# FGC STEP Purchase Cards (“PCards”) process External data and integration requirements

**About this document**

This document was written to provide clarity around what data is provided to the process and by the process, and how that data is exchanged. This will enable Cambridgeshire Fire and Rescue Service (“CFRS”) to consider the migration of the process from our in-house hardware and software stack to our BPM on Cloud solution, which would split the stack between in-house and Cloud. That would change the integration paradigms therefore a change of the Service Oriented Architecture (“SOA”) implementation.

Date of creation: 27 July 2016 by BPM Developer James Thompson.

**Summary of considerations:**

The PCards process is started remotely by message event from IBM Integration Designer (“IID” / “Broker”), which places a specifically structured XML message onto the IBM Business Process Manager (“BPM”) message queue. The details of this are covered below under section 1 “Remotely starting the PCards process”.

The PCards process is a process of essentially one task, delivered to one user once per month at most. Within this task we provide a user interface that requires access to their transaction data plus some supporting reference data (such as VAT Rates). When the user saves their data they invoke a means of updating their transaction data. Details of this are covered below under section 2 “Integration points required/used by the PCards process”.

*Content starts on next page…*

### Remotely starting the PCards process

BPM provides an Event Manager which expects a specifically structured XML structure. This structure has a simple envelope and a flexible type of body/payload. Official documentation for this can be found on IBM’s website at <http://www.ibm.com/support/knowledgecenter/SS964W/com.ibm.wbpm.wle.editor.doc/topics/sending\_message\_events.html>. Here’s a realistic example of the PCards message including envelope:-

>>>  
<**eventmsg**>  
 <**event** processApp="FGCPUCD" ucaname="New\_Statement\_Received\_UCA">  
 554bc144-6046-4b0d-a297-67dafa09d22f  
 </**event**>  
 <**parameters**>  
 <**parameter**>  
 <**key**>XmlDocument</**key**>  
 <**value**><![CDATA[\*\*\*\*\*\*]]></**value**>  
 </**parameter**>  
 </**parameters**>  
</**eventmsg**>  
<<<

Note in the above that the value for key “XmlDocument” contains just “\*\*\*\*\*\*” – that’s because its structure is detailed below, for clarity. Also note that this value should NOT contain an XML declaration; it is therefore equivalent to an XML Element object, not an XML Document object – despite the name of the key.

>>>  
<**Args**>  
 <**Arg**>  
 <**Name**>Account\_Number</**Name**>  
 <**Value**><![CDATA[471532\*\*\*\*\*\*1746]]></**Value**>  
 </**Arg**>  
 <**Arg**>  
 <**Name**>Network\_Login</**Name**>  
 <**Value**><![CDATA[james.a.thompson]]></**Value**>  
 </**Arg**>  
 <**Arg**>  
 <**Name**>FILENAME</**Name**>  
 <**Value**><![CDATA[Demo12112015\_TransactionReport.csv]]></**Value**>  
 </**Arg**>  
 <**Arg**>  
 <**Name**>TLD</**Name**><!-- “Time Loaded Date”, in reference to FILENAME arg. -->  
 <**Value**><![CDATA[2016-07-27]]></**Value**><!-- Must be ISO format “YYYY-MM-DD” -->  
 </**Arg**>  
 <**Arg**>  
 <**Name**>Cardholder\_Name</**Name**>  
 <**Value**><![CDATA[THOMPSON J A]]></**Value**>  
 </**Arg**>  
</**Args**>  
<<<

These parameters arrive in the business process definition (“BPD”) at the start of the PCards process, and are converted to the following JavaScript types:-

* Account\_Number = String
* Network\_Login = String
* FILENAME = String
* TLD = Date
* CardHolder\_Name = String

Regardless of how the PCards process might be remotely started when deployed to the BPM on Cloud environment, it currently requires at a minimum those five input parameters in a format that is amenable to those native JavaScript data types.

*See the next section, overleaf, for details of the integration points required/used by the process…*

### Integration points required/used by the PCards process

In a nutshell the PCards process requires two data READ services and one data WRITE service, as follows:-

* “Get Transactions within Period by Card Holder”: 3 inputs, 2 outputs.
* “Get Available VAT Rates for ALL Transactions in List”: 1 input, 2 outputs.
* “Save Transactions”: 2 inputs, 2 outputs

Details for each are below. Details of the data structures referred to in the below can be found in section 3 “Data Structres”.

**“Get Transactions within Period by Card Holder”**

Inputs:-

1. account\_number = String
2. tld = String (in format “01/Jan/2001”) //Oracle date format - note change from ISO format
3. filename = String

Outputs:-

1. transactions = Array of Transaction
2. errorStr = String or Null

*NOTE: The name of this service is mis-leading because it no longer accepts a date-range is input. All the pertinent data is now extracted from the database using just those three input parameters.*

SQL:-  
>>>  
select

rawtohex(TID) TRANSACTION\_ID,

ACCOUNT\_NUMBER,

CARDHOLDER\_NAME,

REFERENCE\_NUMBER,

DATE\_OCCURRED,

DATE\_POSTED,

MERCHANT,

MCC,

ORIGINAL\_CURRENCY\_CODE\_NUM, --eg 826.

ORIGINAL\_CURRENCY\_CODE, --eg USD.

ORIGINAL\_CURRENCY, --eg US Dollars.

CONVERSION\_RATE, --eg 0.6; use this to convert between ORIGINAL\_AMOUNT and SETTLEMENT\_AMOUNT as needed.

