Amadeus Web Services Implementation Guide

SOAP 4.0

Version 1.8

Web Services Consultancy

February 2019

Amadeus sas
485, Route du Pin Montard
Les Bouillides
B.P. 69
06902 Sophia Antipolis CEDEX
France

Index

_ Introduction 4

_ Prerequisites 5

- Office ID Settings 5

_ Concepts/Background 6

- Header structure 6

_ Password digest generation algorithm 12

_ Flows 15

- Context-less (Stateless) 15
- Context-full 17

_ Common errors 20

_ Code samples 21

Document control

Security level	Restricted			
Company	Amadeus IT Group SA			
Department	Web Services Consultancy			
Author	Katarzyna Rembowicz, Łukasz Siemiradzki, Andreas Bonk			
Reviewed by	Bindesh Shah		**Date**	27 / 10 / 2015
Approved by	Rafał Arndt		**Date**	05 / 11 / 2015
Version	**Date**	**Change**	**Comment**	**By**
1.0	30/09/2015	New document		Katarzyna Rembowicz
1.1	22/10/2015	Updates	New content	Łukasz Siemiradzki
1.2	27/10/2015	Updates	Additional content	Bindesh Shah
1.3	30/10/2015	Merge	Merging new content	Łukasz Siemiradzki
1.4	27/11/2015	Correction	Correct two samples with AgentDutyCode	Łukasz Siemiradzki
1.5	14/08/2017	Correction, Updates	Correcting samples, noded2, TLS1.2, terminology	Łukasz Siemiradzki
1.6	25/10/2017	Correction, Updates	Correction and update for Commmon errors, PHP and Perl samples.	Łukasz Siemiradzki
1.7	23/11/2017	Update	Noded3	Łukasz Siemiradzki
1.8	08/02/2019	Update	Links adjustments	Łukasz Siemiradzki

Introduction

This document provides basic information necessary to implement SOAP header 4.0 for Amadeus Web Services.

This document is primarily for developers, technical and functional consultant who plan for migration, development or evaluation of Soap Header version 4.0. It is assumed that the reader has basic knowledge about XML, SOAP protocol and travel industry domain.

In the scope of for this document the technical and functional features of SOAP header 4.0 are being presented.

Prerequisites

Application which will benefit from SOAP header 4.0 implementation, it does require specific set up of the Web Services Access Point (WSAP). By default, newly created WSAPs are set up to handle SOAP header 4.0 communication. For existing Application on SOAP Header 2.x, there are facilities to help customer migrate to SOAP header 4.0 functionality.

For additional query regarding SOAP header 4.0, please contact your Amadeus representative or dedicated implementation consultant.

Office ID Settings

Not relevant for SOAP header 4.0.

Concepts/Background

Amadeus Web Services uses SOAP communication protocol over HTTP/HTTPS to transfer SOAP messages The structure of SOAP message is following: the root node is Envelope which contains nodes: Header and Body; Response message, in case of problem may contain Fault node. This document will cover only Header node.

The references to other parts of the SOAP message may appear only to help better understand the topic.

This section provides details concerning the format and purpose of elements of SOAP header 4.0

Header structure

SOAP header 4.0 may contain following parts:

- WS-Addressing header
- Amadeus Session header
- WS-Security header
- Amadeus Security header

WS-Addressing header

WS-Addressing header is a W3C recommendation standard defined in http://www.w3.org/TR/ws-addr-soap/

For Amadeus Web Services following elements may be used:

- MessageID
- Action
- To
- From
- RelatesTo

These elements use stable namespace URI: `http://www.w3.org/2005/08/addressing`

MessageID element conveys the message identifier. Please note that this field always appear in the request and response. It should be different for each message. It is recommended to use Universally unique identifier (UUID). Example:

```
<wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">urn:uuid:3a145510-42f1-1e94-
f14e-ac4479bb043b</wsa:MessageID>
```

Action element conveys the operation identifier. Please note that this field always appear in the request and response. Available operations are defined in WSDL file in following XPath: /wsdl:definitions/wsdl:binding/wsdl:operation/soap:operation. Example:

```
<wsa:Action>http://webservices.amadeus.com/ABCDEF_XX_Y_1A</wsa:Action>
```

To element provides the URI of the destination endpoint. Please note that this field always appear in the request and response. In the request the value of this element is the same as endpoint, it can be obtained from WSDL file using following XPath: /wsdl:definitions/wsdl:service/wsdl:port/soap:address

In the response this element is always set to following URI: `http://www.w3.org/2005/08/addressing/anonymous`. Example:

```
<wsa:To>https://noded3.test.webservices.amadeus.com/1ASIWAAABBB</wsa:To>
```

From element conveys the URI of the source endpoint. Please note that this field always appear in the response. It contains element Address which copies the value of the To element of the request. Example:

```
<wsa:From><wsa:Address>https://noded3.test.webservices.amadeus.com/1ASIWAAABBB</wsa:Address><
/wsa:From>
```

