The Progress Electronic Magazine

In this issue:

Did you sign up to receive this E-Zine? Send email to sauge@amduus.com to subscribe or fill out the forms at http://www.amduus.com/online/dev/ezine/EZineHome.html ! It's free! (Though donations are certainly welcome – whatever you feel is fair!)

Publisher's Statement:

You may notice that there is no date on this issue. That is because, I plan on putting out issues as they come to me. I will certainly try to put them out at least once a month, but sometimes I just can't wait to spill some goodies out there into the internet!

As a side note – if you have some idea of an article – or even better yet an article, pass them along to me! Since the Zine is free, I can't pay you, but the rest of the world might be interested if you're a consultant or simply wish to share your works with others for the benefit for the whole community.

In this issue we play with a preprocessor definition that comes from the C world. Often include files will require the use of other include files. What if those include files already have been included? We explore a little way to make sure we don't get errors from including a file that already has been included!

Also, as an added bonus, some code to conveniently make Lookups on a web page.

In the management article, we examine some of the items to take into account when pricing a proposal.

To your success,

Scott Auge
Founder, Amduus Information Works, Inc.
sauge@amduus.com

Amduus Information Works, Inc.
http://www.amduus.com
scott_auge@yahoo.com sauge@amduus.com

Creation of modules and products for re-sale
as well customized Internet/Intranet programming
for E-Business in the marketing/manufacturing/
service and law enforcement industries.

Reach over 700 programmers and companies.

Your ad could be here!

Advertise in the E-Zine for $10.00 per issue!

Analysts Express, Inc.

Webspeed Training and progress programming.

Call James Arnold at
888-889-9091
or
jarnold@mylinuxisp.com

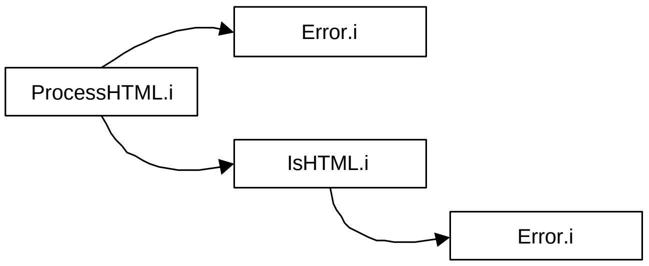
Coding Article: Protecting Multilevel includes

Written by Scott Auge

Often times in applications include files are used. These include files may contain variable definitions, internal procedure, and/or function definitions. A lot of times, one function definition may rely on another function or one internal procedure may rely on another internal procedure.

For program modularity, I tend to put each function, internal procedure, or set of variables into their own include file. When I define a function A that is dependent on another function B, I merely include the file for function B into the include that defines the function A. This way, when a programmer wants to use function A, all they need be concerned about is getting it's include file into the program using it. All the sub-functions would automatically be included for them. (This happens a lot in C/C++ programming.) No extra fluff is in the resulting .r for functions and procedures that might not be used.

Now the problem comes up where one might want to include the include files for functions A and B, both of which use function C. Under the programming methodology I use, function C would be included twice and upon compiling, an error would occur.



Problem where modular use of Error.i can create a compile problem!

With the pre-processor statements now available, one can over-come this problem. One wraps preprocessor statements around code that might be included already. It is very simple - composed of three lines.

The first line determines if a &GLOBAL-DEFINE that should be named unique to each include file has been defined. The naming convention I use is the name of the file name for the include it's self. So as shown below, if PostFix.i is the name of the file, the global define would be POSTFIX_I (since . cannot be used as part of the name in the P4GL.)

If the global define has been defined, that means the module has already been included someplace else where in the source code and will not bother to compile it. If not, then it will allow the compiler to work on source within the &IF and &ENDIF of the pre-processor.

The next line is the actual definition of the &GLOBAL-DEFINE that the above &IF checked for.

The last line is at the end of the source code, which finishes the &IF with an &ENDIF.

Below is a sample program that returns the postfix of a given file name. The preprocessor statements controlling it's compilation are shown in bold.

