Playing With HTTP Libraries

Adrian Kosmaczewski

2008-03-26

It's fun to find out how to tackle the same task in different programming languages; in this case, it's all about doing HTTP requests over a network: fortunately, there are networking libraries in virtually all major programming languages. In my current project, I'm generating wrappers easing the access to the core of the project itself, a RESTful API. This way, developers interested in using the API can just take a wrapper, include it in their projects, and start coding right away. No need to know this (relatively low-level) stuff; just use the API. The wrappers themselves are auto-generated from the API definition itself, but that's another story ;)

Below there is a sample of the different ways I've found to do a network access to a remote server, using HTTP Basic Authentication and a couple of headers, in PHP, Ruby, Python, JavaScript, and even Objective-C! I'm even generating ActionScript 3.0 code, but I'm not a Flash coder :) So I'll post the wrappers that work best at the moment, and in the future I'll include other examples, particularly for .NET, C++ and Java.

In all the cases below, there is a "request" function or method that takes an HTTP verb (`GET`, `POST`, `PUT`, `DELETE`, etc), a URL (without the slash "/" at the beginning) and some parameter data, in the form of a dictionary. The function wraps the underlying libraries of each programming language, offering a simpler interface, and allowing for HTTP Basic Authentication (for HTTP Digest Authentication it would be much, much more complex!). There are synchronous (useful for server or command-line applications) and asynchronous versions (for GUI systems). Off to the code!

In PHP (synchronous):

```
$conf = array(
    "server" => "localhost",
    "username" => "",
    "password" => ""
);

function request($verb, $url, $parameters = array()) {
    global $conf;
```

```
    $headers = array(
    "Content-Type: text/html; charset=utf-8",
    "Accept: application/javascript",
    "Cache-Control: no-cache",
    "Pragma: no-cache",
    "Content-length: " . strlen($xml_data),
    "Authorization: Basic " . base64_encode($conf["username"] . ":" . $conf["password"])
    );

    $path = $conf["server"] . "/" . $url;
    $conn = curl_init();
    curl_setopt($conn, CURLOPT_URL, $path);
    curl_setopt($conn, CURLOPT_HTTPHEADER, $headers);
    curl_setopt($conn, CURLOPT_USERAGENT, "Identify yourself!");
    curl_setopt($conn, CURLOPT_CUSTOMREQUEST, strtoupper($verb));
    curl_setopt($conn, CURLOPT_POSTFIELDS, $xml_data);
    curl_setopt($conn, CURLOPT_RETURNTRANSFER, 1);
    $data = curl_exec($conn);
    $code = curl_getinfo($conn, CURLINFO_HTTP_CODE);

    if($code == 200 && strtoupper($verb) == "GET") {
    return json_decode($data);
    }
    else {
    return $data;
    }

    curl_close($conn);

    return $data;
}
```

The curl library has a distinctive C smell :) The good thing is that after doing this wrapper, doing the C++ one will be fairly straightforward!

In Ruby (synchronous):

```
require 'net/http'
require 'uri'
require 'json'

$conf = {
    "server" => "localhost",
    "username" => "",
    "password" => ""
}
```

```ruby
def request(verb, url, parameters = nil)
    Net::HTTP.start($conf["server"]) do |http|
    headers = {
        "Accept" => "application/javascript",
        "User-Agent" => "Identify yourself!"
    }
    path = "/" + url
    req = nil
    case verb.upcase
        when "GET":
        req = Net::HTTP::Get.new(path, headers)
        when "POST":
        req = Net::HTTP::Post.new(path, headers)
        when "PUT":
        req = Net::HTTP::Put.new(path, headers)
        when "DELETE":
        req = Net::HTTP::Delete.new(path, headers)
        else
        raise Exception.new("Invalid HTTP verb")
    end
    req.basic_auth $conf["username"], $conf["password"]
    req.set_form_data(parameters) if not parameters == nil
    response = http.request(req)
    if response.code == "200" && verb.upcase == "GET"
        JSON.parse(response.body)
    else
        print response.code + " " + response.message + " " + response.body
    end
    end
end
```

The Ruby library is the only one I found so far that features the four HTTP verbs in the interface!