RelatesTo element conveys the MessageID of the related message. Please note that this field always appear in the response. It copies the MessageID of the request. Example:

```
<wsa:RelatesTo
RelationshipType="http://www.w3.org/2005/08/addressing/reply">urn:uuid:3a145510-42f1-1e94-
f14e-ac4479bb043b</wsa:RelatesTo>
```

Sample WS-Addressing header of the request:

```
<soap:Header>
  <add:MessageID xmlns:add="http://www.w3.org/2005/08/addressing">917c4526-31cf-47c-ca1e-
e5015f962a82</add:MessageID>
  <add:Action
xmlns:add="http://www.w3.org/2005/08/addressing">http://webservices.amadeus.com/ABCXYZ_XX_Y_1
A</add:Action>
  <add:To
xmlns:add="http://www.w3.org/2005/08/addressing">https://noded3.test.webservices.amadeus.com/
1ASIWAAABBB</add:To>
  ... (other nodes are being omitted for clarity purpose) ...
</soap:Header>
```

Sample WS-Addressing header of the response:

```
<soap:Header>
  <wsa:To>http://www.w3.org/2005/08/addressing/anonymous</wsa:To>
  <wsa:From>
    <wsa:Address>https://noded3.test.webservices.amadeus.com/1ASIWAAABBB</wsa:Address>
  </wsa:From>
  <wsa:Action>http://webservices.amadeus.com/ABCXYZ_XX_Y_1A</wsa:Action>
  <wsa:MessageID>urn:uuid:3a145510-42f1-1e94-f14e-ac4479bb043b</wsa:MessageID>
  <wsa:RelatesTo RelationshipType="http://www.w3.org/2005/08/addressing/reply">917c4526-31cf-
47c-ca1e-e5015f962a82</wsa:RelatesTo>
  ... (other nodes are being omitted for clarity purpose) ...
</soap:Header>
```

Amadeus Session header

Amadeus Session header is an Amadeus extension defined within Session tag.

For Amadeus Web Services following elements may be used:

- SessionID
- SequenceNumber
- SecurityToken

and following attribute:

- TransactionStatusCode

These elements use namespace URI: `http://xml.amadeus.com/2010/06/Session_v3`.

Please note that these fields always appear in the response. These fields appear in the request if the session is statefull.

SessionID element conveys 10 character string which identifies the session. Example:

```
<awsse:SessionId
xmlns:awsse="http://xml.amadeus.com/2010/06/Session_v3">016CHRLMPV</awsse:SessionId>
```

SequenceNumber element conveys the counter which is set to "1" in the first reply and should be increased in each request. Example:

```
<awsse:SequenceNumber
xmlns:awsse="http://xml.amadeus.com/2010/06/Session_v3">1</awsse:SequenceNumber>
```

SecurityToken element conveys 26 character unique element generated by Amadeus when new SessionID is being created. Example:

```
<awsse:SecurityToken
xmlns:awsse="http://xml.amadeus.com/2010/06/Session_v3">3IE87GTUZ7T8B3551F3VNKL2J4</awsse:Sec
urityToken>
```

TransactionStatusCode attribute conveys the value which contains the status of the session. In the request it may contain one of the following values: "Start", "InSeries" or "End". In the response it may contain one of the following values: "InSeries" or "End".

Sample Session header of the request:

```
<soapenv:Header>
  ...
  <awsse:Session xmlns:awsse="http://xml.amadeus.com/2010/06/Session_v3"
TransactionStatusCode="InSeries">
    <awsse:SessionId>01HFHCODVI</awsse:SessionId>
    <awsse:SequenceNumber>2</awsse:SequenceNumber>
    <awsse:SecurityToken>1SP31VH87S9T310EWJIH27JLRI</awsse:SecurityToken>
  </awsse:Session>
</soapenv:Header>
```

Sample Session header of the response:

```
<soapenv:Header>
...
<awsse:Session TransactionStatusCode="End"
xmlns:awsse="http://xml.amadeus.com/2010/06/Session_v3">
  <awsse:SessionId>01HFHCODVI</awsse:SessionId>
  <awsse:SequenceNumber>3</awsse:SequenceNumber>
  <awsse:SecurityToken>1SP31VH87S9T310EWJIH27JLRI</awsse:SecurityToken>
</awsse:Session>
</soapenv:Header>
```

WS-Security header

WS-Security header is a OASIS standard defined in

`http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf` and
`http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf`

For Amadeus Web Services element Security contains element UsernameToken which may contain following elements:

- Username
- Nonce
- Password
- Created

Elements: Security, UsernameToken, Username, Nonce and Password are defined in namespace:
`http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd`

Please note that these elements may appear only in the request. In case of statefull conversation these elements appears only in the initial request.

