Network

International Payment Solutions

PAYMENT GATEWAY INTEGRATION BITMAP SAMPLE CODE C#

USER MANUAL

VERSION: 1.0.0

21-JUNE-2017

Contents

Introduction

The 'Payment gateway Integration Guide' provides guidance on the process of merchant integration using the Bitmap methodology. It also focuses on specific areas of integration such as encryption/decryption, encoding/decoding of transactions. In this integration, as a merchant, or a software developer from the merchant's side, you need to capture details of your customers on your website. We have created payment page and you can perform transactions using this 'Payment gateway Integration Guide' and send the encrypted transaction details to the payment gateway and get the response in decrypted and easily understandable format.

Prerequisites

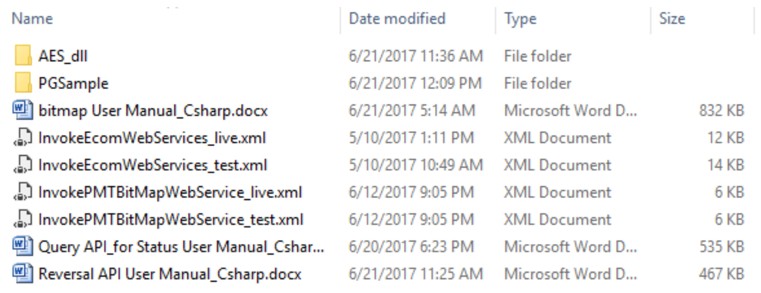
Requires .Net framework: 4.0 or above.

Visual Studio IDE

Aes.DLL version 1.0.0.0

Installation

Extract the zip folder and copy the folder to your local folder. Once the copying is done, if needed you can remove the documentation file`s like (pdf`s, doc, .txt files).

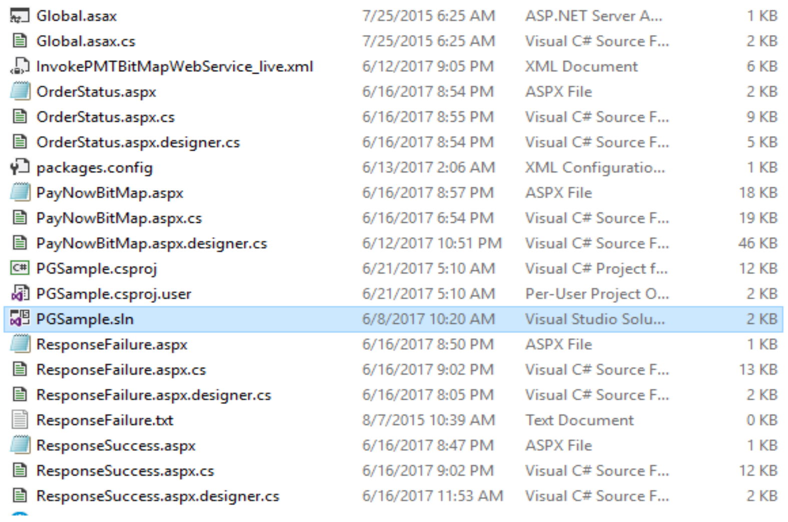


Name	Date modified	Type	Size
AES_dll	6/21/2017 11:36 AM	File folder	
PGSample	6/21/2017 12:09 PM	File folder	
bitmap User Manual_Csharp.docx	6/21/2017 5:14 AM	Microsoft Word D...	832 KB
InvokeEcomWebServices_live.xml	5/10/2017 1:11 PM	XML Document	12 KB
InvokeEcomWebServices_test.xml	5/10/2017 10:49 AM	XML Document	14 KB
InvokePMTBitMapWebService_live.xml	6/12/2017 9:05 PM	XML Document	6 KB
InvokePMTBitMapWebService_test.xml	6/12/2017 9:05 PM	XML Document	6 KB
Query API_for Status User Manual_Cshar...	6/20/2017 6:23 PM	Microsoft Word D...	535 KB
Reversal API User Manual_Csharp.docx	6/21/2017 11:25 AM	Microsoft Word D...	467 KB

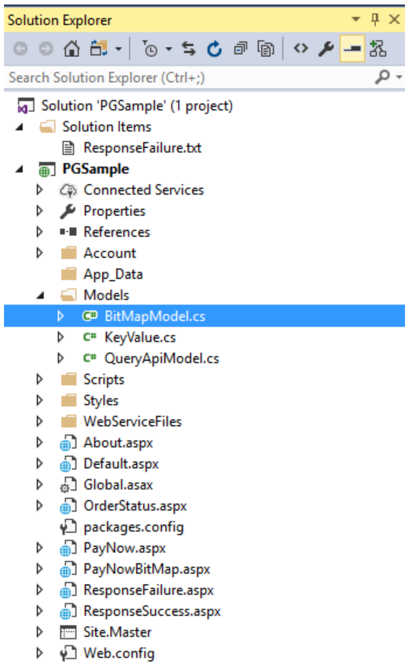
Settings & Executing the file

Once the installation is done please open the solution using PGSample.sln file in Visual studio IDE.

Please go to Models folder inside solution and open BitMapModel.cs file assign the following values like **merchantKey merchantId** with KEY & MID which you have.



