Web Services API Overview

fcB2B Technical Team

Version 2.1 - Revised May 13, 2014

fcB2B
Floor Covering Business to Business Association

Web Services API Overview

Table of Contents

Web Services API Overview

Introduction

This document covers the general characteristics and requirements of Floor Covering Business to Business Association Web Services. Web Services are intended to facilitate a common framework which suppliers and their customers can use to establish simple and low-cost electronic data for software applications. This guide provides information common to all services. Individual documents for each service set are referenced below.

Conventions and Notations

The domain name examples presented in this document conform to RFC2606 **[RFC2606]**.

A **Producer** is a server, provider, software solution, or service container that implements the APIs detailed in this specification. It can be thought of as a hub.

A **Consumer** is a client, software solution or interface to a system which uses the APIs detailed in this specification. It can be thought of as a spoke.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC2119 **[RFC2119]**.

Any implementation is not compliant if it fails to satisfy one or more of the MUST or REQUIRED requirements.

What is a Web Service?

Web services are application programming interfaces (APIs) that are accessed via Hypertext Transfer Protocol (HTTP) and executed on a remote system hosting the requested services. Web services most often fall into one of two classifications: "big" or standards-based Web services (typically using such standards as WSDL, SOAP, and XML-Schema among others) and RESTful Web Services.

While nothing in the detailed specification for the service sets precludes using "big" web services for implementation, this document discusses RESTful Web Services.

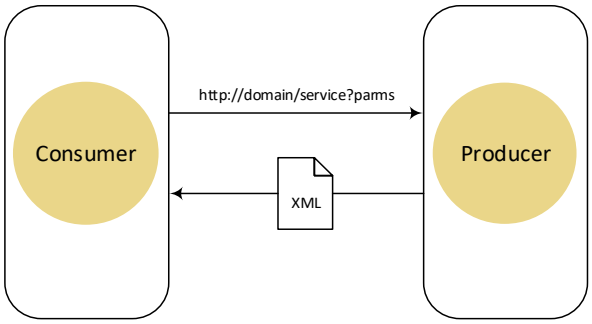

Figure 1 – Producer/Consumer Interaction

REST Described

Representational State Transfer (REST) is a software architecture style introduced and defined in 2000 by Roy Fielding **[Roy Fielding] [Dissertation]**, one of the principal authors of the Hypertext Transfer Protocol (HTTP) specifications 1.0 and 1.1. The REST style was developed in parallel with HTTP/1.1 and can be considered one of the features that made the World Wide Web a success. REST principles follow those of the Web in that all interactions are performed through the standard HTTP operations of GET, PUT, POST, DELETE, etc. Systems conforming to the principles of REST are commonly referred to as being "RESTful".

Put simply, a RESTful Web Services uses HTTP to request a specific URL, passing URL parameters as arguments to the services. The services return XML or JSON for processing by the consuming application.

REST Request Format

A REST request consists of a Uniform Resource Identifier (URI) in the form:

```
http://example.com:8042/over/there?name=ferret#nose
\___/  \_____________/ \_______/  \_________/ \___/
  |          |              |          |        |
scheme     authority       path       query   fragment
```

The **Producer** implementation, most likely on a specific application and web server, will determine the authority in its configuration.
The implementation MAY require a different form of the path than that contained in this specification. The implementation MUST support the query strings as contained in this specification.

REST services operate primarily in terms of resources and operations on them. The standard HTTP methods (GET, POST, PUT, DELETE, etc.) are used to retrieve and

change server state. A service and a resource can be considered synonymous. Services MAY or MAY NOT implement the ability to change the state of the server via POST, PUT or DELETE.

REST Response Format

No single data representation is ideal for every client. This specification currently defines representations for each resource in the XML format.

- Implementations MUST support the XML format.
- Implementations MAY support other widely support formats like JSON and Atom / AtomPub.

Status and Error Handling

In general, a REST **Producer** will use the HTTP status codes referenced in the HTTP/1.1 **[RFC2616]** specification to denote the status of results from a service and **Consumer** applications MUST be prepared to parse and act upon these codes.

When there are errors or additional status information related to a request the **Producer** MUST provide a response with details conforming to the **fcB2B-MessageList.xsd** schema appended to the XML response and set the HTTP Content-Type to **application/xml**.

The following lists detail the base HTTP status codes along with the common fcB2B status codes. Additional service specific codes are detailed in their respective documentation. REST-style services which *change* the state of a resource may use additional codes.

