Using the Amazon Product Advertising API with 4D

By Keisuke Miyako, 4D Japan.

Technical Note 09-41

Table of Contents

Abstract

The purpose of this Technical Note is to illustrate how advanced Web Services Client functionality can be implemented in 4D. Amazon's Product Advertising API is used as an example, as the service requires extra coding in order to be called from 4D. More specifically, HMAC-SHA256 digest signatures, REST type web services, complex SOAP headers and other topics related to web services are discussed in this document.

Introduction

Amazon Product Advertising API (formally Amazon Associates Web Service) is an API (Application Programming Interface) designed to offer external applications the ability to access inventory and other kinds of information provided by Amazon. Members of the Amazon Associates Program can use this feature in a creative way to earn higher referral fees as an affiliate to Amazon's commercial services.

https://affiliate-program.amazon.com/gp/advertising/api/detail/main.html

The API conforms to the popular REST (Representational State Transfer) protocol, in addition to SOAP (Simple Object Access Protocol). Both basically offer the same services, so it is up to the developer to decide which type of Web Services to use.

http://en.wikipedia.org/wiki/Representational_State_Transfer

http://en.wikipedia.org/wiki/SOAP

The built-in web services client feature of 4D (CALL WEB SERVICE/GET WEB SERVICE RESULT) cannot be used to call the Amazon Product Advertising API, for reasons that are explained later. Instead, a combined solution of XML commands, Internet Commands and other methods is used to achieve our objective. But before moving any further, let's step back a little and get a better understanding of how the service works.

Getting Started with AWS

AWS (Amazon Web Services) is a collection of APIs that enables external applications to access various resources provided by Amazon. Services most recently added to this suite include the EC2, S3 and SQS. The Product Advertising API specifically offers means for external applications to access the e-commerce branch of Amazon's web services.

Create an Amazon account

To use AWS, you first need an account at Amazon. You may obtain one by signing up at one of their major marketplaces such as:

https://associates.amazon.ca/gp/flex/advertising/api/sign-in.html

https://partenaires.amazon.fr/gp/flex/advertising/api/sign-in.html

https://affiliate-program.amazon.com/gp/flex/advertising/api/sign-in-jp.html

https://affiliate-program.amazon.co.uk/gp/flex/advertising/api/sign-in.html

Register as an AWS developer

Every API at AWS expects a unique AWS account Key that will identify you as an AWS developer. You can register as a developer at the URL given below.

https://aws-portal.amazon.com/gp/aws/developer/account/index.html

At this point you should have the following 3 items.

- An Amazon Account ID
- An AWS Access Key ID
- A secret Access Key associated to the AWS Access Key ID

AWS and the 4D Web Services Wizard

Calling published SOAP Web Services from 4D is made remarkably simple thanks to the Web Services Wizard. In most cases, you are literally a few clicks away from using such services once you know its WSDL location.

Amazon does publish its Web Services by WSDL, so you might expect the Wizard to automatically create a set of proxy methods, which unfortunately is not going to be the case (otherwise this Tech Note would end here). The following aspects about the built-in SOAP features of 4D must be considered. First, it cannot process complex SOAP responses. Second, it cannot manage SOAP headers with multiple elements at the primary level. Third, 4D does not have a native mechanism for creating HMAC-SHA256 signatures that is required to access the Product Advertising API.

Complex SOAP web services pose a problem

4D has a built in Web Services Wizard that can parse a WSDL (Web Services Description Language) file and auto generate proxy methods that behave like a local method but internally calls a remote procedure. Amazon does publish their services as WSDL, but unfortunately is not fully analyzed by 4D. This is because

the protocol is of what is called a complex type and there is a limit as to how deep into the description file the Web Services Wizard is willing to explore.