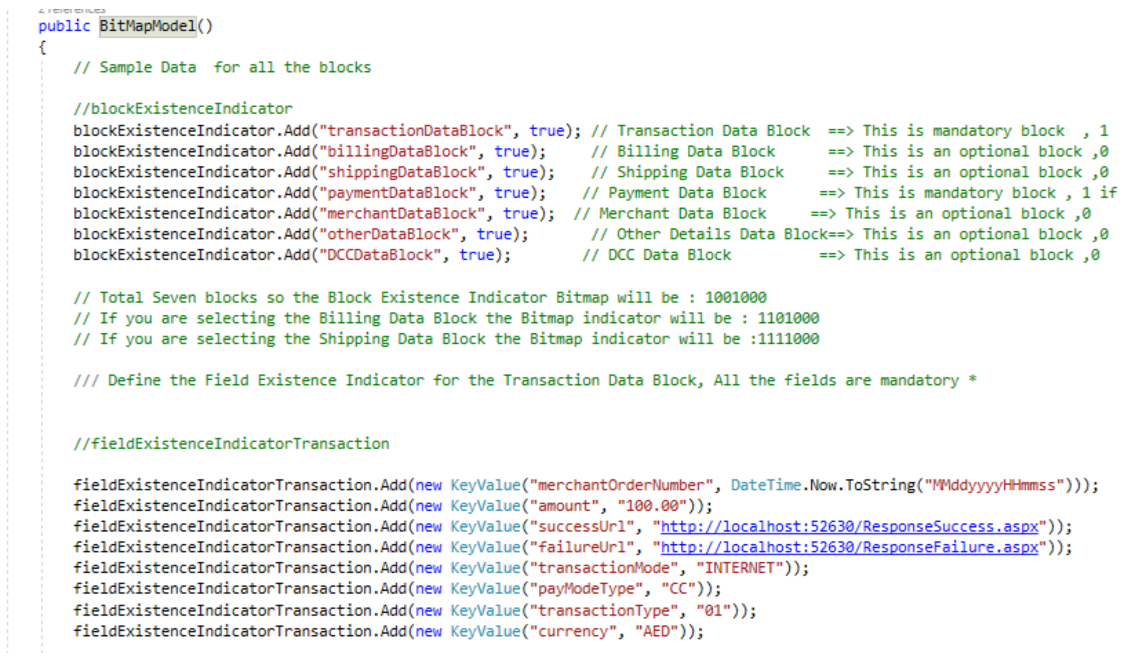
Global.asax	7/25/2015 6:25 AM	ASP.NET Server A...	1 KB
Global.asax.cs	7/25/2015 6:25 AM	Visual C# Source F...	2 KB
InvokePMTBitMapWebService_live.xml	6/12/2017 9:05 PM	XML Document	6 KB
OrderStatus.aspx	6/16/2017 8:54 PM	ASPX File	2 KB
OrderStatus.aspx.cs	6/16/2017 8:55 PM	Visual C# Source F...	9 KB
OrderStatus.aspx.designer.cs	6/16/2017 8:54 PM	Visual C# Source F...	5 KB
packages.config	6/13/2017 2:06 AM	XML Configuratio...	1 KB
PayNowBitMap.aspx	6/16/2017 8:57 PM	ASPX File	18 KB
PayNowBitMap.aspx.cs	6/16/2017 6:54 PM	Visual C# Source F...	19 KB
PayNowBitMap.aspx.designer.cs	6/12/2017 10:51 PM	Visual C# Source F...	46 KB
PGSample.csproj	6/21/2017 5:10 AM	Visual C# Project f...	12 KB
PGSample.csproj.user	6/21/2017 5:10 AM	Per-User Project O...	2 KB
PGSample.sln	6/8/2017 10:20 AM	Visual Studio Solu...	2 KB
ResponseFailure.aspx	6/16/2017 8:50 PM	ASPX File	1 KB
ResponseFailure.aspx.cs	6/16/2017 9:02 PM	Visual C# Source F...	13 KB
ResponseFailure.aspx.designer.cs	6/16/2017 8:05 PM	Visual C# Source F...	2 KB
ResponseFailure.txt	8/7/2015 10:39 AM	Text Document	0 KB
ResponseSuccess.aspx	6/16/2017 8:47 PM	ASPX File	1 KB
ResponseSuccess.aspx.cs	6/16/2017 9:02 PM	Visual C# Source F...	12 KB
ResponseSuccess.aspx.designer.cs	6/16/2017 11:53 AM	Visual C# Source F...	2 KB



```
Solution 'PGSample' (1 project)
  Solution Items
    ResponseFailure.txt
  PGSample
    Connected Services
    Properties
    References
    Account
    App_Data
    Models
      BitMapModel.cs
      KeyValue.cs
      QueryApiModel.cs
    Scripts
    Styles
    WebServiceFiles
    About.aspx
    Default.aspx
    Global.asax
    OrderStatus.aspx
    packages.config
    PayNow.aspx
    PayNowBitMap.aspx
    ResponseFailure.aspx
    ResponseSuccess.aspx
    Site.Master
    Web.config
```

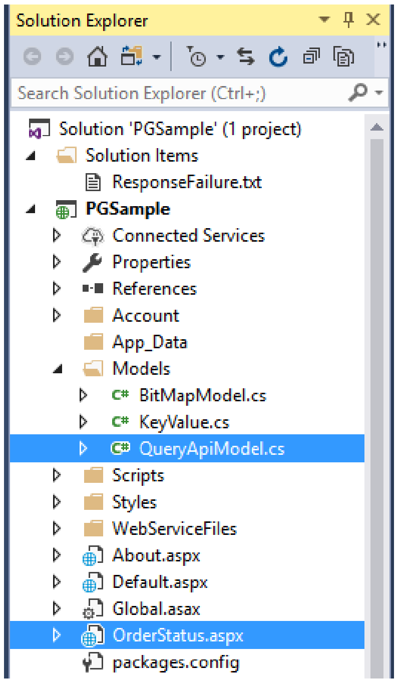
Data can be inserted in two ways.

1. You can set the default values using the below constructor in `BitMapModel` class
2. You can use the below screen to enter values.

