
First Atlantic Commerce Payment Gateway 2 Integration Guide for Developers

Version 2.6.9, 18th July, 2019

Introduction	6
Integration Process Overview	7
Important Integration Information.....	8
Testing Considerations.....	9
Integration (Staging) Platform URLs	12
SOAP Services.....	12
XML POST Service.....	12
Production Platform URLs	13
SOAP Services.....	13
XML POST Service.....	13
FAC Platform URLs: DNS versus IP Addresses.....	14
The Gateway Operations	15
The SOAP Service Operations Overview	17
Implementing the Gateway Operations	21
Important Implementation Details.....	22
CVV2/CVC2 Validation.....	22
Currency Validation.....	23
Address Verification (AVS) Checks	23
3D Secure vs. Non-3D Secure Transactions	25
3D Secure Authentication Only Transactions.....	25
3D Secure authorize/capture (“one-pass”) transactions	26
Fraud Control	27
Fraud Control - Client-side Implementation Requirements for Kount Service	27
The Fraud Check Response	30
BIN Blocking	32
What is a “BIN” and what is “BIN Blocking”?	32
Signature Creation and Verification.....	34
Response Code and Reason Code Responses.....	37
Authorization Original Response Codes	37
The Authorize Operation	38
AuthorizeRequest	39
AuthorizeResponse.....	44
Tokenization using the Authorize (and Authorize3DS) Operation	46
What is a Token?	46
Tokenizing a PAN.....	46
Avoiding Card Numbers.....	47
The Authorize3DS Operation	48
Implementing the 3DS Message.....	48
Authorize3DSRequest.....	49
Authorize3DSResponse	50
The TransactionModification Operation.....	51

Modification Types	51
Capture	51
Refund	51
Reversal	52
Cancel Recurring.....	52
TransactionModificationRequest	53
TransactionModificationResponse	54
Transaction Modification Response and Reason Codes	55
The TransactionStatus Operation	55
TransactionStatusRequest.....	55
TransactionStatusResponse.....	56
HostedPageAuthorize and HostedPageResults	57
The Tokenize Operation.....	58
TokenizeRequest	58
TokenizeResponse	59
The DeTokenize Operation	60
DeTokenizeRequest.....	60
DeTokenizeResponse	61
The UpdateToken Operation	62
UpdateTokenRequest.....	62
UpdateTokenResponse	63
The ExpiringCreditCards Operation	64
ExpiringCreditCardsRequest.....	64
ExpiringCreditCardsResponse	65
Recurring Payments	66
CVV2 and Recurring.....	67
3DS and Recurring	67
Recurring Notifications.....	68
FACPG2 API - Sample Code	71
Programming FACPG in Microsoft.NET VS2010	71
Obtaining a WCF Service Reference	71
Obtaining a SOAP Web Reference	74
Programming Against the Service or Web Reference	76
Programming FACPG in Java.....	77
HTTPS and Java.....	77
Obtaining a Web Reference in Java	77
The Authorize Operation in Java	81
Programming FAC Payment Gateway in PHP	89
SOAP Programming	89
XML Programming.....	89
Authorize Examples.....	91

TransactionModification Examples	96
TransactionStatus Examples.....	100
Programming using SOAP directly	103
Authorize3DS Implementation	104
Calling the Authorize3DS Method in .NET.....	104
The HTML Form Returned	105
The 3DS Response	107
Glossary of Terms.....	108
Appendix A - <i>Field Requirements and Allowable Values</i>	110
1) AVS Field Requirements.....	110
2) BillToState Allowable Values.....	111
3) ISO 4217 Currency Codes.....	112
4) ISO 3166-1 Numeric Country Codes	116
Appendix B – <i>Response and Reason Codes</i>	120
1) CVV2/CVC2 Response Codes	120
2) AVS Check Response Codes	120
3) Response Code and Reason Code Responses.....	122
5) Authorization Original Response Codes	126
6) Transaction Modification Response and Reason Codes.....	130
7) Fraud Control Response and Reason Codes	132
Main FraudControl ResponseCode	132
Third Party FraudResponseCode.....	132
FraudControl Decline or Error Codes	133
Appendix C	134
Test Card Information.....	134

Change Log

Document Version	Description	Release Date
V2.0	Initial version	1 st Aug 2011
V2.1	Tokenization Authorize functions added	22 nd Sep 2011
V2.2	Updated 3DS information	3 rd Oct 2011
V2.3	Tokenize Operation added	6 th Oct 2011
V2.4	Prepayment Operation added	1 st Nov 2011
V2.5	Split of FACPG2 into 5 separate services	3 rd Mar 2012
V2.5.1	Minor Clarification of URLs	10 th May 2012
V2.5.2	Added Clause about XML namespaces	13 th Aug 2012
V2.5.3	Clarification of ExecutionDate and addition of Recurring section and Appendix A	25 th Sep 2012
V2.5.4	Removed outdated sections	24 th Oct 2012
V2.5.5	Removed outdated sections	11 th Dec 2012
V2.5.6	Added Extended Tokenization Operations. Also added Fraud Checking fields to Messages plus notes.	19 th Mar 2013
V2.5.7	Updated formatting, clarified several sections	9 th Jul 2013
V2.5.8	Updated Fraud Control	18 th Jul 2013
V2.5.9	Added ISO 3166-1 Numeric Country Codes	22 nd Aug 2013
V2.6	Added Notifications Service (Recurring)	16 th Sept 2013
V2.6.1	Clarification of AuthorizeRequest and AuthorizeResponse message structure, added new fields BillToFax, BillToMobile, and BillToCounty	25 th Oct 2013
V2.6.2	Updated formatting	8 th Nov 2013
V2.6.3	Added American Express test cards	18 th Jun 2014
V2.6.4	Removed deprecated section: TokenList Operation	26 th Jan 2015
V2.6.5	Added CancelRecurring Transaction Modification Operation	17 th Mar 2015
V2.6.6	Added Host Specific Code for Initial Recurring Transactions where there is a free Trial. Clarification of Notes regarding Reversals.	29 th Apr 2015
V2.6.7	Updated test card responses (cards *3333 and *8888). Also added R0, R1, R3 Visa Recurring ResponseCodes	19 th Aug 2016
v.2.6.8	Added sentences regarding API traffic being encrypted from client to server	13 th Mar 2018
v.2.6.9	Updated Fraud Control for new Kount spec	18 th Jul 2019

Introduction

This document will guide a developer through the integration process required to use First Atlantic Commerce's (FAC) Payment Gateway (PG) Services. These services support and enable the FAC merchant products [cGate® Secure Real-Time](#) and [cGate® Secure Verify](#).

Version 2 of the FAC Payment Gateway introduces two ways of integration; a Payment Gateway Web Service (SOAP based) and an XML POST service. These two methods share the same data structures and attributes, but are called using different URLs and semantics.

Both methods use **HTTPS** as the transport protocol.

To integrate a merchant's system with FAC Payment Gateway, a developer must be able to provide client-side security for receiving cardholder information (credit card number, expiry date, etc.), as well as be able to connect to FAC using HTTPS so as to pass this information using SSL. The developer will also need to be able to write scripts or develop programs for the merchant's web server to provide the necessary integration.

This document does provide some sample code, but it is *not* meant as a cut-and-paste-and-run solution, as each merchant will have specific requirements.

Note that this document provides information on implementing all possible e-commerce transaction types available through FAC's Payment Gateway. These are:

- 3D Secure Authorization Only (authorize only), Authorization & Capture (authorize/capture), Authentication Only or Previously Authenticated
- Authorization Only (non-3D Secure) or Authorization & Capture (authorize/capture)
- Tokenized Authorization Transactions
- Independent Tokenization and De-Tokenization of Card Number
- Updating and Managing Tokenized Card Numbers
- Recurring Transactions
- AVS Verification Only
- Capture
- Reversal
- Refund
- Cancel a Recurring Transaction cycle
- View the results of a previous transaction.
- Fraud Scoring during Authorization
- Independent Fraud Scoring

Integration Process Overview

It is important to note that each integration differs in its requirements, complexity, resources allocated to the integration and the expertise of the technical person(s) working on the implementation. That being said, integrations can take anywhere from 2-3 weeks to 2-3 months. The technical integration process is usually initiated once the merchant's acquiring bank issues a Merchant ID number or a merchant is granted bank approval. Regardless of the payment processing requirements or payment gateway services needed, here is the standard set of steps for a FAC integration:

1. **Technical Integration Guide Review** – FAC will provide the merchant and/or their technical team FAC's integration guide for the technical team to review.
2. **Completion of FAC's Processing Questionnaire** – FAC's business team will have provided you with FAC's processing questionnaire to gain an initial understanding of the merchant's processing requirements.
3. **Integration Call** – Once the integration guide has been reviewed and FAC's processing questionnaire has been completed and returned for FAC, FAC will request an integration call with the technical and business teams. The purpose of the call will serve as an introduction to all the relevant parties involved in the integration project, to discuss the merchant's processing requirements, provide an overview of the integration process and provide a forum to go over any initial questions anyone may have. The call is not always technical in nature but it ensure that both business and technical teams of FAC and the merchant are all understanding the requirements and how to proceed with the integration.
4. **Provision of Test Account** – After the integration call, FAC will set up a test account based on the processing needs of the merchant as indicated on the processing questionnaire and discussed in the integration call. Here, the merchant will be able to begin testing their technical integration to FAC.
5. **Testing on FAC Test Platform** - Testing on the test environment will allow you to test your code thoroughly without performing live transactions. This platform mimics the live environment. It is highly recommended to also perform data validation on the data fields on the payment page prior to transaction submission to FAC. Testing should also include the handling of approvals, declines and failed transactions. Additionally, in this environment cards will not be charged as the transactions do not go out to interchange. It is recommended that you review the following section within this document on testing considerations to help guide you through your testing phase.
6. **Provision of Production Account** – Provided FAC has been given the bank issued Merchant ID (Merchant Number), and after a merchant has successfully tested to their satisfaction in the test environment, FAC will then provide the merchant with their live account credentials.
7. **Testing on FAC Production Platform** - Production testing is restricted to FAC core business hours which are Monday through Friday 8:30 AM ADT – 5:30 PM ADT. Again, it is highly advisable to verify that the data fields on the payment page are validated prior to transaction submission. Testing should also include the handling of approvals, declines and failed transactions.
8. **FAC End-to-End Testing** - FAC will conduct a site review and an end-to-end test from your site. It is usually asked that you set up a test product with a value of \$1.00 USD for FAC to use in their site review and end-to-end test. The end-to-end test validates the full transaction cycle from payment on a merchant's site all the way to validating that the funds of the test transactions are confirmed deposited by the bank in the merchant's account. This process can take up to 5 days depending on the processor and bank.
9. **FAC Business Team Go-Live Approval** - Provided all prior steps have been completed, your FAC Business Development representative will arrange an approved go-live date for the new site. It is important to note that FAC has a strict no go-live policy for Fridays, weekends and Bermuda public holidays.

Important Integration Information

During your integration to FAC, you will be provided a test processing ID and password to be used on our integration (staging) platform, which is a test platform, provided to allow thorough testing without performing live transaction processing. This platform mimics the live platform in every way, except that it is not processing your transactions live, but instead going to a host simulator on our system.

The exception to this rule is \$0 value AVS Verification Only Transactions. These are the only exception, as these will be routed through our live system.

Once your testing is done on the integration platform, we will provide you with a live processing ID and password and will ask that you make the appropriate changes in your code to point to the production platform. We will then ask you to perform some final tests to make sure you have successfully switched your code to point to our production system before officially going live on our system.

Testing Considerations

When testing your payment process it is important that you consider testing for the following:

✓ Data Validation

FAC performs basic data validation on parameter values submitted for credit card payments. The information that will be provided to FAC from your website's payment page should be validated or rendered into the proper format prior to payment submission, as to avoid rejected transactions. If the formatting is improper, it will be processed as is (the data is not modified); therefore, if invalid data is supplied or does not meet FAC's specifications, the transaction will fail or be rejected outright. It is highly recommended that you implement data validation to ensure the data passed is valid, scrubbed or rendered in the proper format. Parameters and their specifications are all outlined in this integration guide.

As an example: The 'CardNumber' specifications are that it be a 16 digit numeric value. FAC will not accept spaces, dashes, alpha characters or other symbols. If submitted, the transaction will result in failure. In the event a cardholder enters an invalid card number like 4111-1111-1111-1111, how do you want to handle this? Will the cardholder see a message on the screen asking them to check their card number and allow them to try again? Will information be provided on the screen to the cardholder to advise them of the acceptable format? Will the cardholder be restricted from entering anything but numbers?

Note: If you are utilizing Address Verification (AVS), it very important that you follow the [guidelines for data formats](#). Issuers only validate standard alpha and numeric values. Ensuring that you are passing the appropriate data formats will not only reduce or eliminate rejections, but you will get a more accurate AVS result.

✓ Transaction Approvals

Testing approvals seems quite simple; however, you may want to consider any of your processes that are initiated by an approval. As an example, are there any additional customer validations that need to take place? Does the cardholder receive an email confirmation or a receipt? Will an internal process be initiated internally with the business administrative staff or trigger a change in inventory?

✓ **Transaction Declines**

When testing Issuer/Processor declines you will want to be certain that your system is handling these according to the businesses requirements. Depending on the returned 'ReasonCode' you may want to display a particular message to the cardholder. You may want to restrict how many times a cardholder reattempt a transaction or notify them to contact a customer service agent to assist with their payment on the website. Your team may want to receive a notification when a cardholder reaches a threshold of attempts.

✓ **Transaction Errors**

Once a transaction has been processed it can have one of three statuses. It can be approved by the Issuer, declined by the Issuer/Processor, and lastly, it could have failed. In case of a transaction failure, there is a problem with processing the transaction. It could be for a number of reasons, but in all cases, FAC will respond with a 'ResponseCode' of 3. In these cases, you will want to check that your payment module is handling these as you require. As an example, in a live real-time environment, you may choose to display a message to the cardholder to try again later and initiate a notification to your team for investigation.

✓ **Website Browser Rendering**

You can design your website and payment page however you wish. It can be very simple and basic, or you can add a lot of complexity to it. This can have an effect on how different browsers display your web pages and the functionality of buttons, fields or drop-downs (if used). Assuming the merchant's customers will be using browsers other than Internet Explorer, it may be wise to test your website and payment page from different internet browsers to validate how they are rendered.

✓ **Captures, Refunds and Reversals**

If you are using the 'TransactionModification' method for processing capturing (in a two-pass processing method), reversing or refunding transactions you will want to test that this is functional, especially

within the production environment. You will want to test cases where the requests are not only approved, but denied, as to ensure your system is handling them as deemed necessary by the business.

As an example, should a refund be denied because the refund cutoff period on your merchant account has expired, how will your system handle this? Will notification be sent out to the technical team? Will the person that processed the refund see on their screen a message stating that the refund did not go through?

As a second example, if you have implemented or are using address verification (AVS), depending on the AVS result, you may want to reverse a transaction to cancel it, or capture a transaction if you wish to proceed with the payment.

✓ **Overall Cardholder Payment Experience**

It is up to the merchant and the business to determine how they want to represent themselves, products, and services to their customers and how they want the overall customer experience to be when making a credit card payment on their site. Although this is outside the scope of FAC, we recommend that ample quality assurance be done on your site to satisfy the needs of the business and that it supports their required functions of the site/payment process.

Integration (Staging) Platform URLs

There are five SOAP services and one XML service that cover all the functions of the Gateway.

SOAP Services

The URL for the FACPG2 web service on our integration (staging) platform is:

<https://ecm.firstatlanticcommerce.com/PGService/<Service Name>>

Example: For our base services, the URL would be:

<https://ecm.firstatlanticcommerce.com/PGService/Services.svc>

IMPORTANT: Note that <Service Name> is the correct service for the function required. See **The Gateway Operations** for information on where each Service operation is located.

For SOAP based integrations, the Web Services Definition Language file location on our integration platform is:

<https://ecm.firstatlanticcommerce.com/PGService/<Service Name>?WSDL>

where <Service Name> is one of the five available Web Services.

Example: WDSL URI for our base services is:

<https://ecm.firstatlanticcommerce.com/PGService/Services.svc?WSDL>

XML POST Service

Base URL for the FACPG2 web service on our integration (staging) platform is:

<https://ecm.firstatlanticcommerce.com/PGServiceXML>

Example: XML POST Service Operation URL (Authorize):

<https://ecm.firstatlanticcommerce.com/PGServiceXML/Authorize>

Production Platform URLs

SOAP Services

The URL for the FACPG web service on our production platform is:

<https://marlin.firstatlanticcommerce.com/PGService/<Service Name>>

Example: For our base services, the URL would be:

<https://marlin.firstatlanticcommerce.com/PGService/Services.svc>

IMPORTANT: Note that <Service Name> is the correct service for the function required. See **The Gateway Operations** for information on where each operation or function is located.

For SOAP based integrations, the Web Services Definition Language file location on our production system is:

<https://marlin.firstatlanticcommerce.com/PGService/<Service Name>?WSDL>

where <Service Name> is one of the five available Web Services.

Example: WDSL URI for our base services is:

<https://marlin.firstatlanticcommerce.com/PGService/Services.svc?WSDL>

IMPORTANT: Note that <Service Name> is the correct service for the function required. See **The Gateway Operations** for information on where each operation or function is located.

XML POST Service

Base URL is:

<https://marlin.firstatlanticcommerce.com/PGServiceXML>

Example: XML POST Service Operation URL (Authorize)

<https://marlin.firstatlanticcommerce.com/PGServiceXML/Authorize>

***All URLs provided in this document will reference the integration (staging) platform. Once it is time to move to our production system, we will remind you of the required URL change.*

FAC Platform URLs: DNS versus IP Addresses

IMPORTANT: When you move to our production system, *marlin.firstatlanticcommerce.com*, this domain name resolves to multiple IP addresses with regards to DNS for redundancy. It is highly recommended to use only the DNS and refrain from filtering by IP address on any sort of firewall.

The Gateway Operations

FAC Payment Gateway Services accept both SOAP and XML POST requests supporting secure transaction requests using the HTTPS protocol. The available service operations are:

Service Operation (SOAP)	Description	XML POST URL	Request Data Type	Response Data Type
Services.svc – Base Payment Gateway Operations				
Authorize	Supports Non-3D Secure authorizations, authorizations with a capture in a single transaction, AVS verification, and pre-authenticated 3D Secure transactions.	<Base>/Authorize	Authorize Request	Authorize Response
Transaction Modification	Captures a previously authorized transaction or reverses a previous authorized and captured transaction or refunds a previously authorized, captured and settled transaction	<Base>/TransactionModification	Transaction Modification Request	Transaction Modification Request
Authorize3DS	Supports 3DS authorizations only, 3DS authorization with a capture, 3D Secure authentication only transactions, and AVS.	<Base>/Authorize3DS	Authorize3DS Request	Authorize3DS Response
Transaction Status	Retrieves the status of a previous transaction.	<Base>/TransactionStatus	Transaction Status Request	Transaction Status Response
Tokenization.svc – Operations for Tokenization/De-Tokenization and Management of PANs				
Tokenize	Tokenizes a Card Number	<Base>/Tokenize	Tokenize Request	Tokenize Response
ExpiringCreditCards	Returns a list of Tokenized Cards whose Expiry Date is soon to be reached.	<Base>/ExpiringCreditCards	ExpiringCreditCards Request	ExpiringCredit Cards Response
DeTokenize	Allows a previously Tokenized Card to be De-Tokenized back to the Card Number	<Base>/DeTokenize	DeTokenize Request	DeTokenize Response
UpdateToken	Allows Expiry Date and Customer Reference to be updated.	<Base>/UpdateToken	UpdateToken Request	UpdateToken Response

TokenList	Get a list of Tokens and optionally filter by Customer Reference	<Base>/TokenList	TokenList Request	TokenList Response
HostedPage.svc – Operations for Authorization of a Hosted Page				
HostedPageAuthorize	Passes confidential data on a Hosted Page transaction to the Gateway and returns a Single-Use Token to allow access the hosted page.	<Base>/HostedPageAuthorization	HostedPageAuthorization Request	HostedPage Authorization Response
HostedPageResults	Allows recovery of full response data independently of the Hosted Page transaction. Requires the Single-use Key as the only parameter.	<Base>/HostedPageResults	Key (String)	HostedPage Results Response
Utilities.svc – Operations for Misc. Tasks				

The SOAP Service Operations Overview

If you generate a Service proxy class using your choice of implementation tools, the Service operations created will have the following calling conventions (VB.NET syntax used for clarity):

Services.svc:

Function Authorize(ByVal Request As AuthorizeRequest) As AuthorizeRequest
 Function TransactionModification(ByVal Request As TransactionModificationRequest) As TransactionModificationResponse
 Function Authorize3DS(ByVal Request As Authorize3DSRequest) As Authorize3DSResponse
 Function TransactionStatus(ByVal Request As TransactionStatusRequest) As TransactionStatusResponse

Tokenization.svc:

Function Tokenize(ByVal Request As TokenizeRequest) As TokenizeResponse
 Function DeTokenize(ByVal Request As DeTokenizeRequest) As DeTokenizeResponse
 Function UpdateToken(ByVal Request As UpdateTokenRequest) As UpdateTokenResponse
 Function TokenList(ByVal Request As TokenListRequest) As TokenListResponse
 Function ExpiringCreditCards(ByVal Request As ExpiringCreditCardsRequest) As ExpiringCreditCardsResponse

HostedPage.svc:

Function HostedPageAuthorize (ByVal Request As HostedPageAuthorizationRequest) As HostedPageAuthorizationResponse
 Function HostedPageResults (ByVal Key As String) As HostedPageResultsResponse

Utilities.svc:

Function BuildForm (ByVal Request As FormRequest) As FormResponse

The calling convention uses message based (sometimes called “document based”) syntax where there is always one input parameter and one return parameter. The parameters are complex data types so can have a hierarchy of data objects embedded.

So, a trivial example of calling one of these soap operations through a generated proxy class in VB.NET could be:

```
Sub DoAuthExample()
    Dim proxy As FACPGService = New FACPGService() ' Visual Studio Generated Service Proxy
    Dim response As AuthorizeResponse
    Dim request As AuthorizeRequest = New AuthorizeRequest()

    "Initialization logic for request goes here.....

    response = proxy.Authorize(request)

    "Process response logic goes here.....

End Sub
```

Note: Depending on the functionality of your SOAP toolkit, it may be possible to call the operation asynchronously. This is a choice you will need to make as the FAC gateway supports both Synchronous (as above) and Asynchronous calling conventions except for Authorize3DS. An Asynchronous calling convention usually involves calling a BeginXXX operation (where XXX is the Operation name) and wiring up a callback for notification of completion. The .NET framework proxy (Service or Web Reference) generation supports both Synchronous and Asynchronous calling conventions.

The WSDL from the Service will give any toolkit enough information to generate a proxy class for you.

We do not intend to go into the detail of the SOAP protocol in this document as this is too complex and this kind of detail is not usually required on any modern platform that provides a SOAP toolkit or interface.

If you need to use raw SOAP (where a toolkit is not available for your server platform), we would suggest calling our service via a proxy class initially (generated using any SOAP toolkit) and capturing the messages sent to our servers using a Network Debugger (e.g. WireShark). You can then use these as a template for “hand crafting” your SOAP messages for your server platform.

We will show examples of some well-known SOAP toolkit methods later in the document.

The XML POST Operations Overview

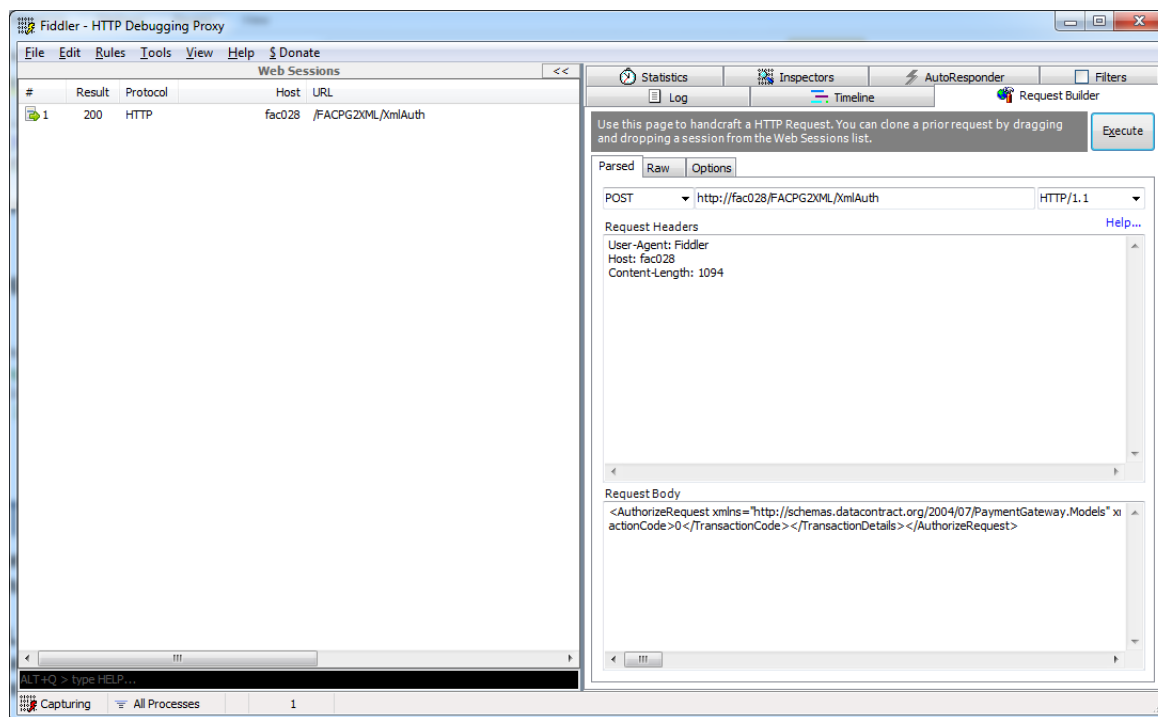
The XML POST Operations route the data sent in the body of an HTTP POST message to one of the SOAP operations and returns the Response class of that operation via the HTTP POST response.

IMPORTANT: You must include the namespace

<http://schemas.firstatlanticcommerce.com/gateway/data> in the root element of the XML Request, otherwise the request will be rejected.

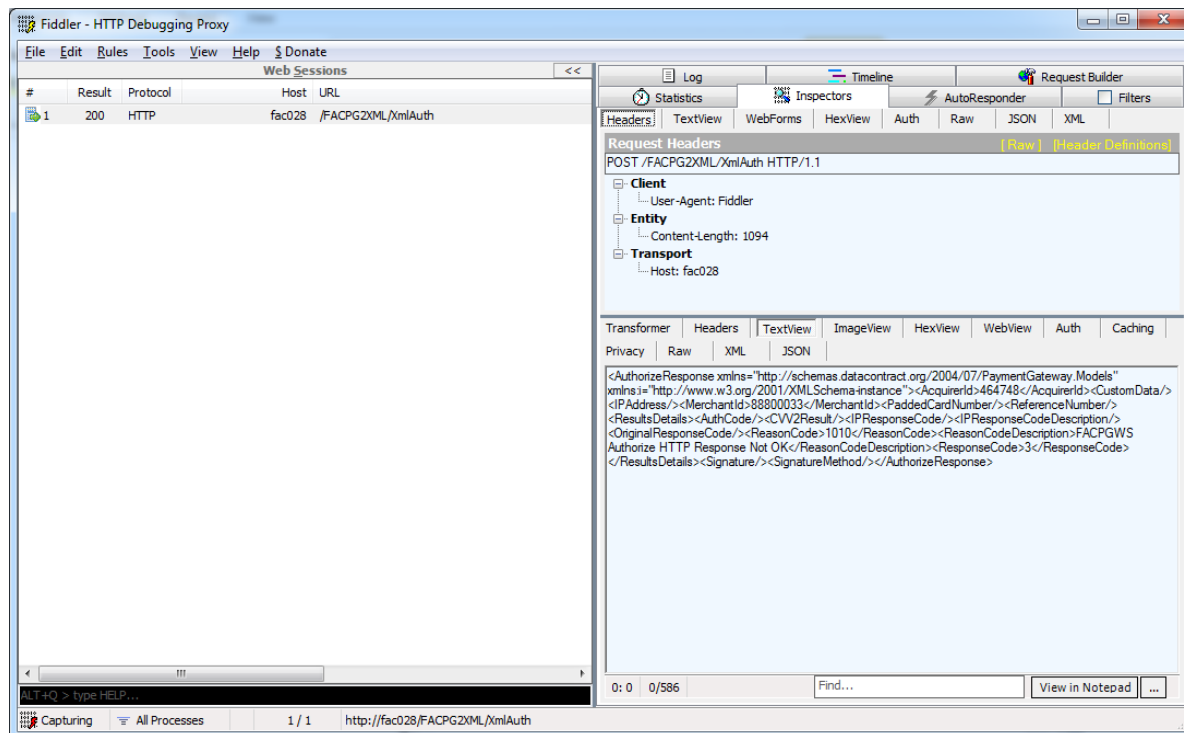
The data used is an XML representation of the XXXRequest and XXXResponse types (where XXX is the Operation Name). The normal way to construct these types in most languages/toolsets is to create classes for the types and use a serialization tool to create the XML from initialized class instances. Alternatively, if this functionality is not available, and as they are just text, any string creation/concatenation method could be used.

Showing an example of this is not so clear when shown programmatically but if you know how to use the Fiddler web debugger (free to download), you could execute a test Authorize as follows (note: this uses a FAC “internal use only” test URL):



The Request Builder of Fiddler allows the creation and sending of an HTTP POST manually. This is a good way of testing initial connectivity and your manually created XML data. Above is an Authorize that has

been executed successfully (see HTTP Result = 200 in left hand pane). This has been routed to the “Authorize” SOAP Service Operation and executed. The returned data can be viewed by double clicking on the Web Session in the left pane.



The returned data mirrors exactly the SOAP AuthorizeResponse message but in XML format (see [AuthorizeResponse](#) section).

In the [code samples](#) sections, we will show how to do this programmatically using some common tools.

Implementing the Gateway Operations

The data for all operations are contained in “Data Contract” Classes and follow a message “Request/Response” semantic. These classes come in pairs; one for the Request and the Response for each Operation.

The following section describes the Request and Response classes for each operation. As stated before, the formats are the same whether you are using the SOAP Service Operation calls or the XML POST URLs. The XML maps exactly onto the Request/Response classes and in fact, are just the classes serialized into XML.

In the section for each Operation will be shown the structure of the Messages in both UML and XML, then, each field of each class will be described in detail.

The UML diagram shows the Classes that would be created by any SOAP toolkit to represent the request data. The XML equivalent is what is used in its place when sending a request via the XML POST mechanism.

IMPORTANT: You must include the namespace <http://schemas.firstatlanticcommerce.com/gateway/data> in the root element of the XML Request otherwise the request will be rejected.