In Python (synchronous):

```python
import simplejson
import httplib, urllib
import base64

conf = {
    "server": "localhost",
    "username": "",
    "password": "",
}

def request(verb, url, parameters = {}):
```

```
    params = urllib.urlencode(parameters)
    base64string = base64.encodestring('%s:%s' % (conf["username"], conf["password"]))[:-1]
    headers = {
        "Accept": "application/javascript",
        "Authorization": "Basic %s" % base64string,
        "User-Agent": "Identify yourself!",
    }
    conn = httplib.HTTPConnection(conf["server"])
    conn.request(verb.upper(), "/" + url, params, headers)
    response = conn.getresponse()
    if response.status == 200 and verb.upper() == "GET":
        obj = simplejson.loads(response.read())
        return obj
    else:
        print response.status, response.reason, response.read()
    conn.close()
```

I much prefer the Ruby way, if you ask me. I don't know, it's more elegant. I can't get used to the indenting!

In JavaScript, using Prototype (asynchronous):

```
var conf = {
    server: "localhost",
    username: "",
    password: ""
};
var request = function(verb, url, parameters, successHandler, errorHandler) {
    // A really ugly way to do HTTP Basic Auth, I know :)
    var api_url = ["http://", conf.username.replace(/@/, "%40"), ":", conf.password, "@", co

    new Ajax.Request(api_url, {
        method: verb.toLowerCase(),
        parameters: parameters,
        requestHeaders: {
            "Accept": "application/javascript",
            "Cache-Control": "no-cache",
            "Pragma": "no-cache",
            "Content-Type": "text/html; charset=utf-8"
        },
        onSuccess: function(transport) {
            if(successHandler) successHandler(transport);
        },
        onFailure: function(transport) {
            if(errorHandler) errorHandler(transport);
        }
    });
```

```
};
```

Again in JavaScript, but this time using jQuery (asynchronous):

```
var conf = {
    server: "localhost",
    username: "",
    password: ""
};
var request = function(verb, url, parameters, successHandler, errorHandler) {
    var api_url = ["http://", conf.username.replace(/@/, "%40"), ":", conf.password, "@", co

    $.ajax({
        type: verb.toUpperCase(),
        url: api_url,
        async: true,
        data: parameters,
        cache: false,
        // jQuery does not allow to change headers otherwise (?) than using the beforeSend(
        beforeSend: function(xhr) {
            xhr.setRequestHeader("Content-Type", "text/html; charset=utf-8");
            xhr.setRequestHeader('Accept', 'application/javascript');
            xhr.setRequestHeader('Cache-Control', 'no-cache');
            xhr.setRequestHeader('Pragma', 'no-cache');
        },
        complete: function(engine, textStatus) {
            if (engine.readyState == 4) {
                if (engine.status == 200 || engine.status == 201) { if(successHandler) succe
                else { if(errorHandler) errorHandler(engine); }
            }
        }
    });

};
```

The last JavaScript one, but this time using bare bones XMLHttpRequest (asynchronous):

```
var conf = {
    server: "localhost",
    username: "",
    password: ""
};
var request = function(verb, url, parameters, successHandler, errorHandler) {
    var api_url = ["http://", conf.username.replace(/@/, "%40"), ":", conf.password, "@", co

    var params = [];
```

```
    for(var item in parameters) {
        params.push(escape(item) + "=" + escape(parameters[item]));
    }
    var engine = null;
    if (window.XMLHttpRequest) {
        engine = new XMLHttpRequest();
    }
    if (window.ActiveXObject) {
        engine = new ActiveXObject("Microsoft.XMLHTTP");
    }
    engine.onreadystatechange = function stateChange() {
        if (engine.readyState == 4) {
            if (engine.status == 200 || engine.status == 201) { if(successHandler) successHa
            else { if(errorHandler) errorHandler(engine); }
        }
    };
    engine.open(verb.toUpperCase(), api_url, true);
    // setRequestHeader() must be called AFTER open() !!
    engine.setRequestHeader("Content-Type", "text/html; charset=utf-8");
    engine.setRequestHeader('Accept', 'application/javascript');
    engine.setRequestHeader('Cache-Control', 'no-cache');
    engine.setRequestHeader('Pragma', 'no-cache');
    engine.send(params.join("&"));
};
```

To use these JavaScript versions, just pass a couple of functions as parameters, and you're done; they will be called in case of success or error, asynchronously.