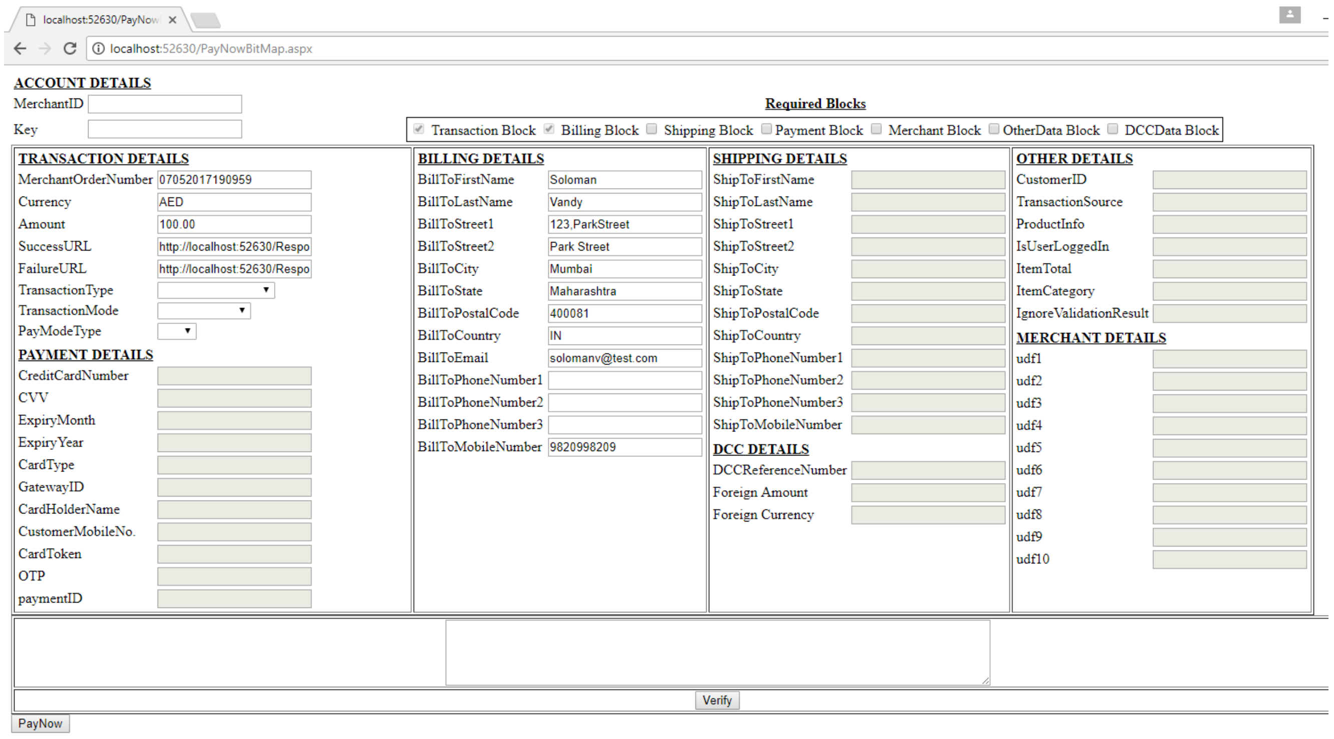


```
public BitMapModel()
{
    // Sample Data  for all the blocks

    //blockExistenceIndicator
    blockExistenceIndicator.Add("transactionDataBlock", true); // Transaction Data Block  ==> This is mandatory block  , 1
    blockExistenceIndicator.Add("billingDataBlock", true);     // Billing Data Block      ==> This is an optional block ,0
    blockExistenceIndicator.Add("shippingDataBlock", true);    // Shipping Data Block     ==> This is an optional block ,0
    blockExistenceIndicator.Add("paymentDataBlock", true);    // Payment Data Block      ==> This is mandatory block , 1 if
    blockExistenceIndicator.Add("merchantDataBlock", true);  // Merchant Data Block     ==> This is an optional block ,0
    blockExistenceIndicator.Add("otherDataBlock", true);       // Other Details Data Block==> This is an optional block ,0
    blockExistenceIndicator.Add("DCCDataBlock", true);        // DCC Data Block          ==> This is an optional block ,0

    // Total Seven blocks so the Block Existence Indicator Bitmap will be : 1001000
    // If you are selecting the Billing Data Block the Bitmap indicator will be : 1101000
    // If you are selecting the Shipping Data Block the Bitmap indicator will be :1111000

    /// Define the Field Existence Indicator for the Transaction Data Block, All the fields are mandatory *


    //fieldExistenceIndicatorTransaction

    fieldExistenceIndicatorTransaction.Add(new KeyValue("merchantOrderNumber", DateTime.Now.ToString("MMddyyyyHHmmss")));
    fieldExistenceIndicatorTransaction.Add(new KeyValue("amount", "100.00"));
    fieldExistenceIndicatorTransaction.Add(new KeyValue("successUrl", "http://localhost:52630/ResponseSuccess.aspx"));
    fieldExistenceIndicatorTransaction.Add(new KeyValue("failureUrl", "http://localhost:52630/ResponseFailure.aspx"));
    fieldExistenceIndicatorTransaction.Add(new KeyValue("transactionMode", "INTERNET"));
    fieldExistenceIndicatorTransaction.Add(new KeyValue("payModeType", "CC"));
    fieldExistenceIndicatorTransaction.Add(new KeyValue("transactionType", "01"));
    fieldExistenceIndicatorTransaction.Add(new KeyValue("currency", "AED"));
```

To get the below form, please set **PayNowBitMap.aspx** page as start page and run the solution



```
Solution 'PGSample' (1 project)
◢ Solution Items
      ResponseFailure.txt
◢ PGSample
   ▷ Connected Services
   ▷ Properties
   ▷ References
   ▷ Account
     App_Data
   ◢ Models
      ▷ BitMapModel.cs
      ▷ KeyValue.cs
      ▷ QueryApiModel.cs
   ▷ Scripts
   ▷ Styles
   ▷ WebServiceFiles
   ▷ About.aspx
   ▷ Default.aspx
   ▷ Global.asax
   ▷ OrderStatus.aspx
     packages.config
```



localhost:52630/PayNowBitMap.aspx

ACCOUNT DETAILS

MerchantID	
Key	

Required Blocks

[✓ Transaction Block] [✓ Billing Block] Shipping Block Payment Block Merchant Block OtherData Block DCCData Block

TRANSACTION DETAILS

MerchantOrderNumber	07052017190959
Currency	AED
Amount	100.00
SuccessURL	http://localhost:52630/Respo
FailureURL	http://localhost:52630/Respo
TransactionType	
TransactionMode	
PayModeType	

PAYMENT DETAILS

CreditCardNumber	
CVV	
ExpiryMonth	
ExpiryYear	
CardType	
GatewayID	
CardHolderName	
CustomerMobileNo.	
CardToken	
OTP	
paymentID	

BILLING DETAILS

BillToFirstName	Soloman
BillToLastName	Vandy
BillToStreet1	123,ParkStreet
BillToStreet2	Park Street
BillToCity	Mumbai
BillToState	Maharashtra
BillToPostalCode	400081
BillToCountry	IN
BillToEmail	solomanv@test.com
BillToPhoneNumber1	
BillToPhoneNumber2	
BillToPhoneNumber3	
BillToMobileNumber	9820998209

SHIPPING DETAILS

ShipToFirstName	
ShipToLastName	
ShipToStreet1	
ShipToStreet2	
ShipToCity	
ShipToState	
ShipToPostalCode	
ShipToCountry	
ShipToPhoneNumber1	
ShipToPhoneNumber2	
ShipToPhoneNumber3	
ShipToMobileNumber	

DCC DETAILS

DCCReferenceNumber	
Foreign Amount	
Foreign Currency	

OTHER DETAILS

CustomerID	
TransactionSource	
ProductInfo	
IsUserLoggedIn	
ItemTotal	
ItemCategory	
IgnoreValidationResult	

