Pacific Masters Web Services

December 22, 2019

Bob Upshaw

# Overview

A web service is an interface to a function that lives on a web server. The user of a web service is a “client”, and the function is known as a “service”. The service operates just like a web server, running on a server listening on a port for HTTP requests. The client operates just like a web client, running on a client machine issuing HTTP requests to the server.

The PAC web services conform to the REST architecture. See <https://en.wikipedia.org/wiki/Representational_state_transfer> for more details.

The original purpose of our web services was to support AGSOTY. Specifically, AGSOTY requires looking up Pacific Masters records set during a specific season. The GetRecords service allows a client (AGSOTY) to send a JSON request to our web server (<https://pacificmasters.org>) and receive back all current and historical swim records for a specific course (SCY, SCM, or LCM.)

This document explains how to use a PAC web service, how to write a service, and the PAC web services infrastructure.

Our web services are accessible on both the dev and production servers.

# Using a PAC Web Service

A client application uses a PAC web service by invoking the desired PACWebService library function in the WebServiceClient PERL package. We will use the GetRecords service as an example. The GetRecords() function is a Perl function because it was initially designed to be used by AGSOTY, a Perl based project. To use GetRecords() the client makes the following call:

require WebServiceClient;

my $JSONdata = WebServiceClient::GetRecords( $trueCourse );

This call passes one parameter to the service: the course (e.g. “LCM”.) It receives back a JSON structure representing the result (an array of associative arrays of PAC records.)

At the time of this writing there are two web services:

* GetRecords – return PAC swim records from our production database.
* GetRecords\_dev – return PAC records from our development database. These data are probably not completely current nor correct.

# Implementing a Service

To implement a new web service you will need to add two interface files: one for the client side (used directly by your client code), and one for the server side (implements the service on the server.).

## Client

The client side of the service should be a very thin Perl function. The client interface is usually written in the same language as the calling client, but keeping it thin allows makes it easier to interface to other languages. Thus the same service can be used by different clients written in different languages, where most of the work is language-agnostic since most of the work occurs on the server.

If the client service is written in Perl it can be added directly to the existing Perl client service file located in PACWebService/Client/Perl/WebServicesClient.pm. Or, if desired, it can be a separate file in the Perl subdirectory, or a different subdirectory if written in a different language. In any case it will call GetData() which already exists for Perl services or a different GetData() if written in a different language. Since the result of GetData() is a JSON structure it’s result is language agnostic.

The client doesn’t really do anything except call GetData(), and GetData() doesn’t really do anything except:

- send a request (GET) to the service server

- receive a JSON result.

The only interesting part of the client is that it will enforce authentication and encryption of both the request and the response. The service will hit our server, and both of them (dev and prod) redirect non-SSL requests to SSL, so that much security is free. But additional security will live on top of that so the redirect isn’t visible.

Of course the client can add error detection/correction if desired, but the more added to the client the more it will cost to support other languages.

See the existing clients for examples.

## Server

The server side is accessible via this URL:

http[s]://{domain}/api/zzzzz/xxxxx.php

where “zzzzz” and “xxxxx” usually indicate the service and is defined in the client’s %OurServices hash. The request will contain a dynamically generated encryption key and zero or more query arguments. For example, for the GetRecords service, the client passeone argument: the course (e.g. “SCY”), and the service on our production server is here:

<https://pacificmasters.org/api/pacrecords/GetRecords.php>

To add a new service you can add PHP code to PACWebService/Server/PHP/zzzzz/xxxxx.php, where “zzzzz” and “xxxxx” represent the service. Or, if you want to write the service in a different language feel free to add a different directory parallel to the PHP subdirectory with the appropriate contents. We use PHP because our server is basically a PHP-based server (Drupal) and the current service (GetRecords) makes use of an existing PHP library.

As explained above, we have allocated the /api/ directory in the root of the web server as the root location for all services. The services are subdivided by language, and then supported by one or more files making up the service code. For example, the production GetRecords service is installed here:

/usr/home/pacmasters/public\_html/pacificmasters.org/api/pacrecords/GetRecords.php

The development service is the same idea.

(Note: we have noticed that the /api/ directory will disappear when major security updates are applied to the Drupal server. But making /api/ a link to a different subdirectory seems to work fine so that’s what we do.)

See an existing service for an example.

# Installing a PAC Web Service

At the time of this writing this is how I install the GetRecords web services on the dev machine. Same idea for the production machine.

1. ssh to the dev platform, then change directory to ~/Automation
2. If a PACWebService subdirectory DOES NOT exist:
   1. git clone <https://github.com/bobup/PACWebService>

or, if it does exist (assume it is already a local git repository):

* 1. cd PACWebService; git pull; cd ..

1. We will install the PHP version of the service:
   1. cd PACWebService/Service/PHP
   2. cp -R PACRecords/\* ~/public\_html/pacmdev.org/api/pacrecords/

Note: if the pacrecords/ directory doesn’t exist then create it.

* 1. cd ../../doc
  2. cp README.txt ~/public\_html/pacmdev.org/api/

Note: if the README.txt file already exists in the /api/ subdirectory you don’t have to copy it again unless it has changed.

* 1. Test it by using your browser to hit the service:

<https://pacmdev.org/api/pacrecords/GetRecords.php?LCM>

You should see the JSON result.

1. The client side of the service can remain where it is as long as the client user (e.g. the AGSOTY code) looks for it in the correct place (which it does at this time.)