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:

When I put out word that issue 16 was to have vi productivity tips I was emailed by someone hoping that it would be syntax highlighting. Well, now that I have got some ideas on how to achieve this – here it is! Hence, how to do Progress syntax highlighting in both the GUI and CHUI worlds of the UNIX environment with the vim editor is included in this issue. The vim editor stands for vi-improved and is available for many platforms. The one I used is on Red Hat 7.1 Linux.

Please keep Amduus Information Works, Inc. in mind when you have a project coming up! There is more than just one guy associated with this corporation! Amduus is already influencing the programming practices of hundreds of programmers in hundreds of companies – can't be all that bad!

Following that article is the completion of the menu maintenance article started in issue 17. This is a simple to use maintenance program to build up the menu that is rendered in the code given in issue 17. With a bit of tweaking, it can probably work with BOMs and other tree based data structures!

Amduus Information Works, Inc. is looking for representatives for the company. We would back you up for support plans and work you may encounter.

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

Coding Article: Syntax coloring in vim

Written by Scott Auge

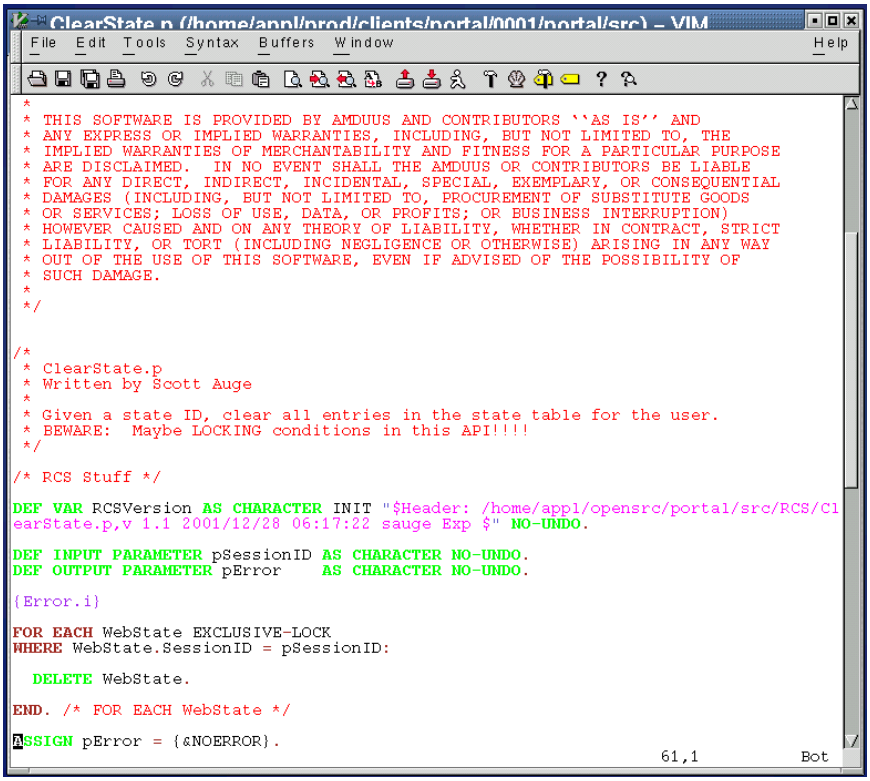
It just doesn't seem fair that the Windows world has syntax highlighting when us in the UNIX world do not! After all, isn't UNIX where e-mail, networking, the world wide web, and all that came from?

Actually – we can do syntax highlighting in the UNIX environment – in this particular instance I am showing it with Red Hat 7.1 and Windows UNIX connectivity tools from WRQ (www.wrq.com) software. An example using a Terminal Emulator is given for the telnet fans, and an example given with X-Windows is given for the GUI fans.

This covers simple 4GL – but in the future, I plan on working on something that handles the syntax for E4GL – something that would definitely by useful – especially for those `` tags tossed in the HTML.

