AjaxStub a Python API

by Carl J. Nobile

August 3, 2007

Version 0.3.0

Contents

1 What is this?

AjaxStub is an API written in Python that generates JavaScript stubs in your web pages. It relieves you of having to write AJAX code in your JavaScript so you can concentrate on your project not the communication process. Using AjaxStub in your web pages is a simple matter of a few steps.

- You instantiate AjaxStub passing it an argument.
- Register your public API.

- Send the initial part of your HTML code.
- Dispatch the AJAX code from AjaxStub.
- Send your JavaScript.
- Send the remaining HTML code.

The remainder of this document will discuss how the internals of AjaxStub work. Examples will be provided for the objects needed to implement AjaxStub in your code.

2 TODO

AjaxStub is in its initial stages of release so there is still much work that needs to be done. Listed below in no particular order is what I would like to include or fix in this package.

1. Subclass AjaxStub with an implementation that uses the Prototype API written by Sam Stephenson `http://www.prototypejs.org/`. If you haven't used his toolkit take a look it's pretty cool.
2. Flesh out registering public APIs written in a class. AjaxStub can currently work with APIs in classes, but the methods in them cannot use different CGI method types yet. This is now partially implemented, but still needs some work.
3. Add the CGI methods PUT and DELETE to the AjaxStub JavaScript class thus making the API a bit more RESTful (Representational State Transfer). See `http://en.wikipedia.org/wiki/REST`.
4. I need to write more example code.

3 Installation

Installation of the AjaxStub is very simple, the only dependency is simplejson by Bob Ippolito `http://cheeseshop.python.org/pypi/simplejson`. Put `AjaxStub.py` and `ajaxstub.js` somewhere in your directory tree and your done.

4 Features

4.1 Python API

4.1.1 Overview of classes used in the AjaxStub Python API

The Python API consists of four classes.

AjaxStubException Catch this exception when using AjaxStub.

StubContainer This class subclasses the Python dict object and is used as a container object for the stub objects.

AjaxStubDispatch This is the base class for AjaxStub and does all the dispatching of data to your web page.

AjaxStub This is the class where almost everything happens.

4.1.2 API Description

4.1.2.1 AjaxStubException The constructor of this class accepts one argument, a message. The default message is `"Error: Default error message."`. There is only one method `getMessage()` used to return the message. The instantiated object can also be used as a string to returning the message.

`__init__(self, msg=__DEFAULT_MESSAGE)` The exception constructor the `msg` argument is passed the error message or uses the default message.

`getMessage(self)` Returns the exception message.

```
try:
  some code here
except AjaxStubException, e:
  print e.getMessage()
```

4.1.2.2 StubContainer This class subclasses the Python `dict` object and is used in exactly the same way. A few additional methods are provided. This class is used internally and is not needed by an implementation using AjaxStub.

`__init__(self, klass=None)` If used with a class pass the instance of the class to the constructor otherwise do not pass anything.

`getClassInstance(self)` Returns the class instance or `None` if your public API is not in a class.

`getMethodObject(self, methodName)` Returns the method object of the named stub.

`getCGIMethod(self, methodName)` Returns the CGI method used for the named stub.

4.1.2.3 AjaxStubDispatch This base class of AjaxStub sets the CGI content type which is used to determine the encode method (JSON, XML, HTML). It also encodes your stub results into one of these three encode types and dispatches everything to your page. This class is used internally and is not needed by an implementation using AjaxStub.

Static Member Objects All these objects are accessible through the AjaxStub interface.

- `AjaxStubDispatch.HTML` is used to set the communication from the server to the client using ContentType text/html. Though this is the default it is not recommended to use this as it has not error checking.
- `AjaxStubDispatch.JSON` is used to set the communication from the server to the client using ContentType text/x-json.
- `AjaxStubDispatch.XML` is used to set the communication from the server to the client using ContentType text/xml.

`__init__(self, cType=HTML)` Passing any of the three static members to the constructor will enable the content type defined by that member static object. If custom content types have been added with the `addContentType()` class method the key name for the custom type can be passes to this argument.

`_getContentType(self)` Returns the content type for this submission.