To verify this point, open the Web Services Wizard from the Design menu and enter one of the Amazon WSDL locations listed below.

http://ecs.amazonaws.com/AWSECommerceService/2009-07-01/AWSECommerceService.wsdl

http://ecs.amazonaws.com/AWSECommerceService/2009-07-01/UK/AWSECommerceService.wsdl

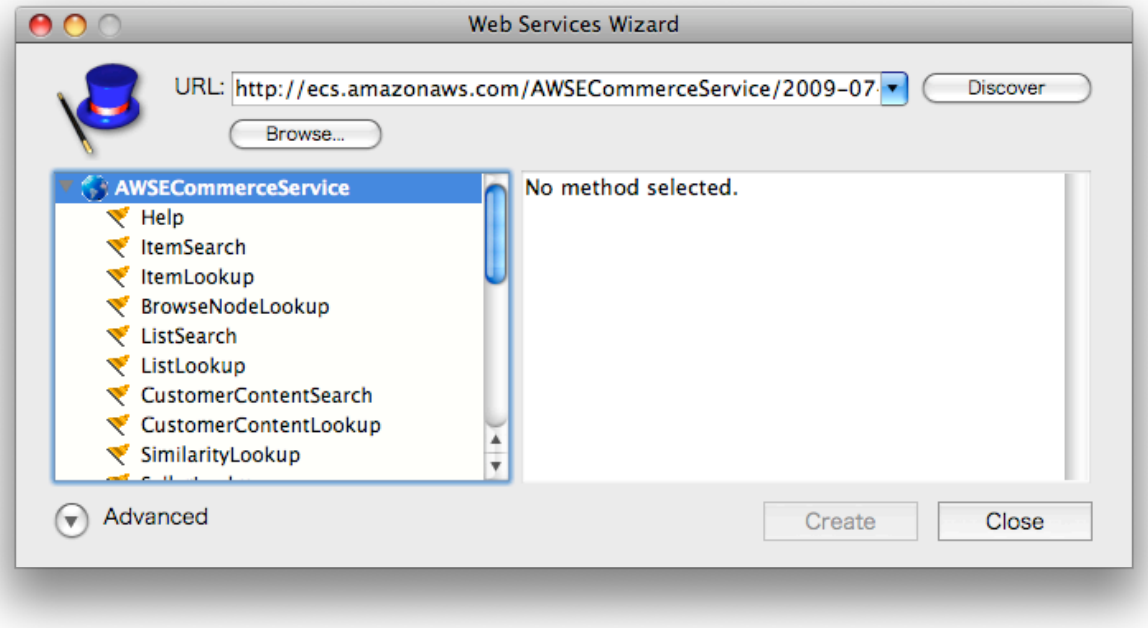
http://ecs.amazonaws.com/AWSECommerceService/2009-07-01/DE/AWSECommerceService.wsdl

http://ecs.amazonaws.com/AWSECommerceService/2009-07-01/JP/AWSECommerceService.wsdl

http://ecs.amazonaws.com/AWSECommerceService/2009-07-01/FR/AWSECommerceService.wsdl

http://ecs.amazonaws.com/AWSECommerceService/2009-07-01/CA/AWSECommerceService.wsdl

The URL will be successfully processed, but an orange alert flag is displayed, predicting that the generated codes are probably going to be imperfect.



Indeed the code that follows the GET WEB SERVICE RESULT command in the proxy method that is generated by the Wizard does not fully analyze the XML response and requires manual completion. In fact, the response does not

contain any meaningful data at all because the request sent by CALL WEB SERVICE is lacking necessary information in the SOAP header.

Complex SOAP headers are not fully supported either

The 4D command to set a specific SOAP header is SET WEB SERVICE OPTION. You call the command with the selector constant 'Web Service SOAP Header' as the first parameter and a root DOM reference to the XML tree as the second. The next CALL WEB SERVICE will use that XML as its SOAP header.

According to the Amazon documentation, one must include valid authentication in the Web Services SOAP header. The structure is illustrated in the example below.

```
<SOAP-ENV:Header xmlns:aws="http://security.amazonaws.com/doc/2007-01-01/">
<aws:AWSAccessKeyId>__access_id__</aws:AWSAccessKeyId>
<aws:Timestamp>__timestamp__</aws:Timestamp>
<aws:Signature>__$signature__</aws:Signature>
</SOAP-ENV:Header>
```