vim in the X-Windows world

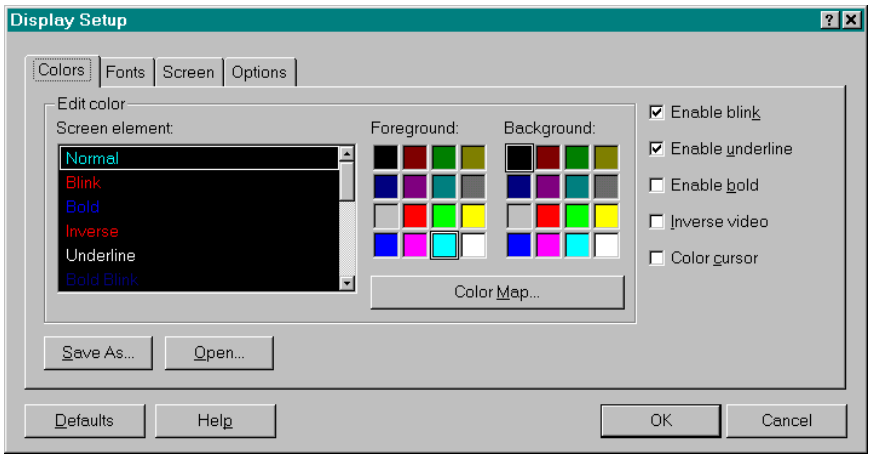
GUI vim on UNIX with syntax highlighting



In the GUI world of UNIX, the vim editor worked quite nicely and I was surprised by it's ease of use. Scrolling, opening and closing of files, etc all could be done via clicks of the mouse. But yet it still had the power of the vi macros, etc.

About character terminal emulators

Character terminals were a little more challenging. As you can see below, I can set up colors for text under my emulation (vt400-7) under certain attributes. I have six attributes I can play with which will give me six colors on my screen. As this does effect the other programs you might use telnet for, you may wish to set up a different parameter file for a session that you would use vim with. Otherwise you might be fiddling with colors as you go between editing and using other programs.

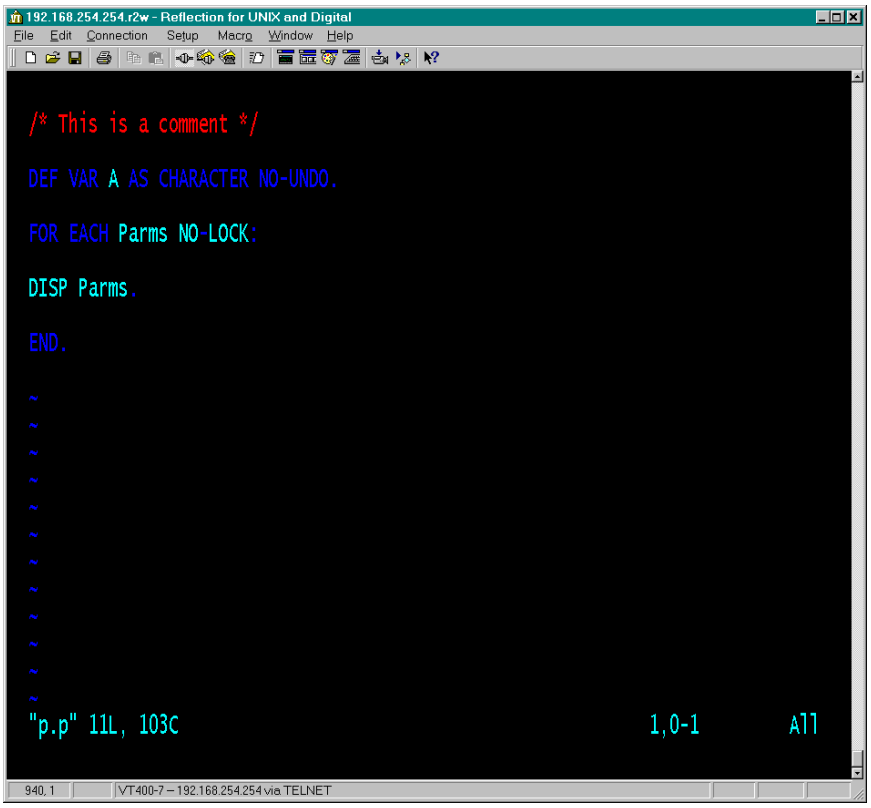


Setting up the colors on VT400-7

Once the colors are chosen, it comes out pretty nicely as can be seen below. Some of the things you may want different colors for are:

- Progress 4GL statements
- Strings and numbers
- Comments

As you can see, these are three items. Philip Uren maintains a syntax file that is already distributed with vim! He has introduced a DEBUG/TODO word used in comments that help those items stand out some.



CHUI (VT400-7) vim with syntax highlighting[1]

Defining vim to use syntax coloring

First be forewarned that some distributions of vim do not have syntax highlighting installed. On my VA Linux 6.2 machine, I had vim, but when I tried to use syntax highlighting it said that option was unavailable! Luckily I had another machine with Red Hat 7.1 installed on it – and it ran just fine!

