The Progress Electronic Magazine

In this issue:

This document may be freely shared with others without modification.

Though intended for users of the software tools provided by Progress Software Corporation, this document is NOT a product of Progress Software Corporation.

© Amduus Information Works, Inc. 2002

Publisher's Statement:

I know that I said in the message to the progress email group that I was going to discuss a web based parameter system. Unfortunately the E-Zine was getting pretty big and I had to make a choice about what to place in the E-Zine. Since the other articles were of more interest to the general progress programmer and to the experimental programmer, I decided to move the web based parameter code to the next E-Zine.

In this issue we further explore natural language interfacing to Progress applications. We do this in a fun way by adding to our adventure game. Doing it in little steps, we update the approach we were taking with the data structures, as well include code for traveling about the map of our adventure world. (The map discussion is located in issue 25.)

Reach nearly one thousand programmers and companies.

Your ad could be here!

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

Imagine, by adding in ScanSoft's Dragon Speak Naturally (http://shop.voicerecognition.com/itemdesc.asp?CartId=7272147-ACCWARE-NVCCE265&ic=DNS+%2D+6%2E0+PROF&Cc=&tpc=) with a natural language interface code to your Progress application/web browser to Webspeed – whoa! An application where someone can actually to talk to the computer to get something done. Could be quite the competitive advantage.

While current database algorithms and techniques handle data pretty well, they don't handle information or knowledge to well. Imagine all the knowledge that is stored in words on web pages, emails, and word processing documents. If one could collate all that information into a database and then have new database algorithms to discover that knowledge conveniently? Some directions to take this code and programming ideas I hope.

We also explore an application message logging system. By using these simple includes, you will be able to follow the flow of execution through your application. You can mention where the current point of execution is, variables, and put messages into a log file. It can be very helpful for deployed applications. When a customer has a problem, ask them to turn on the logging and then have them send it along to you for analysis.

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

Coding Article: Natural Language Interface and Adventure Game Part II

Written by Scott Auge sauge@amduus.com

Interacting with the game with Natural Language

A natural language interface means we wish the user to interact with the program using a human language construct, instead of via buttons, menu’s, check boxes, etc. associated with common user interfaces. I mean, computers have been using this kind of technology for nearly a decade and a half – time for a real update to the way we interact with computers.

Grammar and Vocabulary

The main focus of this effort is the grammar and vocabulary of the language. This can get complicated really fast, so I will keep the explanations very, very simple. The grammar is a set of constructs of different word types and in what order they appear for the given language. The vocabulary are those words that describe actions and identify entities in the game.

Joanju

```
def var x as int init 1.
x eq 12.
display x.
```

If you were using Prolint, you wouldn't have spent the last two hours tracking down this bug.
Proparse Lite - Only US$65
Lint your code before check-in.
Find bugs before your customer does.

w w w . j o a n j u . c o m

Actions are described by verbs. So verbs really will decide which parts of the code the application will execute. Nouns describe entities within the adventure. All together, the verbs will tell the program what to do with what nouns has data structures for. Generally the sentence’s presented by the user are in the first person, that is given form in terms of “I.”

Currently we are focused on moving around in the room map, so the verbs for those actions would be such words as: {move, travel, walk, run, go, trek, and journey}.

One moves about the map by identifying the direction they wish to move or items they wish to manipulate (next article.) The directions are considered nouns. The vocabulary to describe the direction of movement are generally compass points: {north, south, east and west}.

There will be other words that are not really needed by the parser. These generally are articles, prepositions, or simply words that should be ignored. Sample vocabulary might be: {the, a, to}. We will refer to these words as incidental words.

Sample Vocabulary and Grammar

So lets examine a sample phrase we hope the parser to handle:

”Travel to the north”

First our parser will eliminate the incidental words, leaving behind:

“Travel north”

Hence we can handle a grammar of:

Verb Preposition Article Noun
Verb Noun

and combinations of there of to interpret what the user is trying to express to the program.

Try not to make your grammar overly complex or exaggerated. It will be harder to parse it – one wants to be able to issue simple commands to the program and have it achieve something.

