AlpineBits is a definition of an OTA-based interface that is specially tailored for Alpine tourism.
Its purpose is to facilitate data exchange in the tourism sector.

compatible with version 2010A of the OpenTravel Schema by the OpenTravel Alliance.

www.alpinebits.org

Document Change Log

protocol version	documentation release date	description
2011-11	2011-11-18	Minor alterations and release under Creative Commons Attribution-NoDerivs 3.0 Unported License
2011-10	2011-10-20	production release with minor alterations
2011-09	2011-09-08	first draft of redesigned version (using POST instead of SOAP)
2010-10	2010-10-20	second draft
2010-08	2010-08-01	first draft

CREDITS

Copyright Holders

Altea Software Srl - www.altea.it

Peer GmbH - www.peer.biz

SiMedia GmbH - www.simedia.eu

Technical Partners

Seekda GmbH - www.seekda.com

Authors

Chris Mair - www.1006.org

Table of Contents

1. Introduction

This documents describes a standard for exchanging traveling and booking information, called **AlpineBits**.

AlpineBits builds upon established standards:

- client-server communication is done through stateless HTTPS (the client POSTs data to the server and gets a response) with basic access authentication [1] and
- the traveling and booking information are encoded in XML following version 2010A of the OpenTravel Schema [3, 4, 5] (from here on called OTA2010A) by the OpenTravel Alliance [2].

At the current version of the standard, the scope of AlpineBits is limited to exchanging room availability notifications.

AlpineBits relies on its underlying transport protocol to take care of security issues. Hence **the use of HTTPS is mandatory**.

2. POST request and response

An AlpineBits compliant server exposes a **single** HTTPS URL. Clients send POST requests to that URL.

Each POST request transmits the access credentials using **basic access authentication**.

The POST request **must** follow the `multipart/form-data` encoding scheme, as commonly used in the context of HTML forms for file uploads.

Each POST requests **must** have **at least** one parameter named **`action`** specifying the purpose of the request.

According to the value of `action`, zero or more additional parameters are expected to be POSTed.

Following is a capture of an example POST with parameters `action` and `request`:

```
POST / HTTP/1.1
Authorization: Basic Y2hyaXM6c2VjcmV0
Host: localhost
Accept: */*
Content-Length: 1911
Expect: 100-continue
Content-Type: multipart/form-data; boundary=------------------------------9d7042ecb251

------------------------------9d7042ecb251
Content-Disposition: form-data; name="action"

OTA_HotelAvailNotif
------------------------------9d7042ecb251
Content-Disposition: form-data; name="request"

<?xml version="1.0" encoding="UTF-8"?>

<OTA_HotelAvailNotifRQ
     [...]
</OTA_HotelAvailNotifRQ>
------------------------------9d7042ecb251--
```

Note the `Authorization` header, with the username/password string (`chris:secret`) encoded in base64 (`Y2hyaXM6c2VjcmV0`) as defined by the basic access authentication standard [1]. Also note the multipart content with the values of parameters `action` and `request`.

For maximum compatibility across different implementations AlpineBits implementors are asked to handle POST requests using a supporting API in their language of choice. For an example, see appendix A where the usage of libcURL to generate POST requests from PHP is shown.

As a result of the POST request, the server answers with a response.

The expected status code of the response is 200 (Ok), indicating the server could authenticate the user (with or without being able to actually process any action). The content of the response depends on the value of parameter `action` (see subsections 2.1 to 2.4).

In case of authentication failure (either an invalid or missing username/password) the status code is 401 (Authorization Required) and the content is `ERROR`, followed by a colon (`:`) and an error message indicating the reason of the failure, such as no username/password was provided or the password expired, etc.

In case of internal server problem the status code is 500 (Internal Server Error).

An AlpineBits client **must** be able to handle these status codes. It **should** retry a request that has failed (error code 500 or timeout) and only escalate the failure after 2 retries with an appropriate delay.

Subsections 2.1 to 2.4 describe the details of the POST request and the response depending on the value of parameter `action`. Currently, three different values are defined. A server **must** be able to handle them **all**.

2.1 The `getVersion` action

No additional POST parameter besides `action` is needed.

The response content is the string `OK:2011-10`.

2.2 The `getCapabilities` action

No additional POST parameter besides `action` is needed.

The response is a string starting with `OK:` followed by a list of comma separated tokens each of which indicates a single capability of the server.