The use of vim for syntax highlighting can be very powerful, where based on the postfix of the file, it will figure out if it is fortan, c, progress, or what have you. The explanation in this article is very simple as it is a way to tell vim how use the progress syntax and that's all.

First one should create a .vimrc in your home directory. It can use all the items spoken of in the .exrc[2] and .virc files, but adds some more definitions that are vim related.

`$HOME/.vimrc`

[1] DISP & NO-LOCK in this syntax definition has no definition so it is treated as normal. Notice how the arithmetic symbols are highlighted also due to a rule.

[2] See Issue 16 for abbreviations, which can increase your typing speed by 50% to 70%!!!

Inside this file, you can add items that you normally would need to enter in command mode in vim. This provides a nice short cut way of getting vim configured. To enable syntax highlighting, you want to have these three lines in your .vimrc file.

```
syntax enable
syntax on
set syntax=progress
```

The syntax file

Next there is a special file used to define the syntax highlighting rules. You will want to create a .vim/syntax directory in your home directory, with a file named progress.vim. (The `set syntax=` in the .vimrc uses that name plus the postfix .vim to identify the file in this directory.)

```
$HOME/.vim/syntax/progress.vim
```

This can get complicated, so lets start with what we want – some colors!

Colors available for terminals

The first 8 colors are available in terminals with 8 colors. All the colors are available in terminals with 16 colors. You refer to the color by it's name, as you will see later on.

```
0       0       Black
1       4       DarkBlue
2       2       DarkGreen
3       6       DarkCyan
4       1       DarkRed
5       5       DarkMagenta
6       3       Brown, DarkYellow
7       7       LightGray, LightGrey, Gray, Grey
8       0*      DarkGray, DarkGrey
9       4*      Blue, LightBlue
10      2*      Green, LightGreen
11      6*      Cyan, LightCyan
12      1*      Red, LightRed
13      5*      Magenta, LightMagenta
14      3*      Yellow, LightYellow
15      7*      White
```

Colors available for GUI

The terminal colors as well as these are available under GUI .

```
Red         LightRed        DarkRed
Green       LightGreen      DarkGreen       SeaGreen
Blue        LightBlue       DarkBlue        SlateBlue
```

```
Cyan        LightCyan       DarkCyan
Magenta     LightMagenta    DarkMagenta
Yellow      LightYellow     Brown           DarkYellow
Gray        LightGray       DarkGray
Black       White
Orange      Purple          Violet
```

Highlights available on terminals and GUI

Not only can you control the color, but you can control the attributes for the text with these words. Note on the example terminal above, I used these and the color settings on the terminal emulation program to achieve color syntax highlighting.

```
bold
underline
reverse
inverse         same as reverse
italic
standout
NONE            no attributes used (used to reset it)
```

Grouping symbols into sets that are colored

Words that are to be syntax colored are gathered into sets. These sets are named however you wish to name them, and then later on those sets are associated with a color and highlight that should be used when they are displayed in vim. These sets are of three types: keyword, region, and match. The syntax is as follows:

syn [keyword|match|region] SetName SetElements

So, to define some Progress 4GL keywords, one would do the following:

```
syn keyword ProgressStatement DEFINE DEF VAR VARIABLE FOR EACH ASSIGN
syn keyword ProgressStatement CREATE DELETE FIND
```

As you can see, definitions can be continued on multiple lines to the same set called ProgressStatement.

One then defines the color this set would be shown in when rendered on the screen. This is done with the highlight statement which can be abbreviated hi. Of it, there are five attributes that are interesting: term, termfg, termbg, guifg, guibg

These mean terminal foreground and background, gui foreground and background color, as well as the terminal attribute that works in both gui and terminal use of vim. Here is an example used to provide the color syntax in the above pictures:

```
hi ProgressStatement        guifg=green ctermfg=green term=bold
```

As you can see, you can mix and match the gui and chui keywords.

So now that we know how to do individual words, how do we handle sets of words? Such as those items in a progress comment? We do this with the "region" version of the syntax line. It's format is this:

```
syn region SetName start="regexp" end="regexp" contains=ListOfSets
```

An example as follows for progress comments:

```
syn region ProgressComment      start="/\*"  end="\*/"
contains=ProgressComment,ProgressTodo,ProgressDebug,ProgressNeedsWork
```

The contains list is use to prevent the override of those item's color by the regions color definition. For example:

```
syn keyword ProgressTodo      TODO
hi ProgressToDo   guifg=black guibg=yellow
```

will give the word TODO a black text on a yellow background. When TODO is included in a comment, we do not want to loose this coloring, so we assign it to a group and tell the region definition about it.

