

RESTful Interfaces

RESTful interfaces are all the rage among Ruby on Rails and ASP.NET developers building Service Oriented Architectures (SOA) and mashups. In fact, it is just as easy -- or perhaps easier -- to write a REST client using Alpha Five V10 as it is using those "fashionable" Web frameworks.

First of all, let's define our terms. REST stands for [Representational State Transfer](#), and was introduced in [Roy Fielding's Ph.D. thesis](#) in 2000. Reduced to practical terms, RESTful interfaces are accessed using HTTP calls with various verbs: GET, POST, PUT, DELETE, and so on. These verbs operate on the object specified in a [Uniform Resource Identifier](#) (URI).

Suppose we had a digital photo archive with a RESTful interface at <http://digitalphotoarchive.com> and we wanted to retrieve the photo "alphadog.jpg" from album #3 of the alpha account. In this made-up example, we might construct a URI for this resource as <http://digitalphotoarchive.com/alpha/3/alphadog.jpg>. Then we would issue an HTTP GET request to that URI, which would respond with the image as a bit stream with the image/jpeg MIME type. Since HTTP GET is the default action of a Web browser, and most modern Web browsers understand the image/jpeg MIME type, this particular action could be accomplished very simply by pointing a Web browser at the URI.

In a RESTful interface, HTTP POST requests are typically used to add items to the data store. These can also be done by Web browsers, although some HTML needs to be written so that the browser knows what to do. This is normally accomplished by creating an HTML form with an action of POST, pointing the browser at the form, and pressing the Submit button.

What if you wanted to accomplish this programmatically with Alpha Five V10? Alpha supports generic HTTP calls with the `HTTP_FETCH()` function. This is a very flexible function that supports all HTTP verbs, but it is a little complicated to use, because you have to populate several properties of a dot (pointer) variable before calling the function.

It's easier to use the `HTTP_GET()`, `HTTP_POST()`, `HTTP_PUT()`, and `HTTP_DELETE()` functions. These functions are really just wrappers for `HTTP_FETCH()`, but they're more convenient to call. For example, the `HTTP_POST()` function has the following prototype:

```
Result as P = HTTP_POST( URL as C [, Body as C [,
Cookie as C [, Port as N [, Timeout as N [, Show_
Before_Send as L [, Validate_SSL_Certificate as L
]]]] ] )
```

A real-world example that needs a POST is the [IfByPhone API](#). One of the phone mashup services that IfByPhone provides is called a SurVo, or voice survey, and the service is abbreviated CTS, for Click to Survey. Basically, it's a



AlphaSoftware

At Alpha Software our mission is to build cutting edge database software that enables our customers to build breakthrough applications and solve database management problems rapidly and easily.

Alpha Software, Inc.

70 Blanchard Road

Burlington, MA 01803-5100

Phone: 781.229.4500

Fax: 781.272.4876

www.alphasoftware.com

structured interactive voice response call. Once you have created the survey online, you can invoke it with an HTTP_POST() call. So, for example, we could prepare for and make such a call as follows:

```
dim url as C = "https://secure.ifbyphone.com/click_to_xyz.php"
dim req as C = "app=cts&acct=123&survo_id=456&key=78abcdef01"
req = req + "&phone_to_call=" + numberToCall
dim userparms as C = "name|" + name
req = req + "&user_parameters=" + urlencode(userparms)
dim resp as P = http_post( url, req, "", 443, 0, .F., .F.)
IF resp.error_code > 0
retval = retval + "Error " + resp.error_code + " broadcasting message "
    IF prop_valid(resp, "error_text") THEN
        retval = retval + resp.error_text
    END IF
END IF
retval = retval + "Returned: " + resp.body
```

Note that we are actually using the secure HTTPS protocol over port 443, rather than the unsecured HTTP protocol over port 80. This gives us SSL encryption, which protects our account key from snooping.