Unfortunately this form of XML is not compatible with SET WEB SERVICE OPTION. For example, let's create the same XML structure and use it as the SOAP header.

```
$Header:=DOM Create XML Ref("SOAP-
ENV:Header";"http://schemas.xmlsoap.org/soap/envelope/";"xmlns:aws";"http://sec
urity.amazonaws.com/doc/2007-01-01/")
$AWSAccessKeyId:=DOM Create XML element($Header;"aws:AWSAccessKeyId")
DOM SET XML ELEMENT VALUE($AWSAccessKeyId;"AWSAccessKeyId")
$Timestamp:=DOM Create XML element($Header;"aws:Timestamp")
DOM SET XML ELEMENT VALUE($Timestamp;"Timestamp")
$Signature:=DOM Create XML element($Header;"aws:Signature")
DOM SET XML ELEMENT VALUE($Signature;"Signature")

SET WEB SERVICE OPTION(Web Service SOAP Header ;$Header)
```

This is what the SOAP server would actually see.

```
<SOAP-ENV:Header>
<SOAP-ENV:Header xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:aws="http://security.amazonaws.com/doc/2007-01-01/">

  <aws:AWSAccessKeyId>AWSAccessKeyId</aws:AWSAccessKeyId>
  <aws:Timestamp>Timestamp</aws:Timestamp>
  <aws:Signature>Signature</aws:Signature>

</SOAP-ENV:Header>
</SOAP-ENV:Header>
```

Notice the redundant 'SOAP-ENV:Header'. The SET WEB SERVICE OPTION command creates a new 'SOAP-ENV:Header' and converts the root XML that was passed, and as a result, we end up with a nested header element, which breaks the expected format. At this point we are forced to give up on CALL WEB SERVICE.

Is it time to put matters to REST?

Implementation of the REST protocol typically implies that an XML resource can be retrieved from the server by requesting a verbose URL in which all the arguments are included using the "…?name=value&name=value…" syntax.

REST is far more popular compared to SOAP in certain areas of the Internet; the Blogosphere for example. Amazon offers their Web Services in both SOAP and REST, but the REST version is said to account for the majority of network traffic that goes through their system.

REST assured with DOM Parse XML source

REST can be significantly easier to implement than SOAP, since the request is not a structured XML but rather a linear URL. One can make a sophisticated request by adding as many arguments to the base URL as seen necessary. Once the URL is complete, then it can be sent to the server by passing it to the 4D command DOM Parse XML source (pass a network URL instead of a local document path). If the REST server requires requests to be sent via a secure protocol, simply specify the URL by prefixing it with https. With REST we need not care about the complex SOAP header, we just have to make sure the URL contains no errors.

What about HMAC-SHA256 signatures?

Until August 2009, Amazon's Product Advertising API required no additional authentication other than the access ID and secret key. Starting from that month, it is now mandatory to provide cryptographic signatures in order to make a request. The specific procedure though which the signature should be created is described in the online API documentation.

HMAC-SHA256 signatures for REST requests

http://docs.amazonwebservices.com/AWSECommerceService/latest/DG/HMACSignatures.html

Authenticating SOAP requests

http://docs.amazonwebservices.com/AWSECommerceService/latest/DG/SOAP_SOAPAuth.html

The Product Advertising API accepts SOAP request authentication in two ways; either WS-Security, which requires X.509 certificates, or HMAC-SHA256. In any case the request must be made with https. In this Tech Note we focus on HMAC-SHA256, as it is compatible with both REST and SOAP.

Introducing the 4D OpenSSL Plugin

Amazon Web Services requires HMAC-SHA256 authentication. You may be aware that there are no commands in 4D v11 SQL to perform standard encryptions apart from the simple MD5 (part of 4D Pack v11).

AP Get file MD5 digest

http://www.4d.com/docs/CMU/CMU61950.HTM

Well, as a matter of fact, 4D does have the ability to perform a variety of standard security features, including HAMC and SHA. The catch is that the features are not exposed to the developer.

The 4D OpenSSL Plugin, whose executable and source code is provided with this Tech Note, exploits the OpenSSL module that is already embedded in your 4D application. For the purpose of this Tech Note, only the encryption commands, MD5, SHA1, SHA256, SHA384, SHA512 and their HMAC counterparts are provided, but in theory the same technique can be used to fully "port" the OpenSSL library.

