AJAX and It's Application On Webspeed/Progress 4GL

Scott Auge

sauge@amduus.com

http://www.amduus.com

Table of Contents

Introduction

This document should be read by programmers, architects, and analysts.

Ajax is playing a greater and greater role in the development of web applications. A lot of really neat things can be accomplished with AJAX in both user enjoyment of the application and modularization of coding on the system.

This document does not define AJAX nor does it train on the basic understanding of AJAX. There are plenty of other writings that can do that.

This document focuses specifically on using AJAX with Progress Webspeed/AppServer. Some theory is given here and there – but mostly it is a discussion of integrating AJAX applications with Progress data.

Often times, kits that include DHTML goodies like fancy tables and sliders, etc. are confused with AJAX per se. This document focuses purely on AJAX – getting information from the browser to the server and from the server to the browser. How you choose to display it after receiving it is dependent on you.

This document takes more of a tutorial look at things instead of creating an all encompassing theory with every little detail noted. The idea is to introduce the reader to the ideas and less with confusing them with nick nacks – stumbling around in a mental store of items all trying to attract attention.

This code has been found to work on Apple OS X FireFox, Apple OS X Safari, Windows XP FireFox, and Windows XP Internet Explorer 6.0+ and Progress AppServer 10.1 on Linux though Webspeed 9.1x can certainly be used also.

Definition Of What We Are Going To Do

In this paper, there will be a discussion of a web page capable of making at least three AJAX calls depending on what buttons and selections the user is making. This web page is for the updating of users who can use a web application – hence don't get confused with the word “user” meaning a login for the application and “user” meaning the operator of the web application via a browser.

We are not going to talk about the application per se – but at some functionality for the application. The focus is on the AJAX – not the application.

Organizing Your Javascript Routines

When you begin writing your AJAX based web page, you are going to find you are using a lot of javascript. You are going to find it helpful to break up your routines into specific purposes. A naming convention for them can help sort out what does what inside the browser and what does what outside of the browser.

Naming Routines That Create AJAX Calls

Underneath buttons and other widgets of your browser's web page, you will spawn AJAX calls usually by an onclick(), onChange(), etc. event processor.

Needless to say, just about anything that might be involved in generating an AJAX call should be prefixed with ajax_ on it. For example, ajax_Listing_User(), ajax_Update_User(), etc.

Often you can tie all these up into .js libraries available to a web page/web application that can be called. With proper abstraction – one can make them all available under all pages for simplicity or as libraries of related AJAX calls.

Naming Routines That Handle AJAX Call Responses

When one makes an AJAX async call, a special routine must be "attached" to the AJAX XML Requester object to process the call in the background while the scripting of the application goes on it's merry way.

I often call these ajax_handle_ ... for example, ajax_handle_Create_User() or ajax_handle_List_User(). This way it is more obvious which "handler" routine is

associated with which ajax_ routine.

These routines tend to be closer to the web page's operation. Some level of abstraction can be had by having intermediary routines called in the handle routine and then defining those calls in the web page it's self for DOM manipulation.

Naming Routines To Digest AJAX Data

It is NOT at all obvious how to parse an XML result from an ajax call. There is a horrid syntax of objects to traverse to deal with an XML result.

I have a couple of routines in the "useful routines" area later in this document – you may want to rename them to be specific to "getting" data from XML and not from forms, frames, etc. They will work for most well-formed XML but do not pull out attributes associated with a tag[1].

1 I can't be the only one wrapping code around this nasty syntax – if you have a simple to use wrapper please feel free to send em along!

Creating Asynchronized AJAX Calls

These sorts of calls are best when the "thread" of execution has nothing dependent on them. Lets take a look at a call that is used to pull in information about a user. Say the operator has clicked on an item in a selection list of people who can use the application. This function is called by the selection box processing an `onclick="ajax_LookupUser(this)"` event processor.

```
//------------------------------------------------------------------------
// Used to find a user by AJAX
//------------------------------------------------------------------------

function ajax_LookupUser (objselect) {

  LoginID = objselect.value;

  // Catch when the Add User was selected in the list box.  This is
  // actually the creation of a new user so we set up the variables
  // in the browser to tell ajax it's a new user on save.  Note on the
  // Update on the Progress side we recognize a new record to be
  // constructed because the rowid is "new"

  if (LoginID == "new") {

    SetRecordMemory("New");
    SetIdentity ("", "", "");
    SetManagement ("", "", "", "", "");

    // Normally we don't allow updating of the login because it is a
    // key through-out the database.  On new entries though, we need it!

    document.form1.login.readOnly = false;

    return;

  }

  // On a lookup of a user, we want to make sure they can't update the
  // login field

  document.form1.login.readOnly = true;

  // Simply use AJAX to call out to the server to pull back goodies.
  // This is a simple HTTP GET sent to Webspeed

  xmlHttp=GetXmlHttpObject()

  if (xmlHttp==null)
  {
    alert ("Browser does not support HTTP Request")
    return
  }
```

```
  var url="ajax_lookup_user.html";
      url=url+"?UserLoginID="+LoginID;

  //alert (url);

  xmlHttp.onreadystatechange=ajax_handle_lookup_response
  xmlHttp.open("GET",url,true)
  xmlHttp.send(null)

}
```