Username element contains the UserID used for identification of the user. According to the naming convention the element has following format WSXXXYYY. Example:

```
<oas:Username xmlns:oas1="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd">WSXXXYYY</oas:Username>
```

Nonce element contains the unique, random byte stream (including printable alphanumeric string) encoded using Base64 algorithm. The format is string and varies depending on implementation. Example:

```
<oas:Nonce xmlns:oas="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd" EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-
message-security-1.0#Base64Binary">OyhXXWZqIVg=</oas:Nonce>
```

Password element contains the password used to identify UserID. It is generated using raw Nonce, plain text password and contents of Created element. The format is Base64 encoded string, 28 characters long. For the details concerning generation of this element please see section Password digest generation algorithm. Example:

```
<oas:Password xmlns:oas="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd" Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-
profile-1.0#PasswordDigest">LHd+1St4vQzfMnGDKeyNLPawNEo=</oas:Password>
```

Created element conveys date and timestamp of the request. Timestamp should be in the UTC timezone. Example:

```
<oas1:Created xmlns:oas1="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd" oas1:Id="UsernameToken-1">2014-01-24T08:13:28.339Z</oas1:Created>
```

Sample WS-Security header of the request:

```
<soap:Header>
    ... (other nodes are being omitted for clarity purpose) ...
<oas:Security xmlns:oas="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd">
  <oas:UsernameToken xmlns:oas1="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd" oas1:Id="UsernameToken-1">
    <oas:Username>WSAAABBB</oas:Username>
    <oas:Nonce EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-
message-security-1.0#Base64Binary">OyhXXWZqIVg=</oas:Nonce>
```

```
    <oas:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-
token-profile-1.0#PasswordDigest">LHd+1St4vQzfMnGDKeyNLPawNEo=</oas:Password>
    <oas1:Created>2014-01-24T08:13:28.339Z</oas1:Created>
  </oas:UsernameToken>
</oas:Security>
</soap:Header>
```

Amadeus Security header

Amadeus Security header is an Amadeus extension defined within AMA_SecurityHostedUser tag.

For Amadeus Web Services following attributes may be used:

- AgentDutyCode
- POS_Type
- PseudoCityCode
- RequestorType

Please note that AMA_SecurityHostedUser element may appear only in the request. In case of statefull conversation this element appears only in the initial request.

AgentDutyCode attribute contains the duty code of the user which is used by Security element. It is recommended to set this value to "SU" if the created user has more than one Agent Duty Code. If the user has been created with SU Duty Code only, attribute can be omitted

POS_Type attribute contains the Point of Sale (POS) indicator. It contains the details identifying the party or connection channel making the request. This value must be set to "1".

PseudoCityCode attribute contains the OfficeID which will be used in the request. This attribute has to be set to the existing OfficeID.

RequestorType attribute contains the indetifier which of the entity making the request. It can be 'U' for 'UserId' or 'Z' for 'SignId'. As Web Services channel relies on UserId concept for User Identification, it is recommended to set this value to "U".

Sample Amadeus Security header of the request:

```
<soap:Header>
... (other nodes are being omitted for clarity purpose) ...
  <UserID AgentDutyCode="SU" POS_Type="1" PseudoCityCode="CITYY28XX" RequestorType="U"/>
</soap:Header>
```

Password digest generation algorithm

This section describes algorithm used for password digest generation, that is steps required to transform input parameters along with plain text password to the passsword digest which will be transferred in the request in following XPath:
/Envelope/Header/Security/UsernameToken/Password

The digest generation algorithm is following:

Base64(SHA1($NONCE + $TIMESTAMP + SHA1($CLEARPASSWORD)))

Where:

Base64 is a binary-to-text encoding scheme;

SHA1 is a cryptographic hash function;

+ is the concatenation function;

$NONCE is an input parameter generated by the client's application;

$TIMESTAMP is an input parameter which represents current timestamp;

$CLEARPASSWORD is an input paramater - raw password obtained from Amadeus.

Warning

Please take into consideration following notes:

1. $NONCE is generated by the client application. It should be random, unique and the algorithm used for generation should be known only by the client application.
2. The same $NONCE cannot be re-used within 30 minutes. If all input parameters will be correct, but $NONCE has been already used within 30 minutes, Amadeus system will return error. The purpose of this behaviour is to avoid replay attacks (please see note concerning $TIMESTAMP)
3. It is recommended to use internal cryptographic library available for the programming language of choice for $NONCE generation.
4. $NONCE as a parameter for password digest generation algorithm can be printable string, however this string will have to be encoded with Base64 scheme to be transferred in SOAP message in Nonce element.
5. If the $TIMESTAMP parameter which represents current date and time in UTC timezone will differ from Amadeus system time more than 30 minutes, Amadeus system will always return error, even if all other parameters were correct. This feature along with the restriction on $NONCE re-use policy avoids replay attacks.
6. Every time SOAP header is being generated, values in Nonce, and Password elements should be different from the previously created.
7. SHA1 function shouldn't be confused with sha1sum command which returns computed hexadecimal SHA1 message digest in UNIX-like systems.

Below you may find an example of correctly computed password digest and several examples of common errors which may appear during the digest creation.