`addContentType(self, typeMap)` This is a class method that can be called before subclass AjaxStub is instantiated to add custom content types. The `typeMap` argument is a map of custom content types of which he key can be passed to the constructor.

```
AjaxStub.addContentType({'APP': 'application/x-JavaScript'})
as = AjaxStub('APP')
```

Remember you are instantiation the subclass not AjaxStubDispatch.

`initRequest(self)` Sends the content type in response to the request of the client.

`processClientRequest(self)` Sends the results to the client.

`dispatchJavaScript(self)` Sends the generated JavaScript to the client when the page is requested.

`_encodeHTML(self, data)` Encodes the resultant data into the text/html content type. This method can be overridden.

`_encodeJSON(self, data)` Encodes the resultant date into the text/x-json content type. This method can be overridden.

`_encodeXML(self, data)` Encodes the resultant date into the text/xml content type. This method can be overridden.

4.1.2.4 AjaxStub

Static Member Objects

- AjaxStub.HTML is passed to the type keyword in the constructor of the AjaxStub class. It is the default and provides limited functionality for your application.

- AjaxStub.JSON is passed to the type keyword in the constructor of the AjaxStub class. It enables JSON communication packets.
- AjaxStub.XML is passed to the type keyword in the constructor of the AjaxStub class. It enables XML communication packets.
- AjaxStub.COMMAND represents the CGI name used to pass the stub name back to the server and can be used with the `getCGIValue` and `hasCGIKey` methods.
- AjaxStub.ARGUMENT represents the CGI name used to pass a stub variable back to the server and can be used with the `getCGIValue` and `hasCGIKey` methods.
- AjaxStub.GET represents the GET CGI method and can be used with the `setCGIMethod` method or with the register method in he second half of the tuple (method object, `AjaxStub.GET`).
- AjaxStub.POST represents the POST CGI method and can be used with the `setCGIMethod` method or with the register method in he second half of the tuple (method object, `AjaxStub.POST`).

`__init__(self, type=AjaxStubDispatch.HTML, methodClassInstance=None)`
The constructor takes two keyword arguments `type` defaults to the `AjaxStubDispatch.HTML` member object and `methodClassInstance` defaults to `None`. The options that can be passed to the `type` argument are `AjaxStub.JSON`, `AjaxStub.XML`, or `AjaxStub.HTML`. The `methodClassInstance` argument is used to pass in the instantiated object of your public API (stubs) class. *Note: Methods and functions representing your public API can be used concurrently or individually. Methods do not need to be registered individually.*

```
as = AjaxStub(AjaxStub.XML, MyPublicAPI())
```

`getCGIValue(self, key)` Returns the raw value from the CGI parameter list where key is either `AjaxStub.COMMAND` or `AjaxStub.ARGUMENT`.

```
value = as.getCGIValue(AjaxStub.COMMAND)
```

`hasCGIKey(self, key)` Returns `True` if the CGI parameter name exists else returns `False`.

```
if as.hasCGIKey(AjaxStub.COMMAND):
    ...
```

`register(self, *args)` Registers a comma separated list of function objects all using a default CGI method or a comma separated list of tuples of function objects and a CGI method. *Note: This method can optionally be used to add additional functions to your public API. Class methods are automatically registered when passed into the AjaxStub constructor.*

```
def username(name):
    ...

def password(name, passwd):
    ...

as.register(username, (password, AjaxStub.POST))
```

In the example above username will use the default CGI mode and password will use the POST method.

unregister(self, name) Unregister any function added with the register method. *Note: Class methods can not be unregistered.*

```
as.unregister("password")
```

getMethodMaps(self) Returns a tuple of two internal dictionaries the map of the class API and a map of the registered functions.

```
methodMap, functionMap = as.getMethodMaps()
```

setCGIMethod(self, method) Set the default CGI method for your Public API. If no default is set AjaxStub defaults to the GET method. Only two methods are available at this time GET and POST.

```
as.setCGIMethod(AjaxStub.POST)
```

getCGIMethod(self) Gets the default CGI method.

```
cgiMethod = as.getCGIMethod()
```

getStubMethodNameList(self) Gets a list of all the function and method names.

```
stubList = as.getStubMethodNameList()
```

_exportAjaxStubArgsJS(self) Return the two JavaScript variables that need to be set prior to instantiating the JavaScript AjaxStub class.