The Detailed Field Descriptions use the following abbreviations:

For Field Types there are three formats:

A = Alphabetical (a-z, A-Z)

N = Numeric (0-9)

AN = Alpha-Numeric (a-z, A-Z, 0-9)

Each field can be:

R = Required: Must always have a value.

O = Optional: Can be left null or not included (in XML)

C = Conditional: Mandatory under certain circumstances.

It is not necessary to assign default or empty values to fields that you do not need to use (Optional or Conditional) as the gateway can tolerate null values for most optional fields.

Important Implementation Details

Before starting to implement an Authorize or Authorize3DS it is important to know the format and possible values of some of the data fields in more detail.

All developers should be aware that FAC performs only basic data validation on parameter values provided. As such, the data provided to FAC in each transaction request is passed on as is to be processed. This means, for example, that if you provide FAC with a Visa credit card number that is only 15 digits in length, it will be passed through and declined because a Visa credit card is 16 digits in length. As such, **we strongly recommend that you perform data validation before passing transactions to FAC**, some examples of which are provided in the table below:

Parameter	Data Validation
CardNumber	Visa & MasterCard – 16 digits; AMEX – 15 digits; Numbers only - no spaces or dashes or slashes (except where Tokenized Card Number is used)
CardExpiryDate	4 digits (MMYY format) Numbers only - no spaces, dashes or slashes. Must be future dated or for the current month/year; expiry date cannot be in the past (expired).
CardCVV2	Visa & MasterCard – 3 digits; AMEX – 4 digits; Numbers only - no spaces, dashes or slashes
MerchantResponseURL	Must be HTTPS format (https://....)
BillToZipPostCode	Do not include hyphens, slashes or spaces (alphanumeric characters only)

CVV2/CVC2 Validation

The Card Verification Value (CVV2) (Card Verification Code (CVC2) for MasterCard), is the 3-digit code found on the back of all Visa and MasterCard credit cards (4-digit for AMEX). It is listed as a required parameter for the Authorize and Authorize3DS web methods above as it is mandated for all initial transactions processed by a merchant. However, for recurring transactions (i.e. billing for products sold to a repeat customer), this parameter is not required on any subsequent transactions, but is still highly recommended to provide it whenever possible.

Note: While CVV2/CVC2 data is strongly recommended to be sent with each and every transaction, some regions and processors mandate this be sent with every authorization request. If you fail to provide this data, the transactions may be automatically declined by the processor. These details will be discussed during the initial integration call with FAC if they apply.

CVV2/CVC2 Response Codes

After checking a CVV2/CVC2, values are returned in the CVV2Result field as follows:

Code	Definition
M	Match
N	No match.
P	Not Processed
S	Should be on card but was not provided. (Visa only)
U	Issuer not participating or certified.

Currency Validation

For merchants who have a First Atlantic Commerce merchant account configured to support multiple currencies, it is important to provide the correct base currency for each transaction. For example, to process a transaction in U.S. Dollars, the Currency parameter must be set to 840; to process a transaction in Euros, the Currency parameter must be set to 978. Note that FAC's system will process the transaction based on the value provided in the Currency parameter. The currency codes our system uses are the ISO 4217 numeric currency codes. See [Appendix A](#) for a full list of ISO 4217 codes.

Address Verification (AVS) Checks

Address checks are supported where the AuthorizeRequest or Authorize3DSRequest message TransactionCode field includes the AVS Check value.

The result of an AVS Check is stored in the AVSResult field of the Response message ([AuthorizeResponse](#), [TransactionModificationResponse](#), and [TransactionStatusResponse](#)).

If the TransactionCode is set to include an AVS check where the parameters BillToAddress and BillToZipPostCode are not supplied, the AVS check will fail.

If the TransactionCode is set to include an AVS check where the parameters BillToAddress and BillToZipPostCode are not supplied, but any of the parameters, BillToFirstName, BillToLastName, BillToCity, BillToState are supplied, the transaction request will fail and the response will include a ResponseCode of 2, a ReasonCode of 2, and an Original ResponseCode of 91.

When the AVS value is not included in the TransactionCode, the BillToAddress and BillToZipPostCode parameters are ignored.

When including an AVS check in a transaction, be sure not to include hyphens (or any other symbols other than numbers and letters) in the BillToZipPostCode parameter. Doing so will result in a declined transaction.

See [Appendix A](#) for a full list of AVS Field Requirements.

The parameter BillToState is only valid for U.S. based addresses. The allowable values for this parameter can be found in [Appendix A](#).

AVS Check Response Codes

[AVS Codes](#) are returned in the AVSResult field in the Response message of the Operation concerned; one of [AuthorizeResponse](#), [TransactionModificationResponse](#), or [TransactionStatusResponse](#). There are different codes depending on the card type. See [Appendix B](#) for a full list of AVS response codes for Visa, MasterCard, and Amex.

Note: The ability to process AVS checks is processor-specific. Please inform FAC if you wish to use this service and we will advise on its availability through your selected processor.

\$0 AVS Verification Only Transactions

The purpose of a \$0 AVS Verification Only transaction is to retrieve an AVS Result based on the address information provided in the transaction (i.e.: BillToAddress, BillToZipPostCode, etc.). Note that these transactions are not in any way authorized, much like the [3D Secure Authentication Only Transactions](#).

To fully understand what this means a quick rundown of how a normal authorization attempt works is in order. When you send a normal transaction (read: not a \$0 AVS Verification Only transaction) to be authorized, the issuing bank of the credit card is contacted and if the amount requested is available, it is blocked off and a “transaction approved” response is returned.

The authorized amount that is blocked off is freed up within 48-72 hours (depending on the issuing bank) if it is not captured (though some banks will hold this amount for up to 10 days).

When processing a \$0 AVS Verification Only transaction, the system knows that this is only an AVS check and thus does not follow the normal process of attempting to block off the authorization amount on the credit card (as there is no amount to block off). The only action taken here is to verify the address information provided, and return the appropriate AVS Result response.

If you send an AVS Verification Only transaction the amount is automatically set to \$0 regardless of the amount provided in the Amount parameter. The majority of responses for AVS Verification Only transactions return a Response Code of 2 and a Reason Code of 2 or 3, which means transaction declined, as an authorization attempt was never made. This can be ignored as the purpose of this transaction is to get the AVS result, which is returned as expected. Depending on your acquirer some \$0 AVS transaction may return an approved status. This will be discussed in your integration call if you are using this service.

The following rules and conditions apply to AVS Transactions:

- Regardless of the currency a transaction is processed in, the AVS Verification Only transaction must be sent as a U.S. Dollar transaction. This means that the Currency parameter must be set to 840 for every transaction you process through the AVS Only service.
- To process AVS Verification Only transactions you require a different FAC ID than your normal FAC ID. If you wish to use this service, please contact FAC support at support@fac.bm to get this set up.
- For any type of AVS Verification transaction, the “Is AVS” flag value must be included in the TransactionCode parameter value.

- As some issuing banks do not allow \$0 authorization amounts, expect to receive a 1%-2% error rate on \$0 AVS Verification Only Transactions in which the AVS Result is either a 5 or a 9.

3D Secure vs. Non-3D Secure Transactions

Though 3D Secure and Non-3D Secure transactions share many of the same parameters, that's about as far as the similarities go. The process flows of the two transaction types are quite different.

A Non-3D Secure transaction is fairly straightforward. It is a normal synchronous request-response process flow between FAC and you, the merchant. You send us the required parameters via the FACPG2 service and our payment gateway returns an approval (or a decline if the transaction is refused) back to you to do with as you please (i.e. redirect the cardholder to a purchase successful or purchase unsuccessful page depending on the response you receive).

Note: FAC's payment gateway does not itself approve or decline a transaction. The transaction responses come directly from the issuer or processor.

However, a 3D Secure transaction is substantially more involved and this is why the implementation of the Authorize3DS operation is different than any of the other FACPG service operations. You will notice in the request for the Authorize3DS method, in [Authorize3DSRequest](#), that once you receive the response you immediately post it back to the cardholder's browser (there is also a sample of what this response looks like). The response is an HTML page with a form with all of the required fields set as hidden fields with a piece of JavaScript code that forces a posting of this form from the cardholder's browser directly to FAC. It is for this reason that you provide the MerchantResponseURL parameter in 3D Secure transactions as this is where the cardholder's browser will eventually be directed back to at the end of the transaction.

Currently 3DSecure is supported only for Visa and MasterCard transactions. Discover and AMEX transactions must be processed as non-3D Secure. A non-3DS transaction would be implemented through the [Authorize Operation](#).

3D Secure Authentication Only Transactions

On top of 3D Secure Authorization and Authorization/Capture transactions, FAC also provides our merchants the ability to process '3D Secure Authentication Only' transactions.

These transactions are not in any way authorized, but are used to authenticate that the user of the card is indeed the owner of the card by requiring the 3D Secure password to be entered.

A 3-D Secure Authentication Only transaction is made by adding the 3D Secure Only flag value (64) to the TransactionCode parameter in the [Authorize3DSRequest](#) when calling [The Authorize3DS Operation](#).

The response provided (sent to the MerchantResponseURL provided in the initial request – just as in a normal 3D Secure transaction) includes the four 3D Secure related parameters you are looking for from this type of transaction. These are outlined in the table below.

Parameter Name	Value
TransactionStain	A hashed version of the Transaction ID (XID). The XID is a unique tracking number assigned to the authentication request that prevents replay or resubmission of the same transaction.
ECIIndicator	The ECI value indicates the result of the authentication request. Possible values are: Visa: "05" - Full 3D Secure authentication "06" - Issuer and/or cardholder are not enrolled for 3D Secure "07" - 3D Secure authentication attempt failed (numerous possible reasons) MasterCard: "01" - Issuer and/or cardholder are not enrolled for 3D Secure "02" - Full 3D Secure authentication
AuthenticationResult	Valid values are "A", "N", "U" or "Y" "A" = An attempt at authentication was performed (ECIIndicator: V=06, MC=01) "N" = Authentication attempt not supported (ECIIndicator: V=06, MC=01) "U" = Unable to authenticate (ECIIndicator: V=07, MC=01) "Y" = Authentication attempted and succeeded (ECIIndicator: V=05, MC=02)
CAVV	This is a cryptographic value derived by the issuer during payment authentication that provides evidence of the results of the payment authentication process. Note that for MasterCard this field is referred to as UCAF but the field name will still be CAVV.

3D Secure authorize/capture ("one-pass") transactions

Since the 3D Secure transaction process flow involves the cardholder's browser, it is possible for the cardholder to close their browser window after the authentication part of a 3D Secure transaction is complete and before the authorization piece of the transaction is complete. If they do this (it is rare, but can happen), it is possible for the transaction to be approved, without ever getting an approval response back since this response has to come through the cardholder's browser, which has been closed. For this reason, FAC strongly recommends against processing authorize/capture (also known as "one-pass") 3D Secure transactions. Your 3D Secure transaction process flow should be "two-pass", meaning you run an authorization only transaction first and once you receive the response, you run a capture transaction to capture the authorization amount for settlement purposes.

Fraud Control

The FAC Payment Gateway, integrated seamlessly with several Fraud checking services so that the merchant has only one point of integration for sending an Authorization with an included Fraud check, or indeed to do a separate “Fraud Check Only” Authorization message.

Configuring a Fraud check within the context of an Authorization Transaction requires set-up in several systems.

- 1) The Fraud Service Host’s System
- 2) FAC Payment Gateway System
- 3) The Merchant Site (not required by all fraud services)

FAC’s Fraud Control service main component uses a Kount™ (third-party solution & partner) which is a highly rated fraud scoring engine/service and/or BIN Checking (to determine the card Issuer/Country of Issue).

This service can be used together with an authorization or independently as a stand-alone service (no financial transaction).

Can be used for:

- Information purposes/Additional Validation – to help them validate a customer and how to proceed with a payment
- Kount™ Decision Rules - Decision rules can be set by the merchant to provide a merchant with advisement on how to proceed with a transaction.
- FAC Fraud Rules – Rules can be set at the Gateway level to automatically reject a transaction or allow a transaction to proceed for authorization based on the results that come back

Please contact FAC Support if you are interested in Fraud Checking and they will manage the integration process for you.

Fraud Control - Client-side Implementation Requirements for Kount Service

While some fraud services do not require any involvement from the client browser, the Kount fraud service, now integrated to FAC Payment Gateway, does require some work on the part of the Merchant, but this work is minimal. Kount calls this client-side integration the “Data Collector”.

Kount uses some proprietary java-script code to obtain detailed information about the browser and the client machines properties. This enables a more accurate fraud check as specific data is known about the Cardholders environment.

Kount uses some proprietary java-script code to obtain detailed information about the browser and the client machines properties. This enables a more accurate fraud check as specific data is known about the Cardholders environment. The KOUNT Client Collector SDK data collection process is triggered by a load data event in the <body> tag.

To tie this data in with the Authorization data, a session id is required. This is something the Merchant must generate and pass to the Data Collection URL also added to the checkout page.

Practical Example

In this example we are using JQuery/JavaScript to implement the client side requirements. On the server-side, PHP is used. The server side code could just as easily be done in JSP or .NET code behind.

There are several terms you will need to understand the meaning of:

Name	Size	Description	Example
KOUNT_SERVER_URL	N/A	HTTPS URL Path to Kount servers	https://tst.kaptcha.com (Staging Platform) https://ssl.kaptcha.com (Production)
MERCHANT_ID	6	Six digit Merchant Identifier issued by Kount	999999
SESSION_ID	1-32	Unique Session ID created by Merchant.	BDB721BA17E4A4BB58B21A54
MERCHANT_URL	N/A	Merchant's Website URL	https://somemerchant.com/

Client Side:

In any page prior or on the payment page, you must create the session id, and pass this together with your merchant id to the DATA COLLECTOR URL used on the checkout page. The variables "merchantId" and "sessionId" need to be supplied.

Creating a Session ID

Example Script that can be used to generate a session id:

```
<script>
    var uuid = 'xxxxxxxxxx4xxxxyxxxxxxxxxxxxxxxx'.replace(/[xy]/g, function(c) {
        var r = Math.random()*16|0, v = c == 'x' ? r : (r&0x3|0x8);
        return v.toString(16);
    });
</script>
```

This has the advantage of not requiring you to pass in values to an already existing element. The element is created with the parameters in place.

You could also use PHP for creating the session id on the payment form. Here is an example that uses the built in session_id() function.

```
<?php
$sess = session_id();
if ?>
    (!$sess) {
// If the session hasn't already been started, start it now and look up the id
session_start();
$sess = session_id();
}
// The session id is now available for use in the variable $sess
// For more details and examples on working with sessions in PHP, see:
// http://us2.php.net/manual/en/book.session.php
// http://us2.php.net/session_start
// http://us2.php.net/session_id
```

IMPORTANT: The same Session ID must be passed to the KOUNT Data Collection and the FAC Payment Gateway in the AuthorizeRequest or if you are using the Hosted Page functionality, then you must include a hidden field with the session Id assigned to it called "SessionID".

This is required to match up the data retrieved from the cardholder by Kount with the data passed to Kount from the transaction processed through FAC.

DATA Collection Code

The KOUNT Client Collector SDK data collection process is triggered by the load data event.

1. Add the following specified class and attribute to the BODY tag in your Checkout page
<body class="kaxsdc" data-event="load">
2. Include these scripts to the bottom of the <body></body> element:

```
<script type='text/javascript'
src='https://KOUNT_SERVER_URL/collect/sdk?m=merchantId&s=sessionId'></script>
```

Input parameters:

- KOUNT_SERVER_URL - tst.kaptcha.com (staging) or ssl.kaptcha.com (production)
- merchantId - Kount ClientId
- sessionId - Kount SessionId

Add script

```
<script type='text/javascript'>  
    var client=new ka.ClientSDK();  
    client.autoLoadEvents();  
</script>
```

Once the Kount script is finished, it then does another re-direct back to the checkout page on the Merchant's Server.

The Fraud Check Response

Once the Authorization and Fraud check have been completed, the results of both are returned in the AuthorizeResponse message.

You can find details of the fields in the Fraud Control Response in the description of the [AuthorizeResponse](#).

IMPORTANT: How the Fraud Checking system reacts to a bad Fraud score in relation to the Authorization request depends on how you want Fraud Checking set-up. The system is flexible and rules can be applied to cover most scenarios. Please contact FAC Support for more information.

Fraud Control Response and Reason Codes

The ResponseCode, ReasonCode and ReasonCodeDescription fields of the FraudControlResponse message can hold several code and description combinations, found in [Appendix B](#). Fraud Control Response and ReasonCodes may also contain values specified in the Authorization section.

Below is a sample of results :

```
<FraudControlResults>  
    <FraudResponseCode>D</FraudResponseCode>  
    <ReasonCode>2021</ReasonCode>  
    <ReasonCodeDesc>Fraud Decline</ReasonCodeDesc>  
    <ResponseCode>2</ResponseCode>  
</FraudControlResults>
```

Testing Various FraudControlResponses

To test various FraudControlResponses in our test environment you can for back a particular value based on the dollar amount used:

Value	Description	Amount
R	Review Transaction	1.00
D	Deny Transaction	2.00
A	Approve Transaction	3.00
E	Escalate Transaction (Review)	4.00

BIN Blocking

What is a “BIN” and what is “BIN Blocking”?

Every credit card number contains what is called a Bank Identification Number (aka BIN). It is a set of numbers which can provide information on the financial institution which issued that card (Credit Card Issuer) along with the country from where it was issued.

FAC’s Payment Gateway, offers a service which a merchant can subscribe to in order to essentially block or allow transactions from being submitted for authorization based on a credit card’s BIN and country it has been issued. BIN validation occurs prior to an authorization taking place. Depending on the rules set in place by FAC at your account level transactions that are blocked will never proceed to interchange for authorization and automatically return a decline response as outlined below. Alternately transactions that are permitted to proceed for authorization will continue through as normal and return the expected authorization response sets as outlined in the Authorize or Authorize3DS responses.

For example, transactions being blocked by this service the expected denial response and reason codes (along with a sample of BIN Check Results) :

```
<CreditCardTransactionResults>
  <OriginalResponseCode>34</OriginalResponseCode>
  <ReasonCode>2</ReasonCode>
  <ReasonCodeDescription>Transaction is declined.</ReasonCodeDescription>
  <ResponseCode>2</ResponseCode>
</CreditCardTransactionResults>

<FraudControlResults>
  <FraudResponseCode>B</FraudResponseCode>
  <ReasonCode>2021</ReasonCode>
  <ReasonCodeDesc>BinCheck decline</ReasonCodeDesc>
  <ResponseCode>2</ResponseCode>
</FraudControlResults>

<BinCheckResults>
  <BIN>411111</BIN>
  <Brand>VISA</Brand>
  <Country>UNITED STATES</Country>
  <Issuer>TEST BANK NAME</Issuer>
</BinCheckResults>
```

NOTE: Every merchant differs in how they may be able to use this service so it is important to discuss with FAC any way that FAC’s BIN Blocking services and set transaction rules can support your processing.

NOTE: Full response sets included in the appendix of this guide (Appendix Section 7)

Testing Various BINs

To test various BINs in our test environment you can use any of the test card numbers below.

Issuing Country	Card Number
United States	5111111111111111
Honduras	5140001111111111
Poland	5454541111111111
Malaysia	5222221111111111
BIN NOT FOUND	5000001111111111

Signature Creation and Verification

Authorize Request Message Signature Creation

The signature that you send to FAC in your [AuthorizeRequest](#) or [Authorize3DSRequest](#) uses a BASE-64 Encoded SHA1 signature hash to verify the message structure and content. Despite the existence of an API signature, all API traffic should be encrypted the entire path from client to server.

A unique signature is created using the SHA1 algorithm and then this result is BASE-64 encoded. This signature is designed in such a way that any slight change to the message would result in a totally different hash. In order to create this hash, the following 6 fields are required in this order:

Processing Password (**orange**)

FAC ID (**blue**)

Acquirer ID (**green**)

Order ID (**brown**)

Amount (**red**)

Purchase Currency (**purple**)

e.g.: Signature data before SHA1 Hash:

a1B23c1234567890464748FACTEST0100000001200840

Resulting Hash Value: jLA0GGha1I+BbOqSys4au4aM2TY=

Tokenization Message Signature Creation

This message also takes a signature but uses fewer fields as it is independent of a transaction. These are, in this order:

Processing Password (**orange**)

FAC ID (**blue**)

Acquirer ID (**green**)

e.g.: Signature data before SHA1 Hash:

a1B23c1234567890464748

The implementation of this will be exactly the same as the Authorize methods.

Response Message Signature Verification

[Authorize3DSResponse](#) & [AuthorizeResponse](#) Operation Responses also return a verification hash, with approved and declined authorization transactions, to verify that the response is from FAC's system. To use this signature as verification, create a BASE-64 Encoded SHA1 hash from the following fields in this order:

Processing Password (**orange**)

FAC ID (**blue**)

Acquirer ID (**green**)

Order ID (**brown**)

e.g.:

a1B23c1234567890464748FACTEST01

Resulting Hash Value: LOijfhLT2JO2jYaA1bXPIZNWDPg=

General Steps to Implement a Signature

Independently of any framework, the process of creating the signature can be described as follows:

1. Concatenate Data ensuring the data is trimmed of spaces.
2. Create Hash using SHA1 algorithm
3. Convert Hash (usually a byte array) into a Base64 String
4. UrlEncode the String

Code Samples for Generating a SHA1 Signature

The following code samples will demonstrate how to create the hash value required for FAC's system

VB.NET

Imports System.Security.Cryptography

Private Function ComputeHash(ByVal Key As String) As String

Dim objSHA1 As New SHA1CryptoServiceProvider

objSHA1.ComputeHash(System.Text.Encoding.UTF8.GetBytes(Key.ToCharArray))

Dim buffer() As Byte = objSHA1.Hash

Dim HashValue As String = System.Convert.ToBase64String(buffer)

Return HashValue

End Function

C#

```
using System.Security.Cryptography;
private Private
{
    string ComputeHash(string Key)
    {
        SHA1CryptoServiceProvider objSHA1 = new SHA1CryptoServiceProvider();
        objSHA1.ComputeHash(System.Text.Encoding.UTF8.GetBytes(Key.ToCharArray()));
        byte[] buffer = objSHA1.Hash;
        string HashValue = System.Convert.ToBase64String(buffer);
        return HashValue;
    }
}
```

PHP 4.3.9

```
$signature = urlencode(base64_encode(pack("H*",
sha1(MODULE_PAYMENT_FAC_MERCHANT_PASSWORD .
MODULE_PAYMENT_FAC_MERCHANT_ID . '464748' . $orderid . $amount .
MODULE_PAYMENT_FAC_CURRENCY))));
```

Response Code and Reason Code Responses

The ResponseCode, ReasonCode and ReasonCodeDescription fields of the [AuthorizeResponse](#) and [TransactionStatusResponse](#) messages can hold several different code combinations, found in [Appendix B](#).

Authorization Original Response Codes

The response codes for an Authorization are returned in the OriginalResponseCode field of the Response. See [AuthorizeResponse](#), [TransactionModificationResponse](#), or [TransactionStatusResponse](#). They are specific to the Card Issuer - Please see [Appendix B](#) for the full lists of Visa, MasterCard, and Amex Original Response Codes.

The Authorize Operation

The Authorize Operation uses the AuthorizeRequest message to submit a Non-3D secure authorization transaction request. With this Operation it is possible to:

- i) Include a capture as part of the request (called a one-pass transaction)
- ii) Include an address verification (AVS) check with the request
- iii) Perform a \$0 Address Verification request only, with no authorization
- iv) Include the results of a previous 3D Secure authenticated transaction

Due to the stateless and web like nature of the Internet, it is possible that occasionally an authorization transaction request sent to FAC will timeout before you receive a response. If this is the case, do not assume the transaction was not processed on our end. It is possible that we received the request and sent you a response which never reached you due to the fact that the connection between your server and FAC dropped. As such, use The [TransactionStatus operation](#) to check the status of this transaction to verify if it was approved or not.

If the TransactionCode is set to include the AVS Verification Only flag, the amount of the transaction will default to \$0 regardless of the value provided in the Amount parameter. AVS Verification Only transactions are never authorized. For more information see [\\$0 AVS Verification Only Transactions](#).

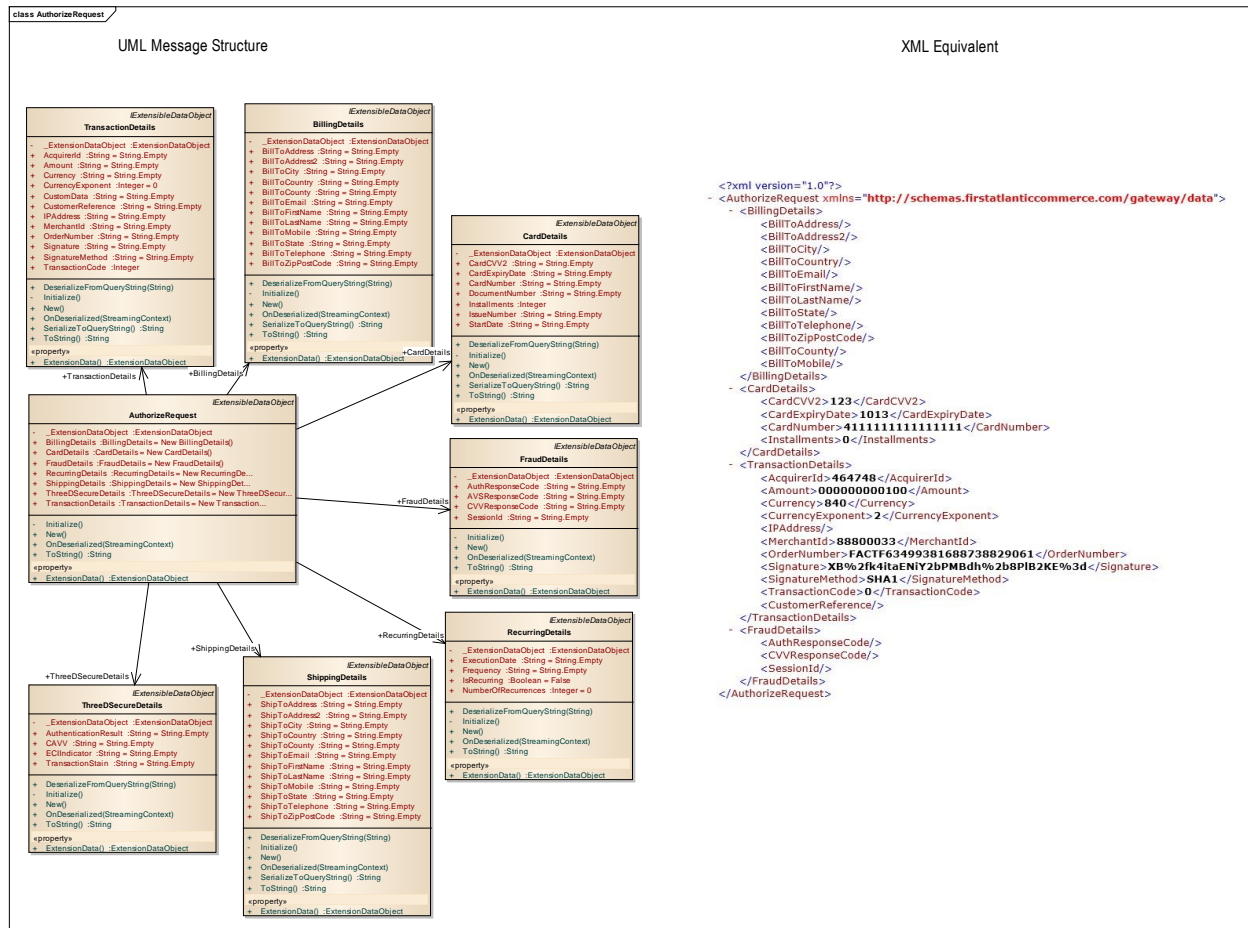
If the TransactionCode is set to include the 'Pre-Authenticated' flag, the ECIIndicator and AuthenticationResult parameters must be provided in the request or the transaction will fail. In addition, if the ECIIndicator value is set to "05" (for Visa) or "02" (for MC), and the AuthenticationResult value is set to "Y", then the TransactionStain and CAVVValue parameters must be provided or the transaction will fail.

You cannot set the TransactionCode to include both AVSOnly and PreAuthenticated flags. Doing so will result in an error response and the transaction will not be processed.

AuthorizeRequest

Message Structure

The Message is divided into seven sub-sections.



The UML diagram shows the Classes that would be created by any SOAP toolkit to represent the request data. The XML equivalent is what is used in its place when sending a request via the XML POST mechanism.