Input parameters:

Plain text password:	AMADEUS
Raw Nonce (may be unprintable)	secretnonce10111
Base64 encoded Nonce	c2VjcmV0bm9uY2UxMDExMQ==
Timestamp/Created	2015-09-30T14:12:15Z

1. Correct result:

Password digest: +LzcaRc+ndGAcZIXmq/N7xGes+k=

Formula: Base64(SHA1($NONCE + $TIMESTAMP + SHA1($CLEARPASSWORD)))

2. All parameters correct, except $NONCE which has the same (Base64) format as in Nonce XML element:

Password digest: AiRk9oAVpkYDX2MXh+diClQ0Lds=

Formula: Base64(SHA1(Base64($NONCE) + $TIMESTAMP + SHA1($CLEARPASSWORD)))

3. SHA1 in hexadecimal encoding instead of raw SHA1 for initial plain password encryption and for concatenated string:

Password digest: NWE1MGRhM2ZmNjFhMDA2ODUyNmIxMGM4MTczODQ0NjE2MWQyM2IxZQ==

Formula: Base64(HEX(SHA1($NONCE) + $TIMESTAMP + HEX(SHA1($CLEARPASSWORD))))

4. SHA1 in hexadecimal encoding instead of raw SHA1 for concatenated string, password not encrypted with SHA1:

Password digest: NzU0ZjJlMTc2ZjkxZmM2OTg4N2E0ZDlkMWY2MWE0YWJkOGI0MzYxZA==

Formula: Base64(HEX(SHA1($NONCE + $TIMESTAMP + $CLEARPASSWORD)))

5. Almost everything is incorrect: SHA1 in hexadecimal encoding instead of raw SHA1 for concatenated string, password not encrypted with SHA1, $NONCE has the same (Base64) format as in Nonce XML element:

Password digest: NGIzYmNiY2I3Njc2ZjZiNzdmNDMwMGVlMTIwODdhZDE1ZmZlOTEwMA==

Formula: Base64(HEX(SHA1(Base64($NONCE) + $TIMESTAMP + $CLEARPASSWORD)))

Below you may find an example of correctly computed password digest and several examples of common errors which may appear during the creation, but this time Nonce is stream of bytes:

<u>Input parameters:</u>

Plain text password:	AMADEUS
Raw Nonce (may be unprintable)	63▒jz▒Í▒U▒>▒%▒▒Q▒▒
Base64 encoded Nonce	NjPfanrqSdmXuFWgPoQlAsHOUbjOBg==
Timestamp/Created	2015-09-30T14:15:11Z

1. Correct result:

Password digest: k/upHztkhZzrsqAKsOBUa45+j1w=

Formula: Base64(SHA1($NONCE + $TIMESTAMP + SHA1($CLEARPASSWORD)))

2. All parameters correct, except $NONCE which has the same (Base64) format as in Nonce XML element:

Password digest: cG36y4hwpLjpD9u2nxJicnMgwPQ=

Formula: Base64(SHA1(Base64($NONCE) + $TIMESTAMP + SHA1($CLEARPASSWORD)))

3. SHA1 in hexadecimal encoding instead of raw SHA1 for initial plain password encryption and for concatenated string:

Password digest: MzM2NjNjYWRhZjZiNjdhMTBkODBkYTM2YzY1ZDllYWJkNjViZDRkNw==

Formula: Base64(HEX(SHA1($NONCE) + $TIMESTAMP + HEX(SHA1($CLEARPASSWORD))))

4. SHA1 in hexadecimal encoding instead of raw SHA1 for concatenated string, password not encrypted with SHA1:

Password digest: ZTUxYmEwN2FkYTJjNWM5MGI3M2UxNDE0Yzc2MDJiMDZiMTc5YmJjNw==

Formula: Base64(HEX(SHA1($NONCE + $TIMESTAMP + $CLEARPASSWORD)))

5. Almost everything is incorrect: SHA1 in hexadecimal encoding instead of raw SHA1 for concatenated string, password not encrypted with SHA1, $NONCE has the same (Base64) format as in Nonce XML element:

Password digest: NTFkMTY5ZDRmOWYwZGU1OWVmZjQ3ZDYxZmFmY2U2OWU1MDI4YWRjNw==

Formula: Base64(HEX(SHA1(Base64($NONCE) + $TIMESTAMP + $CLEARPASSWORD)))

Flows

We may distinguish two types of flows: context-less (stateless) and context-full (statefull).

The context-less flow is the flow which contains context-less queries, that is, the queries which don't require any prior nor later request to be functionally complete.

An example of context-less request is a flight search request. It is possible to search for flight, without needing later reservation.

The context-full flow is the flow which contains at least one context-full request, that is, the request which requires prior or later request to be functionally complete.

An example of context-full request is a Transactional Stored Ticket creation request. To create TST, the pricing context must exist before.

Below you may find two samples of scenarios illustrating two types of flows.

Context-less (Stateless)

Below two context-less requests are being presented, with accompanying responses. We omit functional part of the body.