For more information on OpenSSL, please consult their web site.

http://www.openssl.org/

Points of interest in the Plugin

The plugin is compiled as 32/64 bit Universal Binary for Mac and 32 bit on Windows. The minimum deployment target is OS 10.4 and Windows XP. It uses the new Unicode plugin API, which means it can run on both Unicode and non-Unicode 4D applications.

The syntax for the commands are all the same; it takes 1 (2 for the HMAC implementations) BLOB arguments and return TEXT. To encrypt a document, you must first load it in a BLOB. To encrypt TEXT you need to convert it to BLOB with the CONVERT FROM TEXT command (you should not use the TEXT TO BLOB command anymore, unless you specifically favor the proprietary 4D conversion over the standard ICU library).

The return value is a hexadecimal representation of the digest value. To convert them to other formats, like base64, you need to pack the string into raw binary and then perform the encoding as required. A sample method is provided in the Amazon demo application.

More points of interest in the Plugin

Here is some additional information for plugin developers who want to extend or modify the C source code. Note that 4D is not necessarily committed to support the OpenSSL module, it just happens to be the current tool of choice to implement specific features. Although fairly unlikely, it is possible that the plugin becomes dysfunctional due to a redesign in the main 4D application.

Loading the private OpenSSL module

As discussed earlier, the plugin does not perform the encryption itself, it simply calls the features provided by the OpenSSL module that is included in the 4D application. The procedure is almost the same on both platforms.

1. Identify the main 4D application.

   Windows: GetModuleFileName

   Mac OS X: CFBundleGetMainBundle

2. Find the private frameworks folder.

   Windows: N/A. (the application is not a bundle.)

   Mac OS X: *CFBundleCopyPrivateFrameworksURL*

3. Load the OpenSSL.framework

   Windows: LoadLibraryEx

   Mac OS X: CFBundleCreate

Getting pointers to API functions

First we need to type define the arguments list. For example, we learn from the documentation that there is a function named *EVP_MD_CTX_init*, and that it takes a pointers to an EVP_MD_CTX structure.

```
void EVP_MD_CTX_init(EVP_MD_CTX *ctx);
http://www.openssl.org/docs/crypto/EVP_DigestInit.html
```

So, we type define this function as follows.

```
typedef void (*EVP_MD_FUNC_CTX)(EVP_MD_CTX *);
```

Next, we declare the function in question as of this newly defined type.

```
EVP_MD_FUNC_CTX EVP_MD_CTX_init;
```

Finally, we bind this function with the API entry point. The functions for this operation are *GetProcAddress* on Windows and *CFBundleGetFunctionPointerForName* on Mac. For the sake of simplicity we may declare a platform neutral macro.

```
#if VERSIONWIN
#define GET_PROC(t,vname,fname) (t) GetProcAddress(vname, fname);
#else
#define GET_PROC(t,vname,fname) (t)
CFBundleGetFunctionPointerForName(vname,CFSTR(fname));
#endif
```

This way we can obtain the function pointer address with the same code in our main source code.

```
EVP_MD_CTX_init = GET_PROC(EVP_MD_FUNC_CTX, openssl, "EVP_MD_CTX_init");
```

Now we are ready to use this function, regardless of the platform, directly in our plugin source code.

More about Encryption

Although this method of calling the internal library for encryption should be fairly robust, there are alternative ways to achieve such objectives. One is to call the system API on each platform. The 4D Cryptographic Hash plugin also provided with this Tech Note provides the same set of functions, but independent of the OpenSSL module. If you are interested in a more generic way of performing such tasks, its source code may be useful (Some advanced algorithms require Windows XP SP3 or later).

http://msdn.microsoft.com/en-us/library/aa386983(VS.85).aspx

Even more about Encryption

If you for some reason object to using plugins, the Hash and Digest Component might be your friend. It is an example of how encryption can be performed using LAUNCH EXTERNAL PROCESS, thanks to VBScript and ruby (the openssl shipped with Mac OS X 10.5 does not support SHA256, hence the use of ruby instead).