Note this code is very tight with the DOM of the page it is used within. It is better to abstract around such things.

If the login is not new – then it will create an AJAX call out to Webspeed to pull back the information associated with that login. Lets see what the webspeed program under ajax_lookup_user.html is doing:

```
<!--WSS
PROCEDURE OUTPUT-HEADERS:
  output-content-type("text/xml").
END.

DEFINE VARIABLE cPassword AS CHARACTER NO-UNDO.
DEFINE VARIABLE cUserLoginID AS CHARACTER NO-UNDO.
DEFINE VARIABLE hPassword AS HANDLE NO-UNDO.

ASSIGN cUserLoginID = GET-VALUE ("UserLoginID").

FIND Users NO-LOCK
WHERE Users.UserLoginID = cUserLoginID.

RUN api_password.p PERSISTENT SET hPassword.

RUN GetPassword IN hPassword (INPUT cUserLoginID, OUTPUT cPassword).

DELETE OBJECT hPassword.

--><?xml version="1.0" ?>
<User>
  <Login>`Users.UserLoginID`</Login>
  <Name>`Users.UserName`</Name>
  <EMail>`Users.UserEmail`</EMail>
  <Active>`IF Users.ActiveUser THEN "Yes" ELSE "No"`</Active>
  <LoginExpiration>`Users.UserExpirationDate`</LoginExpiration>
  <LastLoginDate>`Users.LastLoginDate`</LastLoginDate>
  <LastLoginTime>`STRING(Users.LastLoginTime, "HH:MM:SS AM")`</LastLoginTime>
  <Password>`cPassword`</Password>
  <RowID>`STRING(ROWID(Users))`</RowID>
</User>
```

Now, since this is actually happening in the background and the user of the browser can continue their merry way while waiting for the data to return, there is a request call handler that is called everytime the XMLRequest object changes

state[2].

```
//---------------------------------------------------------------------
// We use this for the lookup of a user's information by  AJAX call
//---------------------------------------------------------------------

function ajax_handle_lookup_response () {

  // You MUST have these checks because there are 5 states that can be checked
  // and on each state change this routine is called.  You want to have
  // something to check these states.

  if (xmlHttp.readyState == 4) {
    if (xmlHttp.status == 200) {

      var xmlDoc = xmlHttp.responseXML;

      var Login = GetXMLObjValue(xmlDoc, 'Login', 0);
      var Name = GetXMLObjValue(xmlDoc, 'Name', 0);
      var Email = GetXMLObjValue(xmlDoc, 'EMail', 0);

      SetIdentity (Name, Login, Email);

      var Active = GetXMLObjValue(xmlDoc, 'Active', 0);
      var LoginExpires = GetXMLObjValue(xmlDoc, 'LoginExpiration', 0);
      var LastLoginDate = GetXMLObjValue(xmlDoc, 'LastLoginDate', 0);
      var LastLoginTime = GetXMLObjValue(xmlDoc, 'LastLoginTime', 0);
      var Password = GetXMLObjValue(xmlDoc, 'Password', 0);

      //alert ("Active = " + Active);

      SetManagement (Active, LoginExpires, LastLoginDate, LastLoginTime,
Password);

      // We use this in a hidden field in the form so that when we update
      // we know which record we are updating.

      var RowID = GetXMLObjValue(xmlDoc, 'RowID', 0);

      SetRecordMemory(RowID);

    }
  }

}
```

Basically the above code says that when everything is done and said for[3] – it uses some routines to chop up the XML result into variables. It then places the values of those variables into calls for routines that will populate various page fields – that is the SetIdentity(), SetManagement(), and SetRecordMemory() routines.

---

2 This is important to know and yet is outside the scope of this document. I suggest you find out about the four states an XMLRequest object can be in as well as handling status results in case something goes dreadfully wrong. It is a good thing to know.