Request #1:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:sec="http://xml.amadeus.com/2010/06/Security_v1"
xmlns:typ="http://xml.amadeus.com/2010/06/Types_v1"
xmlns:iat="http://www.iata.org/IATA/2007/00/IATA2010.1"
xmlns:app="http://xml.amadeus.com/2010/06/AppMdw_CommonTypes_v3"
xmlns:ses="http://xml.amadeus.com/2010/06/Session_v3">
   <soapenv:Header xmlns:add="http://www.w3.org/2005/08/addressing">
      <add:MessageID>1e98f38c-db15-4d57-8de5-490c31d5ab92</add:MessageID>
      <add:Action>http://webservices.amadeus.com/ABCDEF_12_3_1A</add:Action>
      <add:To>https://noded3.test.webservices.amadeus.com/1ASIWXXXYYY</add:To>
      <oas:Security xmlns:oas="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd" xmlns:oas1="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-utility-1.0.xsd">
         <oas:UsernameToken oas1:Id="UsernameToken-1">
            <oas:Username>WSYYYXXX</oas:Username>
            <oas:Nonce EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
soap-message-security-1.0#Base64Binary">c2VjcmV0bm9uY2UxMDExMQ==</oas:Nonce>
            <oas:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
username-token-profile-1.0#PasswordDigest">+LzcaRc+ndGAcZIXmq/N7xGes+k=</oas:Password>
            <oas1:Created>2015-09-30T14:12:15Z</oas1:Created>
         </oas:UsernameToken>
      </oas:Security>
      <AMA_SecurityHostedUser xmlns="http://xml.amadeus.com/2010/06/Security_v1">
         <UserID AgentDutyCode="SU" RequestorType="U" PseudoCityCode="OFFICEID"
POS_Type="1"/>
      </AMA_SecurityHostedUser>
   </soapenv:Header>
   <soapenv:Body>
      <!-- Here is the functional part of the request #1 -->
   </soapenv:Body>
</soapenv:Envelope>
```

Response #1:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:awsse="http://xml.amadeus.com/2010/06/Session_v3"
xmlns:wsa="http://www.w3.org/2005/08/addressing">
   <soapenv:Header>
      <wsa:To>http://www.w3.org/2005/08/addressing/anonymous</wsa:To>
      <wsa:From>
         <wsa:Address>https://noded3.test.webservices.amadeus.com/1ASIWXXXYYY</wsa:Address>
      </wsa:From>
      <wsa:Action>http://webservices.amadeus.com/ABCDEF_12_3_1A</wsa:Action>
      <wsa:MessageID>urn:uuid:d063aeeb-1fe8-6d94-1d4c-a175d01ca6ce</wsa:MessageID>
      <wsa:RelatesTo RelationshipType="http://www.w3.org/2005/08/addressing/reply">1e98f38c-
db15-4d57-8de5-490c31d5ab92</wsa:RelatesTo>
```

```
      <awsse:Session TransactionStatusCode="End">
         <awsse:SessionId>00A1D1Z9SS</awsse:SessionId>
         <awsse:SequenceNumber>1</awsse:SequenceNumber>
         <awsse:SecurityToken>1S2CYQAU4625P2PMCJV39XTC3S</awsse:SecurityToken>
      </awsse:Session>
   </soapenv:Header>
   <soapenv:Body>
     <!-- Here is the functional part of the response #1 -->
   </soapenv:Body>
</soapenv:Envelope>
```

Request #2:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:sec="http://xml.amadeus.com/2010/06/Security_v1"
xmlns:typ="http://xml.amadeus.com/2010/06/Types_v1"
xmlns:iat="http://www.iata.org/IATA/2007/00/IATA2010.1"
xmlns:app="http://xml.amadeus.com/2010/06/AppMdw_CommonTypes_v3"
xmlns:ses="http://xml.amadeus.com/2010/06/Session_v3">
   <soapenv:Header xmlns:add="http://www.w3.org/2005/08/addressing">
      <add:MessageID>ed5c950c-62dd-423e-bace-888859915fbc</add:MessageID>
      <add:Action>http://webservices.amadeus.com/ABCDEF_12_3_1A</add:Action>
      <add:To>https://noded3.test.webservices.amadeus.com/1ASIWXXXYYY</add:To>
      <oas:Security xmlns:oas="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd" xmlns:oas1="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-utility-1.0.xsd">
         <oas:UsernameToken oas1:Id="UsernameToken-1">
            <oas:Username>WSYYYXXX</oas:Username>
            <oas:Nonce EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
soap-message-security-1.0#Base64Binary">NjPfanrqSdmXuFWgPoQlAsHOUbjOBg==</oas:Nonce>
            <oas:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
username-token-profile-1.0#PasswordDigest">k/upHztkhZzrsqAKsOBUa45+j1w=</oas:Password>
            <oas1:Created>2015-09-30T14:15:11Z</oas1:Created>
         </oas:UsernameToken>
      </oas:Security>
      <AMA_SecurityHostedUser xmlns="http://xml.amadeus.com/2010/06/Security_v1">
         <UserID AgentDutyCode="SU" RequestorType="U" PseudoCityCode="OFFICEID"
POS_Type="1"/>
      </AMA_SecurityHostedUser>
   </soapenv:Header>
   <soapenv:Body>
      <!-- Here is the functional part of the request #2 -->
   </soapenv:Body>
</soapenv:Envelope>
```

Response #2:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:awsse="http://xml.amadeus.com/2010/06/Session_v3"
xmlns:awsl="http://wsdl.amadeus.com/2010/06/ws/Link_v1"
xmlns:wsa="http://www.w3.org/2005/08/addressing">
   <soapenv:Header>
      <wsa:To>http://www.w3.org/2005/08/addressing/anonymous</wsa:To>
      <wsa:From>
         <wsa:Address>https://noded3.test.webservices.amadeus.com/1ASIWXXXYYY</wsa:Address>
      </wsa:From>
      <wsa:Action>http://webservices.amadeus.com/ABCDEF_12_3_1A</wsa:Action>
      <wsa:MessageID>urn:uuid:267d45fb-40b7-5684-dd4e-bc43ef7b4527</wsa:MessageID>
      <wsa:RelatesTo RelationshipType="http://www.w3.org/2005/08/addressing/reply">ed5c950c-
62dd-423e-bace-888859915fbc</wsa:RelatesTo>
      <awsse:Session TransactionStatusCode="End">
         <awsse:SessionId>001FPN8ACX</awsse:SessionId>
         <awsse:SequenceNumber>1</awsse:SequenceNumber>
         <awsse:SecurityToken>15EZ6S3LYHUIODC656TZ2XYZH</awsse:SecurityToken>
      </awsse:Session>
   </soapenv:Header>
   <soapenv:Body>
     <!-- Here is the functional part of the response #2 -->
   </soapenv:Body>
</soapenv:Envelope>
```