But, back to the region definition, once we have a name associated to the region, we can color the name:

```
hi ProgressComment guifg=red termfg=red term=highlight
```

Sample Definition File

```
" This is a comment
" Ignore word case
syn case ignore
```

```
" Define some progress keywords
syn keyword ProgressStatement DEFINE DEF VAR VARIABLE FOR EACH ASSIGN
syn keyword ProgressStatement CREATE DELETE FIND

" Define a region for Progress comments
syn region ProgressComment      start="/\*"  end="\*/"
contains=ProgressComment

" Associate groups to colors
hi ProgressComment guifg=red termfg=red term=highlight
hi ProgressStatement guifg=green ctermfg=green term=bold
```

More information

:help syntax in the vim editor.

Another vim author for Progress

I borrowed a little from the work of Philip Uren philu@computer.org for this article.

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 based menu system Part II

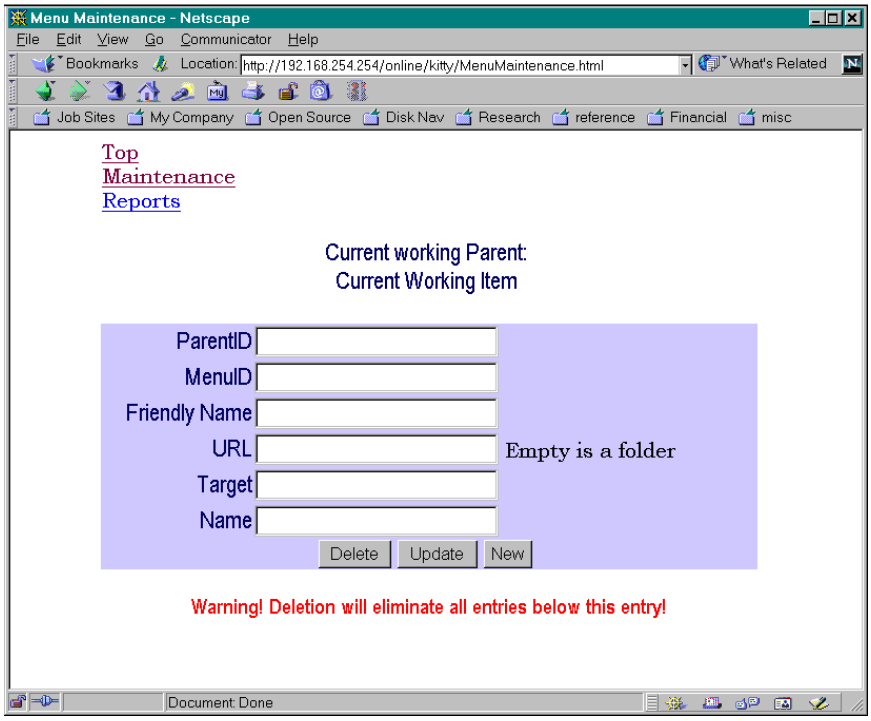
Written by Scott Auge sauge@amduus.com

What we are trying to achieve

Below you see a picture of the maintenance program. It allows recursive work on the menu keeping track of which part of the menu the user is working on.

One of the inputs is the ParentID of a menu item. If the parent ID is empty, then the menu item is placed at the top of the menu structure. Else, the parent ID should be the menu id of a menu item for which that entry should be below.

The menuID uniquely identifies the menu item compared to the other menu items. It never appears to the user, but is used as an internal value to short hand the menu name as a foreign key. It should not contain spaces.



Basic inputs for the page

Following that entry, is the Friendly Name. This is what appears to the user as the text of the hyperlink. It can contain spaces and should be the title of the screen or collection of items the user wishes to drill down into.