SOAP without CALL WEB SERVICE

CALL WEB SERVICE is one of the most powerful and productive commands available in 4D. Thanks to this command, we can make XML-based remote procedure calls without even having to think about XML; analyzing the WSDL, constructing the request SOAP envelope, binding parameters, parsing the response XML and extracting returned values - all of that is managed automatically.

The command is at is best when used for cross 4D communications; one enjoys the extra value of compressed data transmission, array binding, etc. Still, being compatible with industrial standard, there is nothing to stop the developer from

using this command to call web services hosted on remote systems other than 4D - except one important limitation, that is it does not fully support complex types.

Complex web services use structured custom XML elements in their requests and/or responses. If the complexity is in the SOAP body, we can create an XML envelope ourselves using 4D DOM commands prior to a CALL WEB SERVICE with the 'manual in' argument. Likewise it is possible to parse complex responses with 4D DOM commands and the 'manual out' option.

However, if the complexity is in the SOAP header, what we can do is quite limited. SET WEB SERVICE OPTION can help, which allow us to include a custom DOM tree inside the default SOAP header. Still, one must keep in mind that the custom header is not going to replace the default header, it is simply going to be added to the header, which may not be the desired result.

Sometimes, remote procedure calls (also known as proxy methods) created by the web services wizard may end up in requests being rejected. To check whether the request sent was properly formed, one can check the outgoing HTTP with a packet monitor, or more pragmatically, send the same request to the local 4D web server and see what the received HTTP looks like. If the SOAP header does not seem right, and cannot be fixed however you try with SET WEB SERVICE OPTION, then it might be time to consider not using CALL WEB SERVICE.

Call web services with 4D Internet Commands

The *soap call* method, used extensively in the sample database, is an alternative to the built-in command CALL WEB SERVICE. Basically it is an HTTP POST implementation that uses 4D Internet Commands. It gives the developer full control (and responsibility) over what HTTP is sent to make a SOAP web service request. The parameter list and internal logic is customized specifically for SOAP calls. For a more generic version of HTTP POST, please refer to the *download* example discussed later.

```
`$1: end point url, prefixed with the protocol (http or https).
`$2: the method name as in the SoapAction HTTP field.
`$3: the soap header and body (i.e. HTTP body).
`$0: the response HTTP body, if it looks like a SOAP envelope.

$0:=soap call ($endpoint_t;$soap_name_t;$http_body_x)
```

The 4D AWS Library component

This component provides full access via the 4D language to all of the features available in Amazon's Product Advertising API. Both REST and SOAP are supported.

How to use the component

The procedure for calling Amazon's web services is basically uniform. First, you populate and array of arguments, whose size and definition are documented in the method *AWS_xxx_Request* where xxx is the name of the web service you want to call. Then, you call the said method with a single parameter, which is a pointer to the array of arguments. The method will create and return an intermediate request XML.

Next you pass the XML request to either *AWS_xxx_REST* or *AWS_xxx_SOAP*, depending on which implementation of web services you wish to use. Each method require 3 additional parameters, which is your Amazon Account ID, your Amazon Secret Key and the locale {jp|uk|ca|fr|de|us}. In response you will get the response XML as TEXT if the call was REST or BLOB if the call was SOAP.