Please note that response contains the session with attribute **TransactionStatusCode** set to "End". This means that the session is being created by the Amadeus for the processing, once the result is being returned, the session is immediately released. This is noticeable improvement over previous versions of SOAP header where session management logic has to be managed fully by the application.

Context-full

Below three context-full requests are being presented, with accompanying responses. We omit functional part of the body.

Request #1

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:sec="http://xml.amadeus.com/2010/06/Security_v1"
xmlns:typ="http://xml.amadeus.com/2010/06/Types_v1"
xmlns:app="http://xml.amadeus.com/2010/06/AppMdw_CommonTypes_v3"
xmlns:ses="http://xml.amadeus.com/2010/06/Session_v3">
   <soapenv:Header xmlns:add="http://www.w3.org/2005/08/addressing">
      <add:MessageID>60608faa-647a-4494-be0c-96c417bc7bc3</add:MessageID>
      <add:Action>http://webservices.amadeus.com/AAABBB_01_2_IA</add:Action>
      <add:To>https://noded3.test.webservices.amadeus.com/1ASIWXXXYYY</add:To>
      <oas:Security xmlns:oas="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd" xmlns:oas1="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-utility-1.0.xsd">
         <oas:UsernameToken oas1:Id="UsernameToken-1">
            <oas:Username>WSYYYXXX</oas:Username>
            <oas:Nonce EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
soap-message-security-1.0#Base64Binary">UVlwZMUriILQgsbp2Hmupw==</oas:Nonce>
            <oas:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
username-token-profile-1.0#PasswordDigest">vRdsBMfRGmvzOQkfrcz8cKuoTk0=</oas:Password>
            <oas1:Created>2015-10-22T14:20:17Z</oas1:Created>
         </oas:UsernameToken>
      </oas:Security>
      <AMA_SecurityHostedUser xmlns="http://xml.amadeus.com/2010/06/Security_v1">
         <UserID AgentDutyCode="SU" RequestorType="U" PseudoCityCode="OFFICEID"
POS_Type="1"/>
      </AMA_SecurityHostedUser>
      <awsse:Session TransactionStatusCode="Start"
xmlns:awsse="http://xml.amadeus.com/2010/06/Session_v3"/>
   </soapenv:Header>
   <soapenv:Body>
    <!-- Here is the functional part of the request #1 -->
   </soapenv:Body>
</soapenv:Envelope>
```

Response #1

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:awsse="http://xml.amadeus.com/2010/06/Session_v3"
xmlns:wsa="http://www.w3.org/2005/08/addressing">
   <soapenv:Header>
      <wsa:To>http://www.w3.org/2005/08/addressing/anonymous</wsa:To>
      <wsa:From>
         <wsa:Address>https://noded3.test.webservices.amadeus.com/1ASIWXXXYYY</wsa:Address>
      </wsa:From>
      <wsa:Action>http://webservices.amadeus.com/AAABBB_01_2_IA</wsa:Action>
      <wsa:MessageID>urn:uuid:cd10252f-fa15-eb94-a14d-0b1e69b69509</wsa:MessageID>
      <wsa:RelatesTo RelationshipType="http://www.w3.org/2005/08/addressing/reply">60608faa-
647a-4494-be0c-96c417bc7bc3</wsa:RelatesTo>
      <awsse:Session TransactionStatusCode="InSeries">
         <awsse:SessionId>01VWJYKFTY</awsse:SessionId>
         <awsse:SequenceNumber>1</awsse:SequenceNumber>
         <awsse:SecurityToken>2TOAMQEYRSZJL29UWKMJ4E79ZC</awsse:SecurityToken>
      </awsse:Session>
   </soapenv:Header>
   <soapenv:Body>
        <!-- Here is the functional part of the response #1 -->
   </soapenv:Body>
</soapenv:Envelope>
```