You can package all of that code in an Xbasic function that accepts the numberToCall and name parameters as character variables.

It gets better, though. You can invoke that Xbasic function from a grid. One new feature of Alpha Five V10 Ajax grids is that you can tie an action to a button in a grid row through an Ajax callback.

So, for instance, you can add a button control field to a grid ([figure 1](#))

Then you can edit the JavaScript onClick handler for the button ([figure 2](#))

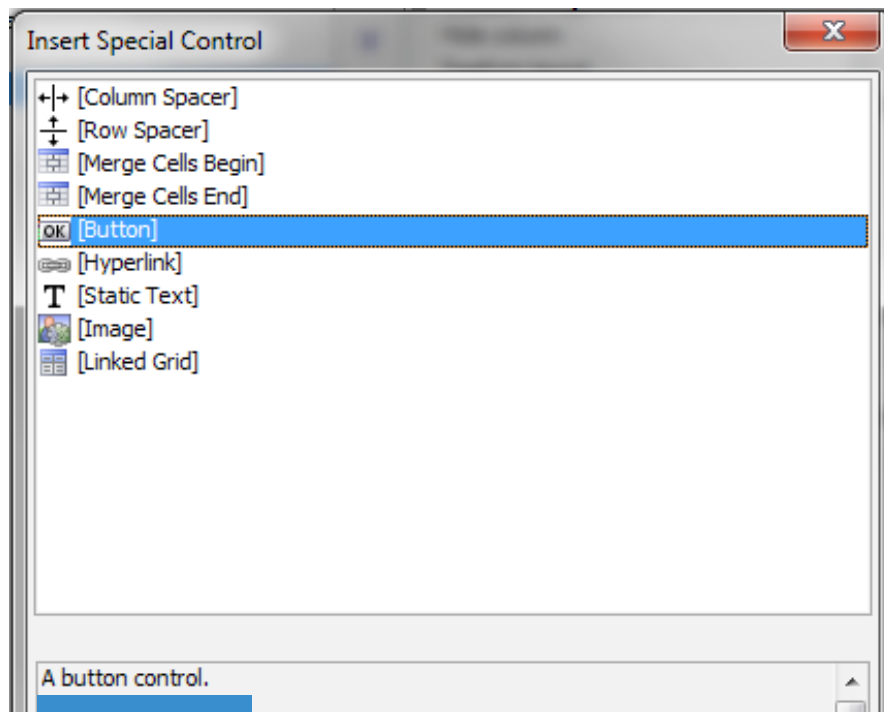


Figure 1

Javascript	
onClick	<Click button to edit>
onDbClick	<Click button to edit>
onMouseDown	<Click button to edit>