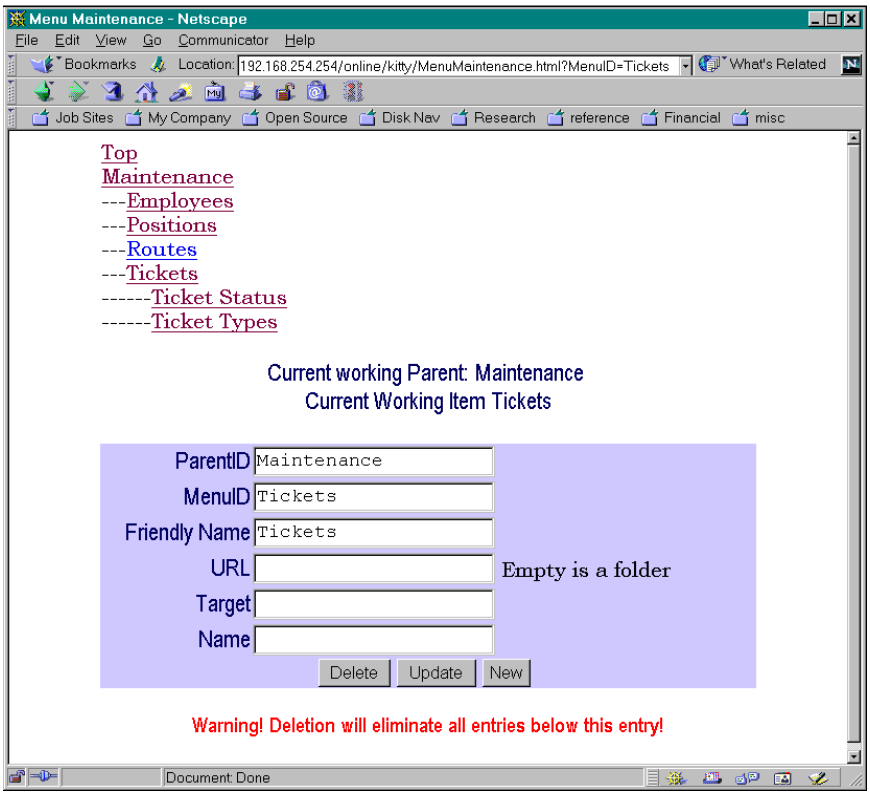
The URL is what lies under the hyperlink. It can be a relative or absolute URL to a web page. If left blank, then the item becomes a folder the user is expected to be able to drill down into.

Obviously we do not want the new web page to appear in the same window or frame as the menu – and so the Target input identifies the FRAME name or Window name the web page noted in URL should open in. Most browsers will open a new window for names not in framesets.

Name is unused in this program.

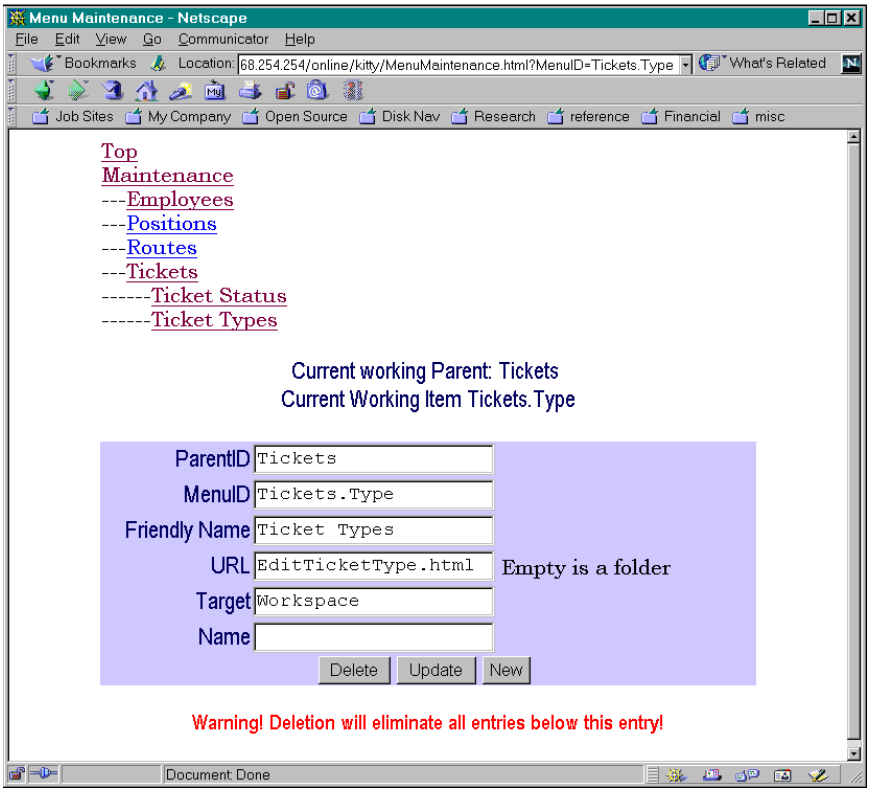
One would use the existing menu items at the top to navigate to a level in which to delete and modify an entry, or to reach a level in which to add new items. The purpose of the navigation is to help pre-populate the data entry fields. If one knows the structure well, s/he could merely populate the data entry fields themselves.

