Fédération Equestre Internationale

HM King Hussein I Building

Chemin de la Joliette 8

CH 1006 Lausanne

FEI21 - WebServices

OrganizerWS - General description

Version 1.1

www.epyx.ch
info@epyx.ch

t +41 21 623 79 50
f +41 21 623 79 55

epyx

Avenue de Provence 4
1007 Lausanne
Switzerland

Table of contents

History

Date	Version	Description
28.01.2013	1.0	Initial document (copied from the document for the FEI NF WebServices).
03.03.2015	1.1	Adjustment in the URLs for the Integration and Validation environments in section 2.

1. Introduction

This document is the base document about the **FEI21 WebServices for Organizers and their IT Providers**.

It describes general information and principles for these WebServices such as URLs to connect to the WebServices and security.

The **FEI21 WebServices application** is made of three WebServices (".asmx" file): one for the methods specifically developed to fulfil Organizers and IT Providers needs (OrganizerWS_**X_Y**.asmx with X and Y representing the version number of the WebServices), a second one shared with the FEI WebServices for NFs, which contain the methods to access static data such as countries, NFs, disciplines codes and so on (CommonWS.asmx) and the last one for authentication (Authentication.asmx).

The documentation about the WebServices is split in the following files:

Document	Description
FEI21 WebServices – OrganizerWS	This is the present document that describes among other things how to connect to the authentication service.
FEI21 WebServices - CommonWS	This document describes the Common Data WebService (CommonWS.asmx) which contains all static data that can be cached on the client application, such as country list, discipline list, document issuing body list and so on. Note that with respect to the current version of the OrganizerWS webmethods, only a few of the methods contained in CommonWS are relevant.
FEI21 WebServices - OrganizerWS X.Y - Methods	This document describes the web methods of this WebService.
FEI21 WebServices - OrganizerWS X.Y - Classes	This document describes the classes used by the methods in OrganizerWS_**X_Y**.asmx.

1.1 Feedback

Feedback about the present documentation (all documents listed above) or about the WebServices themselves can be reported to one of the following contacts:

Company	Person	Contact
FEI	Gaspard Dufour	gaspard.dufour@fei.org
	Jérôme Begey	jerome.begey@fei.org
	Sylvain Wolf	sylvain.wolf@fei.org

1.2 Technology considerations

Web services are implemented using Microsoft C# as the underlying technology, and published using standard WSDL on an IIS server.

2. WebServices URLs

The URL to access the different WebServices is constructed on the module name with the following syntax:

rootURL/_vti_bin/***subfolder***/***serviceName***.asmx

where

- ***serviceName*** is the name of the service, e.g. HorseWS, CommonWS, Authentication.
- ***rootURL*** is the root URL of the Core Application. This URL depends on the environment (integration, validation, production, see below).
- ***subfolder*** is the path to a subfolder used to separate the Authentication WebService (Authentication.asmx in our case) from the FEI21 business WebServices (the other ones).

The following table give the explicit list of URLs:

WebService	**URL**
Authentication	***rootURL***/_vti_bin/Authentication.asmx
CommonWS	***rootURL***/_vti_bin/FEI/CommonWS.asmx
OrganizerWS	***rootURL***/_vti_bin/FEI/OrganizerWS_**X_Y**.asmx

There are currently three environments where the FEI21 WebServices application is available:

Environment name	**Description**	**Root URL**
Integration	This is the environment where the developed applications are integrated and tested.	https://idata.fei.org
Validation	This is the validation environment whose is architecture is close to the production environment	https://vdata.fei.org
Production	This is the production environment	https://data.fei.org

2.1 Generic user account

The following user account is available in both the Integration and the Validation environments (but of course, not in the Production environment):

- Username : OC_WS.
- Password: OC_WS.

To get a user account for the Production environment, look with the FEI (see section 1.1).

3. Authentication and user information

Security is implemented using a session login, and while the session is open no additional security is required. It is anticipated that the session will be initiated by a client application as part of their own central application service, and may therefore be accessible to several users simultaneously. Users are recorded to aid with fault diagnosis, and user information should be supplied in the header for every web service call (see 3.2).

The authentication process is performed through the Authentication.asmx WebService (see section 3.3). We now describe the basic workflow to authenticate and use the returned cookie.

3.1 FEI21 WebServices Authentication

The client application must login into the WebServices application by calling the **Login** method of the **Authentication.asmx** WebService. This method returns a **LoginResult** object that contains an **ErrorCode** to check that the login was successful and a **CookieName** which can be used to get the **authentication cookie** in the cookie container of the WebService proxy caller. This authentication cookie has to be provided to each call of the FEI21 WebServices application.