PostFix.i

```
&IF DEFINED(POSTFIX_I) <> 1 &THEN
&GLOBAL-DEFINE POSTFIX_I Y

DEF VAR RCSVersionPostFix AS CHARACTER INIT "$Header:
/home/appl/BlueDiamond/prgsrc/RCS/PostFix.i,v 1.1 2002/01/16 08:07:22 sauge Exp
$" NO-UNDO.

FUNCTION PostFix RETURNS CHARACTER (INPUT cFileName AS CHARACTER ):

  IF cFileName = "" THEN RETURN "".
  ASSIGN cFileName = SUBSTRING(cFileName, R-INDEX (cFileName, ".")).
  RETURN cFileName.

END. /* FUNCTION PostFix */

&ENDIF
```

About the author: Scott Auge is the founder of Amduus Information Works, Inc. He has been programming in the Progress environment since 1994. His works have included E-Business initiatives and focuses on web applications on UNIX platforms.
sauge@amduus.com

Coding Article: Web programs help pop-ups

When coding character or GUI based applications, one often has an input box. That input box would expect some kind of data that is not easily remembered (such as a part number or inventory location.) With GUI and character applications, it is easy to set up a button that can be used to help the user search a table or view for what they need to input. Under character in Progress, one programs a routine for the F2 key, and under GUI one sets up a lookup button.

But how can one do this in the web world? Below are two routines to simplify this goal.

One function creates a button that would be placed next to the input box that would be populated with the help data. The other creates a hyperlink, that when clicked, will automatically populate the input box.

The general flow of events reads as follows:

1. The user renders a web page upon which is a text box that the user will need a lookup on.
2. Next to the text box, is a call to `LookupButton()` which will render the button.
3. The user can click on this button which will call a URL in a new window.
4. The user can use the pop-up window web page to generate some results encoded in hyperlinks.
5. These hyperlinks are actually generated by the `AnswerHyperlink()` function.
6. Upon clicking of the hyperlink, javascript is activated that populates the calling window's input box and closes the pop-up window.
7. Done!

AnswerHyperLink()

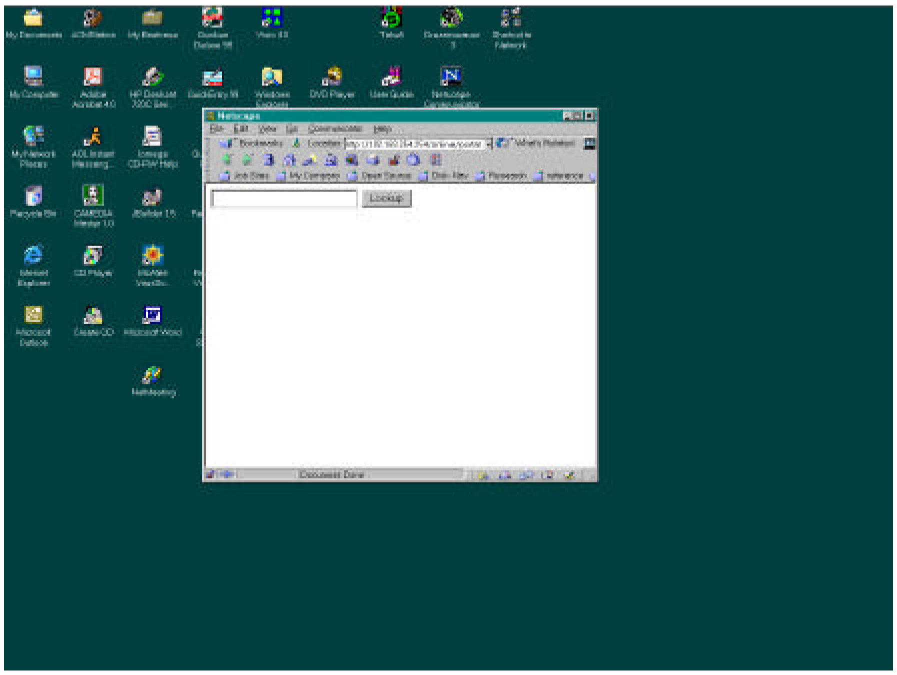
```
/* Chooser Function */
/* SetAnswer - used to set the answer to the field in the calling */
/* window. */

/* This function will create a hyperlink, that when clicked, will place */
/* cValue into FieldName of the window using a button created by the    */
/* LookupButton() function.                                             */

FUNCTION AnswerHyperLink RETURNS CHARACTER
(INPUT FieldName AS CHARACTER,
 INPUT cValue AS CHARACTER,
 INPUT cDescription AS CHARACTER):

  RETURN "<a href=~"#~""
       + " onclick=~'javascript:opener.document.forms[0]."
       + FieldName + ".value=~"" + cValue + "~";window.close();~'>"
       + cDescription + "</a> <br>".

END. /* FUNCTION SetAnswer () */
```