One of the benefits of the approach this method takes, is that works more on semantics of the word than the grammatical form of the words. For example, if you use the word “walk,” our system will recognize it as a verb. It will then lightly use grammar to figure out where to find other words that are of interest to work with that verb. One can think of it as a collection of verbs, and each of these verbs have a collection of nouns they individually would work on.

Since a verb represents an action, that means we are going to perform manipulations of some kind on the data structures (records and variables) within the program that represent nouns. One will associate a verb with a specific program. When that program is executed, it will manipulate the data to make the world appear as if the verb did as asked to be done.

Analysts Express, Inc.

Webspeed Training and progress programming.

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

In our movement example, it would be the re-assignment of the PlayerCurrentRoom value from one room number to another.

This is also a point where we need to discuss the scope of the variable. As you can imagine, the call to the routine will be something like:

```
FIND Verb NO-LOCK
WHERE Verb.Word = WordList.Word
NO-ERROR.

IF AVAILABLE Verb THEN RUN VALUE(Verb.Program).
```

where we are associating a verb to a program routine. We do this in a database table so we can conveniently add more vocabulary that may be synonyms to other vocabulary without having to re-write the application. We can also simply add functionality to the parser by a bit of code and a database entry of a new verb – no need to wander about the code base trying to figure out where to put our new code.

A problem with this approach, is that we need to send the program module the PlayerCurrentRoom variable. We can achieve this in two ways – one is by storing the value in a shared variable, and the other option is in a scratch value record that can be looked up by the routine.

When we have other verbs associated to programs, they may need a totally different set of data to work on – so we really never know before hand what arguments should be sent to the program module (unless it is every possible piece of data it could need to manipulate.)

It is much easier to add a new piece of data representing an entity in the game's world by making a new shared variable at the top level program or better yet, a new record entry in this scratch table.

Since the web program is stateless, it means our variables will loose their values after rendering of the screen. Keeping the values in the database will make it easier to find them again when we need them on subsequent posts to the game via web pages. In the end, the data is more manageable in the database instead of in hidden fields on the web page. After all, the database does describe the world the program is effecting.

Using WebState as the main data structure

Hence, the winner is – data structures about the current state of the game are stored in the database. Here is a possible data table definition to pull this off:

```
=============================================================================
============================ Table: WebState ===========================

             Table Flags: "f" = frozen, "s" = a SQL table
Table                          Dump     Table Field Index Table
Name                           Name     Flags Count Count Label
------------------------------ -------- ----- ----- ----- -------------------
WebState                       webstate           5     2 WebState

  Description: Used to store session information about a web based log in.
 Storage Area: Schema Area


============================= FIELD SUMMARY =============================
============================= Table: WebState ===========================

Flags: <c>ase sensitive, <i>ndex component, <m>andatory, <v>iew component

Order Field Name                Data Type   Flags Format          Initial
----- ------------------------- ----------- ----- --------------- ----------
   10 SessionID                 char        i     x(8)
   20 Category                  char        i     x(8)
   30 Name                      char        i     x(8)
   40 Data                      char              x(8)
   50 CreateDate                date              99/99/99        TODAY

Field Name                      Label                  Column Label
------------------------------- ---------------------- ----------------------
SessionID                       SessionID              SessionID
Category                        Category               Category
Name                            Name                   Name
Data                            Data                   Data
CreateDate                      CreateDate             CreateDate


============================= INDEX SUMMARY =============================
============================= Table: WebState ===========================

Flags: <p>rimary, <u>nique, <w>ord, <a>bbreviated, <i>nactive, + asc, - desc

Flags Index Name                       Cnt Field Name
----- -------------------------------- --- ---------------------------------
      key1                               1 + SessionID

pu    pukey                              3 + SessionID
                                           + Category
                                           + Name

** Index Name: key1
 Storage Area: Schema Area
** Index Name: pukey
 Storage Area: Schema Area
```

Basically, we are going to put our variable name in Name, and the value for that variable in Data. You may want to put the player's login in the SessionID field, or some value identifying the player in the SessionID field. (See GameLogin.html and GameLogin2.html in later issues.)