As an illustration, we give now an application of this workflow in C#:

```
AuthenticationWS.LoginResult result = authenticateWS.Login(username, password);

// Analyse result
if (result.ErrorCode == WebServiceTestApp.AuthenticationWS.LoginErrorCode.PasswordNotMatch)
{
    _errorMessage = "Invalid username/password";
}
else if (result.ErrorCode == WebServiceTestApp.AuthenticationWS.LoginErrorCode.NoError)
{
    cookieContainer = authenticateWS.CookieContainer;

    if (cookieContainer == null || cookieContainer.Count == 0)
    {
        _errorMessage = "The authentication webservice  ("
            + authenticateWS.Url
            + ") did not return any authentication cookie.";
    }

  CookieCollection cc = cookieContainer.GetCookies(new Uri(authenticateWS.Url));
  foreach (Cookie cookie in cc)
  {
      if (cookie.Name == result.CookieName)
      {
          _authenticationCookie = cookie;
          break;
      }
  }

  if (_authenticationCookie == null)
  {
      _errorMessage = "The authentication webservice  ("
          + authenticateWS.Url
          + ") did not return any authentication cookie named " + result.CookieName + ".";
  }
}
else
{
    throw new Exception("The authentication webservice ("
        + authenticateWS.Url
        + ") returned an invalid enum value (the return value, "
        + result.ErrorCode.ToString()
        + ", does not belong to " + "LoginErrorCode");
}
```

Where **authenticateWS** is the Authentication WebService proxy.

To complete the illustration, we now show an example of a subsequent call to a FEI21 WebService:

```
private void btnGetHorseColorList_Click(object sender, EventArgs e)
{
  WebServiceTestApp.HorseWS horseWS = new WebServiceTestApp.HorseWS();

  WebServiceTestApp.AuthHeader authHeader = new WebServiceTestApp.AuthHeader();

  authHeader.UserName = "MyName";
  authHeader.Language = "en";

  horseWS.AuthHeaderValue = authHeader;
```

```
  horseWS.CookieContainer = new CookieContainer();
  horseWS.CookieContainer.Add(_authenticationCookie);

  WebServiceTestApp.HorseColor[] _horseColorList = horseWS.getHorseColorList();

  foreach (WebServiceTestApp.HorseColor color in horseColorList)
  {
    // Display color
  }

}
```

The relevant code in this example is printed in orange: this is the correct way (in C#) to use the authentication cookie received during the authentication process.

3.2 End User Information

Before each call to one of the **FEI21 WebServices**, a header should be added. This header contains information about the end user.

Currently, two properties can be filled:

- **Username**: the end user username string that identifies the user.
- **Language**: a two letter ISO 639-1 language code to set the expected language for returned data.

Remark: in version 1.0.0 of the **FEI WebServices application**, only English ("en") is supported.

```
private void AddEndUserInfoToHorseWS(WebServiceTestApp.HorseWS.HorseWS ws)
{
    WebServiceTestApp.HorseWS.AuthHeader authHeader =
new WebServiceTestApp.HorseWS.AuthHeader();

    authHeader.UserName = "John Doe";
    authHeader.Language = "en";

    ws.AuthHeaderValue = authHeader;
}
```

3.3 Authentication WebService Description

The Authentication WebService contains two methods:

- Mode
- Login

The first one, Mode, is obsolete since version 2.0 (see below).

We give now a brief description of the two methods.

3.3.1 Mode WebMethod

Syntax

```
public AuthenticationMode[] Mode()
```

Parameters

N/A

Return value

An AuthenticationMode enum value that identifies the user authentication system that is being used by the WebServices application.

Possible values are:

- None,
- Windows,
- Passport,
- Forms.

Remarks

Since version 2.0, this method is obsolete and is provided only for backward compatibility. It always returns the "Forms" AuthenticationMode and thus it is not necessary anymore to call this method as an opening step to the authentication process.

3.3.2 Login WebMethod

Syntax

```
public LoginResult[] Login(string username, string password)
```

Parameters

- **username**

  The user's login name for the WebServices application (delivered by FEI).

- **password**

  The user's password.

Return value

A LoginResult structure that contains an error code in its ErrorCode field and the name of the authentication cookie in its CookieName property.

The login process is successful if the ErrorCode field contains the value NoError.

The possible values are:

- NoError,
- PasswordNotMatch.

Remarks

The cookie contains an authentication ticket that will be used to authenticate subsequent requests to the server (see section 3.1 for an example).

4. General considerations about WebMethods

4.1 Exceptions and Messages

The behaviour of the FEI21 WebServices is designed as follow:

- For WebMethods that have an argument "out **Message[]** ***Messages***", the client application must check the length of this Array after the call to the WebMethod. It will in fact contain any predicable exception that occurred during the WebMethod request. See below to know how to use the values returned in this parameter.