Figure 2

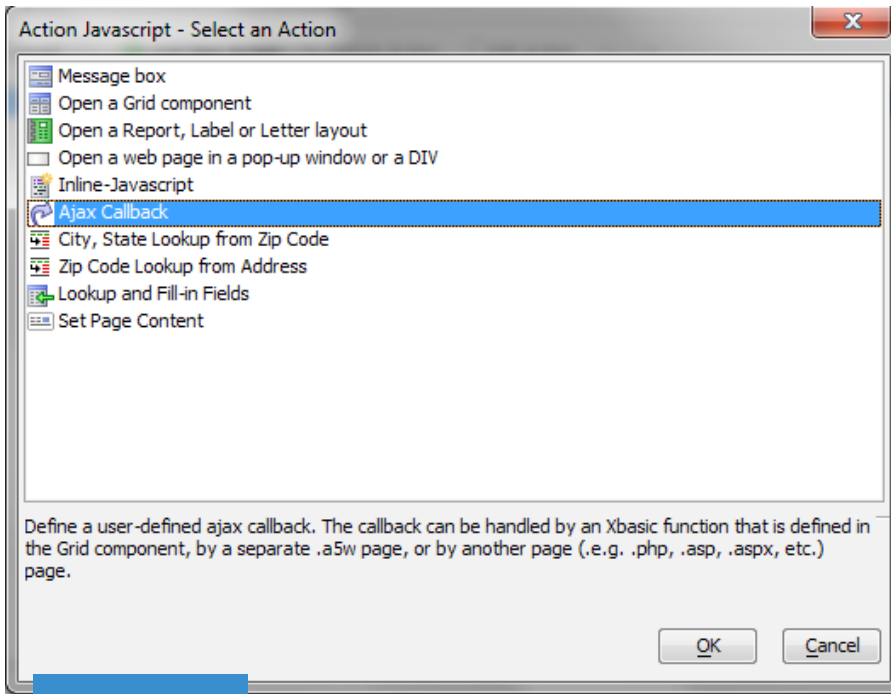


Figure 3

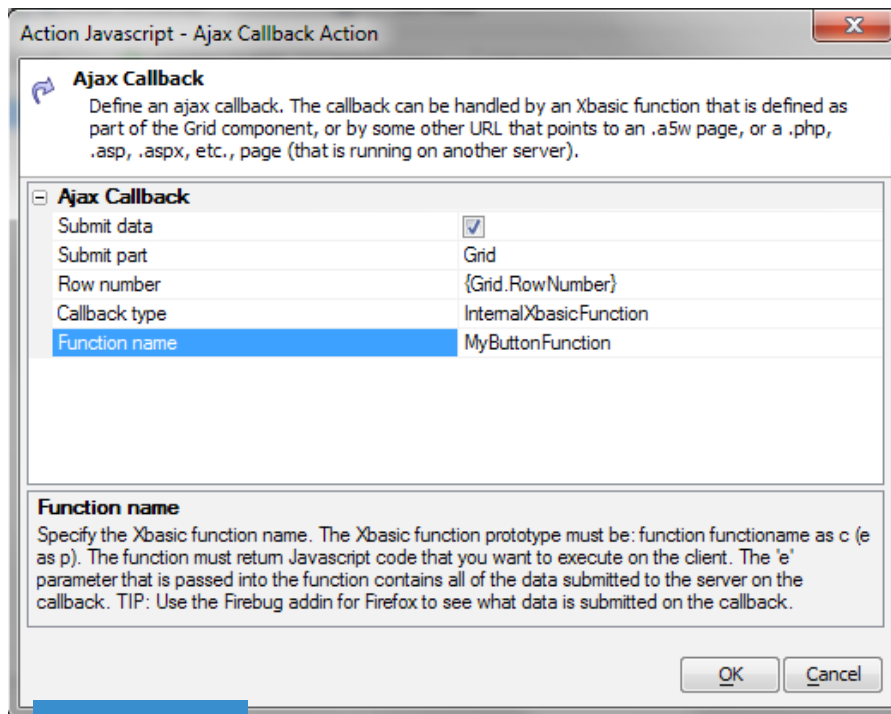


Figure 4

Set the action to an Ajax Callback (figure 3), and set the callback to the name of your function (figure 4)

Note the description of the function prototype at the bottom of this screen. You'll have to write a few lines of code to extract the row data you want from the e variable, but it isn't at all difficult if you follow the hint and use the Firebug addin to Firefox to view the submitted data. Typical code might look something like this:

```
dim row as C = e._
currentRow
dim number as C =
eval("e.V.R" + row +
".numberToCall")
```

```
dim name as C =
eval("e.V.R" + row +
".name")
```

And that's really all there is to it, unless you want to display the result on your Web page. Then you can return a JavaScript expression for the grid to evaluate, something like this:

```
MyButtonFunction =
"$('progress').innerHTML +=
'" + retval + "<br>";"
```

Here we are adding the result of the call to the DIV named "progress" on the calling page. That might be defined in one of the grid's freeform edit regions, or elsewhere on the A5W page that hosts the grid.

Enjoy!