:purOTP&cMerchantID=1001&cMerchantKey=1733540827&cMerchantRequestID=1001&cMCReference=20000000150

5.1 Purchase Create

First trigger to assess customer username and password and if adequate funds available to purchase item. Purchase amount in reserved status subject to customer authorisation.

purCreate			
Request	**Format (Mandatory / Optional)**	**Response**	**Format / Description**
cMerchantID	8 N characters (M)	pcMCReference	11 N characters unique for each mC transaction
cMerchantKey	system generated (M)	pcMerchantRequestID	echo from request
cMerchantRequestID	min 6 max 30 AN characters, must be unique for each request per MerchantID (M)	pdtDateTime	yyyymmdd hh:mm:ss +hh:mm
cCustUsername	AN mC username, must be valid email address (M)	piResponseCode	See relevant table
cCustPasswd	min 8 AN characters (M)	pcStatus	See relevant table
lAutoApprove	Must be set to 'True'	pcReason	See relevant table
cOrderNo	max 30 AN characters (O)	pcRecCustomerMsg	See relevant table
dAmount	N with decimal place (M)	pcCustomField	echo from request
cCustomField	AN 128 characters (O)		

NOTE: Transactions in a Created OK state require a One Time PIN (OTP) to be approved. OTPs expire and must be submitted within a certain period of time after generation (usually set to 10 minutes).

AutoApprove allows for auto authorization of the purchase transaction, if set to TRUE the purApprove API will not be run.

5.2 Purchase OTP

Triggered to resend OTP to the customer for purchase pre-authorisation.

purOTP			
Request	Format (Mandatory / Optional)	Response	Format / Description
cMerchantID	8 N characters (M)	pcMCReference	echo from request
cMerchantKey	system generated (M)	pcMerchantRequestID	echo from request
cMerchantRequestID	min 6 max 30 AN characters, must be unique for each request per MerchantID (M)	pdtDateTime	yyyymmdd hh:mm:ss +hh:mm
cMCReference	from purCreate response	piResponseCode	See relevant table
		pcStatus	See relevant table
		pcReason	See relevant table
		pcRecCustomerMsg	See relevant table

NOTE: There is a maximum amount of purOTP requests that can be sent for each MCReference after which a 'Declined' code 211 would be returned.

5.3 Purchase Pre-Authorise

Triggers a verification of the OTP (the customers confirmation of transaction) and if correct, it reduces the account balance by the initial requested amount.

purPreAuth

Request	Format (Mandatory / Optional)	Response	Format / Description
cMerchantID	8 N characters (M)	pcMCReference	echo from request
cMerchantKey	system generated (M)	pcMerchantRequestID	echo from request
cMerchantRequestID	min 6 max 30 AN characters, must be unique for each request per MerchantID (M)	pdtDateTime	yyyymmdd hh:mm:ss +hh:mm
cMCReference	from purCreate response	piResponseCode	See relevant table
iOTP	6 N characters captured from customer	pcStatus	See relevant table
		pcReason	See relevant table
		pcRecCustomerMsg	See relevant table

5.4 Purchase Cancel

Should the merchant wish to fully cancel transaction before approval stage, or customer does not want to proceed with order prior to approval stage purchase cancel must be triggered to reverse full amount of pending purchase.

If autoapprove=false is used in a purchase's purCreate call, then after purPreAuth either purCancel needs to be triggered or purApprove to complete the steps

purCancel			
Request	Format (Mandatory / Optional)	Response	Format / Description
cMerchantID	8 N characters (M)	pcMCReference	echo from request
cMerchantKey	system generated (M)	pcMerchantRequestID	echo from request
cMerchantRequestID	min 6 max 30 AN characters, must be unique for each request per MerchantID (M)	pdtDateTime	yyyymmdd hh:mm:ss +hh:mm
cMCReference	from purCreate response (M)	piResponseCode	See relevant table
cMerchantReason	3 A characters (O) - See relevant table	pcStatus	See relevant table
		pcReason	See relevant table
		pcRecCustomerMsg	See relevant table

5.5 Purchase Approve

When merchant is ready to ship goods this trigger will fully approve transaction amount (ready for merchant settlement).