MERCHANT DETAILS

udf1	
udf2	
udf3	
udf4	
udf5	
udf6	
udf7	
udf8	
udf9	
udf10	

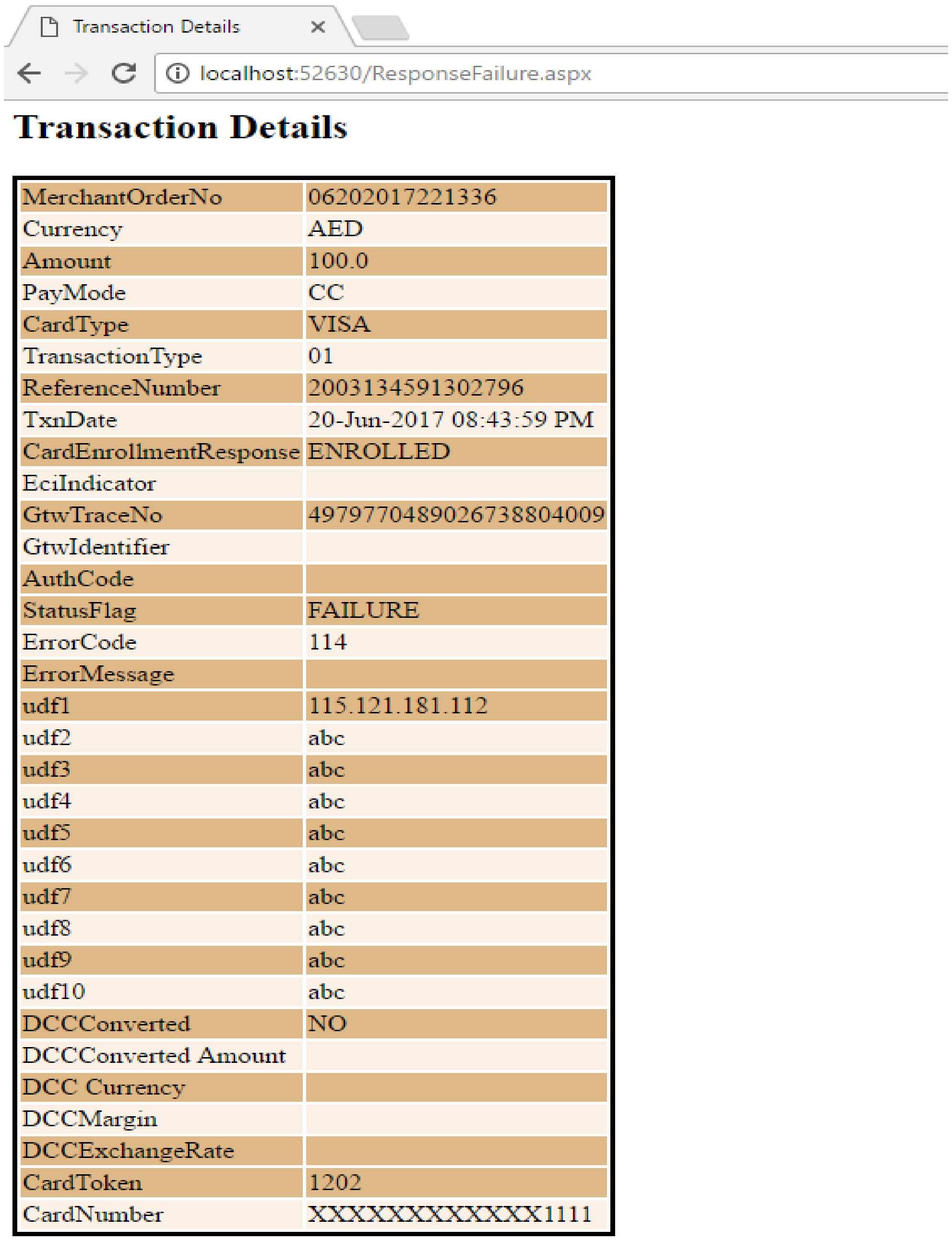
URL ex: http://localhost:port/PayNowBitMap.aspx

Posting URL :-

Test : https://uat-NeO.network.ae/direcpay/secure/PaymentTxnServlet

Live : https://NeO.network.ae/direcpay/secure/PaymentTxnServlet

You will be getting the following results: This UI is provided for the ease of testing and displaying the decoded response in a better understandable format.



Transaction Details

MerchantOrderNo	06202017221336
Currency	AED
Amount	100.0
PayMode	CC
CardType	VISA
TransactionType	01
ReferenceNumber	2003134591302796
TxnDate	20-Jun-2017 08:43:59 PM
CardEnrollmentResponse	ENROLLED
EciIndicator	
GtwTraceNo	4979770489026738804009
GtwIdentifier	
AuthCode	
StatusFlag	FAILURE
ErrorCode	114
ErrorMessage	
udf1	115.121.181.112
udf2	abc
udf3	abc
udf4	abc
udf5	abc
udf6	abc
udf7	abc
udf8	abc
udf9	abc
udf10	abc
DCCConverted	NO
DCCConverted Amount	
DCC Currency	
DCCMargin	
DCCExchangeRate	
CardToken	1202
CardNumber	XXXXXXXXXXXX1111

Merchant Integration using Bit Map Methodology

Two options are available for integration:

- PSP (Payment Service Provider) hosted Integration.
- Merchant hosted integration

Message will have the following structure:

<Merchant ID><Collaborator ID><Encrypted String>

The Encrypted String has the following broad-level structure:

<Block Existence Indicator>||<Data Block 1>||<Data Block 2>||<Data Block 3>||<Data Block 4>||<Data Block 5>||<Data Block 6>||<Data Block 7>

- <Data Block 1> // Transaction Data Block Mandatory One
- <Data Block 2> // Billing Data Block
- <Data Block 3> // Shipping Data Block
- <Data Block 4> // Payment Data Block, It is mandatory if not using PSP Integration
- <Data Block 5> // Merchant Data Block
- <Data Block 6> // Other Details Data Block
- <Data Block 7>// DCC Data Block

Each Data Block has the following structure:

<Field Existence Indicator for the Block>|<Field data 1 for the Block>|<Field data 2 for the Block>|<Field data 3 for the Block…|<Field data n for the Block>

Block Existence Indicator (BEI)

The Block Existence Indicator indicates to the Message Parser whether a particular block of data is present in the given Input message, or Response message. As defined in the tables, each message will have the fields grouped into Data Blocks. The existence or absence of the block in a particular message will be indicated by a 1 or a 0 in the position for the block, where: **1 = presence of Data Block & 0 = absence of Data Block**

Block Existence Indicator (BEI) implies the following

The Block Existence

- Transaction Data Block
- Billing Data Block
- Shipping Data Block
- Payment Data Block
- Merchant Data Block
- Other Details Data Block
- DCC Data Block

Merchant ID and Collaborator ID are mandatory for all messages in the request.

Transaction Data Block and Payment Data Block are mandatory blocks for 'Merchant-hosted' integration, where the payment details are captured at the merchants' page, and therefore any encrypted string must, at least, have the Block Existence Indicator (BEI) as '1001000'.

PSP hosted integration, the payment details are captured at the PSP's payment page, and therefore any encrypted string must mandatorily have, at least, a BEI value of '1000000'.

Field Existence Indicator (FEI)

The **Field Existence Indicator** indicates to the Message Parser whether a particular field of a **Data Block is present**, or absent in the given Input message, or **Response message**. Each message will have the fields grouped into Data Blocks. The existence, or absence of the field in the Data Block in a particular message will be indicated by a 1 or a 0 in the position of the field for the Data Block, where: 1 = presence of a field in the Data Block & 0 = absence of a field in the Data Block

Transaction Data Block implies the following

1. Merchant Order Number
2. Amount
3. Success URL
4. Failure URL
5. Transaction Mode
6. PayMode Type
7. Transaction Type
8. Currency

*<DataBlock1> = <Filed Existence Indicator><Data>

*<Filed Existence Indicator> 11111111: Total 8 fields

*<DataBlock1>=11111011|MerchantOrderNumber|Amount|SuccessURL|Failure URL|Transaction Mode|Transaction Type|Currency

Billing Data Block implies the following

1. BillToFirstName
2. BillToLastName
3. BillToStreet1
4. BillToStreet2
5. BillToCity
6. BillToState
7. BillToPostalCode
8. BillToCountry
9. BillToEmailID
10. BillToMobileNumber