LookupButton()

```
/* This function will create a button that when clicked, will bring up a */
/* pop-up window with the PageName URL.  From that page, one should use  */
/* GET-VALUE("FieldName") to determine the FieldName that is to be pop-  */
/* ulated with the answer (as generated by AnswerHyperLink()).           */

FUNCTION LookupButton RETURNS CHARACTER
(INPUT FieldName AS CHARACTER,
 INPUT PageName AS CHARACTER):

  RETURN "<input type=~"button~" name=~"Submit2~" value=~"Lookup~" "
       + " onclick=~"javascript:window.open(~'"
       + PageName + "?FieldName=" + FieldName + "~',"
       + "~'LookupWindow~',~'scrollbars=yes,width=300,height=300~')~">".

END. /* FUNCTION LookupButton */
```

Web page needing a lookup button

Below is a simple web page that requires some lookup functionality. It simply presents an input box and the Lookup button.



Web page using the LookupButton() function

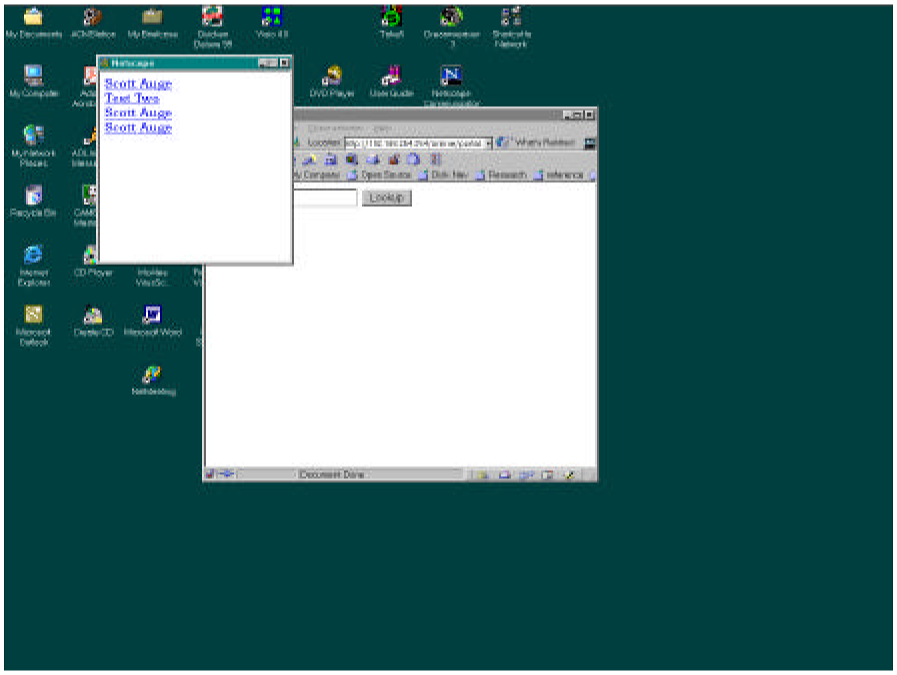
Below is the E4GL code used to render this web page. The code relevant to generating the Lookup button to properly call the Lookup page is presented in bold.

```
<!--WSS

{LookupTools.i}

-->
<html>
<body>
<form>
<input type="text" name="ListMember"> `LookupButton ("ListMember",
"Lookup.html")`
</form>
</body>
</html>
```

As you can see, simply call the `LookupButton()` function with the name of the input, and the web page that will help the user choose a value.

Web page that acts as the lookup

Below is the help page that popped up upon the click of the Lookup button. In order for this to work, the browser must have javascript turned on.



The user would choose the hyperlink that fit their input needs. Of course, one can make the web page more complicated and visually pleasing.

Below is the E4GL that is needed in the pop-up window shown in bold.