If autoapprove=false is used in a purchase's purCreate call, then after purPreAuth either purCancel needs to be triggered or purApprove to complete the steps

AutoApprove allows for auto authorization of the purchase transaction, if set to TRUE the purApprove API will not be run.

purApprove			
Request	Format (Mandatory / Optional)	Response	Format / Description
cMerchantID	8 N characters (M)	pcMCReference	echo from request
cMerchantKey	system generated (M)	pcMerchantRequestID	echo from request
cMerchantRequestID	min 6 max 30 AN characters, must be unique for each request per MerchantID (M)	pdtDateTime	yyyymmdd hh:mm:ss +hh:mm
cMCReference	from purCreate response	piResponseCode	See relevant table
		pcStatus	See relevant table
		pcReason	See relevant table
		pcRecCustomerMsg	See relevant table

5.6 Purchase Refund

This is triggered to either partially or fully refund a previously fully approved purchase. purCancel is used to cancel a purchase that is not yet fully approved.

purRefund			
Request	Format (Mandatory / Optional)	Response	Format / Description
cMerchantID	8 N characters (M)	pcMCReference	New MCReference

			for Refund
cMerchantKey	system generated (M)	pcMerchantRequestID	from request
cMerchantRequestID	min 6 max 30 AN characters, must be unique for each request per MerchantID (M)	pcRefundReference	MCReference from request
cMCReference	from purCreate response (M)	pdtDateTime	yyyymmdd hh:mm:ss +hh:mm
dAmount	N with decimal place (M)	piResponseCode	See relevant table
cMerchantReason	3 A characters (O) - See relevant table	pcStatus	See relevant table
		pcReason	See relevant table
		pcRecCustomerMsg	See relevant table

5.7 Purchase Refund Reversal

This is triggered to be able to either partially or fully reverse a refund incorrectly processed.

purRefundReverse			
Request	**Format (Mandatory / Optional)**	**Response**	**Format / Description**
cMerchantID	8 N characters (M)	pcMCReference	New MCReference for Refund Reversal
cMerchantKey	system generated (M)	pcMerchantRequestID	from request
cMerchantRequestID	min 6 max 30 AN characters, must be unique for each request per MerchantID (M)	pcRefundReference	MCReference from request
cMCReference	from purRefund response (M)	pdtDateTime	yyyymmdd hh:mm:ss +hh:mm
dAmount	N with decimal place (M)	piResponseCode	See relevant table
cMerchantReason	3 A characters (O) - See relevant table	pcStatus	See relevant table
		pcReason	See relevant table
		pcRecCustomerMsg	See relevant table

5.8 Purchase Query

This is triggered to find out the state of a purchase transaction. It is merely an information request and can be used in case of communication failures.

purQuery			
Request	**Format (Mandatory / Optional)**	**Response**	**Format / Description**
cMerchantID	8 N characters (M)	pcMCReference	echo from request

cMerchantKey	system generated (M)	pcPurchaseState	Current state of transaction (i.e Created OK, Pre-Authorised OK, Approved OK, Paid Up OK, Declined OK, Cancelled OK)
cMCReference	from purCreate response (M)	pdtDateTimeChanged	date time TZ transaction entered current state
		pdTranAmt	Amount relevant to the transaction queried
		pdPurchaseBal	Current purchase balance
		piResponseCode	See relevant table
		pcStatus	See relevant table
		pcReason	See relevant table
		pcRecCustomerMsg	See relevant table

5.9 Get Current Interest Rate

It is recommended that this API be called once at the beginning of every day to confirm the interest rate of the mobicred product.

getCurrentIntRate			
Request	**Format (Mandatory / Optional)**	**Response**	**Format / Description**
cMerchantID	8 N characters (M)	pcIntRate	N with 2 decimal places
cMerchantKey	system generated (M)	piResponseCode	See relevant table
	from purCreate response (M)	pcStatus	See relevant table
		pcReason	See relevant table
		pcRecCustomerMsg	See relevant table

5.10 Web Hook

This API can be used to automate triggering of the first purchase that the customer had in their basket when they applied for mobicred from the merchant's site.

In order to use this functionality the following additional elements need to be appended to the redirect URL (refer to 2. Application Page Redirects):

- &callbackURL=value
  - (the encoded URL the merchant wants the web hook sent to)
- &autoApprove=value
  - (optional, will default to true)
- &orderNo=value
  - (basket ID)
- &amount=value
  - (purchase amount)

The above information is stored linked to the customer after submitting the first page of the application.

Once the application is complete, the mobicred call center follow the process of vetting that customer.

If the customer passes the vetting then the mobicred system changes the account's situation and sends an SMS to the customer.

At that point if all the information exists that is required by the web hook then internally the mobicred system does the purCreate to create the transaction on the customer's account.

If that is successful the mobicred system generates an OTP linked to the transaction (no OTP is sent to the customer).

Then the web hook call is sent to the callbackURL telling the merchant they can continue the automated transaction process using the data provided.