11. BillToPhoneNumber1

12. BillToPhoneNumber2

13. BillToPhoneNumber3

*<DataBlock2> = <Filed Existence Indicator><Data>

*<Filed Existence Indicator> 1110111111111: Total 13 fields

*<DataBlock2> =

110111111111|BillToFirstName|BillToLastName|BillToStreet1|BillToCity|BillToState|BillToPostalCode|BillToCountry|BillToEmailID|BillToMobileNumber|BillToPhoneNumber1|BillToPhoneNumber2|BillToPhoneNumber3

Shipping Data block implies the following

1. ShipToFirstName
2. ShipToLastName
3. ShipToStreet1
4. ShipToStreet2
5. ShipToCity
6. ShipToState
7. ShipToPostalCode
8. ShipToCountry
9. ShipToPhoneNumber1
10. ShipToPhoneNumber2
11. ShipToPhoneNumber3
12. ShipToMobileNumber

*<DataBlock3> = <Filed Existence Indicator><Data>
*<Filed Existence Indicator> = 111011110001: Total 12 fields
*<DataBlock3> =
111011110001|ShipToFirstName|ShipToLastName|ShipToStreet1|ShipToCity|ShipToState|ShipToPostalCode|ShipToCountry|ShipToMobileNumber

Payment Data block implies the following

1. Card Number
2. Expiry Month
3. Expiry Year
4. CVV
5. Card Holder Name
6. Card Type
7. Customer Mobile Number
8. Payment ID
9. OTP
10. Gateway ID
11. Card Token

*<DataBlock4> = <Filed Existence Indicator><Data>

*<Filed Existence Indicator> = 00000000000 for PSP

*<Filed Existence Indicator> = 11111111111 for MHP

Case 1: for PSP hosting provider

*<DataBlock4> = 00000000000 since all the values are optional we don`t need to sent the data for NEO under this block

Case 2: for MHI hosting provider

*<DataBlock4> = 11111111111|Card Number|Expiry Month|Expiry Year|CVV|Card Holder Name|Card Type|Customer Mobile Number

Merchant Data Block implies the following

1. UDF1
2. UDF2
3. UDF3
4. UDF4
5. UDF5
6. UDF6
7. UDF7
8. UDF8
9. UDF9
10. UDF10

*<DataBlock5> = <Filed Existence Indicator><Data>

*<Filed Existence Indicator> = 1000000000: Total 10 fields

*<DataBlock5> = 1000000000|UDF1

Other Details Data Block implies the following

1. Cust ID
2. Transaction Source
3. Product Info
4. Is User Logged In
5. Item Total
6. Item Category
7. Ignore Validation Result

*<DataBlock6> = <Filed Existence Indicator><Data>

*<Filed Existence Indicator> = 1111111

*<DataBlock6> = 1111111|Cust ID|Transaction Source|Product Info|Y|ItemTotal|Item|Category|Ignore Validation Result

DCC Data Block implies the following

1. DCC Reference Number

2. Foreign Amount

3. Foreign Currency

*<DataBlock7> = <Filed Existence Indicator><Data>

*<Filed Existence Indicator> = 111

*<DataBlock7> = 111|DCC Reference Number|Foreign Amount|Foreign Currency

Transaction Request

<Merchant> <Collaborator ID> <Encrypted String>

<Merchant> = <YOUR_MERCHANT_ID> Your merchant ID ex: 201408191000001

<Collaborator ID> = NI

<Encrypted String> = encrypt Data (< Block Existence Indicator>||<Data Block 1>||<Data Block 2>||<Data Block 3>||<Data Block 4>||<Data Block 5>||<Data Block 6>||<Data Block 7>)

Transaction Response

The message structure would be similar to the Transaction Request Message, where a BEI and an FEI would be present.

Collaborator ID' is not a part of the Transaction Response message.

Merchant ID and will be sent in plain text along with the response, shown as follows

ResponseBlockBitMap = merchantId||packetBitMap||transactionData||txnResponse ||TxnResponseStatus||merchantData||fraudData||DCCData||additionalData

- packetBitMap = 1110010
- transactionData = FieldBitmap|MerchantOrderNo|Currency|Amount|PayMode|CardType|TransactionType
    - transactionData = 111111|1427091402107|INR|10.00|CC|Visa|01
    - transactionData = 111101|1427091402107|INR|10.00|NB|02
- TxnResponse = FieldBitmap|ReferenceNumber|TxnDate|CardEnrollmentResponse|EciIndicator|GtwTraceNo|GtwIdentifier|AuthCode
    - TxnResponse = 1111111|1001504007769463|14-May-2015 03:09:52 PM|Enrolled|Fully Secure|12345|CitiPG|83100
- TxnResponseStatus = FieldBitmap|StatusFlag|ErrorCode|ErrorMessage
    - TxnResponseStatus = 111|SUCCESS|00000|No Error
- merchantData = FieldBitmap|udf1|udf2|udf3|udf4|udf5|udf6|udf7|udf8|udf9|udf10
    - merchantData = 1111111111|115.121.181.112|abc|abc|abc|abc|abc|abc|abc|abc|abc
    - merchantData = 1000000000|115.121.181.112
- fraudData = FieldBitmap|frauddecision|fraudreason
    - fraudData = 11|REJECT|The order is rejected by Fraud Module
- additionalData = FieldBitmap|CardToken|CardNumber
    - additionalData = 11|1202|XXXXXXXXXXXX1234

Message Structure and its Interpretation

As defined in Table 1, a message will have the following structure:

<Merchant ID><Collaborator ID><Encrypted String>

The Encrypted String has the following broad-level structure:

<Block Existence Indicator>||<Data Block 1>||<Data Block 2>||<Data Block 3>

Each Data Block has the following structure:

<Field Existence Indicator for the Block>|<Field data 1 for the Block>|<Field data 2 for the Block>|<Field data 3 for the Block…|<Field data n for the Block>

Thus, the overall structure of the Encrypted String is as follows:

<Block Existence Indicator>||<Field Existence Indicator for Block 1>|<Field data 1 for Block 1>|<Field data 2 for Block 1..>||<Field Existence Indicator for Block 2>|<Field data 1 for Block 2>|<Field data 2 for Block 2>|<Field data 3 for Block 2..>||<Field Existence Indicator for Block n>|<Field data 1 for Block n>|<Field data 2 for Block n>

EXAMPLE:

Request string

1111111||11111111|06122017132336|100.00|http://localhost:52630/ResponseSuccess.aspx|http://localhost:52630/ResponseFailure.aspx|INTERNET|CC|SALE|AED||11111111110000|Soloman |Vandy|123,ParkStreet|Park Street|Mumbai|Maharashtra|400081|IN|solomanv@test.com|9820998209||111111110001|Soloman|Vandy|123ParkStreet|parkstreet|Mumbai|Maharashtra|400081|IN|9820998209||11111111111|4111111111111111|08|2020|123|Soloman|Visa|9820998209|123456|123456|1026|1202||1111111111|115.121.181.112|abc|abc|abc|abc|abc|abc|abc|abc|abc||1111111|9874513|Web|Book|Y|100.00|CD, Book|FALSE||111|09898787|240.00|USD