Detailed Field Descriptions

Section/Field	Format	R/ O/ C	Value
TransactionDetails			
AcquirerId	N(11)	R	"464748"
MerchantId	N(15)	R	Your FAC ID provided by FAC
OrderNumber	AN(150)	R	A unique identifier assigned by the merchant for the transaction
TransactionCode	N(4)	R	<p>The transaction code is a numeric value that allows any combinations of the flags listed below to be included with the transaction request by summing their corresponding value. For example, to include AVS in the transaction and to tokenize the card number, assign the sum of the corresponding values 1 and 128 to the transaction code. The valid codes for an Authorization request are:</p> <p>0 - None 1 - Include an AVS check in the transaction OR Flag as a \$0 AVS verification only transaction 2 - **HOST SPECIFIC - Flag as a \$0 AVS verification only transaction 4 - Transaction has been previously 3D Secure Authenticated the 3D Secure results will be included in the transaction. 8 - Flag as a single pass transaction (Authorization and Capture as a single transaction) 64 - Flag as a 3DS Authenticate Only transaction (3DS-Only) 128 - Tokenize PAN (Request Token) 256 - Hosted Page Auth + 3DS (applies to Hosted Payment Pages only) 512 - Fraud Check Only 1024 - Fraud Test 2048 - Subsequent Recurring - future recurring payments 4096 - Initial Recurring - First Payment in a recurring cycle 8192 - **HOST SPECIFIC - Initial Recurring for "Free-Trials"</p>
Amount	N(12)	R	Total amount of purchase. Note: The purchase amount must be presented as a character string that is 12 characters long. (i.e. \$12.00 would be provided as "000000001200")
Currency	N(3)	R	The purchase currency ISO country code (i.e. US Dollars = 840) See Appendix A for a full list of codes.
CurrencyExponent	N(1)	R	The number of digits after the decimal point in the purchase amount (i.e. \$12.00 = 2)
SignatureMethod	AN(4)	R	"SHA1"
Signature	AN(28)	R	See Signature Creation and Verification for information on creating this signature
IPAddress	AN(15)	C	IPv4 client address

CustomData	AN(n)	O	Reserved for future use
CustomerReference	AN(256)	O	Used with Tokenization Request to associate a Token with a Customer
CardDetails			
CardNumber	N(19)	R	Cardholder's credit/debit card number or previously Tokenized PAN
CardExpiryDate	N(4)	R	Cardholder's credit card expiry date (formatted as MMY format). If using a Tokenized Card Number, CardExpiryDate can be any future date (e.g. current month or later). It does not have to match actual card expiry date, but must not be blank or past.
CardCVV2	N(4)	R	Cardholder's credit card CVV2 number as it appears on the back of the credit card (3 digits for Visa and MasterCard, 4 digits for AMEX)
IssueNumber	N(2)	C	Cardholder's debit card issue number. Can be one or two digits depending on card type.
StartDate	N(4)	C	Cardholder's credit/debit card start date (formatted as MMY format). Required for some debit cards.
BillingDetails			
BillToAddress	AN(50)	C	Cardholder's address. This value is needed for AVS verification transactions. See Appendix A for all Billing Details field requirements.
BillToAddress2	AN(50)	O	Reserved for FAC Internal use
BillToZipPostCode	AN(10)	C	Cardholder's zip/postal code. This value is needed for AVS verification transactions. See Appendix A for field requirements.
BillToFirstName	AN(30)	O	Cardholder's first name
BillToLastName	AN(30)	O	Cardholder's last name
BillToCity	AN(30)	O	Cardholder's city
BillToState	AN(5)	O	Cardholder's state – MUST BE minimum 2 characters long. See Appendix A for allowable values.
BillToCountry*	N(3)	O	Cardholder's ISO country code (ie. U.S.A. = 840) See Appendix A .
BillToCounty	AN(15)	O	Cardholder's county
BillToEmail*	AN(50)	O	Cardholder's e-mail address
BillToTelephone*	N(20)	O	Cardholder's telephone number
BillToMobile	N(20)	O	Cardholder's mobile number
BillToFax	N(20)	O	Cardholder's fax number
ShippingDetails			
Same fields as BillingDetails but prefixed with "ShipTo"			
ThreeDSecure Details			
ECIndicator	N(2)	C	This value is only needed for pre-authenticated 3D Secure transactions and the transaction code must include the value 4 in its summed value. Possible values include: Visa: "05" - Full 3D Secure authentication "06" - Issuer and/or cardholder are not enrolled for 3D Secure

			<p>"07" - 3D Secure authentication attempt failed (numerous possible reasons) MasterCard: "01" - Issuer and/or cardholder are not enrolled for 3D Secure "02" - Full 3D Secure authentication</p>
AuthenticationResult	A(1)	C	<p>This value is only needed for pre-authenticated 3D Secure transactions and the transaction code must include the value 4 in its summed value. Possible values include: "A" = An attempt at authentication was performed (ECIIndicator: V=06, MC=01) "N" = Authentication attempt not supported (ECIIndicator: V=06, MC=01) "U" = Unable to authenticate (ECIIndicator: V=07, MC=01) "Y" = Authentication attempted and succeeded (ECIIndicator: V=05, MC=02)</p>
TransactionStain	AN(28)	O	<p>A hashed version of the Transaction ID (XID). The XID is a unique tracking number assigned to the authentication request that prevents replay or resubmission of the same transaction.</p>
CAVV	AN(28)	O	<p>This is a cryptographic value derived by the issuer during payment authentication that provides evidence of the results of the payment authentication process. Note that for MasterCard this field is referred to as UCAF but the field name will still be CAVV.</p>
RecurringDetails			
IsRecurring	A(5)	C	<p>Set to "True" or "False" depending on whether a Recurring transaction is required.</p>
ExecutionDate	N(8)	C	<p>When to execute the next authorization after the immediate initial authorization. Example Date format is "20130715" (YYYYMMDD).</p>
Frequency	A(1)	C	<p>Flag to determine how frequently to execute the recurring authorization. Possible values are: "D" – Daily "W" – Weekly "F" – Fortnightly/Every 2 weeks "M" – Monthly "E" – Bi-Monthly "Q" – Quarterly "Y" – Yearly</p>
NumberOfRecurrences	N(3)	C	<p>How many times in total to execute. For example, Frequency = "D", NumberOfRecurrences = 7 will execute every day for a week.</p>
FraudDetails			
AuthResponseCode	A(1)	O	<p>For pre-authorized Fraud-only checks, "A" = Auth, "D" = Declined.</p>

AVSResponseCode	A(1)	O	Standard FAC AVS response code , for pre-authorized Fraud-only checks
CVVResponseCode	A(1)	O	Standard FAC CVV2 validation response code , for pre-authorized Fraud-only checks
SessionId	AN(32)	R	For any Fraud check a unique Session ID is required and it must be the same as the session ID generated for the Data Collector. See Fraud Control

Note: “R/O/C” refers to the field’s presence as being Required, Optional, or Conditional.

*The BillingDetails fields **BillToCountry**, **BillToEmail**, and **BillToTelephone** are only included when the transaction is flagged as an AVS transaction. See [Address Verification \(AVS\) Checks](#) for more information on AVS transactions and please see [Appendix A](#) for all Billing Details field requirements.

The ability to process AVS checks is processor-specific. Please inform FAC if you wish to use this service and we will advise on its availability through your selected processor.

While the [Recurring](#) interface is in place, it may be dependent on processor, so please inform FAC if you wish to start using Recurring transactions.

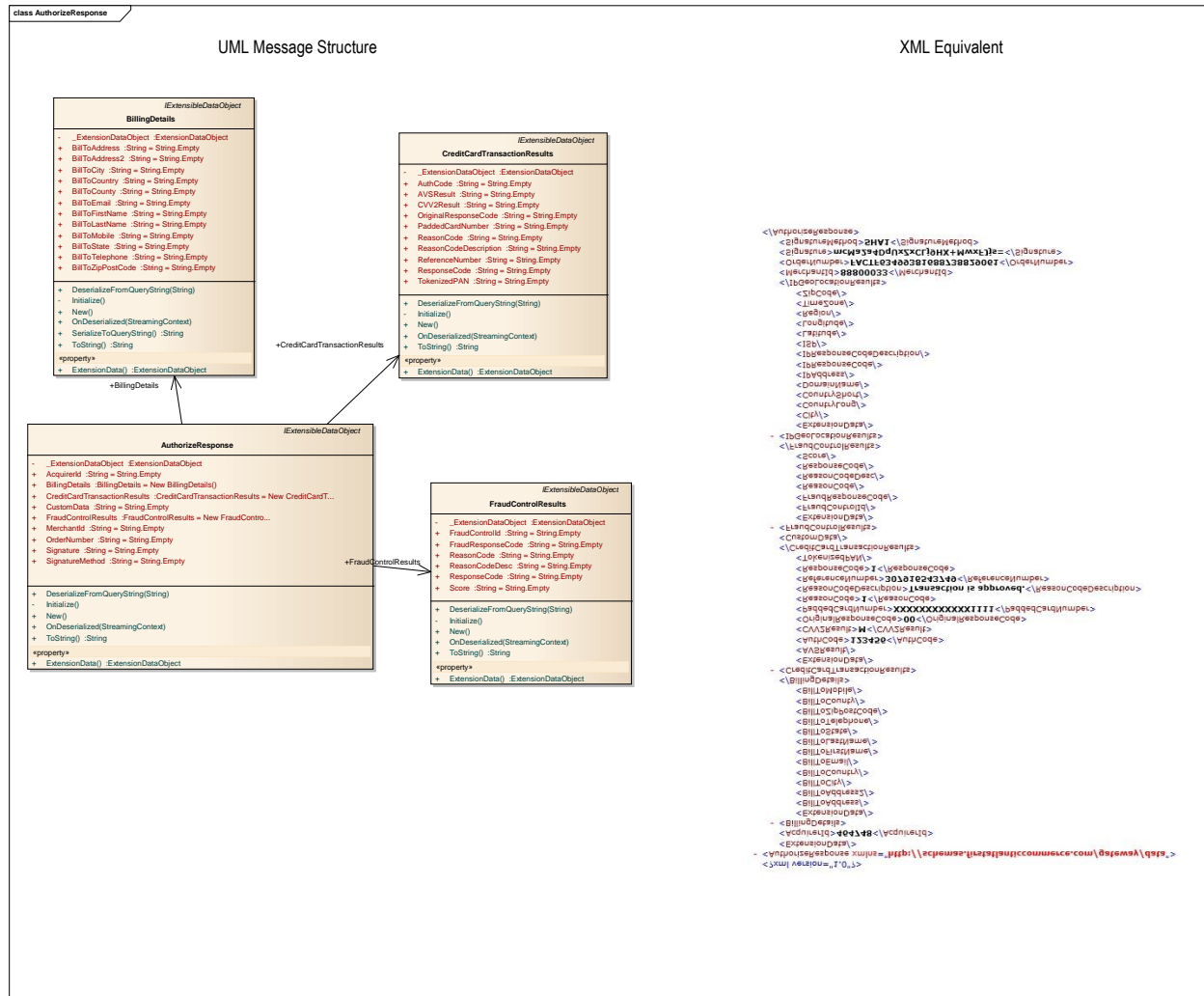
For [Fraud Control](#) services, it will require additional configuration changes to your processing account and require you to subscribe to the service. Please inform FAC if you would like to implement fraud control.

AuthorizeResponse

The data returned from an Authorize Operation is in the form of an AuthorizeResponse.

Message Structure

The AuthorizeResponse message has three sub-sections; the main body; a CreditCardTransactionResults section; a BillingDetails Section; and a FraudControlResults section.



Detailed Field Descriptions

Section/Field Name	Format	Notes
AuthorizeResponse (Main Body)		
AcquirerID	N(11)	The value passed in AcquirerId in the AuthorizeRequest
CustomData	AN(n)	Reserved for future use
IPAddress	N(15)	The value passed in IPAddress in the AuthorizeRequest
MerchantID	N(15)	The value passed in MerchantId in the AuthorizeRequest
Signature	AN(28)	See Signature Creation and Verification for information on how this signature is created
SignatureMethod	AN(4)	Always will be "SHA1"
CreditCardTransactionResults		
AuthCode	AN(6)	The authorization code as provided by the cardholder's issuing bank
AVSResult	AN(1)	Address Verification Results Field. See Address Verification (AVS) Checks
CVV2Result	A(1)	See CVV2/CVC2 Response Codes for possible values for this parameter.
OriginalResponseCode	AN(3)	See Authorization Original Response Codes for possible values for this parameter.
PaddedCardNumber	N(19)	The value passed in CardNumber in the AuthorizeRequest, padded with X's, providing the last 4 numbers only
ReasonCode	N(4)	See Response Code and Reason Code Responses for possible values for this parameter.
ReasonCodeDescription	AN(100)	See Response Code and Reason Code Responses for possible values for this parameter.
ReferenceNumber	N(12)	The unique reference number given this transaction by FAC's system
ResponseCode	N(1)	See Response Code and Reason Code Responses for possible values for this parameter.
TokenizedPAN	N(19)	Tokenized Card Number returned with Tokenization Requests and Tokenized Authorizations
BillingDetails		
BillToAddress	AN(50)	Cardholder's address. This value is needed for AVS verification transactions.
BillToAddress2	AN(50)	Reserved for FAC Internal use
BillToZipPostCode	AN(10)	Cardholder's zip/postal code. This value is needed for AVS verification transactions.
BillToFirstName	AN(30)	Cardholder's first name
BillToLastName	AN(30)	Cardholder's last name
BillToCity	AN(30)	Cardholder's city
BillToState	AN(5)	Cardholder's state – MUST BE minimum 2 characters long
BillToCountry	N(3)	Cardholder's ISO country code (ie. U.S. = 840) See Appendix A .
BillToCounty	AN(15)	Cardholder's county
BillToEmail	AN(50)	Cardholder's e-mail address
BillToTelephone	N(20)	Cardholder's telephone number

BillToMobile	N(20)	Cardholder's mobile number
BillToFax	N(20)	Cardholder's fax number
FraudControlResults		
AuthCode	N(6)	Authorization code of successful transaction
FraudControlId	AN(50)	Fraud System transaction ID
FraudResponseCode	AN(3)	Fraud System Response Code. See Fraud Control Response Codes .
ReasonCode	AN(4)	Fraud reason if ResponseCode = 3 See Fraud Control Reason Codes .
ReasonCodeDesc	AN(255)	Fraud Reason Description. See Fraud Control Reason Codes .
ResponseCode	N(1)	1 or 3, Pass or Fail. See Fraud Control Response Codes .
Score	AN(20)	Fraud Score Number (Kount)

Tokenization using the Authorize (and Authorize3DS) Operation

The FAC Payment Gateway has the ability to Tokenize Card Numbers (PAN) and associated expiry date so that the merchant does not have to store the PAN. In fact, once Tokenized, the merchant cannot access the original PAN at all via FACPG2 and must use the Token from that point on when processing transactions (if the merchant is not storing the PAN itself).

The Tokenization process can start with an Authorization or 3DS Authorization. This is required when the first transaction may need the CVV2 field to be verified and it cannot be saved as part of the Tokenization process (saving CVV2 is not PCI compliant). It is also good practice to validate a card before tokenization. You could use an AVS only or zero/low value Auth without Capture transaction if you do not have a real transaction to process at the time of Tokenization (this may depend on the Payment Host facilities).

The main advantage of Tokenization to a Merchant is never having to store the actual PAN. This puts the Merchant out of scope for most PCI regulations and reduces the cost of PCI compliance considerably.

What is a Token?

When a Card Number is tokenized, the middle 6 characters are replaced with a special sequence of characters that represent the real numbers within the FAC Payment Gateway. This information is stored in a specially encrypted database that is PCI Compliant. Using special sequence of characters you cannot infer the actual card number. E.g.:

Actual Card Number 4111111111111111

Tokenized Card Number 411111_000011111 (Token is _00001)

While it seems odd to still expose most of the number, these are useful for recognition of the card by the customer or merchant. The important part of the number is masked and cannot be derived from the 6 digit token.

Tokenizing a PAN

To Tokenize a PAN, the following steps are required.

- 1) Set bit 8 (value 128) in the TransactionCode field of the AuthorizeRequest.
- 2) You can set a value specific to a customer using the CustomerReference field.
This can be used to retrieve a set of Tokenized PANs by customer (only on request to FAC, see roadmap below).
- 3) Set up the other fields for a standard Authorization.
- 4) Send the transaction to the FAC Payment Gateway.
- 5) Save the returned Token found in the TokenizedPAN field of the Response.

Updating the Expiry Date or Customer Reference

In order to update either the Expiry Date or Customer Reference, use the [UpdateToken](#) method of the Tokenization.svc Service.

Using a Tokenized PAN in a standard Authorization

To Use Tokenized PAN, the following steps are required.

- 1) Set the CardNumber using the TokenizedPAN saved from step 5) above.
- 2) You should still set the Expiry Date, but it will be superseded by the one stored with the Token.
- 3) Set up the other fields for a standard Authorization (inc. CVV2 if required).
- 4) Send the transaction to the FAC Payment Gateway.
- 5) The Tokenized PAN is Detokenized within the FAC Payment Gateway and is processed as normal.

Tokenizing without an Authorization Transaction

It is possible to create a Token outside of an Authorization by using the Tokenization.svc FACPG2 Service. See the [Tokenize Operation](#) and related Operations.

Avoiding Card Numbers

To avoid accessing the card data altogether, see [HostedPageAuthorize and HostedPageResults](#).

These two operations are part of a much wider Hosted Page system for building and using custom payment pages hosted by FAC. This completely avoids card data entering the Merchants system and is a good way of minimizing the potential PCI compliance overhead.

However, this is a large subject and as such has been included in a separate integration document. Please ask for the Hosted Page guide if you are interested in this style of Integration.

The Authorize3DS Operation

The Authorize3DS Operation uses the Authorize3DSRequest message to send 3DS Authorization data that is used to build an HTML form, with hidden fields. This form is returned in the Authorize3DSResponse. With this message it is possible to:

- i) Include a capture as part of the request (called a one-pass transaction)
- ii) Include an address verification (AVS) check with the request
- iii) Perform a 3DS authentication only

Note: 3DS does not support Recurring Transactions at this time.

Note: An AVS check cannot be performed with a 3D Secure Authentication Only transaction. If the TransactionCode parameter is set to include the '3D Secure Authentication Only' flag, any AVS related information provided in the request will be ignored.

Note: See 3D Secure vs. Non-3D Secure Transactions for more on 3DS transactions. See 3D Secure Authentication Only Transactions for more on 3D Secure authentication only transactions.

Implementing the 3DS Message

This method differs from Authorize web method as the response returned is not the final authorization response, but instead is an HTML page with embedded JavaScript code, that must be re-directed back to the cardholder's browser. This is necessary so that the authentication portion of a 3D Secure transaction occurs directly between a cardholder's browser and their issuing bank. Because of this, you must do two things for these web methods that you are not required to do for any of the other web methods:

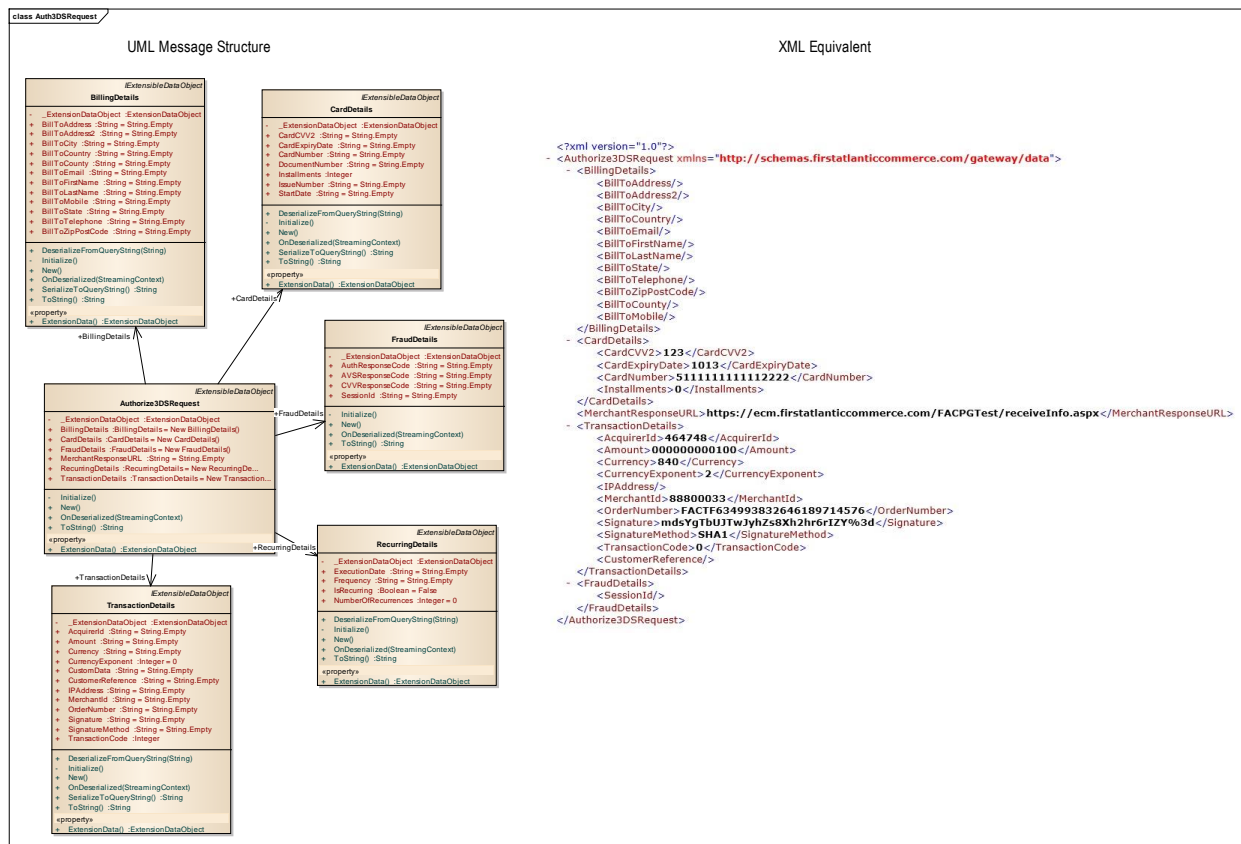
- i) Post the response received from this message, unaltered, back to the cardholder's browser.
- ii) Develop a secure web page using HTTPS, to receive the final 3DS authorization information. The URL of this page must be provided in the MerchantResponseURL field of the Authorize3DS request

Please see [Sample Code](#) for sample code for implementing each of these web methods.

Authorize3DSRequest

Message Structure

The Authorize3DSRequest message has a main body and four sections; TransactionDetails, CardDetails, BillingDetails, and FraudDetails. Note that these are also included in the [AuthorizeRequest](#) and will not be repeated in the detailed field descriptions.



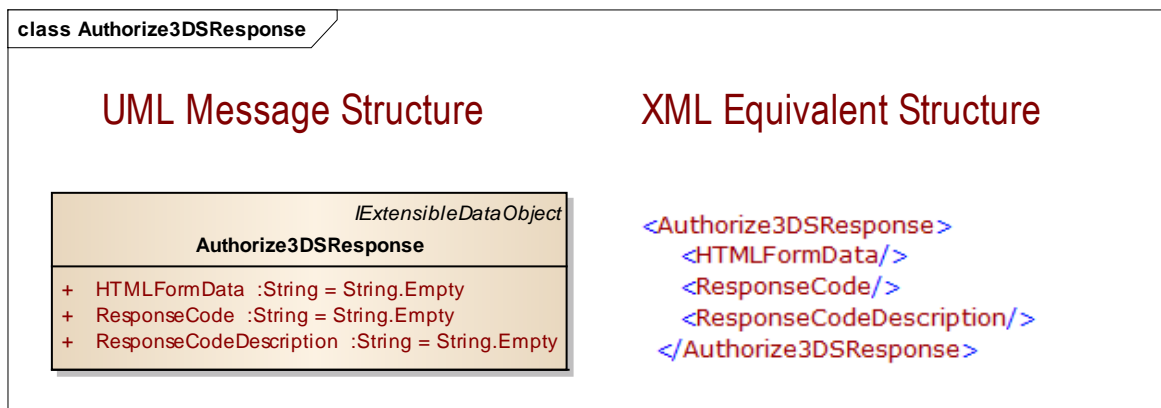
Detailed Field Descriptions

Section	Field Name	Format	Presence	Notes
Authorize3DSRequest (Main Body)	MerchantResponseURL	AN(250)	R	The URL of a secure web page using HTTPS, to receive the final 3DS authorization information.
BillingDetails	See AuthorizeRequest , BillingDetails Section			
CardDetails	See AuthorizeRequest , CardDetails Section			
TransactionDetails	See AuthorizeRequest , TransactionDetails Section			
FraudDetails	See AuthorizeRequest , FraudDetails Section			

Authorize3DSResponse

The Authorize3DS Operation uses the Authorize3DSResponse message to receive the HTML Form that is sent to the user. This form is later used to communicate with the FAC Gateway do the Authorization after the Cardholder has authenticated with the 3DS Systems.

Message Structure



Detailed Field Descriptions

Section	Field Name	Format	Notes
Authorize3DSResponse (Main Body)	HTMLFormData	AN(250)	HTML page with JavaScript that must be posted back to the user's browser.
	ResponseCode	N(4)	Returns "00" on success or "03" if there's a problem.
	ResponseCodeDescription	AN(100)	Description of response.
	TokenizedPAN	N(19)	Tokenized Card Number returned with Tokenization Requests and Tokenized Authorizations

The TransactionModification Operation

The TransactionModification Operation allows changes to a previously Authorized transaction.

The following operations are possible:

- 1) [Capture](#)
- 2) [Refund](#)
- 3) [Reversal](#)
- 4) [Cancel Recurring](#)

Note: If you receive a timeout from a Capture, Reversal or Refund transaction you will have to check FAC Web Reports to see if the transaction was successful or not. You can check the status of a Recurring transaction in Merchant Administration, under Recurring.

All API traffic should be encrypted the entire path from client to server.

Modification Types

Capture

This Capture operation is used to capture a previous successfully authorized transaction that has not yet been captured. If the capture amount requested is equal to the previously authorized transaction amount then this transaction will be sent for settlement at the next settlement period. If the capture amount requested is less than the previously authorized amount then the transaction will not be settled until either the remaining amount of the previously authorized transaction is either reversed, or a new capture request is made for the fully authorized amount. If the capture amount requested is more than the previously authorized transaction amount then an error will be returned.

Note: If you are going to be processing partial captures, please let FAC know as this requires some specific set-up in our payment gateway system. To see a full list of the possible response and reason codes to a capture request, please see [Response Code and Reason Code Responses](#).

Refund

The Refund operation is used to refund a previously captured and settled transaction. A refund transaction will be sent for settlement at the next settlement period.

Note: It is possible to refund a captured transaction that has not yet been settled, but in doing so, both the capture and refund will appear on your customer's credit card statement as a debit, and then a credit respectively. If possible, it is preferred to avoid this so it is recommended that captured transaction be reversed before being settled.

To see a full list of the possible response and reason codes to a refund request, please see [Transaction Modification Response and Reason Codes](#).

Reversal

The Reversal operation can be used to either fully or partially reverse a previously captured transaction. As stated above, if a previously authorized transaction is only partially captured, then the remaining amount must be reversed before the partial capture amount is sent for settlement.

This operation can also be used to reverse a fully captured transaction provided this transaction has not yet been sent to be settled. If it has been sent to be settled, then you will have to refund the transaction using the Refund operation described above.

Note: Not all processors and Issuers support reversals. If reversals are supported, the reversal will void the authorization. Otherwise (if not), the authorization will be dropped automatically by the Issuer after a set period of time (typically 5-10 days).

Note: If you are going to be processing partial reversals, please let FAC know as this requires some specific set-up in our payment gateway system.

To see a full list of the possible response and reason codes to a reversal request, please see [Transaction Modification Response and Reason Codes](#).

Cancel Recurring

The Cancel Recurring Operation is used to cancel/disable a Recurring transaction. This will prevent the Recurring cycle from executing in the future. This does not refund or reverse an Authorization, but cancels (prevents) any future Recurring payments in a cycle. The Order ID and Amount must match that of the original Recurring Authorization request.

To see a full list of the possible response and reason codes to a cancel recurring request, please see [Transaction Modification Response and Reason Codes](#).

IMPORTANT NOTE: It is not possible to re-enable a recurring transaction after it has been disabled, so please be careful if using this feature.

TransactionModificationRequest

Message Structure

class TransactionModificationRequest	
UML Message Structure	XML Equivalent Structure
<div> <i>ExtensibleDataObject</i> TransactionModificationRequest + AcquirerId :String + Amount :String + CurrencyExponent :Integer + MerchantId :String + ModificationType :Integer + OrderNumber :String + Password :String </div>	<pre> <TransactionModificationRequest> <AcquirerId i:nil="true"/> <Amount i:nil="true"/> <CurrencyExponent>0</CurrencyExponent> <MerchantId i:nil="true"/> <ModificationType>0</ModificationType> <OrderNumber i:nil="true"/> <Password i:nil="true"/> </TransactionModificationRequest> </pre>

Detailed Field Descriptions

Section	Field Name	Format	Presence	Notes
TransactionModification Request (Main Body)	ModificationType	N(1)	R	Can be one of 4 values: 1 = Capture 2 = Refund 3 = Reversal 4 = Cancel Recurring
	AcquirerID	N(11)	R	"464748"
	Password	AN(20)	R	Merchant Password
	MerchantID	N(15)	R	Your FAC ID provided by FAC
	OrderNumber	AN(150)	R	A unique identifier assigned by the merchant for the transaction
	Amount	N(12)	R	Total amount of purchase. Note: The purchase amount must be presented as a character string that is 12 characters long. (ie. \$12.00 would be provided as "000000001200")
	CurrencyExponent	N(1)	R	The number of digits after the decimal point in the purchase amount (ie. \$12.00 = 2)

TransactionModificationResponse

Message Structure

class TransactionModificationResponse	
UML Message Structure	XML Equivalent Structure
<div> <div> <div> <div>TransactionModificationResponse</div> <div> + AcquirerId :String + MerchantId :String + OrderNumber :String + OriginalResponseCode :String + ReasonCode :String + ReasonCodeDescription :String + ResponseCode :String </div> </div> <div> <div>IEExtensibleDataObject</div> </div> </div> </div>	<pre> <TransactionModificationResponse> <AcquirerId i:nil="true"/> <MerchantId i:nil="true"/> <OrderNumber i:nil="true"/> <OriginalResponseCode i:nil="true"/> <ReasonCode i:nil="true"/> <ReasonCodeDescription i:nil="true"/> <ResponseCode i:nil="true"/> </TransactionModificationResponse> </pre>

Detailed Field Descriptions

Section	Field Name	Format	Notes
Transaction Modification Response (Main Body)	AcquirerID	N(11)	The value passed in AcquirerId in the AuthorizeRequest
	MerchantID	N(15)	The value passed in MerchantId in the AuthorizeRequest
	OrderNumber	AN(150)	A unique identifier assigned by the merchant for the transaction
	OriginalResponseCode	AN(3)	See Authorization Original Response Codes for possible values for this parameter
	ReasonCode	N(4)	See Transaction Modification Response and Reason Codes
	ReasonCodeDescription	AN(100)	See Transaction Modification Response and Reason Codes
	ResponseCode	N(1)	See Transaction Modification Response and Reason Codes

Transaction Modification Response and Reason Codes

The ResponseCode, ReasonCode and ReasonCodeDescription fields of the [TransactionModificationResponse](#) message can hold several different code and description combinations, which can be found in [Appendix B](#).

The TransactionStatus Operation

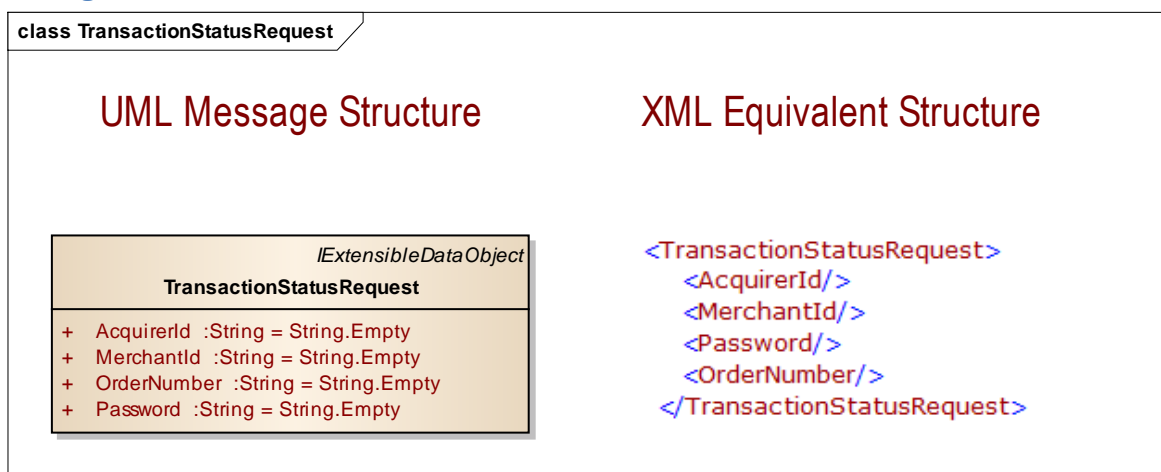
TransactionStatusRequest

The TransactionStatus operation is used to retrieve the status of any previous authorization transaction. Regardless of whether the transaction has since been captured, reversed or refunded, this web method will provide you with the original authorization's response.