HTTP Status Codes

HTTP status code	Meaning
200 (“OK”)	The requested service returned successfully; the consuming application must parse the response content to determine the exact nature of the response, for example no material may be available on the requested date, but the return status is considered OK *because* the service responded.
301 (“Moved Permanently”)	This will occur if the resource has moved, for instance if a new version of the service has been implemented, the old one removed, and a client expecting the old location is improperly used. This is a *non-recoverable* error.

HTTP status code	Meaning
400 ("Bad Request")	This signifies a client error. There is something about the request that cannot be fulfilled because of invalid or missing data – for instance, a properly formed request for data about a product which does not exist has been presented to the server. When a 400 status code is returned, the response body **MUST** contain an instance of the "fcB2B-Status-2.0.xsd" schema. This is a *recoverable* error.
404 ("Not Found")	Resource not found. This will typically because the sub-service requested (e.g. "stockcheck") does not exist, was misspelled, or is not supported in this implementation of the service. It may also occur if the server itself is not available. This is a *non-recoverable* error.
403 ("Forbidden")	This will occur if the requested resource is protected by security and the consuming application failed to provide the correct security credentials. For instance, The request signature the server calculated does not match the signature the client provided, or the (optional) request time was too skewed from the time stamp contained in the request. This is a *recoverable* error, but it will NOT help to repeat this request without changing it.
500 ("Internal Server Error")	Internal server error. This means that the server is available, but the service for some reason could not fulfill the request. The service may be experiencing technical difficulties such as the backend database or the application server behind a fronting web server not being available. This is a *recoverable* error.
503 ("Service Unavailable")	The service is temporarily unavailable, most likely due to an overloading situation, for instance a high rate of requests for product data; the request rate should be reduced. This is a *recoverable* error.

Standard fcB2B Status Codes

The following table lists the standard fcB2B REST status and response codes.

Status Code	Severity	Description	HTTP Status Code
AccessDenied	Error	Access Denied	403 Forbidden
AccountProblem	Error	There is a problem with your fcB2B account that prevents the operation for completing successfully. Provide link to more information.	403 Forbidden
CredentialsNotSupported	Error	This request does not support credentials.	400 Bad Request
ExpiredToken	Error	The provided token has expired.	400 Bad Request
InvalidVersion	Error	The current version configuration specified in the request is invalid.	400 Bad Request
IncompleteBody	Error	You did not provide the correct number of bytes in the message specified by the Content-Length HTTP header	400 Bad Request
InternalError	CritcalError	Internal Error. Please try again later.	500 Internal Server Error
InvalidArgument	Error	Invalid Argument	400 Bad Request
InvalidClientIdentifier	Error	The client identifier does not match our records.	403 Forbidden
InvalidCredentials	Error	The credentials provided are invalid.	403 Forbidden
InvalidTransport	Error	This service must be accessed over a secure HTTPS connection.	400 Bad Request
InvalidToken	Error	The provided token is malformed or otherwise invalid.	400 Bad Request
InvalidURI	Error	The specified URI could not be parsed	400 Bad Request
InvalidXML	Error	The XML you provided was not well-formed or did not validate against our published schema.	400 Bad Request

Status Code	Severity	Description	HTTP Status Code
MalformedPOSTRequest	Error	The body of your POST request is not well-formed multipart/form-data.	400 Bad Request
MaxMessageLengthExceeded	Error	Your request was too big.	400 Bad Request
MethodNotAllowed	Error	The specified method is not allowed against this resource.	405 Method Not Allowed
MissingContentLength	Error	You must provide the Content-Length HTTP header.	411 Length Required
MissingRequestBodyError	Error	This happens when the user sends an empty xml document as a request. The error message is, "Request body is empty."	400 Bad Request
MissingSecurityInfo	Error	Your request was missing a required security element.	400 Bad Request
NoSuchVersion	Error	Indicates that the version ID specified in the request does not match an existing version.	404 Not Found
NotImplemented	Error	Header or request you provided implies functionality that is not implemented.	501 Not Implemented
NotSignedUp	Error	Your account is not signed up for use of the requested service. Provide a link to the sign-up url or more information.	403 Forbidden
OperationAborted	Error	A conflicting conditional operation is currently in progress against this resource. Please try again.	409 Conflict
PermanentRedirect	Warning	The URL you are attempting to access must be addressed using the specified endpoint. Please send all future requests to this endpoint.	301 Moved Permanently
Redirect	Information	Temporary redirect.	307 Moved Temporarily