After encryption

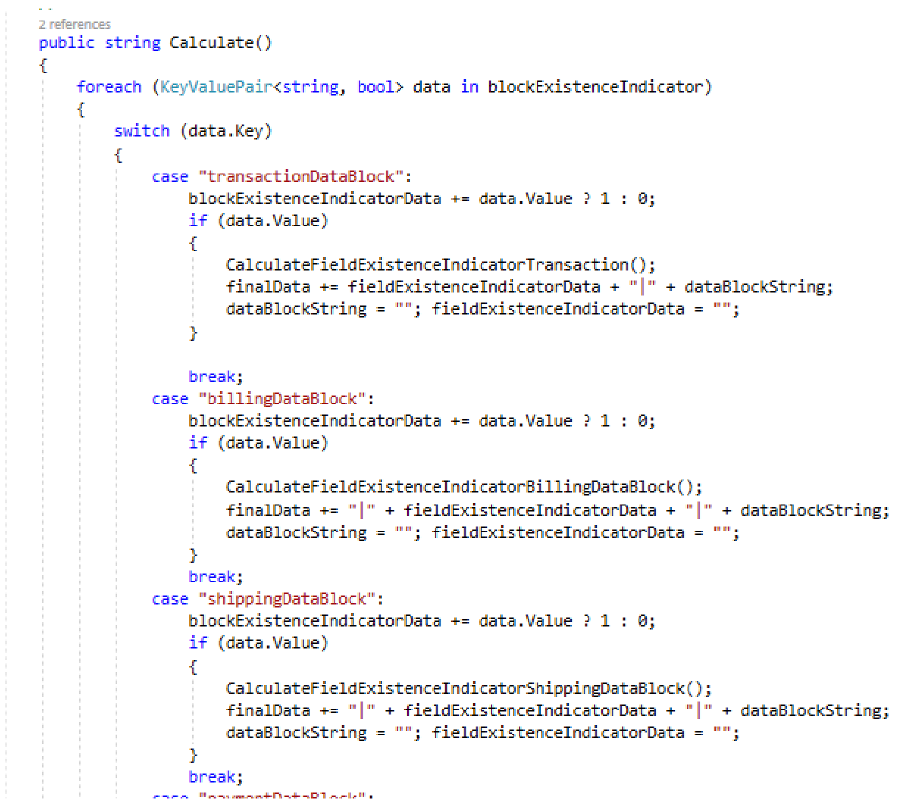
X01703301XXXXX||NI||1akX9jAiZhaTmdjXPCB80gL9cHagJp8Q5dElDBwJiCvIexL1qe0KWdAcjqusRt
z9KmvOsAa2AhSbNn32JQYqZP6f1NfPIqxduhdRQQZasE7iqgGF+EbaYjdOSN1xLxQvcMmKcmIM3Zba
d+39+KuLUAKuvNkVneRHNs+FB+kA2l6DZQaMAk1kr9WwzMuMaROmZETfCgQ8kg/FJXi1s/vZ6V8M
eieBsADtl0UdcqCKOYeV0qbyYdPx0hZ3VCXFbI2ajrTaOkWgtORyuT73xQ++hunBrD8A/pmKdCmFKIS
y7gyzKBPuhi732aca2VJUq8I3t1l//m7mg0k/xKScsAtmmIH2sbLZToarFoW8kmJHkLs+U/IVLt1TZPKM
otkd8PP/1lBwwdvMP1P9I1SAui6MZjtzOdLj5cTKV9SoXHGph7CqEzzFw1yzMIDwKsp48OjhIGjUrvBC
Q+yJF/GxMwds36dFbV+rVyc474DjycDr4oFSjxiXrsMd6Qi5z6XSm4vULrH7WzM6dzA5fU95driRPVyzf
WcRZgo8vvMsOHl5qLOLhltDOTNo3FPb0Xq3fgiQN73/Nh7RmjhNGr1hxW5Cd2bknCqV72GvAGrIf/O
gIpKbmuMCoWZZof/SHXcWDPkeYLNT1d9GYBTZ8syncEkDpR1YcFhaHwcYCExzzUam7gdE8w9xgqh
XppXAe9DVfWolSOH5QB+CDnUfZzpwqrWbnxvAuJOrrQNF7K1K5A7QLCD14n7k7Sr8nc1v766lm3/6
Y4a0/3mOejkPmiiwsDzVsVel7xrLqV2BNRBB4UmIV84=

Response string

X00702091XXXXXXX||1111111||110111|1427091402107|INR|CC|Visa|01||1111111|10015040
07769463|14 - May - 2015 03:09:52 PM|Enrolled|Fully
Secure|12345|CitiPG|83100||111|SUCCESS|00000|No
Error||1111111111|115.121.181.112|abc|abc|abc|abc|abc|abc|abc|abc|abc||11|REJECT|The
order is rejected by Fraud
Module||11111|Y|240.00|AED|0.75|0.7178203||11|1202|XXXXXXXXXXXX1234