Creating a Tickets folder on the menu



To create an entry, one merely fills in the items, and clicks the New button. A record will be made. Dangling records, that is records that cannot be reached by menu items above will not be shown.

Deletion occurs on the Menu Id given on the screen and clicking the Delete button. All entries beginning at that menu id and including the menu id will be removed from the database. The screen will leave the user editing the next highest record, if there is one.



Updating a record is based on the Menu ID. The value in the Menu ID will decide which record is updated. One can walk the navigation tree to reach the record, or merely type it in and click Update.

The menu ID should never be modified once it is created.

A more advanced use would allow the user to move an entire tree of menu items under a different menu item by set the Parent ID directly to the menu id of the entry you wish the tree to fall under.

The MenuMaintenance.html code:

The MenuMaintenance code is pretty straight forward. An "action" is specified, being Blank, Delete, Update, or New – just like the actions of the buttons. One can think of this as web event programming ☺.

It searches for the Menu record identified by the NVP[3] MenuID. One can then delete it by calling DelTree.p or update it using the NVPs for the input boxes, or create one and then updating it via the NVPs of the input boxes.

It then renders the Menu tree placing that HTML into a variable.

That variable is then displayed along with the HTML.

```
DEF VAR cMenuTree AS CHARACTER NO-UNDO.
DEF VAR cParentMenuID AS CHARACTER NO-UNDO.
DEF VAR cMenuID       AS CHARACTER NO-UNDO.

DEF VAR cName          AS CHARACTER NO-UNDO.
DEF VAR cFriendlyName  AS CHARACTER NO-UNDO.
DEF VAR cURL           AS CHARACTER NO-UNDO.
DEF VAR cTarget        AS CHARACTER NO-UNDO.
DEF VAR cAction        AS CHARACTER NO-UNDO.

/* Figure out which menu item we are working with */

ASSIGN cMenuID = GET-VALUE("MenuID").
ASSIGN cParentMenuID = GET-VALUE("ParentMenuID").
ASSIGN cAction = GET-VALUE("Submit").

/* Pre-populate the values if we can */

IF cMenuID <> "" AND cAction = "" THEN DO:
```