Status Code	Severity	Description	HTTP Status Code
RequestTimeout	Error	Your socket connection to the server was not read from or written to within the timeout period.	400 Bad Request
RequestTimeTooSkewed	Error	The difference between the request time and the server's time is too large.	403 Forbidden
SignatureDoesNotMatch	Error	The request signature we calculated does not match the signature you provided. Check your Secret Key and signing method. For more information, see REST Authentication.	403 Forbidden
SlowDown	Warning	Please reduce your request rate.	503 Slow Down
TooManyRequests	Error	Too many requests were made after warning so service usage suspended.	503 Service Unavailable
UnexpectedContent	Error	This request does not support the supplied content.	400 Bad Request

Severity Codes / Enumeration Values

Severity Code	Description
Information	Informational Message (No action required)
Warning	Warning Message (Possible action required)
Error	Error Encountered (Failure and action most likely required)
CritcalError	Critical error or significant non-recoverable failure

fcB2B MessagesList Schema

The fcB2B MessageList Schema is provided as a response to all requests to indicate problems or other information related to the status of a request.

fcB2B MessageList Schema: fcB2B-MessageList.xsd

Name	Description
Message	Root message element
StatusCode	Status Code from the Common or Service-Specific code lists

Name	Description
Severity	Severity Code from the Severity Codes / Enumeration Values list.
Description	Short description related to the reason for the status being returned.
Parameters	Root element containing a list of name/value pairs containing additional information related to the status message as described below
Name	The parameter name
Value	The parameter value as text

The MessageList schema provides a list of extension parameters as a name/value pair. The parameter list is intended to allow for additional status information in **Producer** responses that has not been defined by this specification. It is not intended to act as a functional extension to the fcB2B Request and Response schemas. The following is a list of the recommended reserved fcB2B parameter names:

Name	Description
Reason	Reason for the error or status message
Resource	The resource that generated the status or error message for assistance in troubleshooting
URL	URL related to this error
LinkToMoreInformation	URL with additional information related to this message
ArgumentName	Name of the argument responsible for the message Example: **SupplierItemSKU**
ArgumentValue	Value contained in the argument responsible for the message Example: **NULL**

Content Type

It is recommended that the standard HTTP header for POST and PUT verbs "Content-Type" for REST applications always use a value of "application/xml" or "application/json" (depending on the type of the content).

Security

There are three aspects in general to securing web services:

- Assuring the privacy of the document payload
- Verifying the identity of the requester
- Assuring that the request itself originated from the verified requester (i.e. has not been "spoofed" or tampered with)

There are many ways to achieve these for different protocols, i.e. differing for REST vs. WS* Web Services. WS-Security and related standards MAY be supported by **Producer** implementations for WS* Web Services but are not detailed in this specification.

The currently adopted security configuration for fcB2B is:

- Server-side **HTTP/HTTPS** allowed for Discovery and Stock Check services
- Server-side **HTTPS** transport layer security is required for all services other than the base Discovery and Stock Check services
- Request signing

HTTPS

Using server-side HTTPS with a Web Service API means that a **Consumer** system must be able to form a URL of the form https://rest.example.com but also that the underlying platform or application software must provide support for HTTPS as a client. The **Producer** system must be configured to handle secure HTTPS requests and responses, and must have an appropriate digital certificate configured. The details of such configuration are beyond the scope of this document.

Request Signing

Request signing means that a *signature* is added as a request parameter which is generated using a private (secret) key shared between the **Producer** and **Consumer** systems. The signature is a cryptographic transformation of the rest of the request. The secret key will be associated in each system with a clear-text identity, which itself must be included as a request parameter in order to look up the secret key. The **Producer** system will simply use the same algorithm that the **Consumer** system used to sign the request and compare the resulting signature to that which is passed in the request. Therefore request signing assures that the request was made by the verifiable identity and that it has not been tampered with in transit.

Request signing differs slightly between the GET and POST HTTP verbs because the with GET there is a request query string, while with post there is no request query string, parameters are carried as HTTP headers and the request document is carried in the body of the POST.

The reference implementation of request signing for fcB2B Web Services is based on Amazon Web Services request signing standards.