Code Description

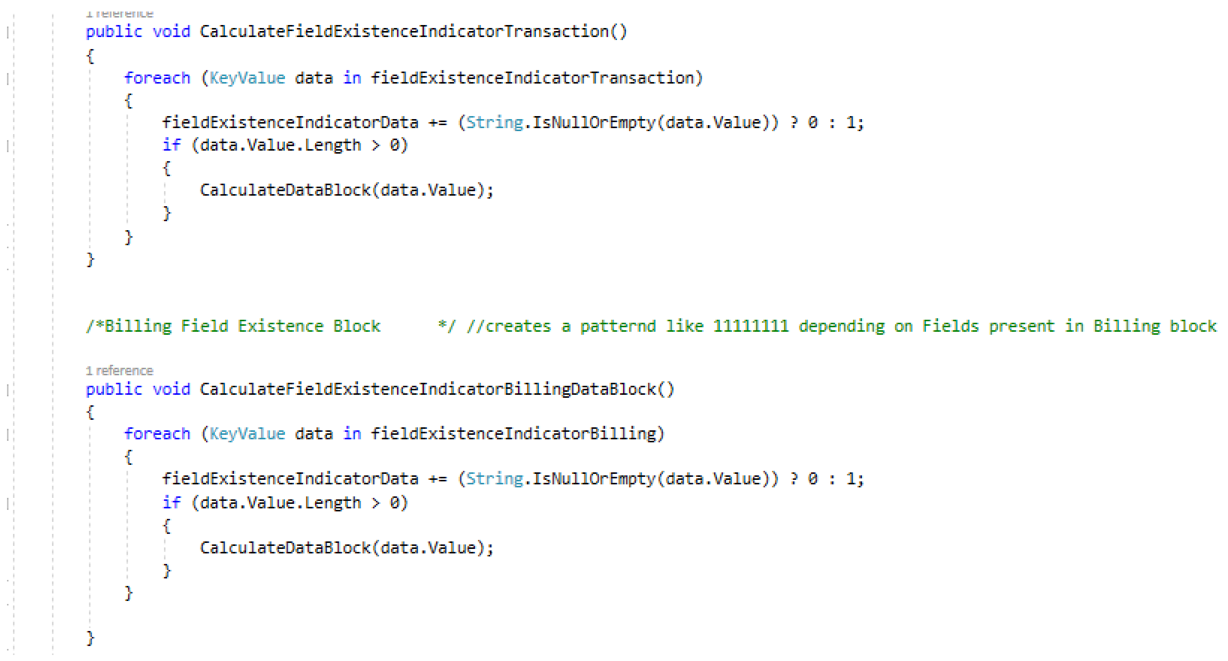
Below calculate() method creates the Block existence indicator (eg: 1101101) where the first block that is transaction block is mandatory.



```
2 references
public string Calculate()
{
    foreach (KeyValuePair<string, bool> data in blockExistenceIndicator)
    {
        switch (data.Key)
        {
            case "transactionDataBlock":
                blockExistenceIndicatorData += data.Value ? 1 : 0;
                if (data.Value)
                {
                    CalculateFieldExistenceIndicatorTransaction();
                    finalData += fieldExistenceIndicatorData + "|" + dataBlockString;
                    dataBlockString = ""; fieldExistenceIndicatorData = "";
                }

                break;
            case "billingDataBlock":
                blockExistenceIndicatorData += data.Value ? 1 : 0;
                if (data.Value)
                {
                    CalculateFieldExistenceIndicatorBillingDataBlock();
                    finalData += "|" + fieldExistenceIndicatorData + "|" + dataBlockString;
                    dataBlockString = ""; fieldExistenceIndicatorData = "";
                }
                break;
            case "shippingDataBlock":
                blockExistenceIndicatorData += data.Value ? 1 : 0;
                if (data.Value)
                {
                    CalculateFieldExistenceIndicatorShippingDataBlock();
                    finalData += "|" + fieldExistenceIndicatorData + "|" + dataBlockString;
                    dataBlockString = ""; fieldExistenceIndicatorData = "";
                }
                break;
```

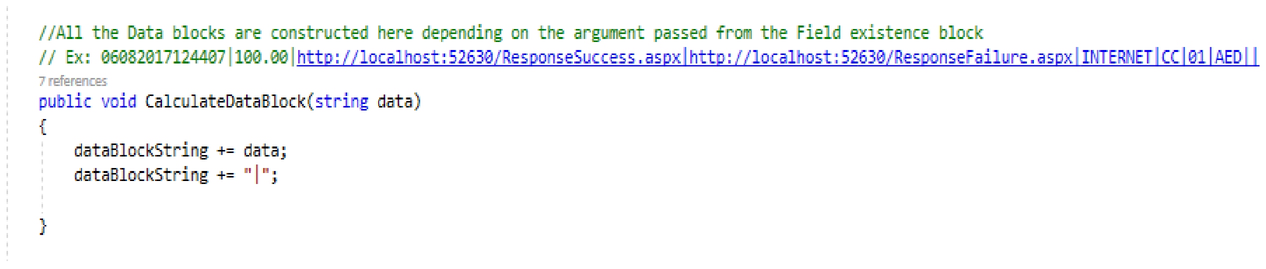
Below methods creates the Field Existence indicator (ex: 111111110001) according to which data block will be present

```
CalculateFieldExistenceIndicatorTransaction()
CalculateFieldExistenceIndicatorBillingDataBlock()
CalculateFieldExistenceIndicatorShippingDataBlock()
CalculateFieldExistenceIndicatorPaymentDataBlock()
CalculateFieldExistenceIndicatorMerchantDataBlock()
CalculateFieldExistenceIndicatorOtherDataBlock()
CalculateFieldExistenceIndicatorDCCDataBlock()
```



```
1 reference
public void CalculateFieldExistenceIndicatorTransaction()
{
    foreach (KeyValue data in fieldExistenceIndicatorTransaction)
    {
        fieldExistenceIndicatorData += (String.IsNullOrEmpty(data.Value)) ? 0 : 1;
        if (data.Value.Length > 0)
        {
            CalculateDataBlock(data.Value);
        }
    }
}


/*Billing Field Existence Block      */ //creates a patternd like 11111111 depending on Fields present in Billing block

1 reference
public void CalculateFieldExistenceIndicatorBillingDataBlock()
{
    foreach (KeyValue data in fieldExistenceIndicatorBilling)
    {
        fieldExistenceIndicatorData += (String.IsNullOrEmpty(data.Value)) ? 0 : 1;
        if (data.Value.Length > 0)
        {
            CalculateDataBlock(data.Value);
        }
    }

}
```

Below method creates data block for all the blocks with the string received

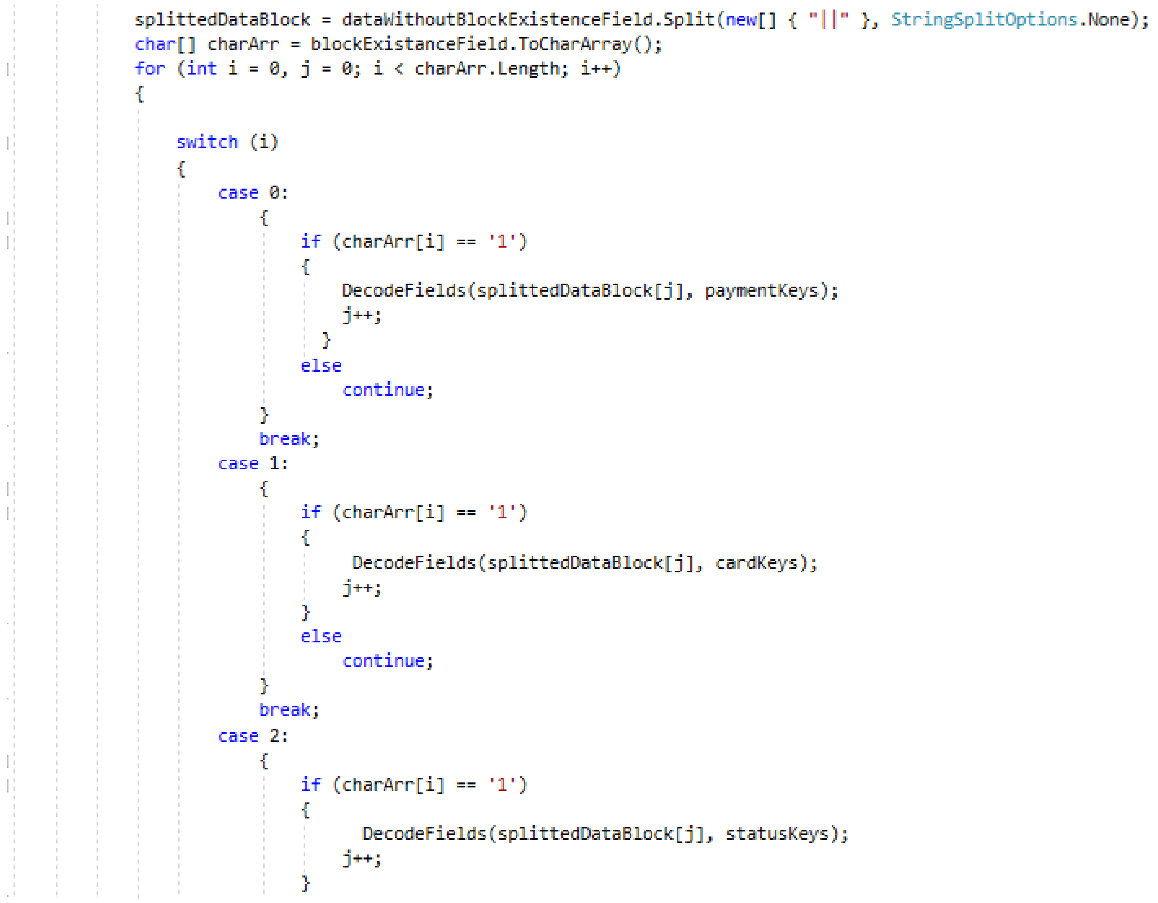


```
//All the Data blocks are constructed here depending on the argument passed from the Field existence block
// Ex: 06082017124407|100.00|http://localhost:52630/ResponseSuccess.aspx|http://localhost:52630/ResponseFailure.aspx|INTERNET|CC|01|AED||
7 references
public void CalculateDataBlock(string data)
{
    dataBlockString += data;
    dataBlockString += "|";

}
```