This will be an HTTP POST with a JSON payload.

The information that is sent to the callback URL:

- pcMCReference
  - (mobicred transaction number to be used in subsequent purchase API calls)
- iOTP
  - (to be used in subsequent purPreAuth, OTP expires after 30 min)
- lAutoApprove
  - (indicates whether merchant must do purPreAuth and purApprove separately or not)
- cOrderNo
  - (echo what we stored at application)
- dAmount
  - (echo what we stored at application)

The raw JSON data sent to the callbackURL is as follows:

```
{
   "cMCReference": "2000000001",
   "iOTP": 1234,
   "lAutoApprove": true,
   "cOrderNo": "1",
   "dAmount": 100.12
}
```

The response from the callbackURL should be in the following format:

Success:

```
{
   "cStatus": "SUCCESS",
   "cErrorMessage": “”
}
```

Error:

```
{
   "cStatus": "ERROR",
   "cErrorMessage": “Basket ID not found for Order No”
}
```

The mobicred system will simply be logging the response, there isn't a different outcome based on success or error.

On the merchant's side they will use the data sent in the web hook to decide whether to send a subsequent purPreAuth (using iOTP provided in webhook data for security) or purCancel call to mobicred. This would be based on whether the basket, purchase amount and stock availability are still valid etc.

After the merchant triggers the appropriate purchase API, the customer will get sent an SMS letting them know what is happening with their first purchase at the merchant.

6 Code Samples

6.6 PHP

```
<?php
    Protected function mcrAPI($api = false, $parameters = false, $dataMode = 'json',
$test = true){
        $curl = curl_init();
        $url = 'https://'.($test ? 'test' : 'live').'.mobicred.co.za/web_mcr'.($test
? 'tst' : 'liv').'/rest.w';

        curl_setopt($curl,CURLOPT_POST, true);
        if (!$api || !$parameters || $this -> username ==  '' || $this -> password ==
'') return false;

        $dataMode_param       ='VAR/'.($dataMode == 'xml' ? 'XML' : 'JSON') ;
        $service_param        = 'ilDataService:'.$api;
        $authentication_param = "user:{$this -> username}|{$this ->
password}|GSMUS|&login_company_obj=-1&login_company_branch_obj=-
1&process_date=".date('Y/m/d');

        $data = '?rqDataMode='       . urlencode($dataMode_param) .
                '&rqService='        . urlencode($service_param) .
                '&rqAuthentication=' . urlencode($authentication_param);

        foreach($parameters as $key=>$value){
            $parameters_str .= '&' . $key . '=' . urlencode($value);
        }

        $link = $url.$data.$parameters_str;

        curl_setopt_array($curl,array(
                                    CURLOPT_URL              => $link,
                                    CURLOPT_RETURNTRANSFER   => true,
                                    CURLOPT_ENCODING         => "",
                                    CURLOPT_MAXREDIRS        => 10,
                                    CURLOPT_TIMEOUT          => 30,
                                    CURLOPT_HTTP_VERDION     =>
CURL_HTTP_VERSION_1_1,
                                    CURLOPT_CUSTOMREQUEST    => "POST"
                                    )
                            );

        $response = curl_exec($curl);
        $err = curl_error($curl);
        curl_close($curl);

        $response = json_decode($response);

        if ($err) {
                    $this->log('cURL Error: '. $err . PHP_EOL);
```

```
        }
        return $response->rqResponse;
    }
?>
```