3 The above code does NOT handle errors very smoothly. You need to check the other states to determine if an error has occurred.

A quick look at these routines will show they merely populate form fields:

```
//---------------------------------------------------------------------
// Set the information in the identity part of the page.
//---------------------------------------------------------------------

function SetIdentity (Name, Login, Email) {

   document.form1.name.value = Name;
   document.form1.login.value = Login;
   document.form1.email.value = Email;

}
```

As noted above, if you make libraries of AJAX routines, you can “tie” them into a specific page's DOM structure by having a set of routines like SetIdentity() between the call data and the DOM names. This way you don't find yourself re-constructing AJAX call functions over and over.

Creating Synchronized AJAX Calls

These sorts of calls are best when there is another action dependent on the AJAX call. For example, we have a call that gets a list of users called ajax_List_Users() and another call named ajax_Update_User().

When we call ajax_Update_User() we want to also re-populate the user list box on the web page when a creation or name update is done.

If we do this asynchronously – we may actually get the listing back BEFORE the call for creating the user was even finished! This is because AJAX would be making two web hits behind the scenes totally out of step with each other. This can make for some screwy and confusing web pages.

So we need to make the ajax_Update_User() a synchronized call – that is the browser makes the XML call out and waits for a response – and THEN we go and make the ajax_List_Users() call (which has no dependencies so it can be a-synchronized.)

Here for example is some code that does this. It is called by a button with `onclick="ajax_UpdateUser()"`.

```
//----------------------------------------------------------------------
// Used to update the user by AJAX
//----------------------------------------------------------------------

function ajax_UpdateUser() {

  if (document.form1.Password1.value != document.form1.Password2.value) {
    alert ("Passwords do not match!");
    return;
  }

  // Validations Checked - now we send out the request to the AJAX handler

  xmlHttp=GetXmlHttpObject()
```

```
  if (xmlHttp==null)
  {
    alert ("Browser does not support HTTP Request")
    return
  }

  NVPs = "Password=" + escape(document.form1.Password1.value)
       + "&Active=" + (document.form1.Active.checked ? "YES" : "NO")
       + "&RowID=" + escape(document.form1.RowID.value)
       + "&Email=" + escape(document.form1.email.value)
       + "&Name=" + escape(document.form1.name.value)
       + "&LoginExpiration=" + escape(document.form1.LoginExpire.value)
       + "&UserLoginID=" + escape(document.form1.login.value);

  // alert (NVPs);

  // Needs to be synchronized because we do a List Users.  It is possible to
  // List Users before a new user is even made so we wait for it to be done.
  // Because we wait in synchronized - the onreadystatechange handler
  // is never called.

  xmlHttp.open("POST","ajax_update_user.html",false)  // Can't async cuz might
list before user made
  xmlHttp.setRequestHeader('Content-Type','application/x-www-form-urlencoded');

  xmlHttp.send(NVPs)

  var xmlDoc = xmlHttp.responseXML;
  var ErrorResult = GetXMLObjValue(xmlDoc, 'ErrorResult', 0);

  if (ErrorResult != "") alert (ErrorResult);

  // If we created a new user, we need to relist the userlist
  // box

  if (document.form1.RowID.value == "New")  ajax_ListUsers();

}
```

As can be seen in the above code, the ajax_Update_User() opens with the async flag set to false (the `xmlHttp.open(... , false)` bit) . This tells the browser to wait for an answer before going on with execution of the script.

Once it is done – it will check a value in the page on the browser to determine if it was updating an existing record or a new one. If it is a "New" record, it will call out to ajax_ListUsers() which does some work in another AJAX call to populate another part of the screen. Note you don't need a browser event to initiate an AJAX call – you call them when you need them.

Gotcha: When you make a call synchronized the onreadystatechange handler is NEVER called. This is because one never needs to check for state changes for the

AJAX call. If you put business logic into this routine – it will never get executed[4].

Gotcha Too: Note the above routine is heavily dependent on the DOM of the page it is called from. A better way would be to put some arguments to it for a layer of abstraction for use on multiple pages.

4 Yes, I spent some time scratching my head wondering what happened when I could see action occurring on the Webspeed side but none of my statements were running on the browser side!

Returning HTML under AJAX

Sometimes it is simply easier to return some generated HTML and stuff it into a web page.

An example is a section of the web page detailing what reports a user has access too. This actually an HTML table with some nice formatting and the like. It would be mondo javascript (and a PIA to maintain) to regenerate this information for different users if it was executed for every user information was requested about.