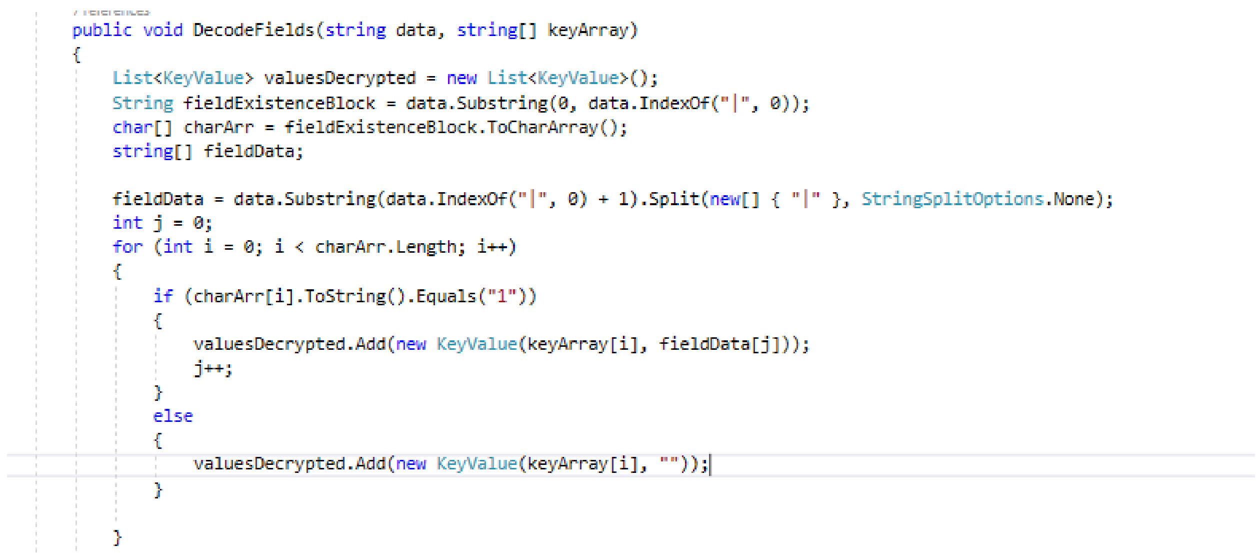
Response String Decoding

splittedDataBlock[] array contains the response data separated using (||) symbol. Below are the keys for different blocks which will be passed to DecodeFields() method to attach ito the keys by creating a key value pair and then printed to response page.

```
string[] paymentKeys = { "MerchantOrderNo", "Currency", "Amount", "PayMode", "CardType", "TransactionType" };
string[] cardKeys = { "ReferenceNumber", "TxnDate", "CardEnrollmentResponse", "EciIndicator", "GtwTraceNo",
"GtwIdentifier", "AuthCode" };
string[] statusKeys = { "StatusFlag", "ErrorCode", "ErrorMessage" };
string[] merchantKeys = { "udf1", "udf2", "udf3", "udf4", "udf5", "udf6", "udf7", "udf8", "udf9", "udf10" };
string[] fraudKeys = { "frauddecision", "fraudreason" };
string[] dccKeys = { "DCCConverted", "DCCConverted Amount", "DCC Currency", "DCCMargin", "DCCExchangeRate" };
string[] tokenKeys = { "CardToken", "CardNumber" };
```



```
            splittedDataBlock = dataWithoutBlockExistenceField.Split(new[] { "||" }, StringSplitOptions.None);
            char[] charArr = blockExistanceField.ToCharArray();
            for (int i = 0, j = 0; i < charArr.Length; i++)
            {

                switch (i)
                {
                    case 0:
                        {
                            if (charArr[i] == '1')
                            {
                                DecodeFields(splittedDataBlock[j], paymentKeys);
                                j++;
                              }
                            else
                                continue;
                        }
                        break;
                    case 1:
                        {
                            if (charArr[i] == '1')
                            {
                                 DecodeFields(splittedDataBlock[j], cardKeys);
                                j++;
                            }
                            else
                                continue;
                        }
                        break;
                    case 2:
                        {
                            if (charArr[i] == '1')
                            {
                                  DecodeFields(splittedDataBlock[j], statusKeys);
                                j++;
                            }
```



```
        public void DecodeFields(string data, string[] keyArray)
        {
            List<KeyValue> valuesDecrypted = new List<KeyValue>();
            String fieldExistenceBlock = data.Substring(0, data.IndexOf("|", 0));
            char[] charArr = fieldExistenceBlock.ToCharArray();
            string[] fieldData;

            fieldData = data.Substring(data.IndexOf("|", 0) + 1).Split(new[] { "|" }, StringSplitOptions.None);
            int j = 0;
            for (int i = 0; i < charArr.Length; i++)
            {
                if (charArr[i].ToString().Equals("1"))
                {
                    valuesDecrypted.Add(new KeyValue(keyArray[i], fieldData[j]));
                    j++;
                }
                else
                {
                    valuesDecrypted.Add(new KeyValue(keyArray[i], ""));
                }

            }
```