AlpineBits specifies the following capabilities:

- `action_getVersion`
  the server implements the `getVersion` action
- `action_getCapabilities`
  the server implements the `getCapabilities` action
- `action_OTA_HotelAvailNotif`
  the server implements the `OTA_HotelAvailNotif` action
- `OTA_HotelAvailNotif_accept_rooms`
  accept booking limits for specific rooms (see section 3)
- `OTA_HotelAvailNotif_accept_categories`
  accept booking limits for categories of rooms (see section 3)
- `OTA_HotelAvailNotif_accept_los`
  accept length of stay restrictions (see section 3)
- `OTA_HotelAvailNotif_accept_dow`
  accept day of the week restrictions (see section 3)

AlpineBits **requires** a server to support all currently defined actions, hence the capabilities `action_getVersion`, `action_getCapabilities` and `action_OTA_HotelAvailNotif` **must** all be returned.

Moreover, at least one out of the two `OTA_HotelAvailNotif_accept_rooms` and `OTA_HotelAvailNotif_accept_categories` **must** be supported. These indicate whether the server can handle the availability of rooms at the level of distinct rooms, at the level of categories of rooms or both (see section 3).

All other capabilities are optional. It is a **client's responsibility** to check for server capabilities before trying to use them. A server implementation is free to ignore information that requires a capability it doesn't declare. A server **must**, however, implement all capabilities it declares.

2.3 The `OTA_HotelAvailNotif` action

When the value of the `action` parameter is `OTA_HotelAvailNotif` the client intends to send a room availability notification request to the server. The parameter name is `request` and its value is the notification request encoded as an OTA2010A `OTA_HotelAvailNotifRQ` XML document.

The content of the response is an OTA2010A `OTA_HotelAvailNotifRS` XML document indicating success or failure.

Both, the value of the `request` parameter and the content of the response **must** conform to OTA2010A. Server implementors are encouraged to offer an option to actually validate incoming requests against the OTA2010A schema to aid debugging.

The business logic of an AlpineBits server, i.e. how the server processes and stores the information it receives is implementation-specific. There is, however, a minimum set of elements and attributes that AlpineBits **requires** to be present in a request. If these are not present, the business logic is bound to fail and will return a response indicating error even though the request is formally valid. OTA2010A, for instance, allows requests that do not indicate the hotel. Such a request would certainly trigger an error from an AlpineBits server.

The business logic of room availability notifications is explained in more detail in section 3.

In case of success, the `OTA_HotelAvailNotifRS` response will contain an empty `Success` element. In case of failure the `OTA_HotelAvailNotifRS` response will contain one or more `Error` elements each spotting a `Type` attribute that is implementation-specific, but **must** follow the OTA2010A error code table that comes with the OTA2010A documentation package [3].

Following are two examples of responses, one indicating success and one indicating an error at the business level (files `test_ok.xml` and `test_ko_invcode_missing.xml` in the example set):

```
<?xml version="1.0" encoding="UTF-8"?>
<OTA_HotelAvailNotifRS xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                       xmlns="http://www.opentravel.org/OTA/2003/05"
                       xsi:schemaLocation="http://www.opentravel.org/OTA/2003/05 OTA_HotelAvailNotifRS.xsd"
                       Version="1.001">
    <Success/>
</OTA_HotelAvailNotifRS>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<OTA_HotelAvailNotifRS xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                       xmlns="http://www.opentravel.org/OTA/2003/05"
                       xsi:schemaLocation="http://www.opentravel.org/OTA/2003/05 OTA_HotelAvailNotifRS.xsd"
                       Version="1.001">
    <Errors>
        <Error Type="113">
            missing information, need AvailStatusMessage element for all rooms
        </Error>
    </Errors>
</OTA_HotelAvailNotifRS>
```

An AlpineBits server **may silently ignore** information contained in a request that it has no capabilities to process. It is an AlpineBits **client's responsibility** to check whether the server has the capabilities to process the information it is sending (or deal with the consequence that the server might ignore them).

2.4 Un unknown or missing action

The response content is the string `ERROR:unknown or missing action`.

3. An Annotated AlpineBits `OTA_HotelAvailNotif` request

The file discussed here is provided as file `test_ok.xml` in the example set.