Below are the states that we will be using to store information used by the various portions of code in the application.

SessionID	Category	Name	Value

Randomly Generated and Assigned to the User	Map	CurrentRoom	Current room number the player is in.
	FeedBack	Message	Message from the application back to the user.
	FeedBack	Command	Command the user gave to the system.
	Player	Login	Login ID for the player

The application uses the WriteState.p and Error.i from www.amduus.com/OpenSrc/SrcLib/BlueDiamond/BlueDiamond/plussrc

See previous issues of the E-Zine for more discussion on the use of WebState.

Matching verbs to programming

Lets get back to our verb table. We want to associate some word to some kind of action, hence the table is actually pretty simple:

```
============================= Table: Verb ===============================

              Table Flags: "f" = frozen, "s" = a SQL table


Table                          Dump      Table Field Index Table
Name                           Name      Flags Count Count Label
------------------------------ --------- ----- ----- ----- -------------------
Verb                           verb                2     1 Verb

 Storage Area: Schema Area


============================= FIELD SUMMARY =============================
============================= Table: Verb ===============================

Flags: <c>ase sensitive, <i>ndex component, <m>andatory, <v>iew component

Order Field Name                Data Type   Flags Format          Initial
----- ------------------------- ----------- ----- --------------- ----------
   10 Word                      char        i     x(8)
   20 Program                   char              x(8)

Field Name                      Label                  Column Label
------------------------------- ---------------------- ----------------------
Word                            Word                   Word
Program                         Program                Program


============================= INDEX SUMMARY =============================
============================= Table: Verb ===============================
Flags: <p>rimary, <u>nique, <w>ord, <a>bbreviated, <i>nactive, + asc, - desc

Flags Index Name                       Cnt Field Name
----- -------------------------------- --- ---------------------------------
pu    pukey                              1 + Word

** Index Name: pukey
 Storage Area: Schema Area
```

By matching a verb to “action” we are matching a verb to some form of programming that will cause actions on the data in the application to appear as the verb has been performed.

There is ancillary code at the bottom of this article to pre-populate this information.