  For unexpected exceptions in those WebMethods, a Soap Exception is raised and must be processed by the client application.

- For unexpected exceptions in all other WebMethods, Soap Exceptions are raised that must be processed by the client application

Use of the returned value in the "*Messages*" parameter:

If no errors or warnings are generated then the value of *Messages* will be null.

If there were errors or warnings then *Messages* will contain an array of message object. For each *Message* object, the property "UID" contains the identifier of the message type and the property "Detail" contains detailed information about the exception. This identifier must be used to get a *MessageType* object by using the getMessageTypeList() method in CommonWS.asmx and finding the corresponding message type (the match must be performed between the "UID" property of the Message object and the "Id" property of the MessageType object). On this object, the description and criticality of the exception can be read in the "Description" and "IsWarning", "IsError" and "IsCritical" properties.

The inspection of the whole *Messages* array will show how to deal with the problem:-

- If the worst case message is a warning (W) then the user can deal with it, (e.g. by more restrictive search conditions).
- If the worst case message is an exception (E) the client application program should have dealt with it before calling the method, and needs amendment.
- If the worst case message is a critical unhandled exception (C) an unexpected fault has occurred at the server (e.g. power or network failure) and the service is likely to be down until the fault is fixed. The FEI web service administrators should be notified immediately.

4.2 Transaction control number

To ensure this concurrency control mechanism, the FEI system appends to each record a special property called TCN, which means "Transaction control number". This property is provided in each class returned by the WebServices. You can use this number to check whether the local copy of a given record (for example a horse record) has changed since you got it: if the TCN has changed that means that some data has changed and that you should replace (or merge) your local data with the received data; and if it has not changed, you know then that the received data matches your local version of the data.

4.3 Note about the getVersion() method

Note that this method returns the version number of the FEI Core Application. The FEI WebServices for Organizers and IT Providers is part of it and has its own version number contained in the name of the asmx file.

4.4 Use of the getLookupDateList() method

The method getLookupDateList() returns the last modification date for each type of lookup data (such as coutries, disciplines, horse colors, message types, …). Those dates must thus be stored by the client applications. A periodic call to the getLookupDateList() will thus shows which lookup data must be refreshed: the lookup data for which the date has changed must be reloaded. To do that, the corresponding method (getCountryList(),getDisciplineList(), getHorseColorList(), getMessageTypeList(), …) must be called and the result stored in the client system.

Be sure not to call one of the getXXXList() method each time the lookup data is required but to store it and apply the procedure exposed above.

4.5 .NET ASMX vs WCF WebServices

Since version 2010 of Microsoft Visual Studio, Microsoft tends to promote the WCF Services Reference instead of the "old fashion" Web References. Indeed, in a Visual Studio 2010 project, when right clicking on the "References" node, you are only proposed to "Add Service Reference…" and not anymore to "Add Web Reference".

If you want to add a Web Reference, you have to click on "Add Service Reference…" and then click on the "Advanced…" button on the bottom left corner of the dialog box to reach another dialog box where you can finally click on "Add Web Reference…" to end up with the usual interface to add a Web Reference to your project.

If you want to develop your client application with a WCF Service, use the interface on the first modal dialog box. Note however that you will have to deal with the cookie persistence like described at section 3.1 but with different objects. Look at the following articles for more information:

- http://megakemp.com/2009/02/06/managing-shared-cookies-in-wcf/
- http://msdn.microsoft.com/fr-fr/library/system.servicemodel.basichttpbinding.allowcookies.aspx

4.6 Workaround for certificate issues

Due to a SSL certificate issue in the Integration and Validation environments (see section 2), you may experience some trouble connecting to the Web Services in those environments if you are developing with Microsoft Visual Studio as you will indeed get a security issue during the hand shaking between your application and the FEI WebServices hosting server.

Of course, this issue (and the following workaround) is not relevant in the Production environment: if there is a certificate validation error this means that there really is a problem and you should contact FEI.

A workaround to this issue is the following:

- Add a custom validation handler on the class `ServicePointManager`:

```
ServicePointManager.ServerCertificateValidationCallback +=
 new System.Net.Security.RemoteCertificateValidationCallback(customXertificateValidation);}
```

This can for example be done in the `Application_Start` method of your `Global.asax.cs` file (if your are in an ASP.NET project).

- In the delegate `customXertificateValidation`, always returns "true", thus ignoring SSL certificate issues:

```
private static bool customXertificateValidation(object sender, X509Certificate cert,
X509Chain chain, System.Net.Security.SslPolicyErrors error)
{
    return true;
}
```

Remark:

- According to our experience, there is for example no such issue with PHP (with the default configuration of SSL).

*** End of document ***