ORIGINAL\_AMOUNT, --Gross; in the local transaction currency (see below for currency cols).

SETTLEMENT\_AMOUNT, --Gross; in the native statement currency.

SETTLEMENT\_AMOUNT\_EXCL\_VAT, --Net; in the native statement currency.

VAT\_RATE\_PERC,

rawtohex(VAT\_RATE\_ID) VAT\_RATE\_ID,

VAT\_AMOUNT,

RECOGNISED,

RECEIPT\_AVAIL,

REASON,

PURCH\_DESCRIPTION,

coalesce( GL\_CODE, (COST\_CODE||CARDHOLDER) ) GL\_CODE,

DOC\_REF

from

gpc\_transaction\_vw

where

account\_number = ? --account\_number param. Varchar.

and tld = to\_date( ?, 'dd/Mon/yyyy' ) --tld param. Varchar (before cast to date).

and filename = ? --filename param. Varchar.

order by

date\_occurred desc   
<<<

**“Get Available VAT Rates for ALL Transactions in List”**

Inputs:-

1. transactions = Array of Transaction

Outputs:-

1. availableVatRates = Array of AvailableVatRatesForASingleTransaction //”a list of lists”.
2. errorStr = String or Null

NOTE: This service is a wrapper which calls a single-transaction-specific service “**Get VAT Rates valid for a given date**” for each transaction in the provided array. The inputs and outputs of this are defined below:-

Inputs:-

1. givenDate = String (in format “01/Jan/2001”) //value from Trans. col. “DATE\_OCCURRED”

Outputs:-

1. rates = list of VATRate
2. errorStr = String or Null.

SQL:-

>>>  
select

rawtohex(rd.datum\_id) as rate\_id,

rd.datum\_name as rate\_label,

rd.datum\_value as rate\_value

from

reference\_data\_vw rd

where

rd.context\_id = hextoraw('/\*UUID here meaning “VAT Rates” reference data type\*/') --unique ID for the context of "Vat Rates".

and

(

to\_date( ?, 'dd/Mon/yyyy' ) >= rd.start\_tms --provided date is greater than or equal to our vat rates' start date

and

(

rd.end\_tms is null -- EITHER our vat rate does NOT have an end date (meaning it is current), OR...

or

to\_date( ?, 'dd/Mon/yyyy' ) <= rd.end\_tms -- OR our vat rate has an end date and our date falls on or before it.

)

)

order by

(regexp\_replace(rd.datum\_value,'[^0-9.]',NULL,1,0)/100) desc --this regexp converts the string values of "20%" into a numeric value like "0.2" which can then be sorted properly.  
<<<

**“Save Transactions”**

Inputs:-

1. transactions = Array of Transaction
2. doc\_ref = String

Outputs:-

1. failedUpdates = Array of Transaction
2. errorStr = String or Null.

NOTE: This service is a wrapper which calls a single-transaction-specific service “**Update Transaction Status**” for each transaction in the provided array. The inputs and outputs of this are defined below:-

Inputs:-

1. transaction = Transaction
2. doc\_ref = String

Outputs:-

1. errorStr – String or Null

SQL:-

>>>  
call U\_TRANS\_VERIFICATION(

?, --Transaction ID [from the GPC\_TRANSACTION\_VW view]

?, --bIsRecognised [1|0]

?, --bHasReceipt [1|0]

?, --sWhyNoReceipt

?, --sDescription

?, --sGLCode [max 7 chars]

?, --VAT Rate ID from the REF\_DATA table

?, --VAT Amount

? --doc\_ref

)  
<<<

The SQL above is calling a Stored Procedure, with the following parameters:-

1. Transaction.id = VARCHAR
2. Transaction.bCardHolderRecognisesThisTransaction = VARCHAR (“0”|”1”|NULL)
3. Transaction.bCardHolderHasReceipt = VARCHAR (“0”|”1”|NULL)
4. Transaction.sWhyNoReceipt = VARCHAR
5. Transaction.purchaseDescription = VARCHAR
6. Transaction.generalLedgerCode = VARCHAR
7. Transaction.VatRate.id = VARCHAR
8. Transaction.VatAmount = VARCHAR
9. doc\_ref = VARCHAR

### Data Structures

**AvailableVATRatesForASingleTransaction**

* transaction\_index = Int, Unsigned (>= 0)
* transaction\_id = String
* transaction\_date = Date
* possibleVATRates = Array of VATRate

**VATRate**

* id = String
* label = String
* description = String
* value = Decimal / Float etc

**Transaction**

* id = String
* accountNumber = String
* cardHolderName = String
* reference\_number = String
* originalDate = Date
* recordedDate = Date
* merchant = String
* mcc = String
* currency = Currency //Localised to place of purchase at time of purchase. The overall “Statement” currency is always GBP, as set in the BPM process app, therefore see child property conversionRateAgainstNativeCurrency
* amountGrossOrig = String
* amountGross = String
* vatRate = VATRate
* vatAmount = String
* amountNet = String
* bCardHolderRecognisesThisTransaction = Boolean (or Null)
* bCardHolderHasReceipt = Boolean (or Null)
* sWhyNoReceipt = String
* purchaseDescription = String
* generalLedgerCode = String
* doc\_ref = String

**Currency**

* id = String
* label\_short = String
* label\_long = String
* conversionRateAgainstNativeCurrency = Decimal / Float etc.

End of document.