The reference implementation uses HMAC-SHA-256, meaning that the signature is a Hash-based Message Authentication Code, in which the hash is computed using a Secure Hash Algorithm of 256 bits.

IMPORTANT NOTE: Common to both GET and POST verb implementations is the fact that the endpoint used in signing and verifying the signature is URI of the *first HTTP* or HTTPS receiver of the request, *that which the sender was given to point to*. Thus if an environment has a content router or load balancer in which an incoming request is routed to multiple web servers to be fulfilled, the endpoint used in signing is the first place a request "hits" in the infrastructure, which may be different than the endpoint that finally processes the request. This is to prevent spoofing of requests. In the examples below, an HTTP trace tool is in use. Its URI is "localhost:7070" but the final URI which processes the request is the endpoint to which the tunnel tool points, "localhost:9090". ***"localhost:7070" must be used for both the signing and verification endpoint.***

GET Verb

Request signing for the GET verb is based on Version 2 of the AWS request signing standards.

It has the following features:

- a query parameter named "apiKey" MUST be present in the request
  - apiKey MUST be able to be linked within the **Producer** and **Consumer** systems to the private/secret shared key used for signing and verification
  - apiKey MAY be the same as the clientIdentifier but need not be (e.g. the clientIdentifier may serve to identify a sub-entity used in business logic)
- a query parameter named "Timestamp" will be added to the signed request
  - Timestamp MAY be used by the **Producer** system to reject a request received outside an agreed window from the timestamp, thus precluding bookmarking or "replay" attacks

In practice, the request to be signed will be formed as a string; this string MUST consist of a series of name/value pairs, with the pairs being delimited by the "&" character and the name/value within each pair being delimited by the "=" character. This is of course the form of a query string in an HTTP "GET" verb. However this form will be used for *all* HTTP verbs in REST requests, so for "POST" and "PUT" verbs, the request attributes normally passed as either request headers or request body will be temporarily formed into a pseudo-query-string for the purposes of signing. This allows the exact same algorithm to be used to sign all requests regardless of the HTTP verb.

NOTE: The request body (e.g. an XML or JSON document) in a "POST" or "PUT" request will be formed into a pseudo-query-string by simply creating a name/value pair whose name is "Body" and whose value is the document content.

There are no specific requirements placed on the construction of the secret key, but it is recommended that it at least follow the conventions of strong passwords.

The reference implementation performs the following steps:

1. Split the original query string on its "=" signs into a map of the name/value pairs of the parameters and URL-decode each (in case they were previously URL-encoded)
2. Create an ISO 8601 timestamp and add it to the map
3. Add the required "apiKey" to the map
4. Canonicalize the map – that is, perform a lexicographical sort on the map, that is a sort which takes into account case
5. URL-encode each map entry using UTF-8 character encoding
6. Re-create the query string format from the map
7. Create a "string-to-sign" consisting of:
    - The HTTP verb "GET", "POST", "PUT" or "DELETE" followed by a newline character (\n)
    - Plus the authority portion of the request (less http://) plus a newline character
    - Plus the path plus a newline character
    - Plus the query string that has been lexicographical sorted in step 6 but do not add the newline character
8. Execute the HMAC-SHA-256 algorithm on the total string from step 7
9. URL-encode the signature string from step 6
10. Assemble the final string consisting of http(s):// plus the URI plus the request path plus the query string plus the signature

For example:

- GET request with query string
- apiKey = ABC12345
- secretKey = ABC@12&68
- endpoint = localhost:8080
- path = fTech/stockcheck
- request parameters:
    - SupplierItemSKU=ACBBFFFGNTL2
    - ClientIdentifier = C12345

The input query string will be:
"SupplierItemSKU=ACBBFFFGNTL2&ClientIdentifier=C12345"

Completing steps 1 thru 5 will result in an encoded, sorted map of:

ClientIdentifier=C12345
SupplierItemSKU=ACBBFFFGNTL2
Timestamp=2011-01-22T23%3A03%3A48.000Z
apiKey=C12345

Steps 6 & 7 will result in:

GET
localhost:8080
/fTech/stockcheck
ClientIdentifier=C12345&SupplierItemSKU=ACBBFFFGNTL2&Timestamp=2011-01-22T23%3A20%3A51.000Z&apiKey= ABC12345

Steps 8 & 9 will produce a signature of:
Signature=XJaO%2F2EJilYlzqIh0CqI4zQ6pD8vsL6oH%2Bxf3svx56w%3D

The final request string (without the word wrap as seen below) will then be:

http://localhost:8080/fTech/stockcheck?ClientIdentifier=C12345&SupplierItemSKU=ACBBFFFGNTL2&Timestamp=2011-01-22T23%3A32%3A12Z&apiKey=ABC12345&Signature=XJaO%2F2EJilYlzqIh0CqI4zQ6pD8vsL6oH%2Bxf3svx56w%3D

Version Management

There are three alternatives for handling versions in REST services.