6.7.NET

```
using System;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Threading.Tasks;

  class PurResult
  {
     public string     pcMCReference  { get; set; }
     public string     pcMerchantRequestID  { get; set; }
     public DateTime    pdtDateTime  { get; set; }
     public int        piResponseCode  { get; set; }
     public string     pcStatus  { get; set; }
     public string     pcReason  { get; set; }
     public string     pcRecCustomerMsg  { get; set; }
     public string     pcCustomFieldI:\tmp\jameelm\.Net purCreate REST consumption.txtd  { get; set; }

  }



  class Program
  {

     // Creating authentication variables
      public string rqDataMode = "VAR/JSON",
            usernameAndPassword = "redwood|password123|GSMUS|",
            loginCompany = "-1",
```

```
        loginCompanyBranch = "-1",
        processDate = "2013/07/22";

  // Creating input parameters
 public string rqservice = "ilDataService:purCreate",
        cMerchantID = "1001",
        cMerchantKey = "1733700827",
        cMerchantRequestID = "11111",
        cCustUsername = "abrah@mweb.com",
        cCustPasswd = "fkD02",
        lAutoApprove = "true",
        cOrderNo = "1",
        dAmount = "100",
        cCustomField = "test";

  // These parameters (input) are used for testing only
  public string testrqservice = "ilDataService:purQuery",
        testcMerchantID = "1001",
        testcMerchantKey = "1733700827",
        testcMCReference = "20000000358";
  // Testing parameters end


string restURL = "https://test.mobicred.co.za/web_mcrtst/rest.w?rqDataMode="
+ rqDataMode
+ "&rqAuthentication=User:" + usernameAndPassword
+ "|%26login_company_obj%3D"+ loginCompany
+ "%26login_company_branch_obj%3D" + loginCompanyBranch
+ "%26process_date%3D" +  processDate
+ "&rqservice=" + rqservice
+ "&cMerchantID=" + cMerchantID
+ "&cMerchantKey=" + cMerchantKey
```

```
    + "&cMerchantRequestID=" + cMerchantRequestID
    + "&cCustUsername=" + cCustUsername
    + "&cCustPasswd=" + cCustPasswd
    + "&lAutoApprove=" + lAutoApprove
    + "&cOrderNo=" + cOrderNo
    + "&dAmount=" + dAmount
    + "&cCustomField=" + cCustomField ;

    static void Main()
    {
       RunAsync().Wait();
    }

    static async Task RunAsync()
    {
       using (var client = new HttpClient())
       {
          client.BaseAddress = new Uri(restURL);

         // HTTP GET
          HttpResponseMessage response = await client.GetAsync();
          if (response.IsSuccessStatusCode)
          {
             PurResult purResult = await response.Content.ReadAsAsync<PurResult>();
             Console.WriteLine("{0}\t${1}\t{2}\t{3}\t{4}\t{5}\t{6}\t{7}", purResult.pcMCReference,
purResult.pcMerchantRequestID, purResult.pdtDateTime, purResult.piResponseCode, purResult.pcStatus,
purResult.pcReason, purResult.pcRecCustomerMsg, purResult.pcCustomField);
          }

       }
    }
  }
```

6.8JAVA CODE

```
// Call rest service and return JSON object as String
    private String sendAndReceiveJSON() {

        HttpURLConnection conn = null;

        try {

            /* Notes
             * %26 = &
             * %3d = :
             Notes end */

            // Creating authentication variables
            String rqDataMode = "VAR/JSON",
                    usernameAndPassword = "redwood|password123|GSMUS|",
                    loginCompany = "-1",
                    loginCompanyBranch = "-1",
                    processDate = "2013/07/22";

            // Creating input parameters
            String rqservice = "ilDataService:purCreate",
                    cMerchantID = "1001",
                    cMerchantKey = "1733700827",
                    cMerchantRequestID = "11111",
                    cCustUsername = "abrah@mweb.com",
                    cCustPasswd = "fkD02",
                    lAutoApprove = "true",
                    cOrderNo = "1",
                    dAmount = "100",
                    cCustomField = "test";

            // These parameters (input) are used for testing only
            String testrqservice = "ilDataService:purQuery",
                    testcMerchantID = "1001",
                    testcMerchantKey = "1733700827",
                    testcMCReference = "20000000358";
            // Testing parameters end

            // Creating the string for the url
            String urlString = "https://test.mobicred.co.za/web_mcrtst/rest.w?";

            // Creating the StringBuilder to build the entire string
            StringBuilder stringBuilder = new StringBuilder(urlString);

            // Filling in the authentication parameters
            stringBuilder.append("rqDataMode=");
            stringBuilder.append(rqDataMode);
            stringBuilder.append("&");
            stringBuilder.append("rqAuthentication=user:");
            stringBuilder.append(usernameAndPassword);
            stringBuilder.append("%26");
            stringBuilder.append("login_company_obj");
```

```
            stringBuilder.append("%3d");
            stringBuilder.append(loginCompany);
            stringBuilder.append("%26");
            stringBuilder.append("login_company_branch_obj");
            stringBuilder.append("%3d");
            stringBuilder.append(loginCompanyBranch);
            stringBuilder.append("%26");
            stringBuilder.append("process_date");
            stringBuilder.append("%3d");
            stringBuilder.append(processDate);
            stringBuilder.append("&");

        // Filling in the input parameters
            stringBuilder.append("rqservice=");
            stringBuilder.append(rqservice);
            stringBuilder.append("&");
            stringBuilder.append("cMerchantID=");
            stringBuilder.append(cMerchantID);
            stringBuilder.append("&");
            stringBuilder.append("cMerchantKey=");
            stringBuilder.append(cMerchantKey);
            stringBuilder.append("&");
            stringBuilder.append("cMerchantRequestID=");
            stringBuilder.append(cMerchantRequestID);
            stringBuilder.append("cMCReference=");
            stringBuilder.append("&");
            stringBuilder.append("cCustUsername=");
            stringBuilder.append(cCustUsername);
            stringBuilder.append("&");
            stringBuilder.append("cCustPasswd=");
            stringBuilder.append(cCustPasswd);
            stringBuilder.append("&");
            stringBuilder.append("lAutoApprove=");
            stringBuilder.append(lAutoApprove);
            stringBuilder.append("&");
            stringBuilder.append("cOrderNo=");
            stringBuilder.append(cOrderNo);
            stringBuilder.append("&");
            stringBuilder.append("dAmount=");
            stringBuilder.append(dAmount);
            stringBuilder.append("&");
            stringBuilder.append("cCustomField=");
            stringBuilder.append(cCustomField);

            // The actual URL
//            System.out.println(stringBuilder.toString());
            URL url = new URL(stringBuilder.toString());

            // Begin sending
            conn = (HttpURLConnection) url.openConnection();
            conn.setRequestMethod("GET");
            conn.setRequestProperty("Accept", "application/json");

            if (conn.getResponseCode() != 200) {
                throw new RuntimeException("Failed : HTTP error code : "
```

```
                    + conn.getResponseCode());
        }

        BufferedReader br = new BufferedReader(new InputStreamReader(
                (conn.getInputStream())));

        String output;
        StringBuilder jsonReponseStringBuilder = new StringBuilder();
        int lineCount = 0;

        while ((output = br.readLine()) != null) {
            jsonReponseStringBuilder.append(output);

            lineCount++;
            if (lineCount > 1) {
                jsonReponseStringBuilder.append("\n");
            }
        }

        return jsonReponseStringBuilder.toString();

    } catch (MalformedURLException e) {
        System.out.println("URL error: MalformedURLException\n");
        System.out.println(e.getMessage());

    } catch (IOException e) {
        System.out.println("IO error: IOException\n");
        System.out.println(e.getMessage());

    } finally {
        if (conn != null) {
            conn.disconnect();
        }
    }

    return null;
}
```