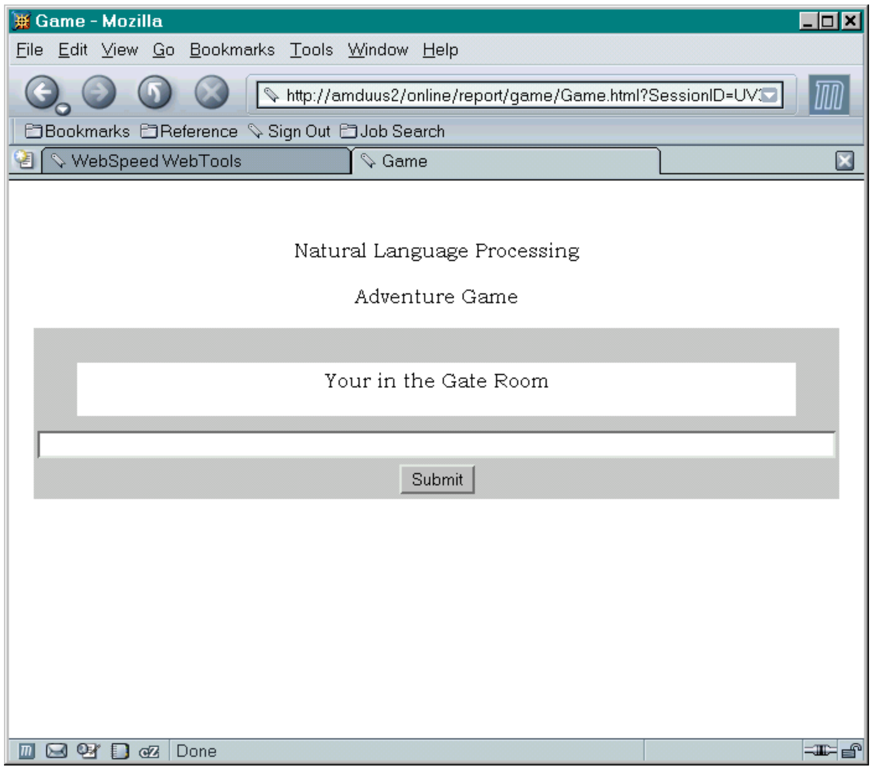
Word Types available

When the user enters in words, we need to decide what part of the grammar they belong to. This table will identify words as verbs, nouns, articles, prepositions, etc. See the ancillary code at the end of this article for pre-populating this information.

```
12/31/02 23:22:42       PROGRESS Report
Database: amduus (PROGRESS)
=========================================================================
============================= Table: WordType ===========================
              Table Flags: "f" = frozen, "s" = a SQL table
Table                        Dump     Table Field Index Table
Name                         Name     Flags Count Count Label
---------------------------- -------- ----- ----- ----- -------------------
WordType                     wordtype           2     1 ?
  Description: Words and grammar type
 Storage Area: Schema Area
============================= FIELD SUMMARY =============================
============================= Table: WordType ===========================
Flags: <c>ase sensitive, <i>ndex component, <m>andatory, <v>iew component
Order Field Name               Data Type   Flags Format          Initial
----- ------------------------ ----------- ----- --------------- ----------
   10 Word                     char        i     x(8)
   20 WordType                 char              x(8)
Field Name                    Label                  Column Label
----------------------------- ---------------------- ----------------------
Word                          Word                   Word
WordType                      WordType               WordType
============================= INDEX SUMMARY =============================
============================= Table: WordType ===========================
Flags: <p>rimary, <u>nique, <w>ord, <a>bbreviated, <i>nactive, + asc, - desc
Flags Index Name                       Cnt Field Name
----- -------------------------------- --- ---------------------------------
pu    pukey                              1 + Word
** Index Name: pukey
 Storage Area: Schema Area
============================= FIELD DETAILS =============================
============================= Table: WordType ===========================
** Field Name: WordType
  Description: Grammar for the word
```

Game.html

This is the main routine of the game. It acts as the information gathering point from the user, and presents to the user a view of the gaming world via text.


Game - Mozilla
File Edit View Go Bookmarks Tools Window Help
http://amduus2/online/report/game/Game.html?SessionID=UV
Bookmarks Reference Sign Out Job Search
WebSpeed WebTools
Game
Submit
Done