- Make a unique resource URI for each version of a resource (e.g. http://rest.example.com/v1/stockcheck)
- Use HTTP protocol headers to convey version (e.g. include version parameters in the HTTP request and response headers)
- Provide a version number as a request parameter

Some authorities feel that it "breaks" the REST architecture to embed versions in the URI. This is a matter of taste, but it is also potentially a change management challenge for application server configuration. The use of HTTP protocol headers can be technically daunting. Therefore, the recommended implementation is to provide an optional "Version" parameter in the requests and provide a version attribute in the response schema.

Version Numbers, Namespaces and Compatibility

In general, there are two kinds of changes to data schema:

- **Compatible** – an optional addition to content which a legacy **Consumer** can ignore
- **Incompatible** – a mandatory addition to content or change in structure which a legacy **Consumer** cannot ignore

Compatible changes MUST be identified by a minor version or 2nd digit increment in the version number, e.g. "1.0" becomes "1.1". The target namespace of the document schema MUST remain the same, e.g. if it was "1.0" before the change it will be "1.0" after the change.

Incompatible changes MUST be identified by a major version or 1st digit increment in the version number, e.g. "1.0" becomes "2.0". The target namespace of the document schema MUST be changed according to the major version change, e.g. if it was "1.0" before the change it must become "2.0" after the change.

NOTE: While compatible changes in theory can be ignore by a legacy **Consumer**, in practice depending on the exact nature of the document processor in the implementation they may not. For example, if an addition is made to an enumeration and it is only sent to new **Consumers** and not to legacy **Consumers**, the legacy Consumer can successfully handle the change. If the new enumeration is sent to all **Consumers** however, and if the consuming processor validates the document against the old schema, it may break. In practice therefore it may be necessary to assure the optional content in a compatible change is not sent to legacy **Consumers**.

Service Discovery

Service Discovery is a means by which a **Consumer** can discover which services are implemented by a given **Producer** and the implementation details of each service. The **Consumer** calls the Service Discovery service and gets a response detailing the list of services the **Producer** supports. It is recommended that **Producer** implementations support Service Discovery. A **Producer** implementation MAY depart from the path portion of the following URI for service discovery if required.