There are some important details to note about this Operation:

- This operation does not work for 3D Secure Authenticate Only transactions or \$0 AVS Verification Only transactions.
- This operation does not return the 3D Secure related parameters for a 3D Secure transaction.
- The response provided from this operation will be the same as was provided when the transaction was originally sent for authorization.
- The response provided from this operation will return the most recent authorization attempt for the provided OrderNumber.

Message Structure



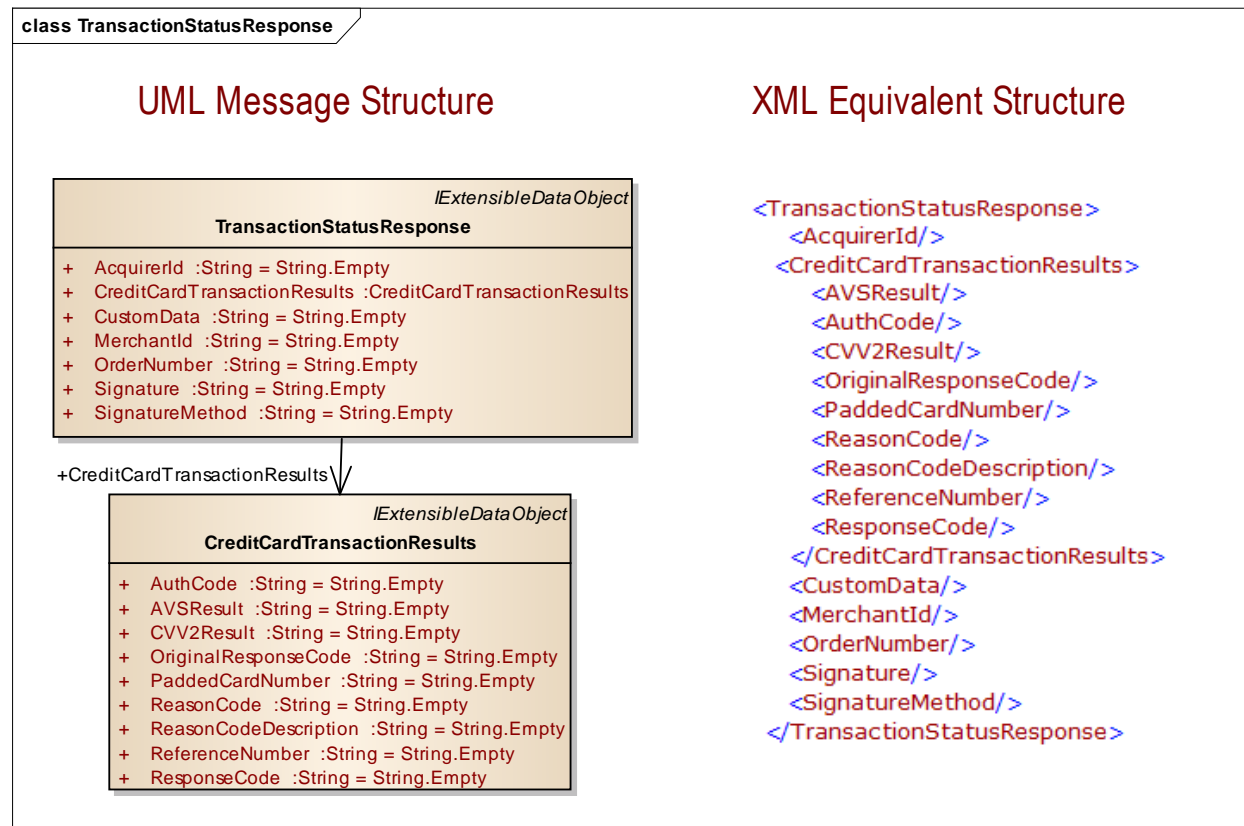
Detailed Field Descriptions

Section	Field Name	Format	Presence	Notes
	AcquirerId	N(11)	R	"464748"
	Password	AN(20)	R	Merchant Password

TransactionStatus Request (Main Body)	MerchantId	N(15)	R	Your FAC ID provided by FAC
	OrderNumber	AN(150)	R	A unique identifier assigned by the merchant for the transaction

TransactionStatusResponse

Message Structure



Detailed Field Descriptions

Section	Field Name	Format	Notes
Transaction StatusResponse (Main Body)	AcquirerID	N(11)	The value passed in AcquirerId in the AuthorizeRequest
	CustomData	AN(n)	Reserved for future use
	MerchantID	N(15)	The value passed in MerchantID in the AuthorizeRequest
	OrderNumber	AN(150)	Merchants original order number.
	Signature	AN(28)	See Signature Creation and Verification for information on how this signature is created
	SignatureMethod	AN(4)	Always will be "SHA1"

CreditCard TransactionResults	AuthCode	AN(6)	The authorization code as provided by the cardholder's issuing bank
	AVSResult	AN(1)	Address Verification Results Field. See Address Verification (AVS) Checks and AVS Check Response Codes
	CVV2Result	A(1)	See CVV2/CVC2 Response Codes for possible values for this parameter
	OriginalResponseCode	AN(3)	See Authorization Original Response Codes for possible values for this parameter
	PaddedCardNumber	N(19)	The value passed in CardNumber in the AuthorizeRequest, padded with X's, providing the last 4 digits only
	ReasonCode	N(4)	See Response Code and Reason Code Responses
	ReasonCodeDescription	AN(100)	See Response Code and Reason Code Responses
	ReferenceNumber	N(12)	The unique reference number given this transaction by FAC's system
	ResponseCode	N(1)	See Response Code and Reason Code Responses

HostedPageAuthorize and HostedPageResults

These two operations are part of a much wider Hosted Page system for building and using custom payment pages hosted by FAC. This completely avoids card data entering the Merchant's system and is a good way of minimizing the potential PCI compliance overhead.

However, this is a large subject and as such has been included in a separate integration document. Please ask for the Hosted Page guide if you are interested in this style of Integration.

The Tokenize Operation

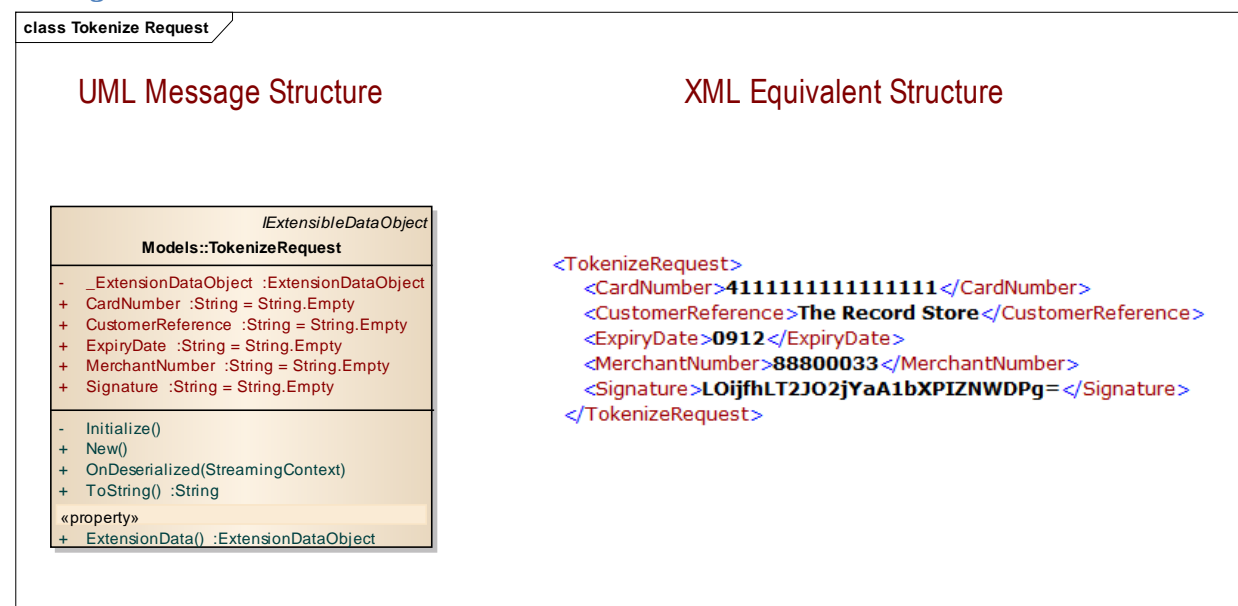
The Tokenize Operation takes a Card Number (PAN) and returns a Tokenized Card Number. See section [Tokenization using the Authorize \(and Authorize3DS\) Operation](#) for more information on Tokenization and formats/usage.

While you can Tokenize using both the Authorize or Authorize3DS methods, some merchant may want to convert their stored Card Numbers “en-masse” during a “PCI Compliance simplification exercise”. This method can be used to do this.

Once you create a Token, you can use this in place of the Card Number and Expiry Date in all future transactions for the same Cardholder. If you wish to de-scope parts of your system for PCI compliance, you should not store the original Card Number and discard it as soon as you have Tokenized it.

TokenizeRequest

Message Structure



Detailed Field Descriptions

Field Name	Format	Presence	Value
CardNumber	N(19)	R	Card Number to be Tokenized
Customer Reference	AN(255)	R	Your internal Customer ID or Name. Should be unique per customer (Cardholder).
ExpiryDate	N(4)	R	Expiry Date (MMYY) of Card
Merchant Number	N(8)	R	Your FAC ID provided by FAC
Signature	AN(28)	R	See Signature Creation and Verification

TokenizeResponse

The Tokenize Operation returns a TokenizeResponse object. Tokenize operations can fail for numerous reasons. Inspect the Success field and the contents of ErrorMsg for information on failed Tokenization attempts.

Message Structure

class Tokenize Response

UML Message Structure

<i>IExtensibleDataObject</i>	
Models::TokenizeResponse	
-	_ExtensionDataObject :ExtensionDataObject
+	ErrorMsg :String = String.Empty
+	Success :Boolean = False
+	Token :String = String.Empty
<hr/>	
-	Initialize()
+	New()
+	OnDeserialized(StreamingContext)
+	ToString() :String
<hr/>	
«property»	
+	ExtensionData() :ExtensionDataObject

XML Equivalent Structure

```
<TokenizeResponse>
  <ErrorMsg/>
  <Success>true</Success>
  <Token>411111_000011111</Token>
</TokenizeResponse>
```

Detailed Field Descriptions

Field Name	Format	Presence	Value
ErrorMsg	AN(255)	O	Will contain any error information
Success	Boolean	R	Either True or False
Token	AN(19)	R	The Tokenized Card Number

The DeTokenize Operation

The DeTokenize Operation takes a Tokenized Card Number (TokenPAN) and returns a Card Number (PAN). See section [Tokenization using the Authorize](#) (and Authorize3DS) Operation for more information on Tokenization and formats/usage.

While you can Tokenize and De-Tokenize using both the Authorize or Authorize3DS methods, you may want to just use the Tokenize/Detokenize functionality of the Gateway without using the financial transactions.

DeTokenizeRequest

Message Structure

class DetokenizeRequest	
UML Message Structure	XML Equivalent
<pre> classDiagram class DetokenizeRequest { <i>IExtensibleDataObject + _ExtensionDataObject :ExtensionDataObject + MerchantNumber :String = String.Empty + Signature :String = String.Empty + TokenPAN :String = String.Empty - Initialize() + New() + OnDeserialized(StreamingContext) + ToString() :String «property» + ExtensionData() :ExtensionDataObject </pre>	<pre> <DeTokenizeRequest> <MerchantNumber>88800033</MerchantNumber> <Signature>m%2f%2fjB6Dpgl3OyXeA%2fthFQNeIXb0%3d</Signature> <TokenPAN>411111_000471111</TokenPAN> </DeTokenizeRequest> </pre>

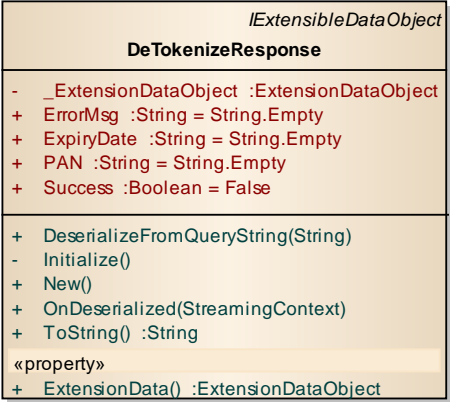
Detailed Field Descriptions

Field Name	Format	Presence	Value
TokenPAN	N(19)	R	Tokenized PAN to be De-Tokenized
Merchant Number	N(8)	R	Your FAC ID provided by FAC
Signature	AN(28)	R	See Signature Creation and Verification

DeTokenizeResponse

The DeTokenize Operation returns a DeTokenizeResponse object. DeTokenize operations can fail for numerous reasons. Inspect the Success field and the contents of ErrorMsg for information on failed DeTokenization attempts.

Message Structure

class DeTokenizeResponse	
UML Message Structure	XML Equivalent
 <pre> classDiagram class DeTokenizeResponse { <i>IExtensibleDataObject - _ExtensionDataObject :ExtensionDataObject + ErrorMsg :String = String.Empty + ExpiryDate :String = String.Empty + PAN :String = String.Empty + Success :Boolean = False + DeserializeFromQueryString(String) - Initialize() + New() + OnDeserialized(StreamingContext) + ToString() :String <<property>> + ExtensionData() :ExtensionDataObject </pre>	<pre> <DeTokenizeResponse> <ExtensionData /> <ErrorMsg /> <ExpiryDate>1213</ExpiryDate> <PAN>4111111111111111</PAN> <Success>true</Success> </DeTokenizeResponse> </pre>

Detailed Field Descriptions

Field Name	Format	Presence	Value
ErrorMsg	AN(255)	O	Will contain an error information
Success	Boolean	R	Either True or False
PAN	AN(19)	R	The Untokenized Card Number
ExpiryDate	N(4)	R	The Card Expiry Month/Year (MMYY)

The UpdateToken Operation

The UpdateToken operation allows the stored tokens attributes to be updated periodically so that details such as the expiry date and the customer reference can be maintained.

UpdateTokenRequest

Message Structure

UML Message Structure	XML Equivalent
<pre> class UpdateTokenRequest <!-- IExtensibleDataObject --> UpdateTokenRequest - _ExtensionDataObject :ExtensionDataObject + CustomerReference :String = String.Empty + ExpiryDate :String = String.Empty + MerchantNumber :String = String.Empty + Signature :String = String.Empty + TokenPAN :String = String.Empty - Initialize() + New() + OnDeserialized(StreamingContext) + ToString() :String «property» + ExtensionData() :ExtensionDataObject </pre>	<pre> <UpdateTokenRequest> <CustomerReference>Mr Jim Smith</CustomerReference> <ExpiryDate>1217</ExpiryDate> <MerchantNumber>88800033</MerchantNumber> <Signature>m%2f%2fB6Dpgl3OyXeA%2fthFQNeIXb0%3d</Signature> <TokenPAN>411111_000471111</TokenPAN> </UpdateTokenRequest> </pre>

Detailed Field Descriptions

Field Name	Format	Presence	Value
TokenPAN	N(19)	R	Tokenized PAN associated with attributes to be updated
Merchant Number	N(8)	R	Your FAC ID provided by FAC.
Signature	AN(28)	R	See Signature Creation and Verification
Customer Reference	AN(20)	R	The existing or new Customer Reference. Do not leave blank.
ExpiryDate	N(4)	R	The existing or new value of the Expiry Date - Month/Year (MMYY). Do not leave Blank.

UpdateTokenResponse

The UpdateToken Response returns the Success or Failure of the Update operation and confirms the Tokenized PAN updated.

Message Structure

class UpdateTokenResponse	
UML Message Structure	XML Equivalent
<pre> classDiagram class UpdateTokenResponse { <i>IExtensibleDataObject +_ExtensionDataObject :ExtensionDataObject +ErrorMsg :String = String.Empty +Success :Boolean = False +TokenPAN :String = String.Empty -Initialize() +New() +OnDeserialized(StreamingContext) +ToString() :String <<property>> +ExtensionData() :ExtensionDataObject </pre>	<pre> <UpdateTokenResponse> <ExtensionData /> <ErrorMsg /> <Success>true</Success> <TokenPAN>411111_000471111</TokenPAN> </UpdateTokenResponse> </pre>

Detailed Field Descriptions

Field Name	Format	Presence	Value
ErrorMsg	AN(255)	O	Will contain an error information if Success = False
Success	Boolean	R	Either True or False
TokenPAN	AN(19)	R	The Tokenized Card Number whose attributes were updated

The ExpiringCreditCards Operation

The ExpiringCreditCards operation selects any cards near or at the Expiry month so that the merchant can run updates on the cards once they have the new expiry date from the customer.

ExpiringCreditCardsRequest

Message Structure

UML Message Structure	XML Equivalent
<pre> classDiagram class ExpiringCreditCardsRequest { <i>IExtensibleDataObject - _ExtensionDataObject :ExtensionDataObject + ExpiryDate :String + MerchantId :String + Signature :String = String.Empty - Initialize() + New() + OnDeserialized(StreamingContext) + ToString() :String <<property>> + ExtensionData() :ExtensionDataObject </pre>	<pre> <ExpiringCreditCardsRequest> <ExpiryDate>0215</ExpiryDate> <MerchantId>88800033</MerchantId> <Signature>m%2f%2fb6Dpgl3OyXeA%2fthFQNeIXb0%3d</Signature> </ExpiringCreditCardsRequest> </pre>

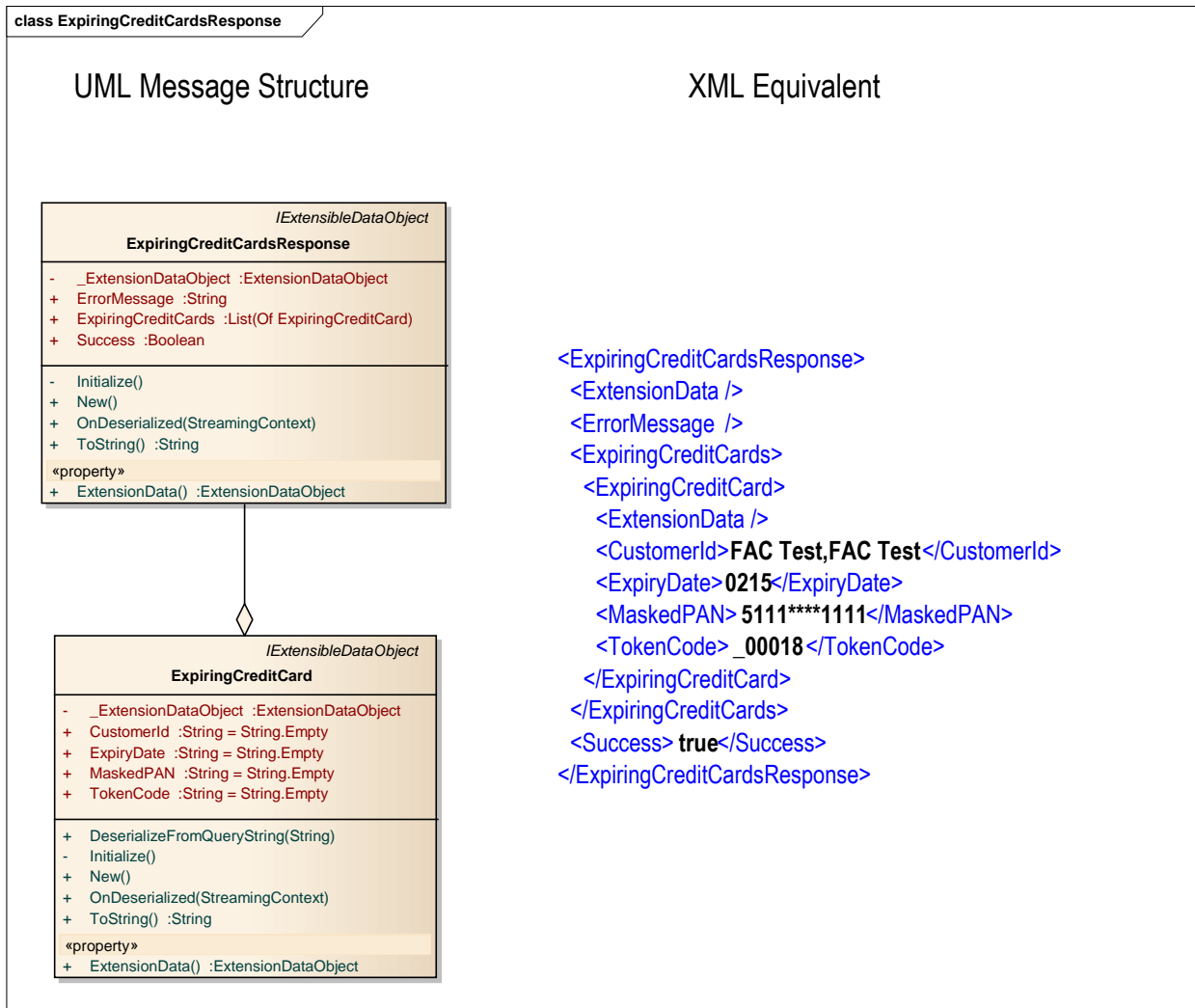
Detailed Field Descriptions

Field Name	Format	Presence	Value
MerchantId	N(8)	R	Your FAC ID provided by FAC.
Signature	AN(28)	R	See Signature Creation and Verification
ExpiryDate	N(4)	R	Month and Year of Expiry in format MMY

ExpiringCreditCardsResponse

The ExpiringCreditCardsResponse contains a list of Cards near or at Expiry date and the success or not of the operation.

Message Structure



Detailed Field Descriptions

Field Name	Format	Presence	Value
ErrorMsg	AN(255)	O	Will contain an error information if Success = False
Success	Boolean	R	Either True or False
ExpiringCreditCards	List	O	Will return a list of ExpiringCreditCards if any found.

Recurring Payments

Recurring payments are scheduled payments which are set to automatically charge a card at a defined regular interval. Check with your acquiring bank for any updates to their policies, card association regulations or best practices for management of recurring transactions.

The first transaction is used as a template for the subsequent transactions on this particular card, and as a result, it determines the amount to charge the cardholder over the course of the recurring cycle. At present, you cannot change the amount once the execution date has been established.

There are four fields which are required by merchants that wish to provide recurring payment to their customers (See [AuthorizeRequest](#), RecurringDetails section):

- **ExecutionDate** – This determines when to execute the next authorization after the immediate initial authorization. Example Date format is “20130715” (YYYYMMDD).
- **IsRecurring** – True or False.
- **Frequency** – Daily, Weekly, Fortnight (Every 2 Weeks), Monthly, Bi-Monthly (Every Other Month), Quarterly, Yearly.
- **NumberOfRecurrences** – How many times in total to charge the card (whether this includes the initial immediate authorization or not depends on the ExecutionDate – see the following note).

It is important to note:

- If the ExecutionDate is set to the current (today’s) date, then the immediate initial authorization is captured and becomes the first recurring payment in the series.
- If the ExecutionDate is set to any date in the future, the immediate initial authorization will not be captured and should not be manually captured, as it serves simply as an authorization on the card. The first recurring payment in the series will then take place on the set ExecutionDate in the future.
- An ExecutionDate prior to the current date (any past date) will result in a “ExecutionDate format is invalid” failed response.
- Any declines that occur after the recurring cycle has started will be resubmitted daily until they are approved.

Basically, there are two scenarios that can happen with recurring payments:

Scenario 1 - ExecutionDate set to current date (same day as transaction)

- 1) The original recurring transaction request (sent by you) that has the ExecutionDate is set to the same day the transaction is received, is taken as the first recurring payment and will be captured.
- 2) Subsequent payments will be processed at the time interval defined from the ExecutionDate (date of initial transaction) forward.

OR

Scenario 2 - ExecutionDate set in the future

- 1) The original recurring transaction request (sent by you) is an authorization-only transaction and will not be captured and should not be manually captured.
- 2) The first captured recurring payment will be done on the specified ExecutionDate and subsequently every time interval specified.

While the Recurring interface is in place, it may be dependent on processor so please inform FAC if you wish to start using Recurring transactions.

CVV2 and Recurring

The Card Verification Value (CVV2) (Card Verification Code (CVC2) for MasterCard), is the 3-digit code found on the back of all Visa and MasterCard credit cards (4-digit for AMEX). It is listed as a required parameter for the Authorize and Authorize3DS web methods above as it is mandated for all initial transactions processed by a merchant.

For merchants who are initiating/managing their own recurring transactions (not using FAC's recurring service), you must still provide the CVV2/CVC2 with the initial payment, but you cannot store the CVV2/CVC2 for future payments. The CVV2/CVC2 parameter is therefore not required on subsequent transactions in this case, but is still highly recommended to provide it whenever possible.

3DS and Recurring

3DS does not support Recurring Transactions at this time.

Canceling a Recurring Cycle

Please see the [Transaction Modification Operation](#) – Cancel Recurring.

Special Recurring Response Codes

If you receive any of the following responsecodes for a recurring trancto the corresponding actions should be taken as mandated by Visa:

R0 : Consider the transaction declined and cancel the recurring payment transaction. No further recurring payments should be sent for this transaction

R1 : Consider the transaction declined and cancel all the recurring payment transaction for that card and that merchant. No further recurring payments should be sent for this card from this merchant.

R3 : Consider the transaction declined and cancel all the recurring payment transaction for that card from all merchants. No further recurring payments should be sent for this card from this acquirer ID.

Recurring Notifications

Merchants are responsible for regularly managing and verifying their transaction activity including recurring transactions. FAC's notification service has been designed to be used in conjunction with a merchant's regular transaction monitoring processes/procedures. This service offers merchants additional real-time transaction status data after a recurring transaction is processed. As a recurring transaction is processed notification data is passed back to a response URL which is hosted on the merchant's side.

To implement this, a merchant will require developing and hosting a URL to which can handle a generic POST request and read XML data. The merchant will need to provide FAC with the URL(s) and be required to notify FAC of any future changes of the URL(s).

The URL(s) **MUST** be https (not plain http) and support the following fields:

Detailed Field Descriptions

Field Name	Format	Value
ResponseCode	N(1)	See Response Code and Reason Code Responses for possible values for this parameter.
ReferenceNo	N(12)	The unique reference number given this transaction by FAC's system
AuthId	AN(6)	Authorization code of successful transaction
RecurringAdviceIndicator	AN(100)	Descriptive text advising on the status of the recurring transaction (Successful/Active, Declined or Failed): <ul style="list-style-type: none"> • "Recurring Transaction Active" <ul style="list-style-type: none"> ○ Transaction was successful ○ Recurring transaction is active • "Recurring Transaction declined" <ul style="list-style-type: none"> ○ Transaction was declined ○ System will retry daily up-to the default number of tries (5) unless otherwise set by FAC • "Recurring Transaction Blocked as Expiry Date Exceeded." <ul style="list-style-type: none"> ○ Recurring transaction was rejected as the credit card expiry has passed ○ Verify the status of the recurring transaction to confirm disabled ○ A new recurring schedule will have to be submitted with updated card/card details or an update to Tokenized Card Expiry (if Tokenization is used) • "Recurring Transaction declined and deactivated - reached maximum retries."

		<ul style="list-style-type: none"> ○ A recurring transaction had declined and was reattempted for the default number of tried (5) or as set by FAC ○ Recurring cycle has been deactivated. Merchant should validate the status via FAC portal ○ A new recurring schedule will have to be submitted with updated card/card details or an update to Tokenized Card Expiry (if Tokenization is used) ● “Recurring Transaction Raised Error. Check Merchant Admin for Status.” <ul style="list-style-type: none"> ○ Error - Recurring Transaction Failed ○ Merchant to verify the transaction via FAC portal and action accordingly ○ Contact FAC support if assistance is required
RecurringPeriod	N(2)	The numeric value of the recurring cycle period. This increments by one as the recurring period is increased.
CardExpiring	Boolean	True or False
OriginalResponseCode	AN(3)	See Authorization Original Response Codes for possible values for this parameter.
OrderId	AN(150)	Order IDs are generated by the merchant and become a fixed order ID for all recurring orders tied to the original transaction. Every subsequent recurring transaction will have the same OrderID as a prefix but have the recurrence number (period) appended to the end with the format “-###” Example Order ID : TestTrxn-2, TestTrxn-3, etc.
MerchantId	N(15)	The value passed in MerchantId in the AuthorizeRequest
AttemptNumber	N(2)	Number of attempts on a transaction. A recurring transaction can be reattempted up to 5 times as the default however could differ if set differently at the request of a merchant
ReasonCode	N(4)	See Response Code and Reason Code Responses for possible values for this parameter.
ReasonCodeDesc	AN(100)	See Response Code and Reason Code Responses for possible values for this parameter.

The data sent to these URLs is in plain text XML format as follows:

```
<AuthorizationResponseData xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <ResponseCode>1</ResponseCode>
  <ReferenceNo>315818511716</ReferenceNo>
  <AuthId>123456</AuthId>
  <RecurringAdviceIndicator>Recurring Transaction Active</RecurringAdviceIndicator>
  <RecurringPeriod>2</RecurringPeriod>
```

```
<CardExpiring>false</CardExpiring>  
<OriginalResponseCode>00</OriginalResponseCode>  
<OrderId>FACTF635062161436423349174-2</OrderId>  
<MerchantId>88800033</MerchantId>  
<AttemptNumber>1</AttemptNumber>  
<ReasonCode>1</ReasonCode>  
<ReasonCodeDesc>Transaction is approved.</ReasonCodeDesc>  
</AuthorizationResponseData>
```

FACPG2 API - Sample Code

Please use this section as a guide only. It is not possible to show coding techniques for all available platforms, so only the most popular will be shown here. For a comprehensive list of SOAP Toolkit implementations see:

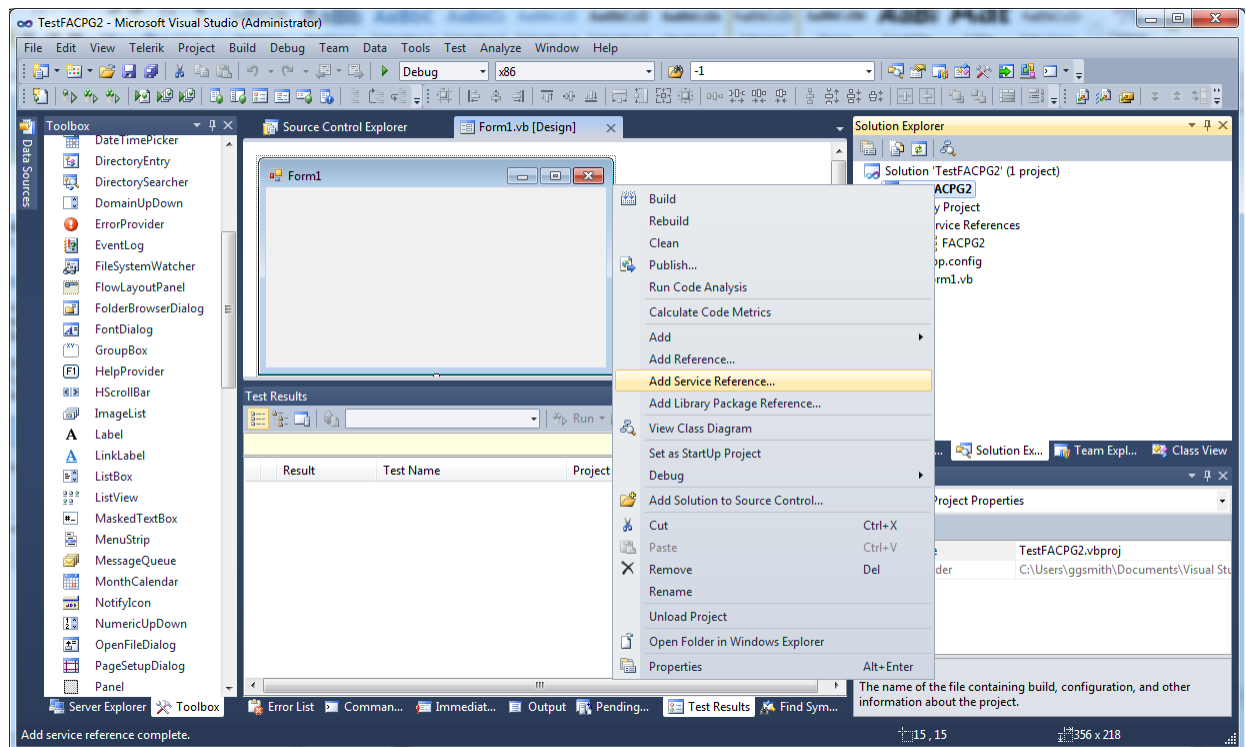
<http://www.soapware.org/directory/4/implementations>

Programming FACPG in Microsoft.NET VS2010

The key to accessing FAC Payment Gateway services in .NET is obtaining a Service or Web Reference. Either will do. A service reference is the WCF (Windows Communication Foundation) approach to creating a reference to either a Web Service or WCF Service.