Request #2

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:sec="http://xml.amadeus.com/2010/06/Security_v1"
xmlns:typ="http://xml.amadeus.com/2010/06/Types_v1"
xmlns:app="http://xml.amadeus.com/2010/06/AppMdw_CommonTypes_v3"
xmlns:ses="http://xml.amadeus.com/2010/06/Session_v3">
   <soapenv:Header xmlns:add="http://www.w3.org/2005/08/addressing">
      <add:MessageID>1e925868-78ca-11e5-92af-56d0e78cc150</add:MessageID>
      <add:Action>http://webservices.amadeus.com/BBBCCC_01_2_IA</add:Action>
      <add:To>https://noded3.test.webservices.amadeus.com/1ASIWXXXYYY</add:To>
      <awsse:Session TransactionStatusCode="InSeries"
xmlns:awsse="http://xml.amadeus.com/2010/06/Session_v3">
         <awsse:SessionId>01VWJYKFTY</awsse:SessionId>
         <awsse:SequenceNumber>2</awsse:SequenceNumber>
         <awsse:SecurityToken>2TOAMQEYRSZJL29UWKMJ4E79ZC</awsse:SecurityToken>
      </awsse:Session>
   </soapenv:Header>
   <soapenv:Body>
     <!-- Here is the functional part of the request #2 -->
   </soapenv:Body>
</soapenv:Envelope>
```

Response #2

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:awsse="http://xml.amadeus.com/2010/06/Session_v3"
xmlns:wsa="http://www.w3.org/2005/08/addressing">
   <soapenv:Header>
      <wsa:To>http://www.w3.org/2005/08/addressing/anonymous</wsa:To>
      <wsa:From>
         <wsa:Address>https://noded3.test.webservices.amadeus.com/1ASIWXXXYYY</wsa:Address>
      </wsa:From>
      <wsa:Action>http://webservices.amadeus.com/BBBCCC_01_2_1A</wsa:Action>
      <wsa:MessageID>urn:uuid:1b56bf46-36bb-8494-7546-0d94269ca97d</wsa:MessageID>
      <wsa:RelatesTo RelationshipType="http://www.w3.org/2005/08/addressing/reply">1e925868-
78ca-11e5-92af-56d0e78cc150</wsa:RelatesTo>
      <awsse:Session TransactionStatusCode="InSeries">
         <awsse:SessionId>01VWJYKFTY</awsse:SessionId>
         <awsse:SequenceNumber>2</awsse:SequenceNumber>
         <awsse:SecurityToken>2TOAMQEYRSZJL29UWKMJ4E79ZC</awsse:SecurityToken>
      </awsse:Session>
   </soapenv:Header>
   <soapenv:Body>
        <!-- Here is the functional part of the response #2 -->
   </soapenv:Body>
</soapenv:Envelope>
```

Request #3

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:sec="http://xml.amadeus.com/2010/06/Security_v1"
xmlns:typ="http://xml.amadeus.com/2010/06/Types_v1"
xmlns:app="http://xml.amadeus.com/2010/06/AppMdw_CommonTypes_v3"
xmlns:ses="http://xml.amadeus.com/2010/06/Session_v3"
   <soapenv:Header xmlns:add="http://www.w3.org/2005/08/addressing">
      <add:MessageID>b0089730-78ca-11e5-a3e0-58d0e78cc150</add:MessageID>
      <add:Action>http://webservices.amadeus.com/CCCDDD_01_2_IA</add:Action>
      <add:To>https://noded3.test.webservices.amadeus.com/1ASIWXXXYYY</add:To>
      <awsse:Session TransactionStatusCode="End"
xmlns:awsse="http://xml.amadeus.com/2010/06/Session_v3">
         <awsse:SessionId>01VWJYKFTY</awsse:SessionId>
         <awsse:SequenceNumber>3</awsse:SequenceNumber>
         <awsse:SecurityToken>2TOAMQEYRSZJL29UWKMJ4E79ZC</awsse:SecurityToken>
      </awsse:Session>
   </soapenv:Header>
   <soapenv:Body>
     <!-- Here is the functional part of the request #3 -->
   </soapenv:Body>
</soapenv:Envelope>
```

Response #3

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:awsse="http://xml.amadeus.com/2010/06/Session_v3"
xmlns:wsa="http://www.w3.org/2005/08/addressing">
   <soapenv:Header>
      <wsa:To>http://www.w3.org/2005/08/addressing/anonymous</wsa:To>
      <wsa:From>
         <wsa:Address>https://noded3.test.webservices.amadeus.com/1ASIWXXXYYY</wsa:Address>
      </wsa:From>
      <wsa:Action>http://webservices.amadeus.com/CCCDDD_01_2_IA</wsa:Action>
      <wsa:MessageID>urn:uuid:b6f47895-8e88-edb4-2541-e58640a8b94f</wsa:MessageID>
      <wsa:RelatesTo RelationshipType="http://www.w3.org/2005/08/addressing/reply">b0089730-
78ca-11e5-a3e0-58d0e78cc150</wsa:RelatesTo>
      <awsse:Session TransactionStatusCode="End">
         <awsse:SessionId>01VWJYKFTY</awsse:SessionId>
         <awsse:SequenceNumber>3</awsse:SequenceNumber>
         <awsse:SecurityToken>2TOAMQEYRSZJL29UWKMJ4E79ZC</awsse:SecurityToken>
      </awsse:Session>
   </soapenv:Header>
   <soapenv:Body>
        <!-- Here is the functional part of the response #3 -->
   </soapenv:Body>
</soapenv:Envelope>
```