Discovery Schema: DiscoveryResponse.xsd

Request

http://rest.example.com/services

Response

Commented [TG1]: Need to update based on new schema with wrapping elements

```
<?xml version="1.0" encoding="UTF-8"?>
<ServiceProfiles
  xsi:schemaLocation="http://fcb2b.com/schemas/2.0/core DiscoveryResponse.xsd"
  xmlns="http://fcb2b.com/schemas/2.0/core" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <ServiceProfile schemaVersion="2.0">
    <Name>StockCheck</Name>
    <Description>Basic unconstrained single item request</Description>
    <Versions>
      <Version version="1.0" date="2010-08-01">
        <DefaultNamespace>http://fcb2b.com/schemas/1.0/inventoryServices</DefaultNamespace>
        <InputSchema>InventoryInquiryRequest.xsd</InputSchema>
        <OutputSchema>InventoryInquiryResponse.xsd</OutputSchema>
        <OutputSchemaLocaton>InventoryInquiryResponse.xsd</OutputSchemaLocaton>
        <HTTPRequestPath>http://localhost:8079/fTech/stockcheck</HTTPRequestPath>
        <HTTPSRequestPath>https://localhost:8442/fTech/stockcheck</HTTPSRequestPath>
      </Version>
      <Version version="2.0" date="2014-08-01">
```

```
        <DefaultNamespace>http://fcb2b.com/schemas/2.0/InventoryServices</DefaultNamespace>
        <InputSchema>InventoryInquiryRequest.xsd</InputSchema>
        <OutputSchema>InventoryInquiryResponse.xsd</OutputSchema>
        <OutputSchemaLocaton>InventoryInquiryResponse.xsd</OutputSchemaLocaton>
        <HTTPRequestPath>http://localhost:8080/fTech/stockcheck</HTTPRequestPath>
        <HTTPSRequestPath>https://localhost:8443/fTech/stockcheck</HTTPSRequestPath>
      </Version>
    </Versions>
    <AnonymousAccessPermitted>true</AnonymousAccessPermitted>
  </ServiceProfile>
  <ServiceProfile>
    <Name>InventoryInquiry</Name>
    <Description>Basic unconstrained multiple item request</Description>
    <Versions>
      <Version version="2.0" date="2014-08-01">
        <DefaultNamespace>http://fcb2b.com/schemas/2.0/InventoryServices</DefaultNamespace>
        <InputSchema>InventoryInquiryRequest.xsd</InputSchema>
        <OutputSchema>InventoryInquiryResponse.xsd</OutputSchema>
        <OutputSchemaLocaton>InventoryInquiryResponse.xsd</OutputSchemaLocaton>
        <HTTPRequestPath>http://localhost:8080/fTech/inventoryinquiry</HTTPRequestPath>
        <HTTPSRequestPath>https://localhost:8443/fTech/inventoryinquiry</HTTPSRequestPath>
      </Version>
    </Versions>
    <AnonymousAccessPermitted>false</AnonymousAccessPermitted>
  </ServiceProfile>
  <ServiceProfile>
    <Name>RelatedItems</Name>
    <Description>Basic unconstrained related item request</Description>
    <Versions>
      <Version version="2.0" date="2014-08-01">
        <DefaultNamespace>http://fcb2b.com/schemas/2.0/InventoryServices</DefaultNamespace>
        <InputSchema>InventoryInquiryRequest.xsd</InputSchema>
        <OutputSchema>InventoryInquiryResponse.xsd</OutputSchema>
        <OutputSchemaLocaton>InventoryInquiryResponse.xsd</OutputSchemaLocaton>
        <HTTPRequestPath>http://localhost:8080/fTech/relateditems</HTTPRequestPath>
        <HTTPSRequestPath>https://localhost:8443/fTech/relateditems</HTTPSRequestPath>
      </Version>
    </Versions>
    <AnonymousAccessPermitted>false</AnonymousAccessPermitted>
  </ServiceProfile>
  <ServiceProfile>
    <Name>SolutionCheck</Name>
    <Description>Constrained single item request</Description>
    <Versions>
      <Version version="2.0" date="2014-08-01">
        <DefaultNamespace>http://fcb2b.com/schemas/2.0/InventoryServices</DefaultNamespace>
        <InputSchema>InventoryInquiryRequest.xsd</InputSchema>
        <OutputSchema>InventoryInquiryResponse.xsd</OutputSchema>
        <OutputSchemaLocaton>InventoryInquiryResponse.xsd</OutputSchemaLocaton>
        <HTTPRequestPath>http://localhost:8080/fTech/solutioncheck</HTTPRequestPath>
        <HTTPSRequestPath>https://localhost:8443/fTech/solutioncheck</HTTPSRequestPath>
      </Version>
    </Versions>
    <AnonymousAccessPermitted>false</AnonymousAccessPermitted>
  </ServiceProfile>
  <ServiceProfile>
    <Name>SolutionInquiry</Name>
    <Description>Constrained multiple item request</Description>
    <Versions>
      <Version version="2.0" date="2014-08-01">
        <DefaultNamespace>http://fcb2b.com/schemas/2.0/InventoryServices</DefaultNamespace>
        <InputSchema>InventoryInquiryRequest.xsd</InputSchema>
        <OutputSchema>InventoryInquiryResponse.xsd</OutputSchema>
        <OutputSchemaLocaton>InventoryInquiryResponse.xsd</OutputSchemaLocaton>
        <HTTPRequestPath>http://localhost:8080/fTech/solutioninquiry</HTTPRequestPath>
        <HTTPSRequestPath>https://localhost:8443/fTech/solutioninquiry</HTTPSRequestPath>
      </Version>
```

```
    </Versions>
    <AnonymousAccessPermitted>false</AnonymousAccessPermitted>
  </ServiceProfile>
</ServiceProfiles>
```

Testability

It is recommended to use the reference implementation of request signing in concert with the well-known client known as "cURL".
cURL is available for free download from http://curl.haxx.se/.
The idea as will be shown in the examples is to run the request signing helper and then paste its output into the command line for cURL. There are more elaborate ways to do this as well, but using this technique is a good way to get started.