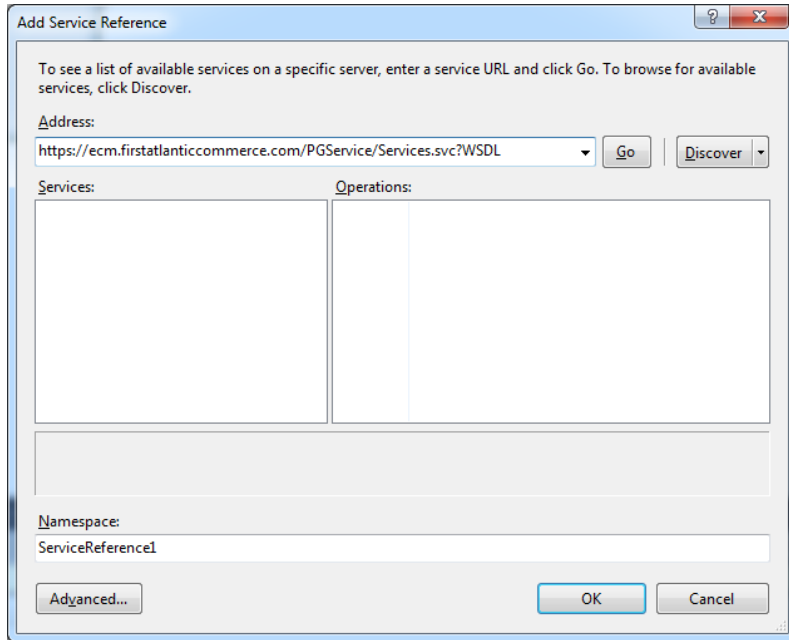
As FAC Payment Gateway v2.0 is implemented in WCF, using the SOAP binding, both a WCF Service reference and a Web Reference will work. However, if you are using .NET 4.0, a WCF Service reference should produce more efficient code and use faster serialization libraries under the hood.

Obtaining a WCF Service Reference

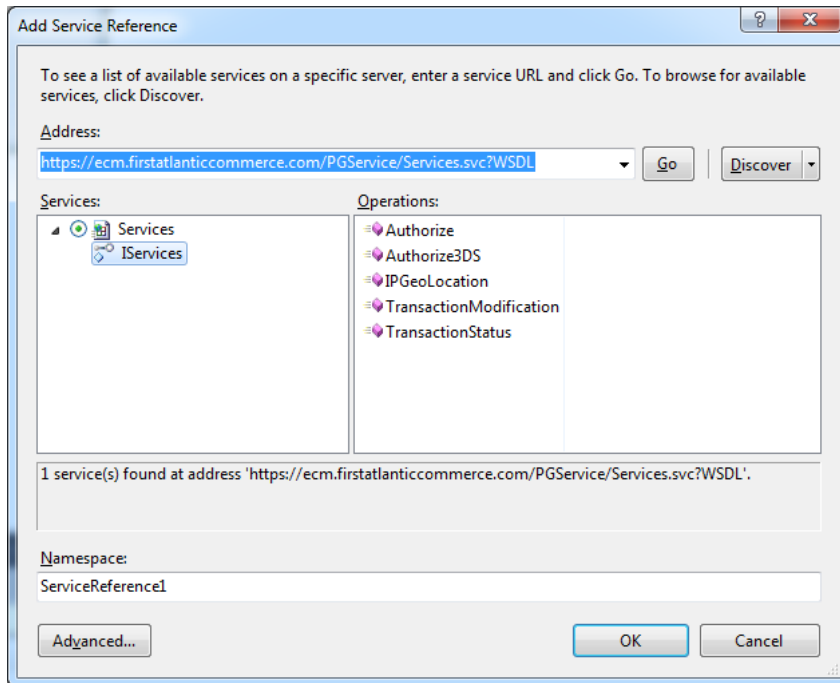


In your .NET project (or create a test project as above), in the context menu for the project (from the Solution Explorer), Select “Add Service Reference...”

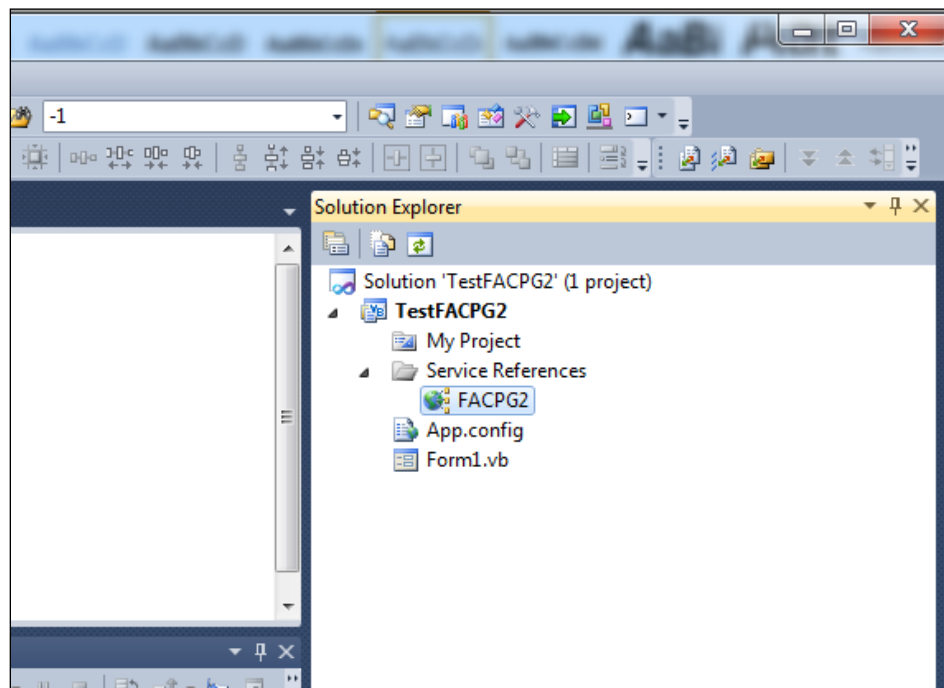
Enter the full URL and press “Go” in the Add Service Reference Dialog.



A Services node, named “Services” will appear in the left pane. Expand this to show the operations and make sure they are all available.

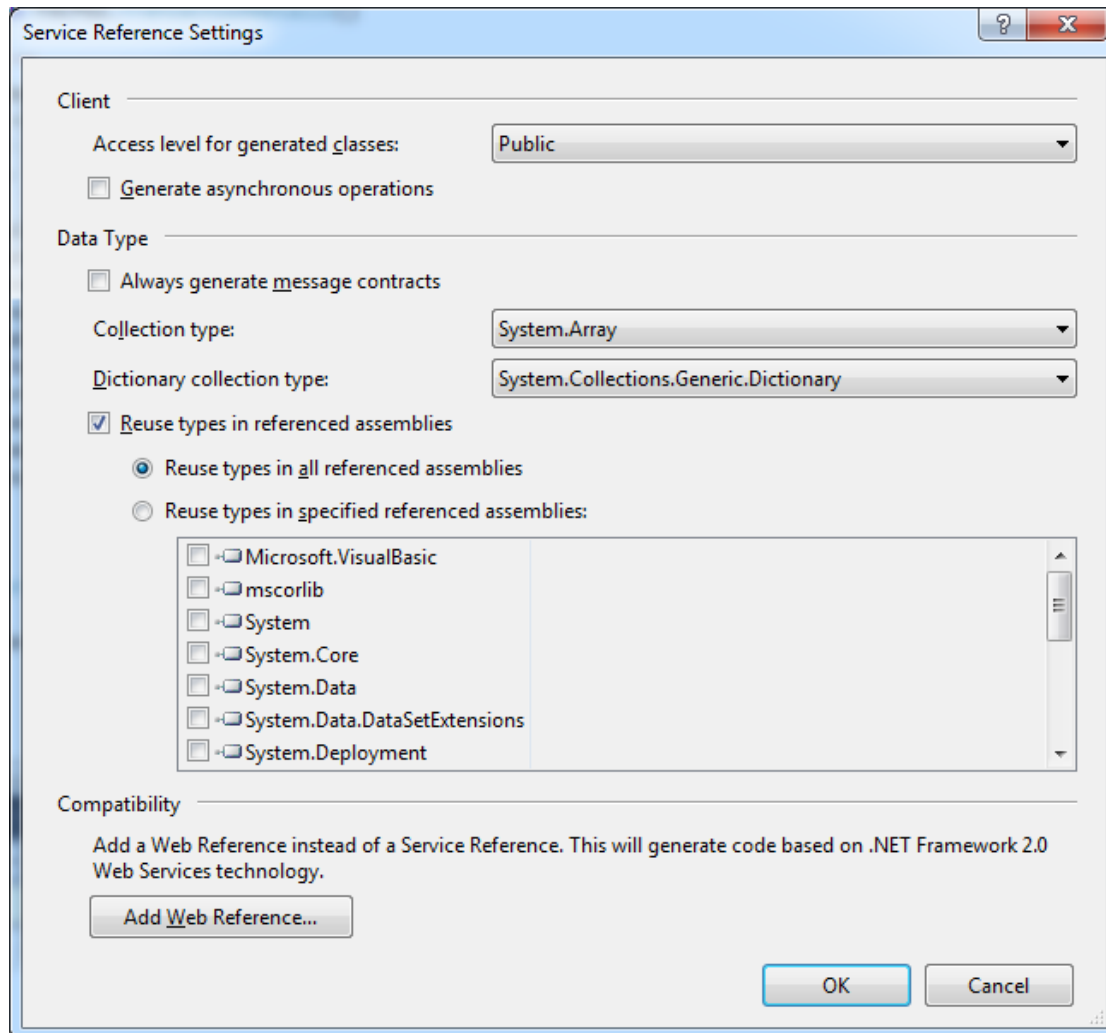


Rename the Namespace to something meaningful like “FACPG2” and press OK. A service reference will be added to your project.



Obtaining a SOAP Web Reference

This has the same starting point as for a Service Reference. Select the “Add Service Reference...” option from the project context menu but this time, click on the “Advanced...” button. You will see the Service Reference Settings dialog:



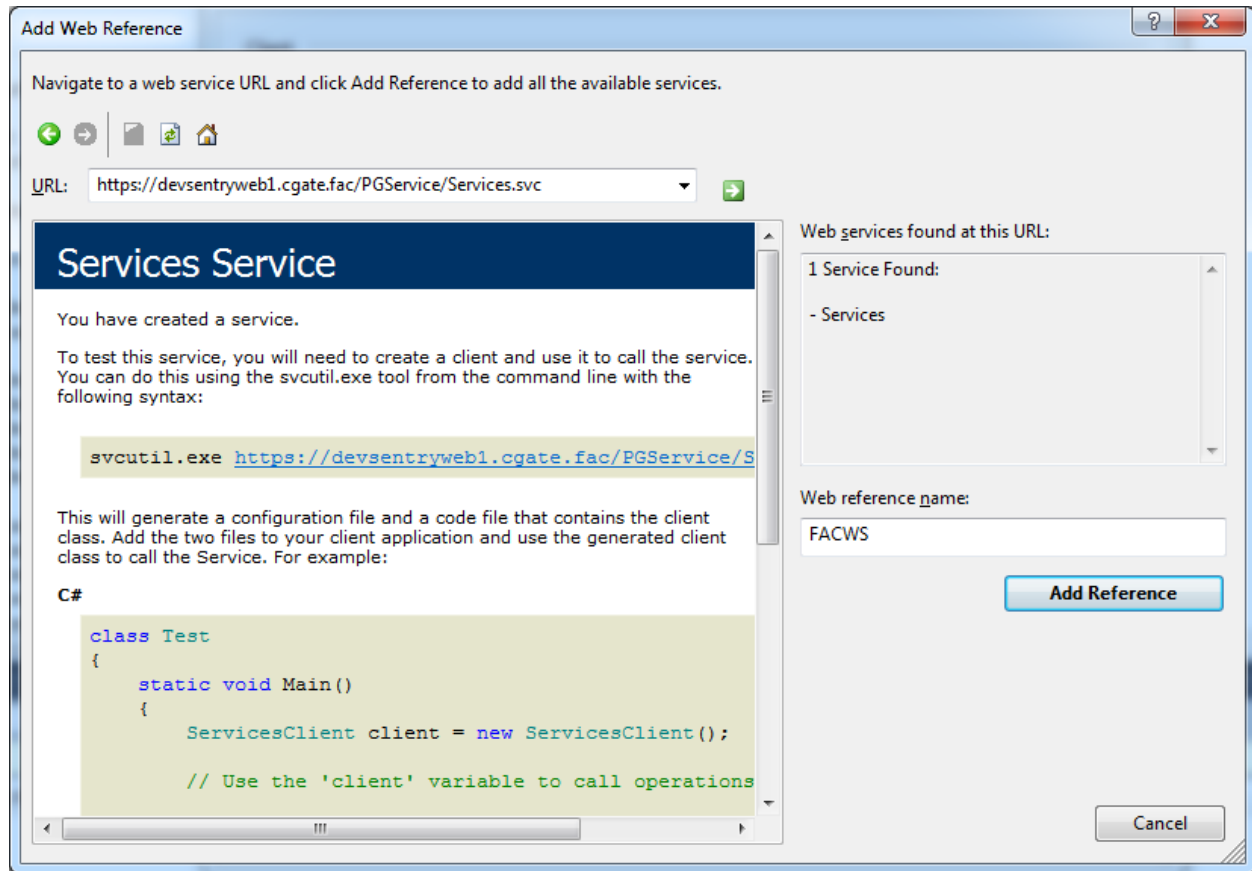
The image shows the "Service Reference Settings" dialog box. It has a title bar with a question mark and a close button. The dialog is divided into three sections: "Client", "Data Type", and "Compatibility".

- Client**:
 - Access level for generated classes:
 - ☐ Generate asynchronous operations
- Data Type**:
 - ☐ Always generate message contracts
 - Collection type:
 - Dictionary collection type:
 - ☒ Reuse types in referenced assemblies
 - ☒ Reuse types in all referenced assemblies
 - ☐ Reuse types in specified referenced assemblies:
 - ☐ Microsoft.VisualBasic
 - ☐ mscorlib
 - ☐ System
 - ☐ System.Core
 - ☐ System.Data
 - ☐ System.Data.DataSetExtensions
 - ☐ System.Deployment

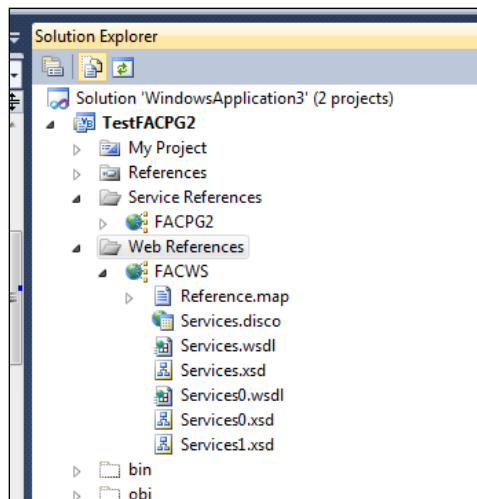
- Compatibility**:
- Add a Web Reference instead of a Service Reference. This will generate code based on .NET Framework 2.0 Web Services technology.
-

At the bottom right are "OK" and "Cancel" buttons.

Select the “Add Web Reference...” button at the bottom of the dialog (on the left). You will then see the “Add Web Reference” dialog. Enter the URL in the URL text Box and start a search for the Services.



You will get a confirmation that the “Services” service reference has been found. Enter a Web Reference Name and select “Add Reference”. A Web reference will be added under a “Web References” folder in the project (see Solution Explorer).



Programming Against the Service or Web Reference

Whether or not you have referenced the services as a WCF Service or SOAP Web Service, the coding is the same except that the class created for the client is called “Services” in SOAP Web Reference and “ServicesClient” when using a WCF Service Reference.

It is important to note that this code is a guide only and must not be used as production code. It is not production standard code and we offer no guarantees as to its correctness or performance in a production environment.

Once you have a reference, you can access all the operations described earlier in the document. Here is an example of coding an Authorize:

```
'
' The Client is the ServicesClient class created by the Service Reference inst
antiatiion.
' The server URL is determinded by entries in App.config or Web.config
'
Dim client As FACPG2.ServicesClient = New FACPG2.ServicesClient()
'
' Create a request object to send the data to the server
'
Dim request As FACPG2.AuthorizeRequest = New FACPG2.AuthorizeRequest()

With request
    '
    ' Message sections must be instantiated before use
    '
    .CardDetails = New FACPG2.CardDetails()
    .TransactionDetails = New FACPG2.TransactionDetails()
    '
    ' Make sure all mandatory fields are completed
    '
    .CardDetails.CardCVV2 = ""
    .CardDetails.CardExpiryDate = "0914"
    .CardDetails.CardNumber = "4111111111111111"
    .CardDetails.IssueNumber = ""
    .CardDetails.StartDate = ""

    .TransactionDetails.AcquirerId = "464748"
    .TransactionDetails.Amount = "000000000100"
    .TransactionDetails.Currency = "840"
    .TransactionDetails.CurrencyExponent = "2"
    .TransactionDetails.IPAddress = ""
    .TransactionDetails.MerchantId = "<Your FAC ID Here>"
    .TransactionDetails.OrderNumber = "FACPGTEST" + DateTime.Now.Ticks.ToStrin
g("000000000000")
    ' Compute Hash from required fields
    .TransactionDetails.Signature = Helper.ComputeHash("<Your Processing
Password Here>",
        .TransactionDetails.MerchantId,
        .TransactionDetails.AcquirerId,
        .TransactionDetails.OrderNumber,
```

```
.TransactionDetails.Amount,  
.TransactionDetails.Currency)  
.TransactionDetails.SignatureMethod = "SHA1"  
.TransactionDetails.TransactionCode = "0"  
End With  
  
' Then it's a simple method call to get an Authorization  
Dim response As FACPG2.AuthorizeResponse = client.Authorize(request)  
'  
' Always good practice to close connected object  
'  
client.Close()
```

The minimum you need to initialize for an Authorization are the CardDetails and TransactionDetails message sections. These need to be instantiated first before assigning values to the fields. Obviously this is using test data. It fabricates a Merchant Order number from a date/time and uses test card numbers. The ComputeHash function is a utility method written in-house to create the hashed signature for a message. You will need to create your own “ComputeHash” or similar function to achieve the same.

Programming FACPG in Java

Java is a platform independent programming and is free to use. This has made it a popular choice for web site implementers with the staff who have the expertise to use a fully object orientated, enterprise grade system.

HTTPS and Java

Unlike other systems such as .NET, Java requires trusted servers to have their certificate installed locally to valid the server connections. This requires downloading the FAC server certificate and installing it in your local system.

To make this job easier, use the Portecle utility to do this. Information and downloads are available from:

<http://portecle.sourceforge.net/>

Once you have the Certificate installed in your cert store, you will be able to connect to the FAC server (you will need to repeat the process when you set-up your system to use our production server as opposed to the staging server).

Obtaining a Web Reference in Java

As there are many Java IDE, we will not explain here how to use each of them. Most IDE have the ability to add a Web Reference in a similar manner to Microsoft’s Visual Studio.

However, this is not always as straightforward as it could be. Some Java IDE do not create the proxy classes correctly for FAC Payment Gateway operations as these contain complex data contracts.

However, if your IDE gives you an error when trying to create the service proxy from the WSDL url, you can always use the command line tool “wsimport”. We have tested this at FAC and it works well.

Here’s how:

Create a Binding file

Create a file called Binding.xml and paste in the following XML

```
<jaxb:bindings version="2.0"
  xmlns:jaxb="http://java.sun.com/xml/ns/jaxb"
  xmlns:xjc="http://java.sun.com/xml/ns/jaxb/xjc"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <jaxb:bindings
    schemaLocation="https://ecm.firstatlanticcommerce.com/PGService/Services.svc?xsd=xsd2">
    <jaxb:bindings node="/xs:schema">
      <jaxb:globalBindings>
        <xjc:generateElementProperty>>false</xjc:generateElementProperty>
      </jaxb:globalBindings>
    </jaxb:bindings>
  </jaxb:bindings>
</jaxb:bindings>
```

Execute the WSIMPORT utility

Ensure your path has an entry for your Java SDK bin directory

For information in the use of wsimport, execute wsimport without any parameters.

```
Administrator: C:\Windows\system32\cmd.exe
C:\Users\ggsmith\Documents\JavaClient>wsimport
Missing WSDL_URI

Usage: wsimport [options] <WSDL_URI>

where [options] include:
  -b <path>                specify jaxws/jaxb binding files or additional schemas
                           (Each <path> must have its own -b)
  -B<jaxbOption>            Pass this option to JAXB schema compiler
  -catalog <file>           specify catalog file to resolve external entity references
                           supports TR9401, XCatalog, and OASIS XML Catalog format.
  -d <directory>           specify where to place generated output files
  -extension                allow vendor extensions - functionality not specified
                           by the specification. Use of extensions may
                           result in applications that are not portable or
                           may not interoperate with other implementations
  -help                    display help
  -httpproxy:<host>:<port>  specify a HTTP proxy server (port defaults to 8080)
  -keep                    keep generated files
  -p <pkg>                 specifies the target package
  -quiet                   suppress wsimport output
  -s <directory>           specify where to place generated source files
  -target <version>        generate code as per the given JAXWS spec version
                           e.g. 2.0 will generate compliant code for JAXWS 2.0 spec
  -verbose                output messages about what the compiler is doing
  -version                print version information
  -wsdllocation <location> @WebServiceClient.wsdlLocation value

Extensions:
  -XadditionalHeaders      map headers not bound to request or response message to
                           Java method parameters
  -Xauthfile              file to carry authorization information in the format
                           http://username:password@example.org/stock?wsdl
  -Xdebug                 print debug information
  -Xno-addressing-databinding enable binding of W3C EndpointReferenceType to Java
  -Xnocompile              do not compile generated Java files

Examples:
wsimport stock.wsdl -b stock.xml -b stock.xjb
wsimport -d generated http://example.org/stock?wsdl

C:\Users\ggsmith\Documents\JavaClient>
```

To create the proxy as java code (not bytecode), Execute the wsimport with the following parameters:

Wsimport -Xnocompile -b Binding.xml <https://ecm.firstatlanticcommerce.com/PGService/Services.svc?WSDL>

This is what you should see:

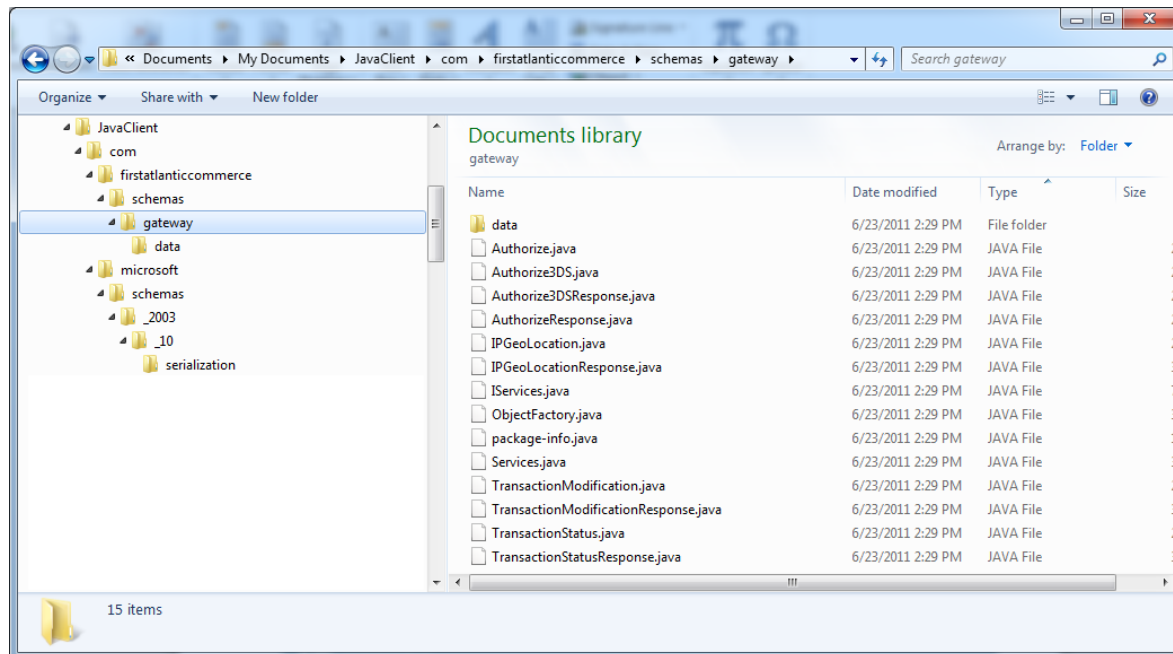
```
Administrator: C:\Windows\system32\cmd.exe
C:\Users\ggsmith\Documents\JavaClient>wsimport -Xnocompile -b Binding.xml https://ecm.firstatlanticcommerce.com/PGService/Services.svc?WSDL
parsing WSDL...

generating code...

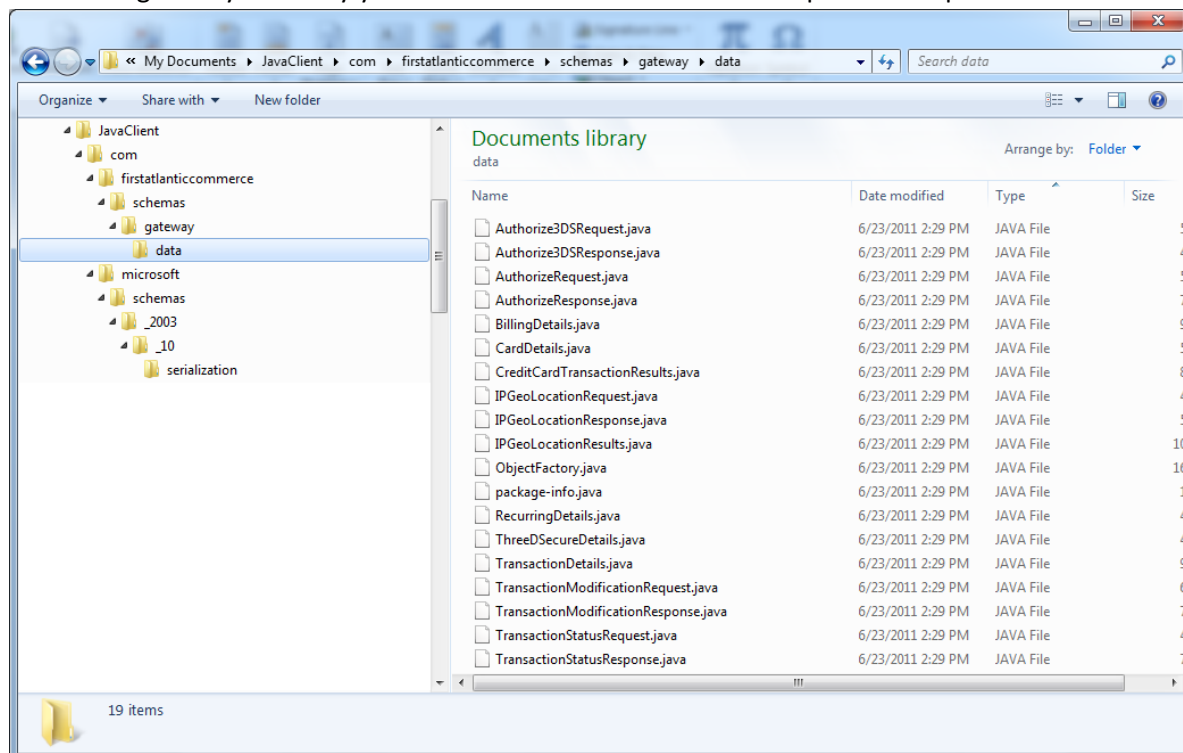
C:\Users\ggsmith\Documents\JavaClient>
```

This will create a sub-directory under the current one called “com” with all the code for the Web Reference.

Using File Explorer (in Windows), we can see what's been created:



Under the gateway directory you'll find the Service Interface and Operation implementations.