It all starts with the page that needs the HTML inserted handling the event of choosing a user:

```
<select name="UserList" size="25" class="Data" style="width: 300px"
onclick="ajax_ListUserModules(this.value)">
```

We can see it calls out into an ajax_ routine which reads as follows:

```
function ajax_ListUserModules (UserLoginID) {

  xmlHttp=GetXmlHttpObject()

  if (xmlHttp==null)
  {
    alert ("Browser does not support HTTP Request")
    return
  }

  var url="ajax_list_user2module.html"
             + "?UserLoginID=" + escape(UserLoginID);

  xmlHttp.open("GET",url,false)
  xmlHttp.send(null)

  var TheHTML = xmlHttp.responseText;

  document.getElementById("ModuleManagement").innerHTML = TheHTML;

}
```

This routine calls out to a Webspeed program called ajax_list_user2module.html that will generate some HTML.

Lets take a look at the code that generates the HTML that will be inserted into the page above. (It might be interesting to note it also includes calls to ajax routines it's self!)

```
<!--WSS
PROCEDURE OUTPUT-HEADERS:
  output-content-type("text/html").
END.

/*********************************************************/
/* Help code readability for check boxes.                */
/*********************************************************/

FUNCTION IsChecked RETURNS CHARACTER (INPUT L AS LOGICAL):

  IF L THEN RETURN "CHECKED".
  ELSE RETURN "".

END. /* FUNCTION IsChecked */

DEFINE VARIABLE cUserLoginID AS CHARACTER NO-UNDO.

ASSIGN cUserLoginID = GET-VALUE("UserLoginID").

/*********************************************************/
/* For this routine, we simply use the responseText value */
/* for the HTTPXML objects                               */
/*********************************************************/
-->

        <table width="90%" border="0" align="center">
          <tr>
            <td colspan="2" bgcolor="#CCCCCC" class="Field_Label"><div
align="center">Module Management </div></td>
          </tr>

          <!--WSS
          FOR EACH Modules NO-LOCK:

          FIND UserModule NO-LOCK
          WHERE UserModule.UserLoginID = cUserLoginID
            AND UserModule.ModuleID = Modules.ModuleID
            NO-ERROR.
          -->
          <tr>
            <td valign="top">
              <div align="right">
                <input type="checkbox" name="Module_`Modules.ModuleID`"
value="YES" onclick="ajax_set_User2Module(`Modules.ModuleID`, this.checked)"
`IsChecked(AVAILABLE UserModule)`>
              </div></td>
            <td valign="top"><div
class="Section_Header">`Modules.ModuleName`</div>
              <blockquote>
                <p class="Field_Label"><em>`Modules.ModuleDescription` </em><br>
                </p>
                <hr>
```

```
        </blockquote></td>
      </tr>
      <!--WSS
      END. /* FOR EACH Modules */
      -->
    </table>
```

We will end up getting this response in a responseText attribute for the XMLRequest object. How to get the HTML from there into the web page? We create a named `<DIV>` marker where the HTML will be inserted:

```
 <td valign="top">
                <div id="ModuleManagement">

                     </div>
</td></tr>
```

If you look in the ajax routine running on the browser, you will see this DIV can be simply populated with:

```
var TheHTML = xmlHttp.responseText;

document.getElementById("ModuleManagement").innerHTML = TheHTML;
```

What About Converting Existing Webspeed Pages

The underlying principle of AJAX - pulling back only what information is needed and performing only what specific action is requested by the user – is so foreign to commonly developed web pages that conversion is almost guaranteed to be a rewrite.

For example, one application had a set of pages for updating users of a web application. One page was a listing of users, another was for updating an existing user – and perhaps yet another for creating a user.

Under AJAX, all of these functions can often be done within the same page with another that acts as an AJAX handler.

Now, it is a toss-up to which involves faster development time! I have found that having an organized set of AJAX routines as well as proper abstraction around them can get the ball rolling pretty quickly – BUT one has to make the investment in time to make the underlying pieces.

You will want to think out your webspeed AJAX handling routines. I would recommend something like:

```
ajax_yourlibrary.html?action=[add,delete,list]& ...
```

so the same webspeed component can be hit for like data. This is better than a ajax_adduser.html, ajax_deluser.html, and ajax_updateuser.html.

Benefits of Using AJAX under Webspeed

Another benefit of AJAX based pages, is that they can interact with Webspeed and yet be fully under control of a Web Site Designer (to a degree – just make sure they don't mess up the javascript.)