[3] NVP is Name/Value Pair. Name associated to an input widget and its value.

```
  FIND Menu NO-LOCK
  WHERE Menu.MenuID = cMenuID
  NO-ERROR.

  ASSIGN
  cName = Menu.Name
  cFriendlyName =  Menu.FriendlyName
  cURL =  Menu.URL
  cTarget = Menu.Target
  cParentMenuID = Menu.ParentMenuID.

END.

IF cAction = "Delete" THEN DO:

  RUN DelTree.p (INPUT cMenuID, OUTPUT cMenuID).

END.

ELSE IF cAction = "Update" THEN DO:

  FIND Menu EXCLUSIVE-LOCK
  WHERE Menu.MenuID = cMenuID
  NO-ERROR.


  ASSIGN
  cName = GET-VALUE("Name")
  cFriendlyName = GET-VALUE("FriendlyName")
  cURL = GET-VALUE("URL")
  cTarget = GET-VALUE("Target")
  cParentMenuID = GET-VALUE("ParentMenuID").

  ASSIGN
  Menu.ParentMenuID = cParentMenuID
  Menu.MenuID = cMenuID
  Menu.URL = cURL
  Menu.Target = cTarget
  Menu.FriendlyName = cFriendlyName
  Menu.Name = cName.

END.

ELSE IF cAction = "New" THEN DO:
```

```
  CREATE Menu.

  ASSIGN
  cName = GET-VALUE("Name")
  cFriendlyName = GET-VALUE("FriendlyName")
  cURL = GET-VALUE("URL")
  cTarget = GET-VALUE("Target").
  cParentMenuID = GET-VALUE("ParentMenuID").
  cMenuID = GET-VALUE("MenuID").

  ASSIGN
  Menu.ParentMenuID = cParentMenuID
  Menu.MenuID = cMenuID
  Menu.URL = cURL
  Menu.Target = cTarget
  Menu.FriendlyName = cFriendlyName
  Menu.Name = cName.

END.

RUN MtnGenTree.p
(INPUT cMenuID,
 INPUT "MenuMaintenance.html",
 OUTPUT cMenuTree ).
-->

<html>
<head>
<title>Menu Maintenance</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-
1">
</head>

<body bgcolor="#FFFFFF">
<table width="80%" border="0" cellspacing="0" align="center">
  <tr>
    <td>
       <a href="MenuMaintenance.html">Top</a><br>
       `cMenuTree`
    </td>
  </tr>
</table>
<form method="post">
```

```
  <p align="center"><font face="Arial Narrow" size="+1"
color="#000066">Current
    working Parent: `cParentMenuID` <br>
    Current Working Item `cMenuID`</font></p>
  <table width="80%" border="0" cellspacing="0" align="center"
bgcolor="#CCCCFF">
    <tr>
      <td>
        <div align="right"><font face="Arial Narrow" size="+1"
color="#000066">
ParentID</font></div>
      </td>
      <td>
        <input type="text" name="ParentMenuID" value="`cParentMenuID`">
      </td>
    </tr>
    <tr>
      <td>
        <div align="right"><font face="Arial Narrow" size="+1"
color="#000066">
MenuID</font></div>
      </td>
      <td>
        <input type="text" name="MenuID" value="`cMenuID`">
      </td>
    </tr>
    <tr>
      <td>
        <div align="right"><font face="Arial Narrow" size="+1"
color="#000066">Friendly
          Name</font></div>
      </td>
      <td>
        <input type="text" name="FriendlyName" value="`cFriendlyName`">
      </td>
    </tr>
    <tr>
      <td>
        <div align="right"><font face="Arial Narrow" size="+1"
color="#000066">URL</font></div>
      </td>
      <td>
        <input type="text" name="URL" value="`cURL`">
        Empty is a folder</td>
    </tr>
```

```
    <tr>
      <td>
        <div align="right"><font face="Arial Narrow" size="+1"
color="#000066">Target</font></div>
      </td>
      <td>
        <input type="text" name="Target" value="`cTarget`">
      </td>
    </tr>
    <tr>
      <td>
        <div align="right"><font face="Arial Narrow" size="+1"
color="#000066">Name</font></div>
      </td>
      <td>
        <input type="text" name="Name" value="`cName`">
      </td>
    </tr>
    <tr>
      <td colspan="2">
        <center>
          <input type="submit" name="Submit" value="Delete">
          <input type="submit" name="Submit" value="Update">
          <input type="submit" name="Submit" value="New">
        </center>
      </td>
    </tr>
  </table>
</form>
<p align="center"><font face="Arial Narrow" size="3"
color="#FF0000"><b>Warning!
  Deletion will eliminate all entries below this entry!</b></font></p>
</body>
</html>
```

MtnGenTree.p

Same as the GenTree.p of before, but has more stringent checks on the existence of records, and handles the depth differently because while editing the menu, we can actually go down one deeper than exists (after all, that is how we create new items and levels.)

```
DEF INPUT PARAMETER cMenuID AS CHARACTER NO-UNDO.
DEF INPUT PARAMETER cHTMLPageName AS CHARACTER NO-UNDO.
```

```
DEF OUTPUT PARAMETER cMenuHTML AS CHARACTER NO-UNDO.

DEF VAR iMaxDepth AS INTEGER NO-UNDO.
DEF VAR cParentMenuID AS CHARACTER NO-UNDO.
DEF VAR cHTML AS CHARACTER NO-UNDO.

ASSIGN iMaxDepth = 0.

FIND Menu NO-LOCK
WHERE Menu.Menu = cMenuID
NO-ERROR.

IF NOT AVAILABLE Menu THEN DO:

  FIND FIRST Menu NO-LOCK
  WHERE Menu.ParentMenuID = ""
  NO-ERROR.

  IF NOT AVAILABLE Menu THEN RETURN.

  ASSIGN iMaxDepth = -1.

END.

RUN FindDepth.p
(Menu.MenuID,
 iMaxDepth,
 OUTPUT iMaxDepth
).

RUN MtnGenTreeHTML.p
(INPUT cHTML,
 INPUT cMenuID,
 INPUT cMenuID,
 INPUT iMaxDepth,
 INPUT cHTMLPageName,
 OUTPUT cMenuHTML
).
```