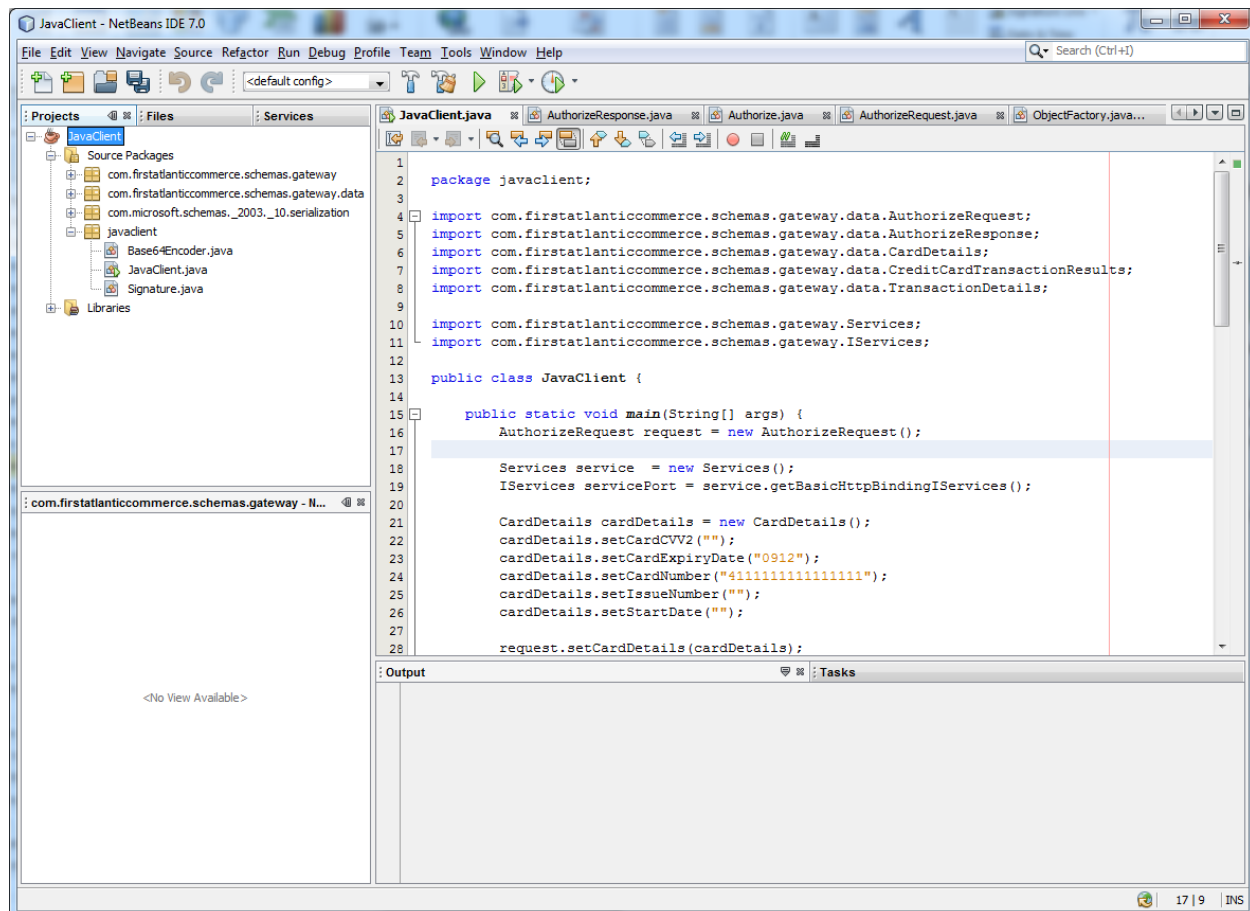


Under the data directory you'll find all the data classes for the request and response for each service Operation.

The Authorize Operation in Java

Now you have the code to access the FAC Payment Gateway, you can create a Java App to use it. The following example uses the free NetBeans IDE. It is important to note that this works just as well on Linux as Windows and the NetBeans IDE runs under both operating systems (as it's built with Java).

Create a project. In the following example we have created one called JavaClient. Add the generated "com" code directory for the Service reference to the "src" directory of the project.



You'll notice that as well as a class called JavaClient.java, there are two other classes (also note the packages and namespaces that have been added by adding the Web Service Reference code).

The class Base64Encoder.java is an open source class that will allow us to encode in Base 64 the hashed Signature required by Authorize and Authorize3DS Web Methods. The code of this class can be downloaded at:

<http://www.source-code.biz/base64coder/java/>

However, we have made a few changes so here is our amended code:

```
package javaclient;

// Copyright 2003-2010 Christian d'Heureuse, Inventec Informatik AG, Zurich, Switzerland
// www.source-code.biz, www.inventec.ch/chdh
//
// This module is multi-licensed and may be used under the terms
// of any of the following licenses:
//
// EPL, Eclipse Public License, V1.0 or later, http://www.eclipse.org/legal
// LGPL, GNU Lesser General Public License, V2.1 or later, http://www.gnu.org/licenses/lgpl.html
// GPL, GNU General Public License, V2 or later, http://www.gnu.org/licenses/gpl.html
// AL, Apache License, V2.0 or later, http://www.apache.org/licenses
// BSD, BSD License, http://www.opensource.org/licenses/bsd-license.php
//
// Please contact the author if you need another license.
// This module is provided "as is", without warranties of any kind.

/**
 * A Base64 encoder/decoder.
 *
 * <p>
 * This class is used to encode and decode data in Base64 format as described in RFC 1521.
 *
 * <p>
 * Project home page: <a href="http://www.source-code.biz/base64coder/java/">www.source-
 * code.biz/base64coder/java</a><br>
 * Author: Christian d'Heureuse, Inventec Informatik AG, Zurich, Switzerland<br>
 * Multi-licensed: EPL / LGPL / GPL / AL / BSD.
 */
public class Base64Encoder {

    // The line separator string of the operating system.
    private static final String systemLineSeparator = System.getProperty("line.separator");

    // Mapping table from 6-bit nibbles to Base64 characters.
    private static final char[] map1 = new char[64];
    static {
        int i=0;
        for (char c='A'; c<='Z'; c++) map1[i++] = c;
        for (char c='a'; c<='z'; c++) map1[i++] = c;
        for (char c='0'; c<='9'; c++) map1[i++] = c;
        map1[i++] = '+'; map1[i++] = '/'; }

    // Mapping table from Base64 characters to 6-bit nibbles.
    private static final byte[] map2 = new byte[128];
    static {
        for (int i=0; i<map2.length; i++) map2[i] = -1;
        for (int i=0; i<64; i++) map2[map1[i]] = (byte)i; }
}
```

```

/**
 * Encodes a string into Base64 format.
 * No blanks or line breaks are inserted.
 * @param s A String to be encoded.
 * @return A String containing the Base64 encoded data.
 */
public static String encodeString (String s) {
    return new String(encode(s.getBytes())); }

/**
 * Encodes a byte array into Base 64 format and breaks the output into lines of 76 characters.
 * This method is compatible with <code>sun.misc.BASE64Encoder.encodeBuffer(byte[])</code>.
 * @param in An array containing the data bytes to be encoded.
 * @return A String containing the Base64 encoded data, broken into lines.
 */
public static String encodeLines (byte[] in) {
    return encodeLines(in, 0, in.length, 76, systemLineSeparator); }

/**
 * Encodes a byte array into Base 64 format and breaks the output into lines.
 * @param in An array containing the data bytes to be encoded.
 * @param iOff Offset of the first byte in <code>in</code> to be processed.
 * @param iLen Number of bytes to be processed in <code>in</code>, starting at <code>iOff</code>.
 * @param lineLen Line length for the output data. Should be a multiple of 4.
 * @param lineSeparator The line separator to be used to separate the output lines.
 * @return A String containing the Base64 encoded data, broken into lines.
 */
public static String encodeLines (byte[] in, int iOff, int iLen, int lineLen, String
lineSeparator) {
    int blockLen = (lineLen*3) / 4;
    if (blockLen <= 0) throw new IllegalArgumentException();
    int lines = (iLen+blockLen-1) / blockLen;
    int bufLen = ((iLen+2)/3)*4 + lines*lineSeparator.length();
    StringBuilder buf = new StringBuilder(bufLen);
    int ip = 0;
    while (ip < iLen) {
        int l = Math.min(iLen-ip, blockLen);
        buf.append (encode(in, iOff+ip, l));
        buf.append (lineSeparator);
        ip += l; }
    return buf.toString(); }

/**
 * Encodes a byte array into Base64 format.
 * No blanks or line breaks are inserted in the output.
 * @param in An array containing the data bytes to be encoded.
 * @return string containing the Base64 encoded data.
 */
public static String encode (byte[] in) {
    return String.valueOf(encode(in, 0, in.length)); }

/**
 * Encodes a byte array into Base64 format.
 * No blanks or line breaks are inserted in the output.
 * @param in An array containing the data bytes to be encoded.
 * @param iLen Number of bytes to process in <code>in</code>.
 * @return A string containing the Base64 encoded data.
 */
public static String encode (byte[] in, int iLen) {
    return String.valueOf(encode(in, 0, iLen)); }

```

```

/**
 * Encodes a byte array into Base64 format.
 * No blanks or line breaks are inserted in the output.
 * @param in    An array containing the data bytes to be encoded.
 * @param iOff  Offset of the first byte in <code>in</code> to be processed.
 * @param iLen  Number of bytes to process in <code>in</code>, starting at <code>iOff</code>.
 * @return     A character array containing the Base64 encoded data.
 */
public static char[] encode (byte[] in, int iOff, int iLen) {
    int oDataLen = (iLen*4+2)/3;          // output length without padding
    int oLen = ((iLen+2)/3)*4;           // output length including padding
    char[] out = new char[oLen];
    int ip = iOff;
    int iEnd = iOff + iLen;
    int op = 0;
    while (ip < iEnd) {
        int i0 = in[ip++] & 0xff;
        int i1 = ip < iEnd ? in[ip++] & 0xff : 0;
        int i2 = ip < iEnd ? in[ip++] & 0xff : 0;
        int o0 = i0 >>> 2;
        int o1 = ((i0 & 3) << 4) | (i1 >>> 4);
        int o2 = ((i1 & 0xf) << 2) | (i2 >>> 6);
        int o3 = i2 & 0x3f;
        out[op++] = map1[o0];
        out[op++] = map1[o1];
        out[op] = op < oDataLen ? map1[o2] : '='; op++;
        out[op] = op < oDataLen ? map1[o3] : '='; op++; }
    return out; }

/**
 * Decodes a string from Base64 format.
 * No blanks or line breaks are allowed within the Base64 encoded input data.
 * @param s    A Base64 String to be decoded.
 * @return     A String containing the decoded data.
 * @throws     IllegalArgumentException If the input is not valid Base64 encoded data.
 */
public static String decodeString (String s) {
    return new String(decode(s)); }

/**
 * Decodes a byte array from Base64 format and ignores line separators, tabs and blanks.
 * CR, LF, Tab and Space characters are ignored in the input data.
 * This method is compatible with <code>sun.misc.BASE64Decoder.decodeBuffer(String)</code>.
 * @param s    A Base64 String to be decoded.
 * @return     An array containing the decoded data bytes.
 * @throws     IllegalArgumentException If the input is not valid Base64 encoded data.
 */
public static byte[] decodeLines (String s) {
    char[] buf = new char[s.length()];
    int p = 0;
    for (int ip = 0; ip < s.length(); ip++) {
        char c = s.charAt(ip);
        if (c != ' ' && c != '\r' && c != '\n' && c != '\t')
            buf[p++] = c; }
    return decode(buf, 0, p); }

/**
 * Decodes a byte array from Base64 format.
 * No blanks or line breaks are allowed within the Base64 encoded input data.
 * @param s    A Base64 String to be decoded.

```

```

* @return    An array containing the decoded data bytes.
* @throws    IllegalArgumentException If the input is not valid Base64 encoded data.
*/
public static byte[] decode (String s) {
    return decode(s.toCharArray()); }

/**
* Decodes a byte array from Base64 format.
* No blanks or line breaks are allowed within the Base64 encoded input data.
* @param in   A character array containing the Base64 encoded data.
* @return     An array containing the decoded data bytes.
* @throws     IllegalArgumentException If the input is not valid Base64 encoded data.
*/
public static byte[] decode (char[] in) {
    return decode(in, 0, in.length); }

/**
* Decodes a byte array from Base64 format.
* No blanks or line breaks are allowed within the Base64 encoded input data.
* @param in   A character array containing the Base64 encoded data.
* @param iOff Offset of the first character in <code>in</code> to be processed.
* @param iLen Number of characters to process in <code>in</code>, starting at <code>iOff</code>.
* @return     An array containing the decoded data bytes.
* @throws     IllegalArgumentException If the input is not valid Base64 encoded data.
*/
public static byte[] decode (char[] in, int iOff, int iLen) {
    if (iLen%4 != 0) throw new IllegalArgumentException ("Length of Base64 encoded input string is
not a multiple of 4.");
    while (iLen > 0 && in[iOff+iLen-1] == '=') iLen--;
    int oLen = (iLen*3) / 4;
    byte[] out = new byte[oLen];
    int ip = iOff;
    int iEnd = iOff + iLen;
    int op = 0;
    while (ip < iEnd) {
        int i0 = in[ip++];
        int i1 = in[ip++];
        int i2 = ip < iEnd ? in[ip++] : 'A';
        int i3 = ip < iEnd ? in[ip++] : 'A';
        if (i0 > 127 || i1 > 127 || i2 > 127 || i3 > 127)
            throw new IllegalArgumentException ("Illegal character in Base64 encoded data.");
        int b0 = map2[i0];
        int b1 = map2[i1];
        int b2 = map2[i2];
        int b3 = map2[i3];
        if (b0 < 0 || b1 < 0 || b2 < 0 || b3 < 0)
            throw new IllegalArgumentException ("Illegal character in Base64 encoded data.");
        int o0 = ( b0      <<2) | (b1>>>4);
        int o1 = ((b1 & 0xf)<<4) | (b2>>>2);
        int o2 = ((b2 & 3)<<6) |  b3;
        out[op++] = (byte)o0;
        if (op<oLen) out[op++] = (byte)o1;
        if (op<oLen) out[op++] = (byte)o2; }
    return out; }

} // end class Base64Coder

```

The Signature.java class is a utility class that helps us creates a signature for the Authorize methods. It uses the Base64Encoder class and the Java MessageDigest class to create the Signature.

```
package javaclient;

import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class Signature
{
    public static String Sign(String password, String merchantId,
                              String acquirerId, String orderNumber,
                              String amount, String currency)
        throws NoSuchAlgorithmException, UnsupportedEncodingException
    {
        String text = password.trim() + merchantId.trim() +
            acquirerId.trim() + orderNumber.trim() +
            amount.trim() + currency.trim();
        MessageDigest md;
        md = MessageDigest.getInstance("SHA-1");
        byte[] shalhash = new byte[40];
        md.update(text.getBytes("iso-8859-1"), 0, text.length());
        shalhash = md.digest();
        return Base64Encoder.encode(shalhash);
    }
}
```

Our `JavaClient.java` Class uses the Web Service reference code to access the remote FAC Payment Gateway operations. Here we are doing a basic Authorization.

```
package javaclient;

import com.firstatlanticcommerce.schemas.gateway.data.AuthorizeRequest;
import com.firstatlanticcommerce.schemas.gateway.data.AuthorizeResponse;
import com.firstatlanticcommerce.schemas.gateway.data.CardDetails;
import com.firstatlanticcommerce.schemas.gateway.data.CreditCardTransactionResults;
import com.firstatlanticcommerce.schemas.gateway.data.TransactionDetails;

import com.firstatlanticcommerce.schemas.gateway.Services;
import com.firstatlanticcommerce.schemas.gateway.IServices;

public class JavaClient {

    public static void main(String[] args) {
        AuthorizeRequest request = new AuthorizeRequest();

        Services service = new Services();
        IServices servicePort = service.getBasicHttpBindingIServices();

        CardDetails cardDetails = new CardDetails();
        cardDetails.setCardCVV2("");
        cardDetails.setCardExpiryDate("0914");
        cardDetails.setCardNumber("4111111111111111");
        cardDetails.setIssueNumber("");
        cardDetails.setStartDate("");

        request.setCardDetails(cardDetails);

        TransactionDetails details = new TransactionDetails();
        details.setAcquirerId("464748");
        details.setAmount("000000000100");
        details.setCurrency("840");
        details.setCurrencyExponent(2);
        details.setIPAddress("");
        details.setMerchantId("<your FAC ID>");
        details.setOrderNumber("FACPG2TEST" + System.currentTimeMillis());

        details.setSignatureMethod("SHA1");
        details.setTransactionCode(0);

        request.setTransactionDetails(details);

        String txSignature;

        try
        {

            // Lastly, set signature from data
            txSignature = Signature.Sign("<your password>",
                                      details.getMerchantId(),
                                      details.getAcquirerId(),
                                      details.getOrderNumber(),
                                      details.getAmount(),
                                      details.getCurrency());

        }
        catch (Exception ex)
```

```

    {
        System.out.println("Unable to sign as " + ex.getMessage());
        return;
    }

    details.setSignature(txSignature);

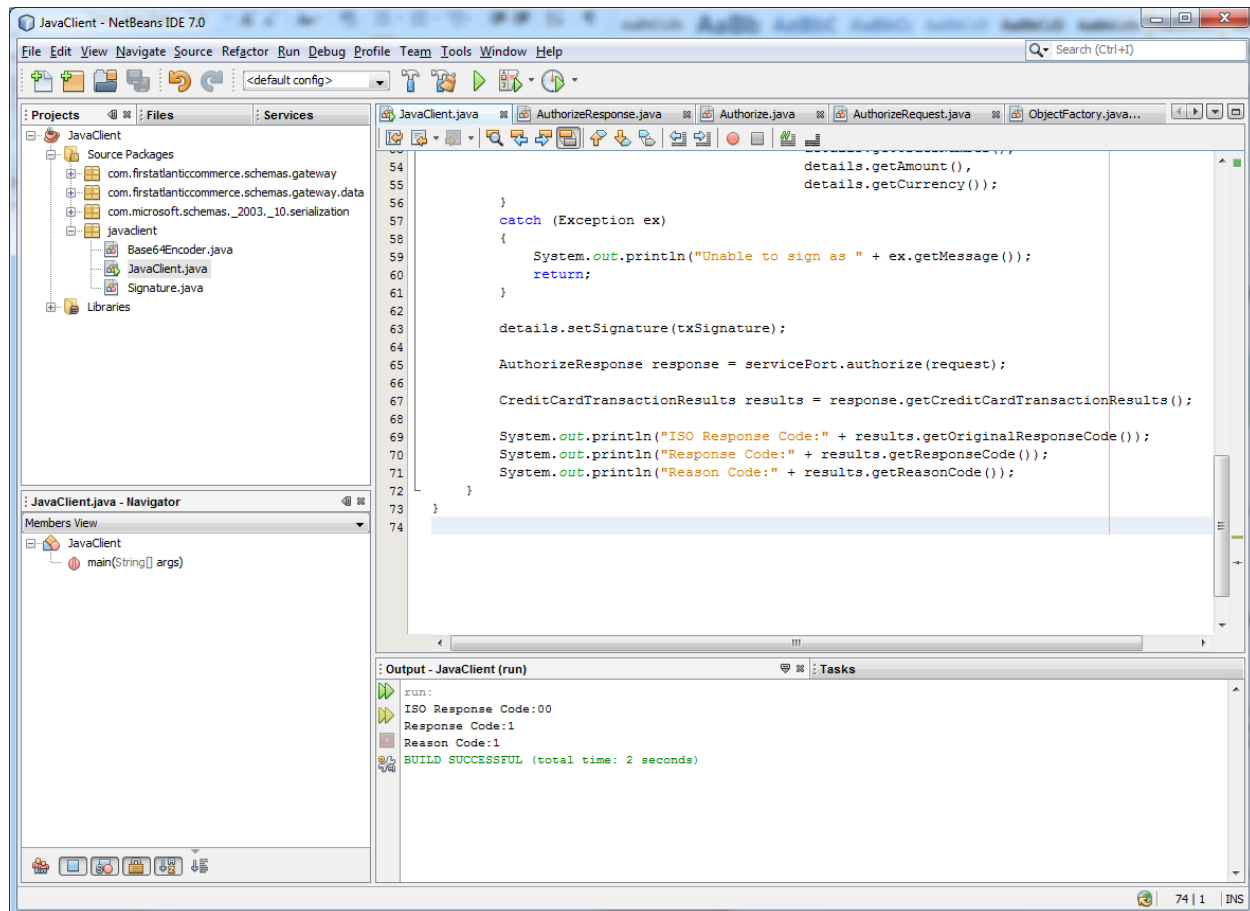
    AuthorizeResponse response = servicePort.authorize(request);

    CreditCardTransactionResults results = response.getCreditCardTransactionResults();

    System.out.println("ISO Response Code:" + results.getOriginalResponseCode());
    System.out.println("Response Code:" + results.getResponseCode());
    System.out.println("Reason Code:" + results.getReasonCode());
}
}

```

The code example outputs the response codes as verification of the Authorization.



Programming FAC Payment Gateway in PHP

PHP is an open source scripting language popular on open source web platforms as a development platform. The PHP language is used as a server scripting language and has many pluggable components.

We have included more samples here than any other platform as there is less help from an IDE/Help files and Linux/PHP is probably the most common Web Merchant platform due to the number of free tools and CMS/Web Cart applications available.

In the following examples, we have used the following set-up and components:

- Ubuntu Desktop
- Apache2 with OpenSSL installed
- PHP 5.3.5
- SoapClient (comes with PHP 5.3.5)
- Libcurl (also called cURL - comes with PHP 5.3.5)
- PEAR5 with XML_Serializer library installed

This document is not a tutorial on Linux so it is assumed that the developer knows how to install Linux components and has done this prior to testing these examples or using them as a template for their own code.

Using these components is not the only way to achieve the same results but these examples have been tested under this environment and are known to work.

SOAP Programming

The basic outline for programming in SOAP with PHP is:

- 1) Set-up options for the SoapClient
- 2) Create a SoapClient using options and WSDL url.
- 3) Create Request object using an Associative Array
- 4) Call Web Method via SoapClient proxy instance
- 5) Process Response

XML Programming

The basic outline for programming in XML with PHP is:

- 1) Load the Serializer and Unserializer code modules
- 2) Create request using an Associative Array
- 3) Serialize the request to XML using Serializer
- 4) Send XML as HTTPS POST using cURL library
- 5) Deserialize reponse XML as PHP Object
- 6) Process Response

As you can see, this is pretty straightforward and quite similar in either method. The XML is probably more efficient as it will require less network round-trips. In SOAP programming there may also be a way of caching the WSDL locally to save the WSDL enquiry every time (the SoapClient calls the server when creating the proxy client instance).

Note: Passwords and FAC IDs have been replaced and will need setting up before this code will run.

Authorize Examples

Authorize using SOAP

```
<?php

// Useful for generation of test Order numbers
function msTimeStamp()
{
    return (string)round(microtime(1) * 1000);
}

// How to sign a FAC Authorize message
function Sign($passwd, $facId, $acquirerId, $orderNumber, $amount, $currency)
{
    $stringtohash =
$passwd.$facId.$acquirerId.$orderNumber.$amount.$currency;
    $hash = sha1($stringtohash, true);
    $signature = base64_encode($hash);

    return $signature;
}

// Ensure you append the ?wsdl query string to the url
$wsdlurl =
'https://ecm.firstatlanticcommerce.com/PGService/Services.svc?wsdl';

// Set up client to use SOAP 1.1 and NO CACHE for WSDL. You can choose
between
// exceptions or status checking. Here we use status checking. Trace is for
Debug only
// Works better with MS Web Services where
// WSDL is split into several files. Will fetch all the WSDL up front.
$options = array(
    'location' =>
'https://ecm.firstatlanticcommerce.com/PGService/Services.svc',
    'soap_version'=>SOAP_1_1,
    'exceptions'=>0,
    'trace'=>1,
    'cache_wsdl'=>WSDL_CACHE_NONE
);

// WSDL Based calls use a proxy, so this is the best way
// to call FAC PG Operations.
$client = new SoapClient($wsdlurl , $options);

// This should not be in your code in plain text!
$password = '<your password>';
```

```
// Use your own FAC ID
$facId = '<your FAC ID>';
// Acquirer is always 464748
$acquirerId = '464748';
// orderNumber must be Unique per order. Put your own format here
$orderNumber = 'FACPGTEST' . msTimeStamp();
// 12 chars, always, no decimal place
$amount = '000000001199';
// 840 = USD, put your currency code here
$currency = '840';
$signature = Sign($password, $facId, $acquirerId, $orderNumber, $amount,
$currency);

// You only need to initialise the message sections you need. So for a basic
Auth
// only Credit Cards and Transaction details are required.

// Card Details. Arrays serialise to elements in XML/SOAP
$CardDetails = array('CardCVV2' => '',
                    'CardExpiryDate' => '0918',
                    'CardNumber' => '4111111111111111',
                    'IssueNumber' => '',
                    'StartDate' => '');

// Transaction Details.
$TransactionDetails = array('AcquirerId' => $acquirerId,
                            'Amount' => $amount,
                            'Currency' => $currency,
                            'CurrencyExponent' => 2,
                            'IPAddress' => '',
                            'MerchantId' => $facId,
                            'OrderNumber' =>
                                $orderNumber,
                            'Signature' => $signature,
                            'SignatureMethod' => 'SHA1',
                            'TransactionCode' => '0');

// The request data is named 'Request' for reasons that are not clear!
$AuthorizeRequest = array('Request' => array('CardDetails' => $CardDetails,
'TransactionDetails' => $TransactionDetails));
// For debug, to check the values are OK
var_dump($AuthorizeRequest);

// Call the Authorize through the Client
$result = $client->Authorize($AuthorizeRequest);

// Check for a fault
if ($client->fault) {
    echo '<h2>Fault</h2><pre>';
}
```

```

    print_r($result);
    echo '</pre>';
} else {
    // Check for errors
    $err = $client->error;
    if ($err) {
        // Display the error
        echo '<h2>Error</h2><pre>' . $err . '</pre>';
    } else {
        // Display the result
        echo '<h2>Result</h2><pre>';
        print_r($result);
        echo '</pre>';
    }
}

?>

```

Authorize using XML

```

<?php

require_once('XML/Serializer.php');
require_once('XML/Unserializer.php');

// Useful for generation of test Order numbers
function msTimeStamp()
{
    return (string)round(microtime(1) * 1000);
}

// How to sign a FAC Authorize message
function Sign($passwd, $facId, $acquirerId, $orderNumber, $amount, $currency)
{
    $stringtohash =
$passwd.$facId.$acquirerId.$orderNumber.$amount.$currency;
    $hash = sha1($stringtohash, true);
    $signature = base64_encode($hash);

    return $signature;
}

// XML Urls are named after the Operation in a Rest-ful manner
$url = 'https://ecm.firstatlanticcommerce.com/PGServiceXML/Authorize';

// This should not be in your code in plain text!
$password = '<your password>';
// Use your own FAC ID
$facId = '<your FAC ID>';

```

```
// Acquirer is always 464748
$acquirerId = '464748';
// Must be Unique per order. Put your own format here
$orderNumber = 'FACPGTEST' . msTimeStamp();
// 12 chars, always, no decimal place
$amount = '000000001199';
// 840 = USD, put your currency code here
$currency = '840';

$signature = Sign($password, $facId, $acquirerId, $orderNumber, $amount,
$currency);

// You only need to initialise the message sections you need. So for a basic
Auth
// only Credit Cards and Transaction details are required.

// Card Details. Arrays serialise to elements in XML/SOAP
$cardDetails = array('CardCVV2' => '',
                    'CardExpiryDate' => '0918',
                    'CardNumber' => '4111111111111111',
                    'IssueNumber' => '',
                    'StartDate' => '');

// Transaction Details.
$transactionDetails = array('Amount' => $amount,
                           'Currency' => $currency,
                           'CurrencyExponent' => 2,
                           'IPAddress' => '',
                           'MerchantId' => $facId,
                           'OrderNumber' =>
                           $orderNumber,
                           'Signature' => $signature,
                           'AcquirerId' => $acquirerId,
                           'SignatureMethod' => 'SHA1',
                           'TransactionCode' => '0');

// The request data is named 'Request' for reasons that are not clear!
$authorizeRequest = array('TransactionDetails' => $transactionDetails,
                          'CardDetails' => $cardDetails);

$options = array(
    "indent" => " ",
    "linebreak" => "\n",
    "typeHints" => false,
    "addDecl" => true,
    "encoding" => "UTF-8",
    "rootName" => "AuthorizeRequest",
    "defaultTagName" => "item",
```

```
"rootAttributes" => array("xmlns" =>
"http://schemas.firstatlanticcommerce.com/gateway/data")
);

$serializer = new XML_Serializer($options);

if ($serializer->serialize($AuthorizeRequest))
{
    $xmlRequest = $serializer->getSerializedData();

    $ch = curl_init($url);

    curl_setopt($ch, CURLOPT_MUTE, 1);
    curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 0);
    curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, 0);
    curl_setopt($ch, CURLOPT_POST, 1);
    curl_setopt($ch, CURLOPT_HTTPHEADER, array('Content-Type: text/xml'));
    curl_setopt($ch, CURLOPT_POSTFIELDS, $xmlRequest);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

    $response = curl_exec($ch);
    curl_close($ch);

    // Let's convert to an Object graph, easier to process
    $options = array('complexType' => 'object');

    $deserializer = new XML_Unserializer($options);
    // Pass in the response XML as a string
    $result = $deserializer->unserialize($response, false);
    // As with Serializing, we must call getUnserializedData afterwards
    $AuthorizeResponse = $deserializer->getUnserializedData();

    // Display the result objects
    echo '<h2>Result</h2><pre>';
    print_r($AuthorizeResponse);
    echo '</pre>';
}

?>
```

TransactionModification Examples

TransactionModification using SOAP

```
<?php
// Ensure you append the ?wsdl query string to the url
$wsdlurl =
'https://ecm.firstatlanticcommerce.com/PGService/Services.svc?wsdl';

// Set up client to use SOAP 1.1 and NO CACHE for WSDL. You can choose
between
// exceptions or status checking. Here we use status checking. Trace is for
Debug only
// Works better with MS Web Services where
// WSDL is split into several files. Will fetch all the WSDL up front.
$options = array(
    'location' =>
'https://ecm.firstatlanticcommerce.com/PGService/Services.svc',
    'soap_version'=>SOAP_1_1,
    'exceptions'=>0,
    'trace'=>1,
    'cache_wsdl'=>WSDL_CACHE_NONE
);

// WSDL Based URL generates a proxy Client. This is the best way
// to call FAC PG Operations.
$client = new SoapClient($wsdlurl , $options);

// This should not be in your code in plain text!
$password = '<your password>';
// Use your own FAC ID
$facId = '<your FAC ID>';
// Acquirer is always this
$acquirerId = '464748';
// Must be a previously Authorized Transaction Order Number
$orderNumber = 'FACPGTEST1307796480663';
// 12 chars, always, no decimal place
$amount = '000000001199';
// Modification Types. psuedo enum
final class ModificationTypes
{
    const Capture = 1;
    const Refund = 2;
    const Reversal = 3;
}

// Transaction Details.
$previousTransactionDetails = array('AcquirerId' => $acquirerId,
    'Amount' => $amount,
```



```

CurrencyExponent' => 2,
'MerchantId' => $facId,

'ModificationType' => ModificationTypes::Capture,
'OrderNumber' => $orderNumber,
'Password' => $password);

// The request data is named 'Request' and not after the
// message request class name for reasons that are not clear!
$TransactionModificationRequest = array('Request' =>
$PreviousTransactionDetails);

// For debug, to check the values are OK
var_dump($TransactionModificationRequest);

// Call the Authorize through the Client
$result = $client->TransactionModification($TransactionModificationRequest);

// Check for a fault
if ($client->fault) {
    echo '<h2>Fault</h2><pre>';
    print_r($result);
    echo '</pre>';
} else {
    // Check for errors
    $err = $client->error;
    if ($err) {
        // Display the error
        echo '<h2>Error</h2><pre>' . $err . '</pre>';
    } else {
        // Display the result
        echo '<h2>Result</h2><pre>';
        print_r($result);
        echo '</pre>';
    }
}
?>

```

TransactionModification using XML

```

<?php

require_once('XML/Serializer.php');
require_once('XML/Unserializer.php');

$url =
'https://ecm.firstatlanticcommerce.com/PGServiceXML/TransactionModification';

// This should not be in your code in plain text!