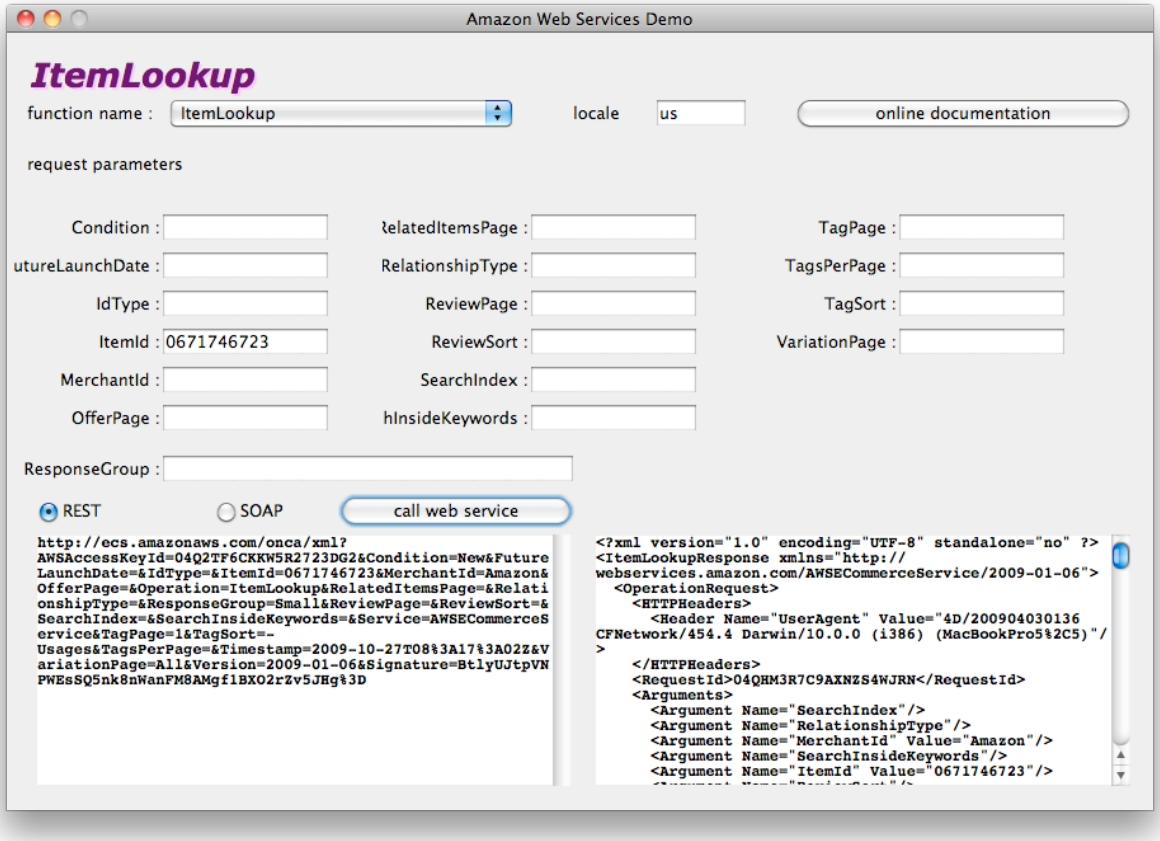
Points of interest in the component

Since the web services interface may evolve over time, variable elements are taken out of individual method as much as possible. For example, the soap namespace, soap endpoint, rest version, rest url, etc. are implemented as return values of methods that bare their name. Rather than directly constructing a request SOAP envelope or a REST URL through hard coding, the component takes a two-step approach, where a generic XML is created and transformed to a SOAP envelope or REST URL using APPLY XSLT TRANSFORMATION. As such, the component can easily be extended to support other forms of object notation, JSON for example, by simply adding or modifying the XSL.

For REST web services, it is imperative to encode the extended characters that appear in the URL. To do this, the component takes advantage of the xalan library that is shipped with 4D, by calling the XSL function *str:encode-uri* through APPLY XSLT TRANSFORMATION. For your information, the full set of XSL functions wrapped as 4D methods is included in the XSLT Goodies component.

Test suite for the component

The 4D AWS Test application is a straightforward GUI for the 4D AWS Library component. It lets you quickly test all of the SOAP/REST the web services API with various parameters and values. The outgoing and incoming data are shown live.



To test a web service call, select a method from the function name popup menu. A minimum set of sample data is provided to make a valid call. You may edit the parameters if necessary. Each time the data is changed, the request preview on the bottom left is updated, which is an URL for a REST request or an XML for SOAP. Click on "call web service" to send the request to Amazon. The response XML will be displayed on the bottom right. Click on "online documentation" to browse API documentation.