Please note that initial response contains the session with attribute **TransactionStatusCode** set to “InSeries”. This means that the session has been created and system expects following requests.

Second request doesn’t contain nodes: **AMA_SecurityHostedUser** or **Security**. Authentication is established basing on the **Session** nodes: **SessionId** and **SecurityToken**. Requests which contains **Session** and **Security** nodes will fail. Attribute **TransactionStatusCode** is set to “InSeries” which indicates that this session should remain open. Setting **TransactionStatusCode** attribute to “Start” for the already opened session is an error and will generate SOAP fault response.

Third request contains **Session** node with attribute **TransactionStatusCode** set to “End” which indicates that after processing the functional part of the request, this session should be closed by Amadeus. Succesful closure of the session is signaled in the response by **TransactionStatusCode** set to “End” in **Session** node.

Common errors

In this section we provide hints which will help avoid common errors.

1. Ensure that element /Header/Security/UsernameToken/Created is a timestamp in UTC, in ISO8601 form.
Below you may see examples of the acceptable forms:

```
yyyy-mm-ddTHH:MM:SSZ or
yyyy-mm-ddTHH:MM:SS+00:00
yyyy-mm-ddTHH:MM:SS:sssZ (where sss denotes miliseconds)
yyyy-mm-ddTHH:MM:SS:ssssssZ (where ssssss denotes microseconds)
```

common problems:

a) time is not synchronized via NTP and deviation between system time and our servers is higher than 30 minutes.
b) application sends local time as UTC

2. System returns "11|Session" in /Envelope/Body/Fault/faultstring element.

This mean the problem with authentication. Common root causes of the problem:

a) Incorrect /Envelope/Header/Security/UsernameToken/Username element (wrong username).
b) Time difference between our server and element /Header/Security/UsernameToken/Created is higher than 30 minutes. Check system time and timezone.
c) The same Nonce element has been used twice within 30 minutes. Check algorithm which generates /Envelope/Header/Security/UsernameToken/Nonce . This element should be a random string.
d) Wrong OfficeId in /Envelope/Header/AMA_SecurityHostedUser/UserID[@PseudoCityCode].
e) Wrong password generation algorithm. Please read section 4 to understand how to generate properly the /Envelope/Header/Security/UsernameToken/Password. You may also refer to OASIS standard - http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf. The only difference from standard is that in line 113 of aforementioned document, application should use SHA1(password) instead of password.
f) Lack of / Envelope/ Header/AMA_SecurityHostedUser/UserID[@AgentDutyCode='SU'] attribute.

3. To properly build code your language of choice may require to download additional schema definitions, not included into the WSDL. This concerns Security element which may require to download schema definitions from OASIS:

http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd

http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd

4. It is strongly recommended to use TLS1.2 cryptograhic protocols – as defined in RFC 5246 and RFC 6176 – for communication security to Amadeus endpoint. Connection attempts with TLS 1.1 or earlier may fail.

Code samples

Samples for Java (Axis2, CXF, JaxWS) and .Net can be found in Amadeus for Developers Portal in the Technical documentation section.

Below you may find samples of password hash function for PHP and Perl.

PHP

```
<?php
$password = "SECRETPASSWORD";

$nonce = random_bytes(32); # requires PHP 7
date_default_timezone_set("UTC");
$timestamp = date(DATE_ATOM);
$encodedNonce = base64_encode($nonce);
$passSHA = base64_encode(sha1($nonce . $timestamp . sha1($password, true), true));
?>
```

`$timestamp` is used for /Header/Security/UsernameToken/Created
`$encodedNonce` is used for /Header/Security/UsernameToken/Nonce
`$passSHA` is used for /Header/Security/UsernameToken/Password

Perl

```
#!/usr/bin/env perl

use strict;
use warnings;
use MIME::Base64 qw(encode_base64 decode_base64);
use Time::Stamp 'gmstamp';
use Digest::SHA qw(sha1 sha1_hex sha1_base64);
use Bytes::Random::Secure ;

my $password = "SECRETPASSWORD";
my $timestamp = Time::Stamp::gmstamp();
my $random = Bytes::Random::Secure->new(
        Bits            => 256,
        NonBlocking     => 1,
);

my $nonce = $random->bytes(32);
my $passSHA = sha1_base64($nonce . $timestamp . sha1($password));
```