One of the problems with Webspeed has been the complete generation of web pages by coding. With AJAX, one can generate portions of the page as well as populate data widgets on the fly. It is a nice solution to the problem HTML-Mapped applications attempted to accomplish.

Of course, one can still mix and mash AJAX pages – some pages might be available only in a generated form (the page merely needs to be "recompiled" into a webspeed program after the designer is done with it) or in a static form allowing the user community better ability to modify it.

One will notice that the Webspeed side of the programming becomes extremely simple compared to when it is generating pages based on E4GL statements. All one needs to do is agree on what form the XML will take.

Warnings About AJAX

Since all sorts of things are happening on the page based on button pushes, check box checking, and selection selecting – be prepared to see your server get HAMMERED ON with many – many AJAX requests.

Often they are small – as they should be – but as exampled with the creation of a user above – this required an AJAX hit to create the data and then domino ed into another AJAX hit to refresh the web page's selection box with that new user. What would have been one page hit on a “normal” web application turned into two hits (granted the one “normal” hit would be a huge amount of computation compared to the two little hits.)

You are also going to be concerned about the version of web browser being used. You will want:

- Version 6.0 or better of IE and
- Version 1.5 or better of FireFox/Mozilla.
- Version 2.0 or better of Apple's Safari browser

Ajax also does a lot of work storing data into memory of the browser. An AJAX based application should be run on a machine not more than three or so years old. Remember – a good part of the application is taking place on the client PC (again) so take that into account.

Do I Still Need Session Control

In short, yes. Interaction between Webspeed and an AJAX based web page still requires some kind of authentication system to access the data and to remember who is connected under what page.

Since the interaction via AJAX back to Webspeed in this paper has been based on an HTTP protocol, the deficiencies of the protocol still remain. If you check the example code, you will see we still GET or POST to an html page using Name/Value Pairs (NVPs).

Basic Routines You May Find Useful

Routines To Manipulate Selection Boxes

I have found a need for these so often, I simply made a set of routines to do so:

```
//-----------------------------------------------------------------------
// Functions to easily manipulate selection boxes.  Doesn't work well on
// selections with the same values though!
//-----------------------------------------------------------------------

function InsertIntoSelect (SelectObj, Display, Value) {

  NumberOfElements = SelectObj.length;
  NewOption = new Option (Display, Value, false, true);
  SelectObj[NumberOfElements] = NewOption;

}

function RemoveFromSelect (SelectObj, Value) {

  Index = FindInSelect(SelectObj, Value);
  if (Index == -1) return false;
  SelectObj[Index] = null;

}

function FindInSelect (SelectObj, Value) {

  for (i = 0; i < SelectObj.length; i++)
    if (SelectObj[i].value == Value) return i;

  return -1;

}

function EmptySelect (SelectObj) {

  i = SelectObj.length;
  for (j = 0; j < i; j++)
     SelectObj[0] = null;
}
```

Routine To Generate An XMLRequest Object

This will work on both IE, Safari, and Mozilla:

```
//---------------------------------------------------------------------
// Need this for the AJAX pulling back of goodies - make it available
// on Mozilla AND Windows
//---------------------------------------------------------------------

function GetXmlHttpObject()
{
  var objXMLHttp=null
  if (window.XMLHttpRequest)
  {
    objXMLHttp=new XMLHttpRequest()
  }
  else if (window.ActiveXObject)
  {
    objXMLHttp=new ActiveXObject("Microsoft.XMLHTTP")
  }
  return objXMLHttp
}
```

You would see these called often in the above chapters.

Routines To Parse Results Of An XMLRequest

This function can make pulling out tag values a bit more readable:

```
//---------------------------------------------------------------------
// In XML, there may not be a value associated with a given entry
// Use this to set it to "" cuz a null throws a javascript error
// This doesn't work for XML Tag attributes
//---------------------------------------------------------------------

function GetXMLObjValue (TheObject, XMLName, EntryNumber) {

  // If an XMLTag is empty or is not available, we will throw a javascript
  // error when we try to use it.  So we need to do the try-catch goodies
  // to catch the error.

  try {

    return TheObject.getElementsByTagName(XMLName)[EntryNumber].firstChild.data;

  } catch (ex) {

    // Some old versions of IE need this.  Who knows why when it is already
    // used?

    var TheError = ex;

    return "";

  }

}
```

It expects the ResponseXML object available from the XML Request object. (See some examples below on how it is called.)