Examples

Example 1 – Simple GET Query

Query string: SupplierItemSKU=ACBBFFFGNTL2&ClientIdentifier=C12345

Request signing command:
java -cp commons-codec-1.4.jar:. org.fcb2b.utilities.RequestSigningHelper -s http -e "localhost:7070" -k "ABC12345" -x "ABC@12&68" -p "/fTech/stockcheck" -q "SupplierItemSKU=ABCBBFFFGNTL2&ClientIdentifier=C12345"

Output:
GET should simply use the signed request as URL:
http://localhost:7070/fTech/stockcheck?ClientIdentifier=C12345&SupplierItemSKU=ABCBBFFFGNTL2&Timestamp=2011-01-25T02%3A52%3A50Z&apiKey=ABC12345&Signature=4aDv3Gqt9VJGLqlVqqR%2BSvyMgY8eb7RmBliGeEvugU8%3D
Previous request verified: true

cURL command:
c:\Tools\cUrl\curl
http://localhost:7070/fTech/relateditems?ClientIdentifier=C12345&SupplierItemSKU=ABCBBFFFGNTL2&Timestamp=2013-04-07T20%3A01%3A54Z&apiKey=ABC12345&Signature=j365YGiZZ0CzJ0%2Fa5LRiSZwjQIAkcaoXltzXeehOwJ4%3D

Raw HTTP traffic resulting from above request:
GET
/fTech/relateditems?ClientIdentifier=C12345&SupplierItemSKU=ABCBBFFFGNTL2&Timestamp=2013-04-

07T20%3A01%3A54Z&apiKey=ABC12345&Signature=j365YGiZZ0CzJ0%2Fa5LRiSZwjQIAkcaoXltzXeehOwJ4%3D HTTP/1.1
User-Agent: curl/7.20.0 (i386-pc-win32) libcurl/7.20.0 OpenSSL/0.9.8l zlib/1.2.3
Host: localhost:7070
Accept: */*

NOTE: You will actually want to test using HTTPS, and that is done merely by changing the URL scheme to https instead of http (and of course assuming that HTTPS is properly configured on the application/web server).
c:\Tools\cUrl\curl "https…"
You are likely to find that cURL throws errors because it does not know how to trace the server-provided SSL certificate to a root CA (Certification Authority). There are several ways to resolve that robustly and correctly, but a workaround is to tell cURL to be "insecure", that is to accept the SSL server certificate without verification. This is done with the "-k" option – see the cURL documentation for further details.
c:\Tools\cUrl\curl –k "https…"

The output of executing the command this way cannot be shown because the raw protocol for obvious reasons is not human readable.

Appendix A – Reference Implementation of Request Signing

Following is the API in Javadoc form for the reference implementation.

Package **Class** Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

org.fcb2b.utilities
Class RequestSigningHelper
java.lang.Object
└ **org.fcb2b.utilities.RequestSigningHelper**

public class **RequestSigningHelper**
extends java.lang.Object

Helper class for signing of fcB2B REST requests

Adapted from Amazon Web Services request signing version 2

Version:
1.1
Author:
William H. Hutchinson

Field Summary

protected java.lang.String	signature
protected java.lang.String	timeStamp

Constructor Summary

RequestSigningHelper(java.lang.String scheme, java.lang.String requestMethod, java.lang.String endpoint, java.lang.String apiKey, java.lang.String secretKey, java.lang.String path)
Creates instance of RequestSigningHelper after checking for all required inputs

Method Summary	
java.lang.String	getSignature() Gets just generated timestamp
java.lang.String	getTimeStamp() Gets just generated signature
static void	main(java.lang.String[] args) Main method for usable for testing but not in production
java.lang.String	sign(java.lang.String requestString)
boolean	verify(java.lang.String checkSignature, java.lang.String checkRequestString) This method does the verification of a signed query NOTE!!: requires that the OLD timestamp is already a parameter and will fail if it is not Gets users information based on user id.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

timeStamp

```
protected java.lang.String timeStamp
```

---

signature

```
protected java.lang.String signature
```

Constructor Detail

RequestSigningHelper

```
public RequestSigningHelper(java.lang.String scheme,
                            java.lang.String requestMethod,
```

```
                        java.lang.String endpoint,
                        java.lang.String apiKey,
                        java.lang.String secretKey,
                        java.lang.String path)
                 throws java.lang.IllegalArgumentException,
                        java.io.UnsupportedEncodingException,
                        java.security.NoSuchAlgorithmException,
                        java.security.InvalidKeyException
```