First, consider the outer part of the XML file:

```
<?xml version="1.0" encoding="UTF-8"?>

<OTA_HotelAvailNotifRQ xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns="http://www.opentravel.org/OTA/2003/05"
       xsi:schemaLocation="http://www.opentravel.org/OTA/2003/05 OTA_HotelAvailNotifRQ.xsd"
       Version="1.002">

<AvailStatusMessages HotelCode="123" HotelName="Frangart Inn">

<!-- ... inner part see below ... -->

</AvailStatusMessages>

</OTA_HotelAvailNotifRQ>
```

Per OTA2010A, an `OTA_HotelAvailNotifRQ` may contain just **one** `AvailStatusMessages` (note the plural) element, hence at most **one** hotel can be dealt with in a single request.

Although OTA doesn't require attributes `HotelCode` or `HotelName` to be present, an AlpineBits compliant server's business logic will **certainly fail** unless at least **one** out of these two is given (and matched from the server's database). The fictitious hotel in the example is the “Frangart Inn” with code “123”. Specifying both — code and name — is redundant, but allowed, as long as both are consistent.

AlpineBits **requires** that a match of `HotelCode`, `HotelName` or `InvCode` (see below) to be **case sensitive**.

Second, consider the inner part of the example that contains `AvailStatusMessage` (note the singular) elements for 3 different rooms.

Let's start with the availabilities for room 101S.

```
<AvailStatusMessage BookingLimit="1" BookingLimitMessageType="SetLimit">
    <StatusApplicationControl Start="2010-08-01" End="2010-08-10"
                              InvCode="101S" IsRoom="true"/>
</AvailStatusMessage>

<AvailStatusMessage BookingLimit="1" BookingLimitMessageType="SetLimit">
    <StatusApplicationControl Start="2010-08-21" End="2010-08-30"
                              InvCode="101S" IsRoom="true"/>
</AvailStatusMessage>
```

The `IsRoom="true"` attribute indicates that we're dealing with a specific room and that `InvCode` indicates the room identifier (101S). Alternatively, `IsRoom="false"` would indicate a category of rooms with `InvCode` being the category name.

An AlpineBits server's business logic **must** be able to treat **at least** one case out of the two cases (`IsRoom="true"` or `IsRoom="false"`). A client should use `getCapabilities` to find out whether the server treats the room case (token `"OTA_HotelAvailNotif_accept_rooms"`), the category case (token `"OTA_HotelAvailNotif_accept_categories"`) or both.

Mixing rooms and categories in a single request is not allowed. An AlpineBits server's business logic **must** return an error if it receives such a mixed request.

Room 101S is thus available from 2010-08-01 to 2010-08-10 and from 2010-08-21 to 2010-08-30.

Regarding the first interval, this means the earliest possible check-in is 2010-08-01 afternoon and latest possible check-out is 2010-08-11 morning (maximum stay is 10 nights).

Since there are no further restrictions, check-ins **after** 2010-08-01 and stays of **less** than 10 nights are allowed as well, provided the check-out is not later than 2010-08-11 morning.

Idem for the other block of 10 nights from 2010-08-21 to 2010-08-30 (latest check-out is 2010-08-31 morning).

Since a specific room is indicated here (`IsRoom="true"`), the only meaningful value of `BookingLimit` is 0 or 1 (the same room can not be available more than once). In the category case, numbers larger than 1 would also be allowed.

`BookingLimit` numbers are always interpreted to be absolute numbers. Differential updates are not allowed.

Note that AlpineBits does **not allow** `AvailStatusMessage` elements with overlapping periods. This implies that the order of the `AvailStatusMessage` elements doesn't matter. It is a **client's responsibility** to avoid overlapping. An AlpineBits server's business logic **may** identify overlapping and return an error or **may** proceed in a implementation-specific way.

Unlike OTA2010A, an AlpineBits server's business logic **requires** the `AvailStatusMessage` attributes `BookingLimit` and `BookingLimitMessageType` to be given. It also requires exactly one `StatusApplicationControl` element with attributes `Start`, `End`, `InvCode` and `IsRoom` for each `AvailStatusMessage` element. It **must** return an error if any of these are missing.

Next: the availabilities for room 102.

```
<AvailStatusMessage BookingLimit="1" BookingLimitMessageType="SetLimit">
    <StatusApplicationControl Start="2010-08-14" End="2010-08-27"
                              InvCode="102" IsRoom="true" Sat="true" />
    <LengthsOfStay>
        <LengthOfStay MinMaxMessageType="SetMinLOS" Time="7" TimeUnit="Day"/>
        <LengthOfStay MinMaxMessageType="SetMaxLOS" Time="7" TimeUnit="Day"/>
    <LengthsOfStay>
<AvailStatusMessage>
```

Room 102 is available from 2010-08-14 to 2010-08-27, but allows arrivals only on Saturdays and requires a stay of 7 nights.