If you have a listing of tags, you can call it like this:

```
//---------------------------------------------------------------------
// In XML, there may be multiple lines of the same entity listed -
// such as a user listing.  One can use this to help determine how many
// entries there are in the XML returned.
//---------------------------------------------------------------------

function GetXMLObjLength (TheObject, XMLName) {

  // If an XMLTag is empty or is not available, we will throw a javascript
  // error when we try to use it.  So we need to do the try-catch goodies
  // to catch the error.

  try {

    return TheObject.getElementsByTagName(XMLName).length;

  } catch (ex) {

    // Some old versions of IE need this.  Who knows why when it is
    // already used?

    var TheError = ex;

    return 0;

  }

}

//---------------------------------------------------------------------
// We use this for the listing of users by AJAX call  TODO
//---------------------------------------------------------------------

function ajax_handle_userlist_response () {

  if (xmlHttp.readyState == 4) {
    if (xmlHttp.status == 200) {

      var xmlDoc = xmlHttp.responseXML;

      EmptySelect (document.form1.UserList);
      InsertIntoSelect (document.form1.UserList, "Add User", "new");

      NumUsers = GetXMLObjLength(xmlDoc, "UserLoginID");

      for (i = 0; i < NumUsers; i++) {

        UserLoginID = GetXMLObjValue(xmlDoc, 'UserLoginID', i);
        UserName = GetXMLObjValue(xmlDoc, 'UserName', i);

        InsertIntoSelect (document.form1.UserList, UserLoginID + " -- " +
UserName, UserLoginID);

      }
    }
  }
}
```

One gets the number of <UserLoginID> tags in the response with

GetXMLObjLength() and then call out to each and every one as shown in the for loop.

See previous examples for more simplistic use in other XML results.

Some Variation In Handling GET and POST Via AJAX

There are slight differences in what needs to be done for calling out for a GET and a POST in AJAX.

Here is an example of a GET:

```
var url="ajax_set_user2module.html"
       + "?UserLoginID=" + escape(UserLoginID)
       + "&ModuleID=" + escape(ModuleID)
       + "&IsSet=" + escape(IsChecked ? "YES" : "NO");

// alert (url);

xmlHttp.open("GET",url,false)
xmlHttp.send(null)

var xmlDoc = xmlHttp.responseXML;
```

and here is an example of a POST:

```
NVPs = “Fname=” + escape(FirstName)
     + “&Lname=” + escape(LastName);

xmlHttp.open("POST","ajax_list_users.html",false)
xmlHttp.setRequestHeader('Content-Type','application/x-www-form-urlencoded');

xmlHttp.send(NVPs)
```

Conclusion

Coming from a pretty heavy javascript and object oriented programming background – I had some AJAX pages running in two days. Most of it was spent writing routines to wrap around the crazy syntax of the XML parsing objects.

One CAN paint one's self into a corner with AJAX programming if you are not careful in your planning. Muddling blindly through a page that I didn't really plan out had me reworking some code to become far more modular than I first expected (for example, the update requiring two AJAX calls instead of making one big sequence of code to handle a button push as well as directly accessing a page's DOM entities.)

The webspeed side of the AJAX architecture becomes quite simple. It is very obvious even in E4GL programming what is being executed on the server and what is being executed on the web page. Data transfer is simply NVPs POSTed or GETted to the server and received in an easy to parse XML document.

About Amduus Information Works, Inc.

Amduus Information Works, Inc. dabbles in many areas from software development, to job placement, to even making movies!

The main technological focus is on UNIX/Linux/Apple OSX, PHP, PostgreSQL, MySQL, and Progress technologies such as Webspeed and the RDBMS among other well known technologies. We also do a lot of integration work.

Amduus is working on SaaS applications for use by our customers. We are open to ideas! Currently we are working on SEIII which can be found off our main page.

We also help place people in companies. Reading this, I assume you are someone who is interested in learning new things and bettering your skills – just the type of people Amduus needs. Feel free to send your resume to jobs@amduus.com. You might be able to work for Amduus directly or to be placed by Amduus. See our placement pages for a sample of job opportunities.

Amduus also creates training media. If you have a piece of software you would like a movie or training documents made available for – let us know! We have found that movies describing how to use our software has been exceptionally received. It is one thing to read about it – it is another to actually see it in action.

Accessing The Code

Sometimes cutting and pasting code is difficult. If there is a demand for the code – it will likely go into the 4GL Wiki found at http://amduus.com/4glwiki – let me know if you want it though!