MtnGenTreeHTML.p

This is very similar to the GenTreeHTML.p program, but since we are working at a lower level than usual (in other words, we can be at a menu level that does not yet exist!) it needed to be touched up a bit.

```
DEF INPUT PARAMETER cHTML AS CHARACTER NO-UNDO.
DEF INPUT PARAMETER cMenuID AS CHARACTER NO-UNDO.
DEF INPUT PARAMETER cPopoutMenuID AS CHARACTER NO-UNDO.
DEF INPUT PARAMETER iMaxDepth AS INTEGER NO-UNDO.
DEF INPUT PARAMETER cHTMLPageName AS CHARACTER NO-UNDO.
DEF OUTPUT PARAMETER cOutHTML AS CHARACTER NO-UNDO.

DEF BUFFER BufMenu FOR Menu.

DEF VAR cString AS CHARACTER NO-UNDO.

ASSIGN cString = "--------------------------------".

FIND Menu NO-LOCK
WHERE Menu.MenuID = cMenuID
NO-ERROR.

IF AVAILABLE Menu THEN RUN MtnGenTreeHTML.p (INPUT cOutHTML,
                                             INPUT Menu.ParentMenuID,
                                             INPUT Menu.MenuID,
                                             INPUT iMaxDepth - 1,
                                             INPUT cHTMLPageName,
                                             OUTPUT cOutHTML).


FOR EACH Menu NO-LOCK
WHERE Menu.ParentMenuID = cMenuID:

  ASSIGN cOutHtml = cOutHTML
                  + SUBSTRING (cString, 1, iMaxDepth * 3).

  ASSIGN cOutHtml = cOutHTML
                  + "<a href=~"" + cHTMLPageName
                  + "?MenuID=" + Menu.MenuID.

  ASSIGN cOutHtml = cOutHTML
                  + "~">" .

  ASSIGN cOutHTML = cOutHTML
                  + Menu.FriendlyName + "</a>"
```

```
                    + "<br>~n".

  IF Menu.MenuID = cPopoutMenuID THEN LEAVE.

END.
```

DelTree.p

Deltree.p is recursive deletion of the records starting at the given MenuID record downwards. It will return the ParentMenuID of the MenuID record given.

```
DEF INPUT PARAMETER cMenuID          AS CHARACTER NO-UNDO.
DEF OUTPUT PARAMETER cParentMenuID   AS CHARACTER NO-UNDO.

DEF BUFFER BufMenu FOR Menu.

FIND Menu EXCLUSIVE-LOCK
WHERE Menu.MenuID = cMenuID
NO-ERROR.

IF NOT AVAILABLE Menu THEN RETURN.

FOR EACH BufMenu EXCLUSIVE-LOCK
WHERE BufMenu.ParentMenuID = Menu.MenuID:

  RUN DelTree.p (INPUT BufMenu.MenuID, OUTPUT cParentMenuID).

END.

ASSIGN cParentMenuID = Menu.ParentMenuID.

DELETE Menu.
```

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 sauge@amduus.com.

Currently there are over 800 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:

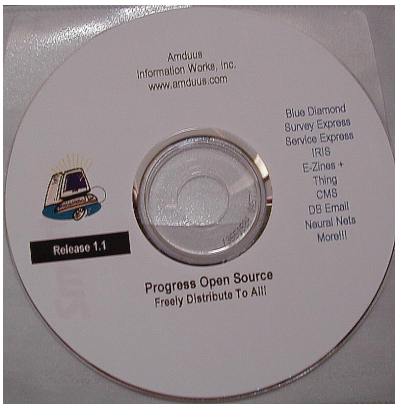
Amduus Information Works, Inc.
1818 Briarwood
Flint, MI 48507
http://www.amduus.com

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.

Order Form for Progress Open Source CD-ROM
COUPON 001A



This is an offer for the CD-ROM at lower than list savings!

Mail this form to:

Amduus Information Works, Inc.
1818 Briarwood
Flint, MI 48507

Please send _____ copies of the Open Source CD-ROM at $15.00 per disk to:

Name _____
Company _____
Address _____
City _____
State _____
Zip _____

Please make your checks/money orders out to: Amduus Information Works, Inc. Cash works too!
This offer only valid in the United States of America.

The CD-ROM includes (all source code included):

- Blue Diamond/IRIS – Webspeed alternatives
- Survey Express – easily create text templates of surveys and then have the program generate the web pages automatically
- Service Express – Web based Help Desk.
- The Progress E-Zines, books on learning to program in Webspeed (PDF/Word/HTML)
- THING – simple tool to manipulate database records with
- CMS – a web content management system
- DB Email – Use pop3 to download emails into a Progress database
- Neural Networks – experiments in spam recognition and text message classification
- GenPDF – create PDF file reports for Webspeed/UNIX CHUI!
- More!