```

```
$password = '<your password>';
// Use your own FAC ID
$facId = '<your FAC ID>';
// Acquirer is always 464748
$acquirerId = '464748';
// Must be a previously Authorized Transaction Order Number
$orderNumber = 'FACPGTEST1307796480663';
// 12 chars, always, no decimal place
$amount = '000000001199';
// Modification Types. psuedo enum
final class ModificationTypes
{
    const Capture = 1;
    const Refund = 2;
    const Reversal = 3;
}

// Transaction Details.
$transactionModificationRequest = array('AcquirerId' => $acquirerId,
                                         'Amount' =>
$amount,
                                         'CurrencyExponent' => 2,
                                         'MerchantId' =>
$facId,
                                         'ModificationType' => ModificationTypes::Refund,
                                         'OrderNumber' =>
$orderNumber,
                                         'Password' =>
$password);

// Note the rootAttributes includes the FAC namespace. Do not use the
// namespace option as it will prefix the name, which is not right and
// will cause the call to fail
$options = array(
    "indent" => " ",
    "linebreak" => "\n",
    "typeHints" => false,
    "addDecl" => true,
    "encoding" => "UTF-8",
    "rootName" => "TransactionModificationRequest",
    "defaultTagName" => "item",
    "rootAttributes" => array("xmlns" =>
"http://schemas.firstatlanticcommerce.com/gateway/data")
);

$serializer = new XML_Serializer($options);
```

```
if ($serializer->serialize($TransactionModificationRequest))
{
    $xmlRequest = $serializer->getSerializedData();

    echo '<pre>';
    echo htmlspecialchars($xmlRequest);
    echo '</pre>';

    $ch = curl_init($url);

    curl_setopt($ch, CURLOPT_MUTE, 1);
    curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 0);
    curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, 0);
    curl_setopt($ch, CURLOPT_POST, 1);
    curl_setopt($ch, CURLOPT_HTTPHEADER, array('Content-Type: text/xml'));
    curl_setopt($ch, CURLOPT_POSTFIELDS, $xmlRequest);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

    $response = curl_exec($ch);
    curl_close($ch);

    // Let's convert to an Object graph, easier to process
    $options = array('complexType' => 'object');

    $deserializer = new XML_Unserializer($options);
    // Pass in the response XML as a string
    $result = $deserializer->unserialize($response, false);
    // As with Serializing, we must call getUnserializedData afterwards
    $TransactionModificationResponse = $deserializer->getUnserializedData();

    // Display the result objects
    echo '<h2>Result</h2><pre>';
    print_r($TransactionModificationResponse);
    echo '</pre>';
}

?>
```

TransactionStatus Examples

TransactionStatus using SOAP

```
<?php
// Ensure you append the ?wsdl query string to the url
$wsdlurl =
'https://ecm.firstatlanticcommerce.com/PGService/Services.svc?wsdl';

// Set up client to use SOAP 1.1 and NO CACHE for WSDL. You can choose
between
// exceptions or status checking. Here we use status checking. Trace is for
Debug only
// Works better with MS Web Services where
// WSDL is split into several files. Will fetch all the WSDL up front.
$options = array(
    'location' =>
'https://ecm.firstatlanticcommerce.com/PGService/Services.svc',
    'soap_version'=>SOAP_1_1,
    'exceptions'=>0,
    'trace'=>1,
    'cache_wsdl'=>WSDL_CACHE_NONE
);

// WSDL Based URL generates a proxy Client. This is the best way
// to call FAC PG Operations.
$client = new SoapClient($wsdlurl , $options);

// This should not be in your code in plain text!
$password = '<your password>';
// Use your own FAC ID
$facId = '<your FAC ID>';
// Acquirer is always 464748
$acquirerId = '464748';
// Must be a previously Authorized Transaction Order Number
$orderNumber = 'FACPGTEST1307796480663';

// Transaction Details.
$previousTransactionDetails = array('AcquirerId' => $acquirerId,
                                     'MerchantId' =>
$facId,
                                     'OrderNumber' =>
$orderNumber,
                                     'Password' =>
$password);

// The request data is named 'Request' and not after the
// message request class name for reasons that are not clear!
```

```
$TransactionStatusRequest = array('Request' => $PreviousTransactionDetails);

// For debug, to check the values are OK
var_dump($TransactionStatusRequest);

// Call the Authorize through the Client
$result = $client->TransactionStatus($TransactionStatusRequest);

// Check for a fault
if ($client->fault) {
    echo '<h2>Fault</h2><pre>';
    print_r($result);
    echo '</pre>';
} else {
    // Check for errors
    $err = $client->error;
    if ($err) {
        // Display the error
        echo '<h2>Error</h2><pre>' . $err . '</pre>';
    } else {
        // Display the result
        echo '<h2>Result</h2><pre>';
        print_r($result);
        echo '</pre>';
    }
}

?>
```

TransactionStatus using XML

```
<?php

require_once('XML/Serializer.php');
require_once('XML/Unserializer.php');

$url =
'https://ecm.firstatlanticcommerce.com/PGServiceXML/TransactionStatus';

// This should not be in your code in plain text!
$password = '<your password>';
// Use your own FAC ID
$facId = '<your FAC ID>';
// Acquirer is always 464748
$acquirerId = '464748';
// Must be a previously Authorized Transaction Order Number
$orderNumber = 'FACPGTEST1307796480663';

// Transaction Details.
```

```
$TransactionStatusRequest = array('AcquirerId' => $acquirerId,
                                   'MerchantId' => $facId,
                                   'OrderNumber' => $orderNumber,
                                   'Password' => $password);

// Note the rootAttributes includes the FAC namespace. Do not use the
// namespace option as it will prefix the name, which is not right and
// will cause the call to fail
$options = array(
    "indent"      => " ",
    "linebreak"   => "\n",
    "typeHints"   => false,
    "addDecl"     => true,
    "encoding"    => "UTF-8",
    "rootName"    => "TransactionStatusRequest",
    "defaultTagName" => "item",
    "rootAttributes" => array("xmlns" =>
"http://schemas.firstatlanticcommerce.com/gateway/data")
);

$serializer = new XML_Serializer($options);

if ($serializer->serialize($TransactionStatusRequest))
{
    $xmlRequest = $serializer->getSerializedData();

    echo '<pre>';
    echo htmlspecialchars($xmlRequest);
    echo '</pre>';

    $ch = curl_init($url);

    curl_setopt($ch, CURLOPT_MUTE, 1);
    curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 0);
    curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, 0);
    curl_setopt($ch, CURLOPT_POST, 1);
    curl_setopt($ch, CURLOPT_HTTPHEADER, array('Content-Type: text/xml'));
    curl_setopt($ch, CURLOPT_POSTFIELDS, $xmlRequest);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

    $response = curl_exec($ch);
    curl_close($ch);

    // Let's convert to an Object graph, easier to process
    $options = array('complexType' => 'object');

    $deserializer = new XML_Unserializer($options);
    // Pass in the response XML as a string
    $result = $deserializer->unserialize($response, false);
}
```

```
// As with Serializing, we must call getUnserializedData afterwards
$TransactionStatusResponse = $deserializer->getUnserializedData();

// Display the result objects
echo '<h2>Result</h2><pre>';
print_r($TransactionStatusResponse);
echo '</pre>';
}

?>
```

Programming using SOAP directly

Programming Web Services with SOAP by James Snell, Doug Tidwell & Pavel Kulchenko takes you through low level programming using SOAP directly. If you have no other option open to you, this is your final port of call. It is not recommended, as it creates additional work for your integration and is open to more errors than would be possible using a Toolkit.

Here is a taste of what is entailed:

<http://oreilly.com/catalog/progwebsoap/chapter/ch03.html>

Authorize3DS Implementation

The Authorize3DS Method of the Payment Gateway (v2) does not do an Authorization. It simply returns an HTML Form that you post back to the Cardholder's browser to kick-off the 3DS process.

It is the Cardholder's browser that controls the process and communicates directly with FAC, which in turn communicates with Visa or MasterCard to get the 3DS Authentication Tokens.

These Tokens are then passed directly to an Authorization Handler in our Gateway (if this is not a 3DS only transaction) and the transaction is completed. The result is passed back to the Merchant site by posting it to the "MerRespUrl" Field in the form returned from Authorize3DS.

It is not mandatory to call Authorize3DS to start a 3DS transaction. As such, you do have the option of creating the HTML page code yourself and passing it back to the cardholder's browser, bypassing the need to call the Authorize3DS method.

In deciding whether to use this method or not, weigh the advantages and disadvantages of doing so. The advantage of using the Authorize3DS method is that you are guaranteed to have a correctly formatted form returned to the cardholder's browser. The disadvantage is the additional time added to the overall 3D Secure transaction time by making this web method call.

Calling the Authorize3DS Method in .NET

We will just concentrate on the .NET way of calling a Service Method/Operation using a Service Reference. It is very similar to the Authorize Operation as it has most of the same parameters.

```
' Create Service Reference
Dim client As FACWS.Services = New FACWS.Services()

' Create Request object for 3DS
Dim request As FACWS.Authorize3DSRequest = New FACWS.Authorize3DSRequest()

' Complete Request Data
With request

    .CardDetails = New FACWS.CardDetails()
    .TransactionDetails = New FACWS.TransactionDetails()

    ' NOTE: This is a Required field for Authorize3DS
    .MerchantResponseURL = "https://www.merchant.com/Response.aspx"

    .CardDetails.CardCVV2 = ""
    .CardDetails.CardExpiryDate = "0918"
    .CardDetails.CardNumber = "4111111111111111"
    .CardDetails.IssueNumber = ""
```



```
.CardDetails.StartDate = ""

.TransactionDetails.AcquirerId = "464748"
.TransactionDetails.Amount = "000000000100"
.TransactionDetails.Currency = "840"
.TransactionDetails.CurrencyExponent = "2"
.TransactionDetails.IPAddress = ""
.TransactionDetails.MerchantId = "<Your Merchant ID Here>"
.TransactionDetails.OrderNumber = "FACPGTEST" + DateTime.Now.Ticks.ToString("000000000000")
.TransactionDetails.Signature = Helper.ComputeHash("<Your Merchant Password
Here>", .TransactionDetails.MerchantId, .TransactionDetails.AcquirerId, .TransactionDetails.OrderNumber, .Transa
ctionDetails.Amount, .TransactionDetails.Currency)
.TransactionDetails.SignatureMethod = "SHA1"
.TransactionDetails.TransactionCode = "0"
End With

' Send Request, Capture Response
Dim response As FACWS.Authorize3DSResponse = client.Authorize3DS(request)
```

You will notice that there's an extra parameter over calling the standard "Authorize" operation and that's the MerchantResponseUrl. This is required for 3DS as this is how the Response is delivered for both 3DS only and 3DS with Authorize.

The HTML Form Returned

Below is a sample response that is returned by the Authorize3DS operation which the Merchant then posts back to the cardholder's browser.

```
<html>
<body>
  <form id="frmHtmlCheckOut" name="frmHtmlCheckOut" action="" method="post">
    <noscript>
      <br>
      <br>
      <h1 align="center">Processing your Transaction</h1>
      <h2 align="center">JavaScript is currently disabled or is not supported by your browser.</h2>
      <br>
      <h3 align="center">Please click on the Submit button to continue processing.</h3>
      <input type="submit" value="Submit" ID="submit" NAME="submit"/>
    </noscript>
    <input id="Version" type="hidden" value="1.0.0" name="Version"/>
    <input id="AcqID" type="hidden" value="464748" name="AcqID"/>
    <input id="MerID" type="hidden" value="1234567890" name="MerID"/>
```

```

<input id="OrderID" type="hidden" value="FACTEST01" name="OrderID"/>
<input id="TransactionCode" type="hidden" value="0" name="TransactionCode"/>
<input id="PurchaseAmt" type="hidden" value="000000001200" name="PurchaseAmt"/>
<input id="PurchaseCurrency" type="hidden" value="840" name="PurchaseCurrency"/>
<input id="PurchaseCurrencyExponent" type="hidden" value="2" name="PurchaseCurrencyExponent"
/>

<input id="CardNo" type="hidden" value="4111111111111111" name="CardNo"/>
<input id="CardExpiryDate" type="hidden" value="0918" name="CardExpiryDate"/>
<input id="CardCVV2" type="hidden" value="123" name="CardCVV2"/>
<input id="IssueNumber" type="hidden" value="1" name="IssueNumber"/>
<input id="SignatureMethod" type="hidden" value="SHA1" name="SignatureMethod"/>
<input id="Signature" type="hidden" value="L5H9kcZHgWXpXRguKKwVsxn06E4=" name="Signature"/
>

<input id="CaptureFlag" type="hidden" value="A" name="CaptureFlag"/>
<input id="MerRespURL" type="hidden" value="https://test.com/Response.aspx" name="MerRespURL"
"/>

```

Note: If performing an AVS check the include the following lines of HTML code (Any of the last 4 lines will only be present if you provide a value for the parameters listed)

```

<input id="BillToAddress1" type="hidden" value="123 Park Lane" name="BillToAddress1"/>
<input id="BillToPostCode" type="hidden" value="12345" name=" BillToPostCode"/>
<input id="BillToFirstName" type="hidden" value="John" name="BillToFirstName"/>
<input id="BillToLastName" type="hidden" value="Doe" name="BillToLastName"/>
<input id="BillToCity" type="hidden" value="New York" name="BillToCity"/>
<input id="BillToState" type="hidden" value="New York" name="BillToState"/>

```

Note: If you are doing a 3D Secure Authentication Only transaction (TransactionCode = 64), you will also see the following line of HTML code*

```

<input id="AuthenticationOnly" type="hidden" value="Y" name="AuthenticationOnly"/>
</form>

<script type="text/javascript" language="javascript">
    CheckOut();
    function CheckOut(){
        document.frmHtmlCheckOut.action =
        "https://ecm.firstatlanticcommerce.com/SENTRY/PaymentGateway/Application/DirectAuthLink.aspx";
        document.frmHtmlCheckOut.submit(); }
</script>
</body>
</html>

```

The 3DS Response

On completion of the 3DS transaction, the Merchant Response URL specified in the HTML Form is used to POST the response data back to the Merchant. The merchant must therefore implement a Page/Handler to accept this data.

IMPORTANT: The Data returned from a 3DS operation is always in Query String format. Here is an example:

```
MerID=<yourFACIDHere>&AcqID=464748&OrderID=TEST013009&ResponseCode=1&ReasonCode=1&ReasonCodeDesc=Transaction+is+approved.&ReferenceNo=903016874912&PaddedCardNo=XXXXXXXXXX  
X0565&AuthCode=030085&CVV2Result=M&AuthenticationResult=Y&CAVVValue=jGqyM2aoUrM1CBAA  
KtVkBkAAAA%3D&ECIIndicator=02&TransactionStain=AgAACQABAwABAgICBQAAAEbx3k%3D&OriginalResponseCode=00&Signature=vntw1t5eS7Bivl%2FVIRic94mr190%3D&SignatureMethod=SHA1
```

Looks messy but is easy to read using built in tools in most languages or scripting engines. Most have a Split function or specific utilities that make this a breeze.

It's important to save all of this information for future reference in the event of customer queries or charge-backs. The CAVVValue and TransactionStain values are the important fields for 3DS Authentication.

Glossary of Terms

3D Secure

3D Secure encompasses both Visa's *Verified by Visa* and MasterCard's *SecureCode* security solutions for online e-commerce transactions. These solutions use personal passwords to help protect cardholders' card numbers against unauthorized use.

Authentication

The process of authenticating is used in 3D Secure transactions to verify that the person attempting a transaction with a given credit card number is the actual cardholder by requiring them to enter a personal password they set up when enrolling in the 3D Secure program (either *Verify by Visa* or *SecureCode*).

Authorization

The process of checking that the credit card being used in a transaction contains sufficient funds to cover the amount of the transaction. Note that if sufficient funds are found, the amount is held for a given period of time, waiting to be withdrawn when settlement occurs (the period of time varies based on the issuing bank of the credit card).

Authorization/Capture

An Authorize/Capture not only checks that the credit card being used in a transaction contains sufficient funds to cover the amount of the transaction, it also flags the transaction as captured meaning it is to be sent for settlement in the next settlement period.

AVS (Address Verification System)

AVS is used as an extra level of security for online credit card transactions that takes the first line of the billing address and the zip/postal code of the cardholder and checks if they are valid as compared to what is stored on file for the given credit card number.

CID (Card Identification Digits)

The 4-digit code found on the front of AMEX cards, the CID is used as an extra security step to help to verify that the person using the credit card is the actual cardholder.

CVC2 (Card Verification Code)

The 3-digit code found on the back of MasterCard cards, the CVC2 is used as an extra security step to help to verify that the person using the credit card is the actual cardholder.

CVV2 (Card Verification Value)

The 3-digit code found on the back of Visa cards, the CVV2 is used as an extra security step to help to verify that the person using the credit card is the actual cardholder.

Capture

When a capture is performed (either in an Authorize/Capture or Capture only transaction) it is the process of flagging an already authorized transaction to be settled in the next settlement period.

One-Pass Transaction

A one-pass transaction (also called an authorize/capture transaction in this document) is a transaction that is both authorized and captured (flagged for settlement) at the same time, in a single transaction request.

FACPG2

The First Atlantic Commerce Payment Gateway Services. These services support and enable the FAC products [cGate® Secure Real-Time](#) and [cGate® Secure Verify](#).

Refund

A refund is the process of refunding a previously settled transaction. This will appear as a credit on the cardholder's credit card statement.

Reversal

A reversal is the process of reversing a previously captured, but not yet settled, transaction. It means that the transaction will never appear on the cardholder's credit card statement.

Settle/Settled/Settlement

The process of settling a transaction is when the money is taken from the cardholder's account and put into the merchant's account. Once a transaction is settled, it will appear as a charge on the cardholder's credit card statement.

SHA1

Secure Hash Algorithm 1. A message digest (hash) function defined in RFC 3174.

Transaction

A transaction is any e-commerce request made by you, the merchant, to FAC. This includes Authorizations (both 3D and Non-3D Secure), Authorization & Captures (both 3D and Non-3D Secure), Captures only, Reversals, Refunds, 3D Secure Authentication Only transactions and AVS Verification Only transactions.

Two-Pass Transaction

A two-pass transaction is a transaction that is processed in two separate transaction requests. The first transaction is the authorization only request and the second transaction (which can come seconds, minutes, hours or even days after the first transaction) captures this transaction and flags it for settlement.

Appendix A - Field Requirements and Allowable Values

1) AVS Field Requirements

The following table lays out the field requirements for Address Verification checks. These requirements may vary somewhat by processor; however, this is the standard across the board:

Field Name	Required or Optional Field	Allowable Character Format	Character Limit	Special Considerations - Restrictions
BillToFirstName	Optional	alphanumeric (a-z, A-Z, 0-9)	up to 30 maximum	No special characters, no accents, no special symbols (Example: æ é à ñ * + & ; ;) and best to avoid all symbols but basic punctuation is acceptable such as periods and dashes (. and -)
BillToLastName	Optional	alphanumeric (a-z, A-Z, 0-9)	up to 30 maximum	No special characters, no accents, no special symbols (Example: æ é à ñ * + & ; ;) and best to avoid all symbols but basic punctuation is acceptable such as periods and dashes (. and -)
BillToAddress	Required	alphanumeric (a-z, A-Z, 0-9)	up to 50 maximum	No special characters, no accents, no special symbols (Example: æ é à ñ * + & ; ;) and best to avoid all symbols but basic punctuation is acceptable such as periods and dashes (. and -)
BillToAddress2	Optional	alphanumeric (a-z, A-Z, 0-9)	up to 50 maximum	No special characters, no accents, no special symbols (Example: æ é à ñ * + & ; ;) and best to avoid all symbols but basic punctuation is acceptable such as periods and dashes (. and -)
BillToCity	Optional	alphanumeric (a-z, A-Z, 0-9)	up to 30 maximum	No special characters, no accents, no symbols (Example : æ é à ñ * + & ; ;)
BillToState	Optional	alphanumeric (a-z, A-Z, 0-9)	2 minimum to 5 maximum	Ideally use the 2 alpha character ISO State Code OR a 3 numeric digit ISO country code if necessary for customers where 'State' would not be applicable OR this field can be omitted. Strictly Alpha numeric. No special characters, no accents, no spaces, no symbols
BillToZipPostCode	Required	alphanumeric (a-z, A-Z, 0-9)	up to 10 maximum	Strictly Alphanumeric only - No special characters, no accents, no spaces, no dashes...etc.
BillToCountry	Optional	Numeric (0-9)	must be 3 digits	This must be 3 digit country code. See ISO 3166-1 Numeric Country Codes .
BillToTelephone	Optional	Numeric (0-9)	up to 20 maximum	Strictly numeric. No special characters, no accents, no spaces, no symbols
BillToEmail	Optional	Alphanumeric (a-z, A-Z, 0-9)	up to 50 maximum	Standard email format (name@domain.com). Basic punctuation is acceptable such as periods and dashes (. and -). No special characters, no accents, no spaces

2) BillToState Allowable Values

The parameter BillToState is only valid for U.S. based addresses. The allowable values for this parameter are as follows:

Abbreviation	State Name	Abbreviation	State Name
AK	Alaska	MS	Mississippi
AL	Alabama	MT	Montana
AR	Arkansas	NC	North Carolina
AS	American Samoa	ND	North Dakota
AZ	Arizona	NE	Nebraska
CA	California	NH	New Hampshire
CO	Colorado	NJ	New Jersey
CT	Connecticut	NM	New Mexico
DC	District of Columbia	NV	Nevada
DE	Delaware	NY	New York
FL	Florida	OH	Ohio
FM	Federate States Of Micronesia	OK	Oklahoma
GA	Georgia	OR	Oregon
GU	Guam	PA	Pennsylvania
HI	Hawaii	PR	Puerto Rico
IA	Iowa	PW	Palau
ID	Idaho	RI	Rhode Island
IL	Illinois	SC	South Carolina
IN	Indiana	SD	South Dakota
KS	Kansas	TN	Tennessee
KY	Kentucky	TX	Texas
LA	Louisiana	UT	Utah
MA	Massachusetts	VA	Virginia
MD	Maryland	VI	U.S. Virgin Islands
ME	Maine	VT	Vermont
MH	Marshall Islands	WA	Washington
MI	Michigan	WI	Wisconsin
MN	Minnesota	WV	West Virginia
MO	Missouri	WY	Wyoming
MP	Northern Mariana Islands		

3) ISO 4217 Currency Codes

Currency	Alphabetic Code	Numeric Code	Minor unit (Currency exponent)
Afghani	AFN	971	2
Algerian Dinar	DZD	012	2
Argentine Peso	ARS	032	2
Armenian Dram	AMD	051	2
Aruban Florin	AWG	533	2
Australian Dollar	AUD	036	2
Azerbaijani Manat	AZN	944	2
Bahamian Dollar	BSD	044	2
Bahraini Dinar	BHD	048	3
Baht	THB	764	2
Balboa	PAB	590	2
Barbados Dollar	BBD	052	2
Belarussian Ruble	BYR	974	0
Belize Dollar	BZD	084	2
Bermudian Dollar	BMD	060	2
Bolivar	VEF	937	2
Boliviano	BOB	068	2
Brazilian Real	BRL	986	2
Brunei Dollar	BND	096	2
Bulgarian Lev	BGN	975	2
Burundi Franc	BIF	108	0
Canadian Dollar	CAD	124	2
Cape Verde Escudo	CVE	132	2
Cayman Islands Dollar	KYD	136	2
CFA Franc BCEAO	XOF	952	0
CFA Franc BEAC	XAF	950	0
CFP Franc	XPF	953	0
Chilean Peso	CLP	152	0
Colombian Peso	COP	170	2
Comoro Franc	KMF	174	0
Congolese Franc	CDF	976	2
Convertible Mark	BAM	977	2
Cordoba Oro	NIO	558	2
Costa Rican Colon	CRC	188	2
Croatian Kuna	HRK	191	2
Cuban Peso	CUP	192	2
Czech Koruna	CZK	203	2
Dalasi	GMD	270	2
Danish Krone	DKK	208	2
Denar	MKD	807	2
Djibouti Franc	DJF	262	0
Dobra	STD	678	2
Dominican Peso	DOP	214	2

Dong	VND	704	0
East Caribbean Dollar	XCD	951	2
Egyptian Pound	EGP	818	2
El Salvador Colon	SVC	222	2
Ethiopian Birr	ETB	230	2
Euro	EUR	978	2
Falkland Islands Pound	FKP	238	2
Fiji Dollar	FJD	242	2
Forint	HUF	348	2
Ghana Cedi	GHS	936	2
Gibraltar Pound	GIP	292	2
Gourde	HTG	332	2
Guarani	PYG	600	0
Guinea Franc	GNF	324	0
Guyana Dollar	GYD	328	2
Hong Kong Dollar	HKD	344	2
Hryvnia	UAH	980	2
Iceland Krona	ISK	352	0
Indian Rupee	INR	356	2
Iranian Rial	IRR	364	2
Iraqi Dinar	IQD	368	3
Jamaican Dollar	JMD	388	2
Jordanian Dinar	JOD	400	3
Kenyan Shilling	KES	404	2
Kina	PGK	598	2
Kip	LAK	418	2
Kuwaiti Dinar	KWD	414	3
Kwacha	MWK	454	2
Kwanza	AOA	973	2
Kyat	MMK	104	2
Lari	GEL	981	2
Latvian Lats	LVL	428	2
Lebanese Pound	LBP	422	2
Lek	ALL	008	2
Lempira	HNL	340	2
Leone	SLL	694	2
Liberian Dollar	LRD	430	2
Libyan Dinar	LYD	434	3
Lilangeni	SZL	748	2
Lithuanian Litas	LTL	440	2
Loti	LSL	426	2
Malagasy Ariary	MGA	969	2
Malaysian Ringgit	MYR	458	2
Mauritius Rupee	MUR	480	2
Mexican Peso	MXN	484	2
Mexican Unidad de Inversion (UDI)	MXV	979	2
Moldovan Leu	MDL	498	2
Moroccan Dirham	MAD	504	2
Mozambique Metical	MZN	943	2

Mvdol	BOV	984	2
Naira	NGN	566	2
Nakfa	ERN	232	2
Namibia Dollar	NAD	516	2
Nepalese Rupee	NPR	524	2
Netherlands Antillean Guilder	ANG	532	2
New Israeli Sheqel	ILS	376	2
New Romanian Leu	RON	946	2
New Taiwan Dollar	TWD	901	2
New Zealand Dollar	NZD	554	2
Ngultrum	BTN	064	2
North Korean Won	KPW	408	2
Norwegian Krone	NOK	578	2
Nuevo Sol	PEN	604	2
Ouguiya	MRO	478	2
Pa'anga	TOP	776	2
Pakistan Rupee	PKR	586	2
Pataca	MOP	446	2
Peso Convertible	CUC	931	2
Peso Uruguayo	UYU	858	2
Philippine Peso	PHP	608	2
Pound Sterling	GBP	826	2
Pula	BWP	072	2
Qatari Rial	QAR	634	2
Quetzal	GTQ	320	2
Rand	ZAR	710	2
Rial Omani	OMR	512	3
Riel	KHR	116	2
Rufiyaa	MVR	462	2
Rupiah	IDR	360	2
Russian Ruble	RUB	643	2
Rwanda Franc	RWF	646	0
Saint Helena Pound	SHP	654	2
Saudi Riyal	SAR	682	2
Serbian Dinar	RSD	941	2
Seychelles Rupee	SCR	690	2
Singapore Dollar	SGD	702	2
Solomon Islands Dollar	SBD	090	2
Som	KGS	417	2
Somali Shilling	SOS	706	2
Somoni	TJS	972	2
South Sudanese Pound	SSP	728	2
Sri Lanka Rupee	LKR	144	2
Sucre	XSU	994	N.A.
Sudanese Pound	SDG	938	2
Surinam Dollar	SRD	968	2
Swedish Krona	SEK	752	2
Swiss Franc	CHF	756	2
Syrian Pound	SYP	760	2

Taka	BDT	050	2
Tala	WST	882	2
Tanzanian Shilling	TZS	834	2
Tenge	KZT	398	2
Trinidad and Tobago Dollar	TTD	780	2
Tugrik	MNT	496	2
Tunisian Dinar	TND	788	3
Turkish Lira	TRY	949	2
Turkmenistan New Manat	TMT	934	2
UAE Dirham	AED	784	2
Uganda Shilling	UGX	800	2
Unidad de Valor Real	COU	970	2
Unidades de fomento	CLF	990	0
Uruguay Peso en Unidades Indexadas (URUIURUI)	UYI	940	0
US Dollar	USD	840	2
Uzbekistan Sum	UZS	860	2
Vatu	VUV	548	0
WIR Euro	CHE	947	2
WIR Franc	CHW	948	2
Won	KRW	410	0
Yemeni Rial	YER	886	2
Yen	JPY	392	0
Yuan Renminbi	CNY	156	2
Zambian Kwacha	ZMK	894	2
Zimbabwe Dollar	ZWL	932	2
Zloty	PLN	985	2

Note: These currency codes are provided for reference only, please check with FAC if you are interested in processing in a specific currency.