That means there are two blocks that can be sold:

- arrival Saturday 2010-08-14, stay 7 nights, departure Saturday 2010-08-21 morning
- arrival Saturday 2010-08-21, stay 7 nights, departure Saturday 2010-08-28 morning

Restrictions on arrivals to certain days of week (DOWs) must be indicated by setting the corresponding DOW attributes to true: `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat` or `Sun`.

A DOW attribute set to false has the same meaning as if missing. If neither of these attributes is set to true, there are no DOW restrictions at all.

An AlpineBits server's business logic **might** ignore DOW restrictions silently. It is the **client's responsibility** to check whether the server supports DOW restrictions using `getCapabilities`: the token to look for is `"accept_dow"`.

Restrictions on length of stay (LOS) must be indicated using one or two `LengthsOfStay` elements with the `MinMaxMessageType` attribute set to `"SetMinLOS"` or `"SetMaxLOS"`. AlpineBits expects the `TimeUnit` to be always `"Day"`.

Note that LOS restrictions are considered to be “on arrival” not “stay through”. That means only the LOS restriction of the arrival day applies.

Again, an AlpineBits server's business logic **might** ignore LOS restrictions silently. It is the **client's responsibility** to check whether the server supports LOS restrictions using `getCapabilities`: the token to look for is `"accept_los"`.

The last `AvailStatusMessage` in the example is about room 103.

```
<AvailStatusMessage BookingLimit="1" BookingLimitMessageType="SetLimit">
    <StatusApplicationControl Start="2010-08-01" End="2010-08-31"
                              InvCode="103" IsRoom="true"/>
</AvailStatusMessage>
```

Room 103 is available in August (earliest check-in 2010-08-01 afternoon and latest check-out 2010-09-01 morning) with no further restrictions.

A final, very important, point is that an AlpineBits server's business logic **requires** either **every room** or **every category** to be mentioned in a request! The server has knowledge about each hotel's complete set of room identifiers and/or complete set of category names and **must** answer with an error to requests that do not mention all rooms/categories or that contain unknown rooms/categories.

This is necessary in order to safeguard against situations where a server's database is not kept up to date with regard to a hotel's capacity.

In the example files the “Frangart Inn” has rooms 101S, 102 and 103 and categories “standard” and “superior”.

A. PHP example client code

While `multipart/form-data` POSTs from HTML are commonly encountered in web programming, it might be less obvious how to perform them from a programming language. To avoid compatibility issues across different implementations AlpineBits implementors are **strongly encouraged** to perform POST requests using a supporting API rather than assemble the header and the parts in a low-level way. The following example shows how this can be done using libcURL from PHP:

```
<?php

$cu = curl_init("http://localhost:8088/alpinebits/server.php");

curl_setopt($cu, CURLOPT_RETURNTRANSFER,    true);
curl_setopt($cu, CURLOPT_POST,              true);
curl_setopt($cu, CURLOPT_HTTPAUTH,          CURLAUTH_BASIC);
                                            # in production should be CURLPROTO_HTTPS only
curl_setopt($cu, CURLOPT_PROTOCOLS,         CURLPROTO_HTTP | CURLPROTO_HTTPS);
curl_setopt($cu, CURLOPT_USERPWD,           "chris:secret");


$data = array(
    "action"  => "OTA_HotelAvailNotif",
    "request" => file_get_contents("test_ok.xml")
);

curl_setopt($cu, CURLOPT_POSTFIELDS, $data);

$output = curl_exec($cu);
$info = curl_getinfo($cu);
curl_close($cu);

if ($info["http_code"] != 200) {
    echo "oops: got http status code " . $info["http_code"] . "<br>\n";
}

echo "server said:<br>\n";
echo $output;

?>
```

B. Links

[1] **HTTP basic access authentication:**
http://en.wikipedia.org/wiki/Basic_access_authentication/

[2] **OpenTravel Alliance:**
http://www.opentravel.org/

[3] **OTA2010A documentation package download:**
http://www.opentravel.org/Specifications/ReleaseNotes.aspx?spec=2010A

[4] **OTA2010A XML schema files online:**
http://www.opentravel.org/Specifications/SchemaIndex.aspx?FolderName=2010A

[5] **browsable interface to the above schema files:**
http://adriatic.pilotfish-net.com/ota-modelviewer/