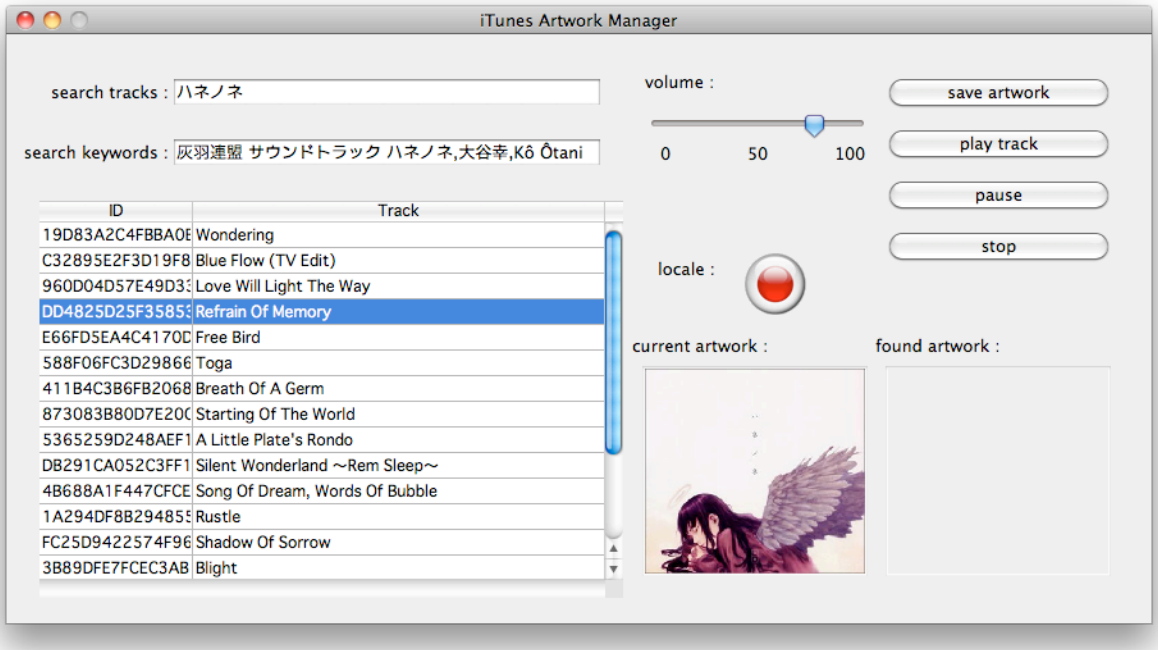
Points of interest in the sample application

The functioning version of the test suite is compiled, as the source code includes the author's Amazon Account information, which should not be disclosed. A source code version is also provided, without the confidential information. To work on the source code version, edit the method *_sample_request_id*.

Example application: iTunes Artwork Manager

4D iTunes.4dbase is a simple application that demonstrates how the component can be implemented. It is cross platform that will access music tracks saved in the local iTunes library, search a related image from Amazon's marketplace using web

services, which can optionally be selected as the album artwork for the specified music track(s).



In the "search tracks" field, type keywords to specify music tracks. Matching tracks are shown in the listbox below. For each selected track, the current artwork, if one exists, is shown on the right. To search Amazon, select the locale, modify the comma-separated keywords in the "search keywords" and validate. The image is shown in the "found artwork" area if the search is successful. Select the track(s) to which you want to associate the image and click "save artwork".

Points of interest in the sample application

The application uses AppleScript on Mac and VBA on Windows, to manipulate iTunes. All the source codes are provided, though the method that actually retrieves images from Amazon is protected (a compiled component), as it contains the author's account information. The source codes for the protected methods are provided without the account data as well. Note that the compiled component is dependant on other components, which should also be complied and placed in the Components folder of the host database in order to be used as a component.

There is a method called *download* in the component "4D AWS Artwork", which might be portable as a generic utility method. It is a high level API that lets you easily receive data from a URL. It is a composite method that demonstrates how the same job (HTTP POST) can be done, either with or without a plugin. Primarily it uses 4D Internet Commands, but alternatively you can switch to AppleScript/VBA.

Conclusion

This Tech Note illustrated how advanced Web Services Client functionality can be implemented in 4D. We used Amazon's Product Advertising API as an example, as the service requires some extra coding in order to be called from 4D. Several ways to implement an HMAC-SHA256 digest feature were explained. The use of DOM parse XML source to call REST over http/https and the used of 4D Internet Command to call complex SOAP services were also described. A high level component to make REST/SOAP calls to the Product Advertising API was provided, together with an accompanying test suite application to verify outgoing and incoming data and a simple application that uses the component.