7 Service Codes

Status	Code	Reason	Notes
Success	0	“”	Used in purQuery (Data request not a command/action)
Pending	001	Created OK	
Pending	002	OTP Sent OK	
Pending	003	Pre-Authorised OK - Account NOT verified	
Pending	004	Pre-Authorised OK - Account verified	
Approved	101	Purchase Approved OK	
Approved	102	Purchase Cancelled OK	
Approved	103	Purchase Refunded OK	
Declined	201	Invalid Username	
Declined	202	Incorrect Password	
Declined	203	Customer account not found	
Declined	204	Account Requires Verification	Customer account is undergoing verification process
Declined	206	Account in Arrears	
Declined	207	Insufficient Funds	
Declined	208	OTP Incorrect	
Declined	209	OTP Expired	Each OTP has a limited time to live, usually around 15 minutes
Declined	210	Maximum OTP incorrect attempts	
Declined	211	Maximum OTP resend requests reached	
Error	0	Error message	Used in purQuery (Data request not a command/action)
Error	301	MerchantID cannot be blank	
Error	302	MerchantID not found	
Error	303	MerchantID disabled	
Error	304	MerchantKey cannot be blank	
Error	305	MerchantKey incorrect	
Error	306	MerchantRequestID cannot be blank	
Error	307	Duplicate MerchantRequestID	
Error	308	Error saving MerchantRequestID	
Error	309	Amount cannot be blank	
Error	310	MerchantID not enabled for autocapture	
Error	311	Effective product transaction not found	

Error	312	OTP not sent	
Error	313	MCReference not valid	
Error	314	Customer account record not found for transaction	
Error	315	Customer record not found for transaction	
Error	316	OTP not required	
Error	317	Error Posting transaction to GL	
Error	318	User login has insufficient privileges	
Error	319	Cannot refund more than purchase balance	
Error	320	Invalid Merchant IP Address	
Error	401	Purchase is in created state	
Error	403	Purchase is in approved state	
Error	405	Purchase in refunded state	
Error	406	Purchase in declined state	
Error	407	Purchase in an unknown state	

MerchantReason	Description
FRD	Suspected Fraud
RTN	Goods returned
NST	No Stock
CAN	Customer cancelled order
DGG	Damaged Goods