Finally, the Cocoa / Objective-C wrapper (both synchronous and asynchronous, header and implementation files):

```
/*
Command-line example that uses this library:
#import <Foundation/Foundation.h>

int main (int argc, const char * argv[]) {
    NSAutoreleasePool * pool;
    pool = [[NSAutoreleasePool alloc] init];

    AKResourceSubClass* resource;
    resource = [[AKResourceSubClass alloc] init];
    [resource get];
    NSDictionary* plist = [resource getPropertyList];
    // Do something with the results...

    [pool drain];
    return 0;
```

```
}
*/

#import <Foundation/Foundation.h>

@interface AKResource : NSObject {
    NSMutableData* receivedData;

    NSString* mimeType;
    NSString* server;
    NSString* username;
    NSString* password;

    BOOL async;
}

-(id)init;
-(void)setServer:(NSString*)serverName;
-(void)setUsername:(NSString*)user
    andPassword:(NSString*)pwd;
-(void)connection:(NSURLConnection *)connection
    didReceiveResponse:(NSURLResponse *)response;
-(void)connection:(NSURLConnection *)connection
    didReceiveData:(NSData *)data;
-(void)connection:(NSURLConnection *)connection
    didFailWithError:(NSError *)error;
-(void)connectionDidFinishLoading:(NSURLConnection *)connection;
-(void)sendRequestTo:(NSString*)urlString
    usingVerb:(NSString*)verb
        withParameters:(NSDictionary*)parameters;
-(void)setAsynchronous:(BOOL)asynchronously;
-(NSDictionary*)getPropertyList;
-(NSString*)getResponseText;

@end
```