Creates instance of RequestSigningHelper after checking for all required inputs

Parameters:

`scheme` - aka protocol, HTTP|HTTPS

`requestMethod` - GET|POST|PUT|DELETE

`endpoint` - aka authority - network location e.g. www.fcb2b.org plus port if any

`apiKey` - public key string (probably not X509 certificate) by which identify of requester is known

`secretKey` - key string (probably not X509 private key) used to sign request, linked in some way in producer system to apiKey this MUST be carefully protected to assure non-repudiation

`path` - the "rest" of the URL after the network location but NOT including the query string

Throws:

`java.lang.IllegalArgumentException`
`java.io.UnsupportedEncodingException`
`java.security.NoSuchAlgorithmException`
`java.security.InvalidKeyException`

Method Detail

sign

```
public java.lang.String sign(java.lang.String requestString)
```

---

verify

```
public boolean verify(java.lang.String checkSignature,
                      java.lang.String checkRequestString)
```

This method does the verification of a signed query NOTE!!: requires that the OLD timestamp is already a parameter and will fail if it is not Gets users information based on user id.

Parameters:

`checkSignature` - the signature against which to compare

`checkRequestString` - the reconstructed request to re-sign for comparison

Returns:

boolean true if the signature verified

---

getTimeStamp

`public java.lang.String` **`getTimeStamp`**`()`

Gets just generated signature

Returns:

signature

---

getSignature

`public java.lang.String` **`getSignature`**`()`

Gets just generated timestamp

Returns:

timestamp

---

main

`public static void` **`main`**`(java.lang.String[] args)`

Main method for usable for testing but not in production

Usage is: "java -classpath classpath-on-your-system -s scheme(http|https) -m method(GET,POST,PUT,DELETE) -e network endpoint e.g. 'localhost:8080' -k apiKey -x secretKey -p path, e.g. '/fTech/stockcheck' -f optional content file -q request or query string, e.g. 'Foo=bar&Bi=bop'"

If -f is specified, text file will be read and request will be considered to be POST and processed that way, otherwise it will processed as a GET and full URL will be returned

It will also call the verify method and report on whether the signature successfully verifies

---

Package **Class** Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Appendix B – XML Schemas

DiscoveryResponse.xsd

fcB2B-MessageList.xsd

Commented [TG2]: Need Reference URL for all v2 Schemas

References

[Roy Fielding] "Roy Thomas Fielding", Wikipedia.
http://en.wikipedia.org/wiki/Roy_Fielding

[Dissertation] "Architectural Styles and the Design of Network-based Software Architectures", Roy Thomas Fielding, 2000.
http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[RFC20] "ASCII format for Network Interchange" RFC 20.
http://tools.ietf.org/html/rfc20

[RFC2119] "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, Bradner, S., March 1997. http://tools.ietf.org/html/rfc2119

[RFC2606] "Reserved Top Level DNS Names", RFC 2606, Eastlake, D. and A. Panitz, June 1999. http://tools.ietf.org/html/rfc2606

[RFC2616] "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616.
http://tools.ietf.org/html/rfc2616

[RFC3986] "Uniform Resource Identifier (URI): Generic Syntax", RFC 3986.
http://tools.ietf.org/html/rfc3986

[WS-*] "Web Services Specifications", Wikipedia.
http://en.wikipedia.org/wiki/List_of_Web_service_specifications

[HTTPS] "HTTP Secure", Wikipedia.
http://en.wikipedia.org/wiki/HTTP_Secure

[RFC2818] "Describes how to use TLS to secure HTTP connections over the Internet", RFC 2818.
http://tools.ietf.org/html/rfc2818

[HMAC] "Hash-based Message Authentication Code", Wikipedia.
http://en.wikipedia.org/wiki/HMAC

[RFC4868] "Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec", RFC4868.
http://www.rfc-archive.org/getrfc.php?rfc=4868

[SHA-2] "SHA-2", Wikipedia.
http://en.wikipedia.org/wiki/SHA-2

Amazon Signature Version 2 Signing Process
http://docs.aws.amazon.com/general/latest/gr/signature-version-2.html