_exportStubsJS(self) Create the stubs and return them as a string.

_handleClientRequest(self) Handle the request from the client and execute the method/function requested from the client.

includeTraceback(self) Enables the including of the traceback in the error result.

4.2 JavaScript API

4.2.1 Overview of the objects used in the AjaxStub JavaScript API

The JavaScript API consists of one function and class it also sets up the `onunload()` JavaScript call.

`window.onunload()` This JavaScript method is called to delete the `AjaxStub` object before the page unloads. If you need to override this call delete the `AjaxStub` object.

`bind(object)` This function binds objects to another object during runtime.

`AjaxStub` This class is where all the AJAX magic takes place.

4.2.2 API Description

4.2.2.1 bind The bind function is needed when the `__handleResponse` callback object needs the `this` object of its class. Since classes in JavaScript do not act like real classes they loose there relation to the class they are used in when invoked as a callback.

4.2.2.2 AjaxStub The `AjaxStub` class is written using closures and auto instantiates when `ajaxstub.js` file is fully loaded. The instantiated object is always `AjaxStub` which is also used to call the constructor of the class–yes after it was instantiated. Douglas Crockford http://www.crockford.com/javascript/private.html is credited for developing this methodology.

Constructor The constructor takes three arguments; `url` is set by the developers application; `contentType` and `funcList` are set by the Python AjaxStub API.

```
var contentType = "text/x-json";
var funcList = ['exception', 'logging', 'username',
                'getExamples', 'login', 'submit'];

window.onload = function() {
  AjaxStub("http://yoururl.com/yourscript.py",
           contentType, funcList);
}
```

In the example above `url` is coded by the developer and the last two arguments `contentType` and `funcList` are pushed to the web page by the Python AjaxStub API.

`log(value)` Will log messages through a callback function named `loggingCB` if you have it defined. You will also need to define a variable named `debug` and set it to `true`. *Note: Do not use `log` if you have not defined `loggingCB`.*

```
var debug = true;

var loggingCB = function() {
  var logging = document.getElementById("logging");
  logging.innerHTML += result + "<br />";
  logging.scrollTop = 1000000;
}

var someFunction = function() {
  ...
  AjaxStub.log('Results of some process.');
  ...
}
```

In the above example first set a variable debug to true then write a callback that does something with your log message so when you use log in your code the message has someplace to go.

`register(list, obj)` This can be used to register stubs, but the stubs must have already been defined by the Python API. The `list` argument contains the stubs you wish to register and `obj` is the object they are in ie. class. The `register` function defaults to the `window` scope to look for callbacks, so the `obj` argument is rarely needed.

It might seem confusing as to why, if you registered your stubs in the Python API, you would need to do it again. Well you don't unless you push JavaScript code up to your page which you may want to delete later. The `register` method is called in the constructor and through introspection finds the stubs and the callbacks. When the class first gets instantiated it will not find the callbacks which you have just pushed up to your page. You must write and pre-register the stubs, however; because the Python API will at the time the page is requested create and put them in the page. I suppose if you wanted to write the stubs yourself and push them up also this may work, but I have never tried this approach myself.

```
var list = ['foo', 'bar'];
AjaxStub.register(list);
```

`unregister(list)` This is used to unregister stubs or stub callback.

```
var list = ['foo', 'bar'];
AjaxStub.unregister(list);
```

`getStubFunction(name)` Returns the object of a registered stub or stub callback.

```
var funcObj = AjaxStub.getStubFunction('foo');
```

`executeStub(name, cgiMethod, args)` This method is generally used in the stubs pushed up to your page by the Python API. It can be used if you push

up your own stubs in an AJAX request to the server. The `name` argument is the function or method name in your Python code, `cgiMethod` is either '`GET`' or '`POST`', and `args` is the argument list of the JavaScript stub calling this method.

```
// Stub for function: foo()
var foo = function() {
  AjaxStub.executeStub('foo', 'GET', foo.arguments);
}
```

The code above is generally auto generated by the Python AjaxStub API.

`getNodeText(doc)` This is a convenience method that gets the text in an XML tag. It will loop through all the nodes in a tag looking for text and cdata nodes and concatenating them together forming the full text of the tag. The `doc` argument is the tag object that you want the text from.