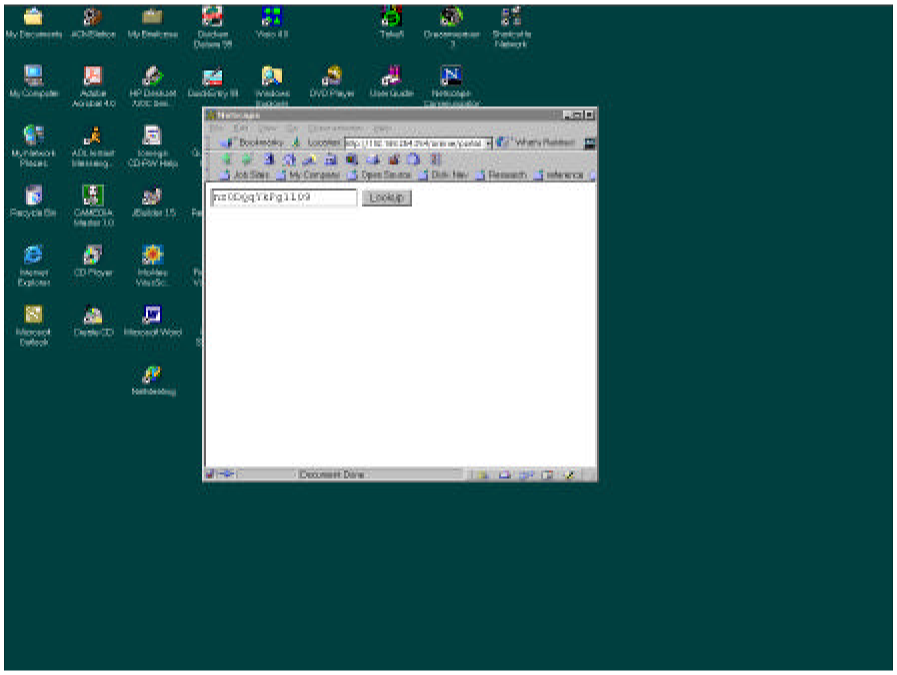
```
<!--WSS
```

```
{LookupTools.i}

DEF VAR cFieldName AS CHARACTER NO-UNDO.

ASSIGN cFieldName = GET-VALUE("FieldName").

-->

<html>

<body>

<!--WSS
FOR EACH MailListMember NO-LOCK:
  FIND Person OF MailListMember NO-LOCK.
-->
`AnswerHyperLink(cFieldName, MailListMember.MailListMemberID,
Person.FirstName +
 " " + Person.LastName)`
<!--WSS
END. /* FOR EACH MailListMember */
-->

</body>

</html>
```

As you can see, there is a Name/Value pair called FieldName that contains the HTML name of the input widget. Be sure to nab this value as it is automatically sent along from the `LookupButton()` function.

One creates the hyperlinks using the AnswerHyperLink() function. It's arguments are the name of the input widget, the value you wish the widget to have upon clicking the hyperlink, as well the human readable text that sits on top of the hyperlink.

Below is resulting screen upon clicking one of the hyperlinks. The input box is populated with the value and the pop-up window is destroyed, allowing the user to continue their work.



About the author: Scott Auge is the founder of Amduus Information Works, Inc. He has been programming in the Progress environment since 1994. His works have included E-Business initiatives and focuses on web applications on UNIX platforms. *sauge@amduus.com*

Management Article: Items to consider in one's pricing

Written by Scott Auge sauge@amduus.com

Often one is prompted to come up with a price for some work. Here are some considerations you can examine and explain to the customer why a certain price came to be.

Skills required

The more skills that are required the higher the price should be. Itemize out the skills needed – it will give you some ideas of what might be required later on. Listing these items out should one be asked is a great way to help the customer identify the complexity of skills needed. Skills to consider are programming languages, software tools used (such as OS, DB Server and Web Server), and industry knowledge.

Originality of the Problem

If it is a problem that has been done before, then the value of the solution will be less. There will be references available to adapt a solution from or to use. If the problem is unknown, then an increase in creativity and problem solving skill will be required. These are valuable items.

Time and Materials

Should one need to provide materials (such as Progress Licenses, computer servers, etc.) – these will definitely be a factor to include in your calculations.

Another item to consider is the value of your time. When you allocate time to a project for a given price, there is some "opportunity cost" in that you may not be able to allocate time for a more valuable project. If you are finding that your time is in high demand, one way to free up some of it is to raise prices. Of course, if you have a lot of interest, and are not converting that interest into customers – you may need to lower the price of your time.

About the author: Scott Auge is the founder of Amduus Information Works, Inc. He has been programming in the Progress environment since 1994. His works have included E-Business initiatives and focuses on web applications on UNIX platforms.
sauge@amduus.com

Publishing Information:

Scott Auge publishes this document. I can be reached at scott_auge@yahoo.com.

Currently there are nearly 600 subscribers and companies that receive this mailing! This mailing is not sent unsolicited, so it is not SPAM.

Amduus Information Works, Inc. assists in the publication of this document:

Amduus Information Works, Inc.
1818 Briarwood
Flint, MI 48507
http://www.amduus.com

Article Submission Information:

Please submit your article in Microsoft Word format or as text. Please include a little bit about yourself for the About the Author paragraph.

Looking for technical articles, *marketing Progress* articles, articles about books relevant to programming/software industry, white papers, etc.

I am looking for work, if you have any knowledge of potential work, I would appreciate hearing from you!

http://www.amduus.com/Resumes/ScottAuge.html

Thanks!