Now the implementation file:

```
#import "AKResource.h"

@implementation AKResource

-(id)init
{
    self = [super init];

    if(self)
```

```
    {
        receivedData = [[NSMutableData alloc] init];

        mimeType = @"application/plist";
        server = @"localhost";
        username = @"";
        password = @"";

        async = NO;
    }

    return self;
}

-(void)setServer:(NSString*)serverName
{
    server = serverName;
}

-(void)setUsername:(NSString*)user andPassword:(NSString*)pwd
{
    username = user;
    password = pwd;
}

-(void)sendRequestTo:(NSString*)resource
    usingVerb:(NSString*)verb
    withParameters:(NSDictionary*)parameters
{
    NSURL* url = [NSURL URLWithString:[NSString
        stringWithFormat:@"http://%@/%@", server, resource]];

    NSMutableDictionary* headers = [[NSMutableDictionary alloc] init];
    [headers setValue:@"text/html; charset=utf-8"
        forKey:@"Content-Type"];
    [headers setValue:mimeType forKey:@"Accept"];
    [headers setValue:@"no-cache" forKey:@"Cache-Control"];
    [headers setValue:@"no-cache" forKey:@"Pragma"];

    NSMutableURLRequest* request;
    request = [NSMutableURLRequest requestWithURL:url
                cachePolicy:NSURLRequestUseProtocolCachePolicy
                timeoutInterval:60.0];
    [request setHTTPMethod:verb];
    [request setAllHTTPHeaderFields:headers];
```

```
    if (parameters)
    {
        NSMutableString* params = [[NSMutableString alloc] init];
        for (id key in parameters)
        {
            [params appendFormat:@"%@=%@&",
                [key stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncoding],
                [[parameters objectForKey:key]
                    stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncoding]];
        }
        [params deleteCharactersInRange:NSMakeRange([params length] - 1, 1)];

        NSData* body = [params dataUsingEncoding:NSUTF8StringEncoding];
        [request setHTTPBody:body];
    }
    if (async)
    {
        NSURLConnection* connection;
        connection = [[NSURLConnection alloc]  initWithRequest:request
                                                delegate:self
                                                startImmediately:YES];

        if (!connection)
        {
            NSLog(@"Could not open connection to resource");
        }
    }
    else
    {
        NSURLResponse* response = [[NSURLResponse alloc] init];
        NSError* error = [[NSError alloc] init];
        NSData* data = [NSURLConnection
            sendSynchronousRequest:request
            returningResponse:&response
            error:&error];
        [receivedData setData:data];
    }
}

-(void)connection:(NSURLConnection *)connection
        didReceiveAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge
{
    NSURLCredential *newCredential;
    newCredential = [NSURLCredential credentialWithUser:username
                        password:password
                        persistence:NSURLCredentialPersistenceNone];
```

```
    [[challenge sender] useCredential:newCredential
        forAuthenticationChallenge:challenge];
}

-(void)connection:(NSURLConnection *)connection
    didReceiveResponse:(NSURLResponse *)response
{
    NSHTTPURLResponse* httpResponse;
    httpResponse = (NSHTTPURLResponse*)response;
    int statusCode = [httpResponse statusCode];
    if (statusCode != 200)
    {
        NSMutableDictionary* info;
        info = [NSMutableDictionary dictionaryWithObject:[response URL]
            forKey:NSErrorFailingURLStringKey];
        [info setObject:@"An error code different than 200 was received!"
            forKey:NSLocalizedDescriptionKey];
        NSError* error = [NSError errorWithDomain:@"API Wrapper"
            code:statusCode
            userInfo:info];
        NSDictionary* errorData = [NSDictionary
            dictionaryWithObject:error
            forKey:@"error"];

        [[NSNotificationCenter defaultCenter]
            postNotificationName:@"ConnectionDidFailWithStatusCodeNotOK"
            object:self
            userInfo:errorData];
    }
    [receivedData setLength:0];
}

-(void)connection:(NSURLConnection *)connection
    didReceiveData:(NSData *)data
{
    [receivedData appendData:data];
}

-(void)connection:(NSURLConnection *)connection
    didFailWithError:(NSError *)error
{
    [connection release];

    NSDictionary* errorData = [NSDictionary
                                dictionaryWithObject:error
                                forKey:@"error"];
```

```
    [[NSNotificationCenter defaultCenter]
        postNotificationName:@"ConnectionDidFailWithError"
        object:self
        userInfo:errorData];

    NSLog(@"Connection failed! Error - %@ %@",
            [error localizedDescription],
            [[error userInfo] objectForKey:NSErrorFailingURLStringKey]);
}

-(void)connectionDidFinishLoading:(NSURLConnection *)connection
{
    [[NSNotificationCenter defaultCenter]
        postNotificationName:@"ConnectionDidFinishLoading"
        object:self];
    [connection release];
}

-(void)setAsynchronous:(BOOL)asynchronously
{
    async = asynchronously;
}

-(NSDictionary*)getPropertyList
{
    NSString* errorStr = nil;
    NSDictionary* propertyList;
    NSPropertyListFormat format;

    propertyList = [NSPropertyListSerialization
                    propertyListFromData:receivedData
                    mutabilityOption: NSPropertyListImmutable
                    format: &format
                    errorDescription: &errorStr];
    return propertyList;
}

-(NSString*)getResponseText
{
    return [[NSString alloc]
            initWithData:receivedData
            encoding:NSUTF8StringEncoding];
}

@end
```

Rather verbose this last one huh? Objective-C has a distinctive characteristic, inherited from Smalltalk (I suppose): methods with named parameters, like `stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncoding` (decidedly, one of the longest method calls ever :) The good thing is that the code reads like a book. This is why I like this language, as well as Ruby: you can read code in these languages very easily.

Another nice aspect of this Objective-C wrapper is that if you have an API that can return "Property Lists" (like the one I've been working on lately), you can send object graphs serialized in XML that are completely Cocoa-compatible. That's what the getPropertyList method does: client code can deal with standard NSDictionary instances, no need to do anything else!

Of course these are not the only ways to do this, but so far these wrappers have helped me a lot. Don't hesitate to leave your comments below! And most important, have fun! As always, use this code at your own risk.