Source: http://www.iso.org/iso/home/standards/currency_codes.htm

4) ISO 3166-1 Numeric Country Codes

Code	Country name	Code	Country name
004	Afghanistan	132	Cape Verde
008	Albania	136	Cayman Islands
010	Antarctica	140	Central African Republic
012	Algeria	144	Sri Lanka
016	American Samoa	148	Chad
020	Andorra	152	Chile
024	Angola	156	China
028	Antigua and Barbuda	158	Taiwan, Province of China
031	Azerbaijan	162	Christmas Island
032	Argentina	166	Cocos (Keeling) Islands
036	Australia	170	Colombia
040	Austria	174	Comoros
044	Bahamas	175	Mayotte
048	Bahrain	178	Congo
050	Bangladesh	180	Congo, the Democratic Republic of the
051	Armenia	184	Cook Islands
052	Barbados	188	Costa Rica
056	Belgium	191	Croatia
060	Bermuda	192	Cuba
064	Bhutan	196	Cyprus
068	Bolivia, Plurinational State of	203	Czech Republic
070	Bosnia and Herzegovina	204	Benin
072	Botswana	208	Denmark
074	Bouvet Island	212	Dominica
076	Brazil	214	Dominican Republic
084	Belize	218	Ecuador
086	British Indian Ocean Territory	222	El Salvador
090	Solomon Islands	226	Equatorial Guinea
092	Virgin Islands, British	231	Ethiopia
096	Brunei Darussalam	232	Eritrea
100	Bulgaria	233	Estonia
104	Myanmar	234	Faroe Islands
108	Burundi	238	Falkland Islands (Malvinas)
112	Belarus	239	South Georgia and the South Sandwich Islands
116	Cambodia	242	Fiji
120	Cameroon	246	Finland
124	Canada	248	Åland Islands

ISO 3166-1 Numeric Country Codes

Code	Country name	Code	Country name
250	France	392	Japan
254	French Guiana	398	Kazakhstan
258	French Polynesia	400	Jordan
260	French Southern Territories	404	Kenya
262	Djibouti	408	Korea, Democratic People's Republic of
266	Gabon	410	Korea, Republic of
268	Georgia	414	Kuwait
270	Gambia	417	Kyrgyzstan
275	Palestine, State of	418	Lao People's Democratic Republic
276	Germany	422	Lebanon
288	Ghana	426	Lesotho
292	Gibraltar	428	Latvia
296	Kiribati	430	Liberia
300	Greece	434	Libya
304	Greenland	438	Liechtenstein
308	Grenada	440	Lithuania
312	Guadeloupe	442	Luxembourg
316	Guam	446	Macao
320	Guatemala	450	Madagascar
324	Guinea	454	Malawi
328	Guyana	458	Malaysia
332	Haiti	462	Maldives
334	Heard Island and McDonald Islands	466	Mali
336	Holy See (Vatican City State)	470	Malta
340	Honduras	474	Martinique
344	Hong Kong	478	Mauritania
348	Hungary	480	Mauritius
352	Iceland	484	Mexico
356	India	492	Monaco
360	Indonesia	496	Mongolia
364	Iran, Islamic Republic of	498	Moldova, Republic of
368	Iraq	499	Montenegro
372	Ireland	500	Montserrat
376	Israel	504	Morocco
380	Italy	508	Mozambique
384	Côte d'Ivoire	512	Oman
388	Jamaica	516	Namibia

ISO 3166-1 Numeric Country Codes

Code	Country name	Code	Country name
520	Nauru	646	Rwanda
524	Nepal	652	Saint Barthélemy
528	Netherlands	654	Saint Helena, Ascension and Tristan da Cunha
531	Curaçao	659	Saint Kitts and Nevis
533	Aruba	660	Anguilla
534	Sint Maarten (Dutch part)	662	Saint Lucia
535	Bonaire, Sint Eustatius and Saba	663	Saint Martin (French part)
540	New Caledonia	666	Saint Pierre and Miquelon
548	Vanuatu	670	Saint Vincent and the Grenadines
554	New Zealand	674	San Marino
558	Nicaragua	678	Sao Tome and Principe
562	Niger	682	Saudi Arabia
566	Nigeria	686	Senegal
570	Niue	688	Serbia
574	Norfolk Island	690	Seychelles
578	Norway	694	Sierra Leone
580	Northern Mariana Islands	702	Singapore
581	United States Minor Outlying Islands	703	Slovakia
583	Micronesia, Federated States of	704	Viet Nam
584	Marshall Islands	705	Slovenia
585	Palau	706	Somalia
586	Pakistan	710	South Africa
591	Panama	716	Zimbabwe
598	Papua New Guinea	724	Spain
600	Paraguay	728	South Sudan
604	Peru	729	Sudan
608	Philippines	732	Western Sahara
612	Pitcairn	740	Suriname
616	Poland	744	Svalbard and Jan Mayen
620	Portugal	748	Swaziland
624	Guinea-Bissau	752	Sweden
626	Timor-Leste	756	Switzerland
630	Puerto Rico	760	Syrian Arab Republic
634	Qatar	762	Tajikistan
638	Réunion	764	Thailand
642	Romania	768	Togo
643	Russian Federation	772	Tokelau

ISO 3166-1 Numeric Country Codes

Code	Country name
776	Tonga
780	Trinidad and Tobago
784	United Arab Emirates
788	Tunisia
792	Turkey
795	Turkmenistan
796	Turks and Caicos Islands
798	Tuvalu
800	Uganda
804	Ukraine
807	Macedonia, the former Yugoslav Republic of
818	Egypt
826	United Kingdom
831	Guernsey
832	Jersey
833	Isle of Man
834	Tanzania, United Republic of
840	United States
850	Virgin Islands, U.S.
854	Burkina Faso
858	Uruguay
860	Uzbekistan
862	Venezuela, Bolivarian Republic of
876	Wallis and Futuna
882	Samoa
887	Yemen
894	Zambia

Note: These country codes are provided for reference only.

Appendix B – Response and Reason Codes

1) CVV2/CVC2 Response Codes

After checking a CVV2/CVC2, values are returned in the CVV2Result field as follows:

Code	Definition
M	Match
N	No match.
P	Not Processed
S	Should be on card but was not provided. (Visa only)
U	Issuer not participating or certified.

2) AVS Check Response Codes

AVS Codes are returned in the AVSResult field in the Response message of the Operation concerned; one of [AuthorizeResponse](#), [TransactionModificationResponse](#), or [TransactionStatusResponse](#). There are different codes depending on the card type.

Visa

Code	Definition
A	Address matches, Zip code does not match.
B	Street addresses match for international transaction. Postal code not verified due to incompatible formats. (Acquirer sent street address and postal code.)
C	Street address and postal code not verified for international transaction due to incompatible formats. (Acquirer sent street address and postal code.)
D	Street addresses and postal codes match for international transaction.
E	Error response for Merchant Category Code.
F	Address does compare and five-digit ZIP codes does compare (UK only)
G	Address information is unavailable for international transaction; non-AVS participant.
I	Address information not verified for international transaction.
M	Street addresses and postal codes match for international transaction.
N	Address and ZIP code do not match.
P	Postal codes match for international transaction. Street address not verified due to incompatible formats. (Acquirer sent street address and postal code.)
R	Retry; system unavailable or timed out.
S	Service not supported by issuer.
U	Address information is unavailable; domestic transactions.
W	Nine-digit ZIP code matches, but address does not match.
X	Exact match, address, and nine-digit ZIP code match.
Y	Address and five-digit ZIP code match.
Z	Five-digit ZIP code matches, but address does not match.
5*	Invalid AVS response (from VISA).
9*	Address Verification Data contains EDIT ERROR.

0	Issuer has chosen not to perform Address Verification for an authorization that was declined.
---	---

MasterCard

Code	Definition
A	Address matches, postal code does not.
N	Neither address nor postal code matches.
R	Retry, system unable to process.
S	AVS currently not supported
U	No data from issuer/Authorization System.
W	For U.S. addresses, nine-digit postal code matches, address does not; for address outside the U.S., postal code matches, address does not.
X	For U.S. addresses, nine-digit postal code and address matches; for addresses outside the U.S., postal code and address match.
Y	For U.S. addresses, five-digit postal code and address matches.
Z	For U.S. addresses, five-digit postal code matches, address does not.
5*	Invalid AVS response (from MasterCard)
9*	Address Verification Data contains EDIT ERROR.
0	Issuer has chosen not to perform Address Verification for an authorization that was declined.

Note: For MasterCard, if a 5 digit zip code is sent and a 9 digit zip code is on the cardholder file (and address matches) a response of 'Y' is returned.

Amex

Code	Definition
A	ADDRESS: Address correct, zip code incorrect
N	NO: Address and zip code are no correct.
R	Retry, system unavailable or timeout.
S	Address Verification Service not valid.
U	Address information is unavailable, account number is not US or Canadian.
Y	YES: Address and zip code are correct.
Z	Zip code correct; address incorrect.
5*	Invalid AVS response (from American Express).
9*	Address Verification Data contains EDIT ERROR.

* These responses (5 & 9) for all credit card types are processor generated responses. Response Code 9 means the record was not sent out for Address Verification. This response will also be returned when address verification has not been requested.

3) Response Code and Reason Code Responses

The ResponseCode, ReasonCode and ReasonCodeDescription fields of the [AuthorizeResponse](#) and [TransactionStatusResponse](#) messages can hold the following code combinations.

NOTE: If you are using [Fraud Control](#) services there will be additional potential ReasonCodes and ReasonCodeDescriptions than described below (refer to [Fraud Response and Reason Codes](#)).

ResponseCode Values

Response Code	Description
1	Approved
2	Declined
3	Error

Reason Code for “Approved” Response Code (1)

Reason Code	Reason Text (ReasonCodeDescription)	Note
1	Transaction is approved.	Normal Approval.

Reason Codes for “Decline” Response Code (2)

Reason Code	Reason Text (ReasonCodeDescription)	Note
2	Transaction is declined.	Normal Decline.
3	Transaction is declined.	Referral. Call for further details on this transaction.
4	Transaction is declined.	Pick up card (if possible) or report to authorities.
35	Unable to process your request. Please try again later.	Merchant exceeds allowed limit.
38	Transaction processing terminated. Please try again later.	Transaction is not permitted to merchant.
39	Issuer or switch not available. Please try again later	Issuing bank or switch not available. Transaction has timed-out.

Reason Codes for "Error" Response Code (3)

Reason Code	Reason Text (ReasonCodeDescription)	Note
5	Connection not secured.	Connection was not secured.
6	HTTP Method not POST.	HTTP Method not POST.
7	"Field" is missing.	Named field is missing.
8	"Field" format is invalid.	Named field format is invalid.
10	Invalid Merchant.	Not such merchant.
11	Failed Authentication (Signature computed incorrectly).	Merchant was found but computed signature does not match one included in the request.
12	Merchant is inactive.	Merchant is not enabled for processing.
14	Merchant is not allowed to process this currency.	Currency supplied is not permitted.
15	Merchant settings are not valid.	Merchant record is not correctly setup in the system.
16	Unable to process transaction.	Unable to authenticate merchant now. Try later.
36	Credit Cardholder canceled the request.	Credit Cardholder canceled the request.
37	Card Entry Retry Count exited allowed limit.	Card Entry Retry Count exited allowed limit.
40	Duplicate Order Not Allowed	Merchant order identification numbers must be unique
42	Illegal Operation by Card Holder. Check Order Status.	Cardholder Pressed the back button while the transaction was processing. Check the status of that order.
60	Duplicate Order Not Allowed.	A transaction for the same card number and same amount was processed previously and thus this transaction has been blocked (optional setting)
90	General Error during processing. Please try again later.	An unexpected error occurred in the system.
98	System is temporarily down. Try later.	System is temporarily down. Try later.
401	Cycle interrupted by the user or client/browser connection not available.	Client Browser connection not available or card holder referred in the process (Back/F5).
994	FACPGWS BeginTransactionStatus Failure	Error while attempting to run the TransactionStatus Operation Try again. If this persists, contact FAC support at support@fac.bm for assistance.
995	FACPGWS EndTransactionStatus Failure	Error while attempting to run TransactionStatus Operation. Try again. If this persists, contact FAC support at support@fac.bm for assistance.
996	Not a web-based transaction	The transaction for which you are requesting the response data is not a web-based transaction. It is a

		MOTO transaction and as such there is no web-based response data for this transaction.
997	FACPGAppWS Failure	Error while attempting to run the TransactionStatus Operation. Try again. If this persists, contact FAC support at support@fac.bm for assistance.
998	Missing Parameter	One of the parameters required by the TransactionStatus Operation was not supplied.
999	No Response	There is no response data for the Order ID provided.
1001	FACPGWS Invalid Protocol. Only HTTPS Allowed	The request was sent via HTTP not HTTPS.
1002	Missing Parameter(s)	One or more of the required parameters is missing in the web method you have called.
1003	Invalid Parameter Settings	Both "AVS Only" and "PreAuthenticated" flags have been included in the TransactionCode when calling the Authorize web method. This is not allowed.
1004	Invalid Amount. Not 12 characters in length	Amount must be exactly 12 characters in length, right-aligned, left-padded with zeros. For example, \$12.00 = 000000001200
1010	FACPGWS Authorize HTTP Response Not OK	Error while attempting to run the Authorize Operation. Try again. If this persists, contact FAC support at support@fac.bm for assistance.
1020	FACPGWS Authorize Failure	Error while attempting to run the Authorize web method. Try again. If this persists, contact FAC support at support@fac.bm for assistance.
1030	FACPG BeginCRRError	Error while attempting to run either the Capture, Reversal or Refund web methods. Try again. If this persists, contact FAC support at support@fac.bm for assistance.
1031	FACPG EndCRRError	Error while attempting to run either the Capture, Reversal or Refund web methods. Try again. If this persists, contact FAC support at support@fac.bm for assistance.

System Response Reason Codes for 3D Secure Transactions

Reason Code	Reason Text (ReasonCodeDescription)	Note
13	Merchant is not allowed to process cards in this Payment system.	Merchant is blocked.
17	Unable to process transaction.	System cannot process a Card Range Request.
18	Unable to process transaction.	System cannot build a Verify Enrollment Request.
19	Unable to process transaction.	System cannot contact Visa Directory.
20	Unable to process transaction.	System cannot build a Payment Authentication.
21	Unable to process transaction.	System could not contact Issuer ACS Server

22	Unable to process transaction.	Issuer ACS responded with invalid data or returned data failed.
23	Unable to process transaction.	System cannot process a Verify Enrollment Request.
31	Authentication successful.	3-D Secure Payment Authentication successful.
32	Authentication failed.	3-D Secure Payment Authentication failed.
33	Authentication successful with attempt.	Attempt authentication was performed.
34	Authentication failed with error.	Authentication result not expected.
41	Card Holder Session Expired.	Cardholder's Session expired while performing a 3DS Transaction. Possibly because he/she closed the window, or pressed the back button in the middle of the transaction.
42	Illegal Operation by Card Holder. Check Order Status.	Cardholder Pressed the back button while the transaction was processing. Check the status of that order.
50	Verify Enrollment response unavailable.	The VeRes message came back from the MPI as "U".
51	BIN Not Enrolled.	The VeRes message came back from the MPI as "N"
52	Card Not Enrolled.	The VeRes message came back from the MPI as "N"
53	Payer Authentication Response Unavailable	The PaRes message came back from the MPI as "U".
96	Merchant URL is Missing	Merchant URL is Missing
98	System is temporarily down. Try later.	System is temporarily down. Try later.
401	Cycle interrupted by the user or client/browser connection not available.	Client Browser connection not available or cardholder referred in the process (Back/F5).
1001	FACPGWS Invalid Protocol. Only HTTPS Allowed	The request was sent via HTTP not HTTPS.
1002	Missing Parameter or Parameters	One or more of the required parameters is missing in the web method you have called.
1004	Invalid Amount. Not 12 characters in length	Amount must be exactly 12 characters in length, right-aligned, left-padded with zeros. For example, \$12.00 = 000000001200
1005	Invalid Capture Flag value provided	The CaptureFlag parameter must be set to either "M" for manual capture (authorize only) or "A" for automatic (authorize/capture)

5) Authorization Original Response Codes

The response codes for an Authorization are returned in the OriginalResponseCode field of the Response. See also [AuthorizeResponse](#), [TransactionModificationResponse](#), or [TransactionStatusResponse](#). They are specific to the Card Issuer.

VISA

Response Code & Description		Response Code & Description	
00	Approved	53	No savings account
01	Refer to issuer	54	Expired card
02	Refer to issuer (special)	55	Incorrect PIN
03	Invalid merchant	56	No card record
04	Pick-up card	57	Transaction not permitted to card
05	Do not honor	58	Transaction not permitted to card
06	Error	59	Suspected fraud
07	Pick-up card (special)	60	Card acceptor contact acquirer
08	Honor with identification	61	Exceeds withdrawal limit
09	Request in progress	62	Restricted card
10	Approved for partial amount	63	Security violation
11	VIP Approval	64	Original amount incorrect
12	Invalid transaction	65	Activity count exceeded
13	Invalid amount	66	Card acceptor call acquirer
14	Card number does not exist	67	Card pick up at ATM
15	No such issuer	68	Response received too late
16	Approved, update track 3	75	Too many wrong PIN tries
17	Customer cancellation	76	Previous message not found
18	Customer dispute	77	Data does not match original message
19	Re-enter transaction	80	Invalid date
20	Invalid response	81	Cryptographic error in PIN
21	No action taken (no match)	82	Incorrect CVV
22	Suspected malfunction	83	Unable to verify PIN
23	Unacceptable transaction fee	84	Invalid authorization life cycle
24	File update not supported by receiver	85	No reason to decline
25	Unable to locate record	86	PIN validation not possible
26	Duplicate file update record	88	Cryptographic failure
27	File update field edit error	89	Authentication failure
28	File temporarily unavailable	90	Cutoff is in process
29	File update not successful	91	Issuer or switch inoperative
30	Format error	92	No routing path
31	Issuer sign-off	93	Violation of law
32	Completed partially	94	Duplicate transmission
33	Expired card	95	Reconcile error
34	Suspected fraud	96	System malfunction
35	Card acceptor contact acquirer	97	Format Error

36	Restricted card	98	Host Unreachable
37	Card acceptor call acquirer	99	Errored Transaction
38	Allowable PIN tries exceeded	N0	Force STIP
39	No credit account	N3	Cash Service Not Available
40	Function not supported	N4	Cash request exceeds issuer limit
41	Pick-up card (lost card)	N7	Decline for CVV2 failure
42	No universal account	P2	Invalid biller information
43	Pick-up card (stolen card)	P5	PIN Change Unblock Declined
44	No investment account	P6	Unsafe PIN
51	Not sufficient funds	XA	Forward to issuer
52	No checking account	XD	Forward to issuer
R0	Stop Payment Order	R1	Revocation of Authorization Order
R3	Revocation of All Authorization Orders		

MasterCard

Response Code & Description		Response Code & Description	
00	Approved	44	No investment account
01	Refer to issuer	51	Not sufficient funds
02	Refer to issuer (special)	52	No checking account
03	Invalid merchant	53	No savings account
04	Pick-up card	54	Expired card
05	Do not honor	55	Incorrect PIN
06	Error	56	No card record
07	Pick-up card (special)	57	Transaction not permitted to card
08	Honor with identification	58	Transaction not permitted to card
09	Request in progress	59	Suspected fraud
10	Approved for partial amount	60	Card acceptor contact acquirer
11	VIP Approval	61	Exceeds withdrawal limit
12	Invalid transaction	62	Restricted card
13	Invalid amount	63	Security violation
14	Card number does not exist	64	Original amount incorrect
15	No such issuer	65	Activity count exceeded
16	Approved, update track 3	66	Card acceptor call acquirer
17	Customer cancellation	67	Card pick up at ATM
18	Customer dispute	68	Response received too late
19	Re-enter transaction	75	Too many wrong PIN tries
20	Invalid response	76	Previous message not found
21	No action taken (no match)	77	Data does not match original message
22	Suspected malfunction	80	Invalid date
23	Unacceptable transaction fee	81	Cryptographic error in PIN
24	File update not supported by receiver	82	Incorrect CVV
25	Unable to locate record	83	Unable to verify PIN
26	Duplicate file update record	84	Invalid authorization life cycle

27	File update field edit error	85	No reason to decline
28	File temporarily unavailable	86	PIN validation not possible
29	File update not successful	88	Cryptographic failure
30	Format error	89	Authentication failure
31	Issuer sign-off	90	Cutoff is in process
32	Completed partially	91	Issuer or switch inoperative
33	Expired card	92	No routing path
34	Suspected fraud	93	Violation of law
35	Card acceptor contact acquirer	94	Duplicate transmission
36	Restricted card	95	Reconcile error
37	Card acceptor call acquirer	96	System malfunction
38	Allowable PIN tries exceeded	97	Format Error
39	No credit account	98	Issuer Unreachable
40	Function not supported	99	Errored Transaction
41	Pick-up card (lost card)	XA	Forward to issuer
42	No universal account	XD	Forward to issuer
43	Pick-up card (stolen card)		

AMEX

Response Code & Description	
000	Approved
001	Approved with ID
100	Deny
101	Expired Card
106	PIN tries Exceeded
107	Please Call Issuer
109	Invalid Service Establishment
110	Invalid Amount
111	Invalid Account
115	Requested Function Not Support
117	Incorrect PIN
121	Limit Exceeded
122	Invalid Manually Entered 4DBC
183	Invalid Currency Code
199	Valid PIN
200	Deny - Pick up Card
290	Refused, Retain Card
300	Successful
301	Not supported by receiver
302	Unable to locate record
303	Duplicate record
304	Field edit error
380	File update not accepted, high

400	Reversal Accepted
800	Accepted
880	File Fully Accepted
881	File Partially Accepted
882	File Fully Rejected
899	Table not found. Default used
900	Advice Accepted

6) Transaction Modification Response and Reason Codes

The ResponseCode, ReasonCode and ReasonCodeDescription fields of the [TransactionModificationResponse](#) message can the following code and description combinations.

Response Code	Description
1	Approved
3	Error

Code	Description	Code	Description
1001	FACPGWS Invalid Protocol. Only HTTPS Allowed	1133	Refund Amount Authentication Invalid
1002	Missing Parameter or Parameters	1134	Refund Amount OrderId Invalid
1004	Invalid Amount. Not 12 characters in length	1135	RefundAmountInvalidMerchantStatus
1030	FACPGWS BeginCRRError	1136	RefundAmountRefundNotAllowed
1031	FACPGWS EndCRRError	1137	RefundAmountRefundCutOffExpired
1100	Failed	1138	ReverseAmountReversalAmtGreaterThanTrxnAmt
1101*	Success	1139	ReverseAmountNonAuthorizedOrder
1102	AuthenticatorSoapHeaderMissing	1140	ReverseAmountNonApprovedOrder
1103	AuthenticatorAcquirerIDMissing	1141	ReverseAmountAlreadyReversed
1104	AuthenticatorAcquirerIDInvalid	1142	ReverseAmountPartialReversalOnTestOrder
1105	AuthenticatorMerchantIDMissing	1143	ReverseAmountGeneralError
1106	AuthenticatorMerchantIDInvalid	1144	ReverseAmountAuthenticationProcessGeneralError
1107	AuthenticatorPasswordMissing	1145	ReverseAmountAuthenticationProcessingError
1108	AuthenticatorPasswordInvalid	1146	ReverseAmountAuthenticationUnknownError
1109	AuthenticateHeaderGeneralError	1147	ReverseAmountAuthenticationFailure
1110	GeneralParamError	1148	ReverseAmountAuthenticationInvalid
1111	CaptureTrxnAmount_TrxnHasBeenClosed	1149	ReverseAmountOrderIdInvalid
1112	CaptureTrxnAmount_CaptureAmtGreaterThanTrxnAmt	1150	ReverseAmountInvalidMerchantStatus
1113	CaptureTrxnAmount_CaptureAmtGreaterThanLeftOverAmt	1151	ReverseAmountReversalNotAllowed
1114	CaptureTrxnAmount_NonAuthorizedOrder	1152	ReverseAmountReversalCutOffExpired

1115	CaptureTrxnAmount_NonApprovedOrder	1153	ReverseAmountPartialReversalNotAllowed
1116	CaptureTrxnAmount_TestOrder	1154	CaptureTrxnAmountOrderInProcessing
1117	CaptureTrxnAmountSuccess	1155	RefundAmountOrderInProcessing
1118	CaptureTrxnAmountGeneralError	1156	ReverseAmountOrderInProcessing
1119	CaptureTrxnAmountAuthenticationProcessGeneralError	1157	CaptureTrxnAmountCaptureReversedNotAllowed
1120	CaptureTrxnAmountAuthenticationProcessingError	1158	CaptureTrxnAmountIsZero
1121	CaptureTrxnAmountAuthenticationUnknownError	1159	ReverseAmountOnInstallment
1122	CaptureTrxnAmountAuthenticationFailure	1160	CaptureTrxnAmount_CaptureAmtLessThanPreviousCapture
1123	CaptureTrxnAmountAuthenticationInvalid	1161	RefundTrxnAmountIsZero
1124	CaptureTrxnAmountOrderIdInvalid	1162	CancelRecurringInvalidMerchantStatus
1125	CaptureTrxnAmountInvalidMerchantStatus	1163	CancelRecurringOrderIdInvalid
1126	CaptureTrxnAmountCaptureCutOffExpired	1164	CancelRecurringAuthenticationFailure
1127	CaptureTrxnAmountPartialCaptureNotAllowed	1165	CancelRecurringAuthenticationInvalid
1128	RefundAmountGeneralError	1166	CancelRecurringAuthenticationUnknownError
1129	Refund Amount Authentication Process General Error	1167	CancelRecurringGeneralError
1130	Refund Amount Authentication Processing Error	1168	CancelRecurringAuthenticationProcessGeneralError
1131	Refund Amount Authentication Unknown Error	1169	CancelRecurringInvalidTransaction
1132	Refund Amount Authentication Failure	11999	CRR General error

* - This is the only Response Reason Code you will see with Response Code 1. All others will be with Response Code 3.

7) Fraud Control Response and Reason Codes

Main FraudControl ResponseCode

The ResponseCode, ReasonCode and ReasonCodeDescription fields of the FraudControlResponse message can hold the following code and description combinations. Fraud Control Response and ReasonCodes may also contain values specified in the Authorization section.

Response Code	Description
1	Fraud Check Successful
2	Decline
3	Error

Third Party FraudResponseCode

These are the actual response codes returned by the Fraud System (third party or FAC)

Third Party Response Code	Third Party System	Description
A	Kount	Authorize
D	Kount	Decline
R	Kount	Review
E	Kount	Escalate
[Various]	PayTrue	See PayTrue Documentation
B	BinCheck	BinCheck decline based on merchant rules and BinCheck data
91	All	Timeout
12	All	Invalid transaction - FraudControl is not enabled for merchant (FOnly)
99	All	Error

FraudControl Decline or Error Codes

Response Code 1 has no ReasonCode or ReasonCodeDescription for Fraud Control.

ReasonCode	ReasonCodeDescription	Details
321	BAD_EMAL	The email address does not meet required format or is
Same as above for all Kount errors – see Kount documentation for the complete list		
2001	Merchant Not Enabled	FraudControl is not enabled for this merchant.
2002	Invalid Fraud Profile	Merchant settings do not specify a valid fraud profile
2003	Missing MerchantId	Could not find fraud-specific MerchantID for this merchant
2004	Invalid Fraud Response	Response from Fraud system was invalid
2005	FraudCheckOnly Not Supported	FraudCheckOnly transactions are not supported with the current merchant configuration.
2006	Simulated Fraud Response	Fraud Response Codes and Score are simulated. For testing only.
2007	BinCheck System Error	BinCheck System Error
2020	FraudControl Decline	FraudControl query succeeded without error but the transaction declined as it did not pass the fraud check rules based on Kount response code.
2021	BinCheck Decline	BinCheck was successful but the transaction declined as it did not pass the bincheck rules based on BIN data.
2091	Response Time Out	Timeout waiting for Fraud System Response or communications error
2091	Response Timeout	Timeout waiting for Fraud System Response or communications error
2097	Format Error (Various)	Various format errors. Details will be in the description.
2096	FraudControl System Error	FraudControl System Exception
2099	FraudControl System Error	FraudControl Internal Error

Appendix C

Test Card Information

The following is a list of test cards you can use to receive specific responses for testing purposes.

It is important to note the following:

- These test cards do not apply to \$0 AVS-Only testing. For \$0 AVS-Only transaction testing, use real credit cards on the production platform so as to get valid AVSResult data.
- Any valid (MMYY, not expired) expiry date and any 3 digit CVV2 value will work for these test cards.
- Note: “Normal Approval” means ResponseCode=1, ReasonCode=1 and “Normal Decline” means ResponseCode=2, ReasonCode=2 in the web responses returned for Auth only and Auth/Capture transactions.
- All card numbers not listed above are defaulted to Normal Approval.
- For every approved transaction you will receive the same dummy authorization ID of 123456.
- These cards are **only to be used in the test environment** (ecm.firstatlanticcommerce.com). Once you are on the production platform, **live** cards must be used.
-
-
- FAC does not provide 3DS-enrolled test cards. To test 3DSecure full authentication, use real valid 3DS-enrolled cards.
-

Visa

Card Number	Response
4111111111111111	Normal Approval, CVV2Result=M
4111111111112222	Normal Approval, CVV2Result=N
4333333333332222	Normal Approval, CVV2Result=U
4444444444442222	Normal Approval, CVV2Result=P
4555555555552222	Normal Approval, CVV2Result=S
4666666666662222	Normal Decline, OriginalResponseCode=05, CVV2Result=N
4111111111113333	Normal Decline, OriginalResponseCode=85, AVSResult=M, CVV2Result=N
4111111111114444	Normal Approval, AVSResult=M
4111111111115555	Normal Approval, AVSResult=A
4111111111116666	Normal Approval, AVSResult=Z
4111111111117777	Normal Approval, AVSResult=N
4111111111118888	Normal Decline, OriginalResponseCode=85, AVSResult=G, CVV2Result=N
4111111111119999	Normal Decline, OriginalResponseCode=98
4111111111110000	Normal Decline, OriginalResponseCode=91
4222222222222222	Normal Approval, CVV2Result=M, AVSResult=N

MasterCard

Card Number	Response
5111111111111111	Normal Approval, CVV2Result=M
5111111111112222	Normal Approval, CVV2Result=N
5333333333332222	Normal Approval, CVV2Result=U
5444444444442222	Normal Approval, CVV2Result=P
5555555555552222	Normal Approval, CVV2Result=S
5555666666662222	Normal Decline, OriginalResponseCode=05, CVV2Result=N
5111111111113333	Normal Decline, OriginalResponseCode=05
5111111111114444	Normal Approval, AVSResult=Y
5111111111115555	Normal Approval, AVSResult=A
5111111111116666	Normal Approval, CVV2Result=M, AVSResult=Z
5111111111117777	Normal Approval, CVV2Result=M, AVSResult=N
5111111111118888	Normal Approval, CVV2Result=N, AVSResult=U
5111111111119999	Normal Decline, OriginalResponseCode=98
5111111111110000	Normal Decline, OriginalResponseCode=91
5222222222222222	Normal Approval, CVV2Result=N, AVSResult=U

American Express

Card Number	Response
3411111111111111	Normal Approval, CVV2Result=M
3422222222222223	Normal Approval, CVV2Result=N
3433333333333335	Normal Approval, CVV2Result=U
3444444444444447	Normal Approval, CVV2Result=P
3455555555555553	Normal Approval, CVV2Result=S
3466666666666665	Normal Decline, OriginalResponseCode=05, CVV2Result=N
341111111111129	Normal Decline, OriginalResponseCode=05
341111111111145	Normal Approval, AVSResult=A
341111111111152	Normal Approval, AVSResult=Z
341111111111160	Normal Approval, AVSResult=N
341111111111186	Normal Decline, OriginalResponseCode=98
341111111111194	Normal Decline, OriginalResponseCode=91
341111111111210	Normal Approval, CVV2Result=M, AVSResult=N