Natural Language Processing

Adventure Game

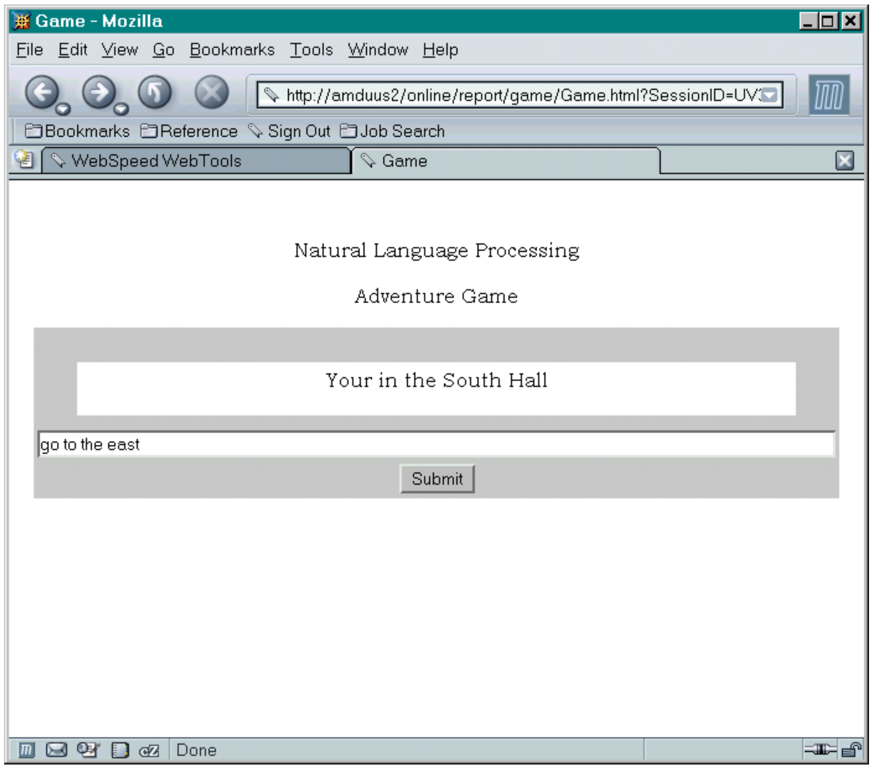
Your in the Gate Room

```
<!--WSS

DEF VAR cRoomDescription    AS CHARACTER NO-UNDO.
DEF VAR cRoomInventory      AS CHARACTER NO-UNDO.
DEF VAR cCommand            AS CHARACTER NO-UNDO.
DEF VAR cSessionID          AS CHARACTER NO-UNDO.
DEF VAR cError              AS CHARACTER NO-UNDO.
DEF VAR cMessage            AS CHARACTER NO-UNDO.

/* Input what the player wants to do */

ASSIGN cCommand = GET-VALUE("Command")
       cSessionID = GET-VALUE("SessionID").

/* Determine what the user's command is all about */

IF cCommand <> "" THEN DO:

  RUN WriteState.p (INPUT cSessionID,
                    INPUT "FeedBack",
                    INPUT "Command",
                  INPUT cCommand,
                  OUTPUT cError).

  RUN game/ParseCommand.p (INPUT cSessionID).

END.


/* Pull up the appropriate room description */

FIND WebState NO-LOCK
WHERE WebState.SessionID = cSessionID
  AND WebState.Category  = "Map"
  AND WebState.Name      = "CurrentRoom"
NO-ERROR.

FIND RoomDesc NO-LOCK
WHERE RoomDesc.RoomNumber = INT(WebState.Data)
NO-ERROR.

IF AVAILABLE RoomDesc THEN
  ASSIGN cRoomDescription = RoomDesc.Description.

/* Pull up any messages from the command parser */

FIND WebState NO-LOCK
WHERE WebState.SessionID = cSessionID
  AND WebState.Category  = "FeedBack"
  AND WebState.Name      = "Message"
NO-ERROR.

ASSIGN cMessage = WebState.Data.

-->

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Game</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-
1">
</head>

<body bgcolor="#FFFFFF">

<p>&nbsp;</p>
<p align="center">Natural Language Processing</p>
<p align="center">Adventure Game</p>
<form name="form1" method="post" action="">
<input type="hidden" value="`cSessionID`" name="SessionID">
  <table width="90%" border="0" align="center" bgcolor="#CCCCCC">
    <tr>
      <td>&nbsp;</td>
    </tr>
    <tr>
      <td><table width="90%" border="0" align="center"
bgcolor="#FFFFFF">
```

```
          <tr>
            <td><p align="center">`cRoomDescription`</p>
              <p align="center">`cRoomInventory`</p></td>
          </tr>
        </table></td>
    </tr>
    <tr>
      <td><center>`cMessage`
      </center></td>
    <tr>
    <tr>
      <td><center>
          <input name="Command" type="text" id="Command" size="90">
        </center></td>
    </tr>
    <tr>
      <td><center>
          <input type="submit" name="Submit" value="Submit">
        </center></td>
    </tr>
  </table>
</form>
<p>&nbsp;</p>
</body>
</html>
```

As you can see below, the program will tell the user where they currently are in the map of the game. The program will accept some natural language input in the input line. It is the sole method of interacting with the game.

Of course, this interface is a web interface. One could easily provide a GUI client or CHUI client interface for the application to receive input and provide data out of. (Perhaps other programmers would like to provide such a beast?)


Game - Mozilla
File Edit View Go Bookmarks Tools Window Help
http://amduus2/online/report/game/Game.html?SessionID=UV
Bookmarks Reference Sign Out Job Search
WebSpeed WebTools
Game
Submit
Done

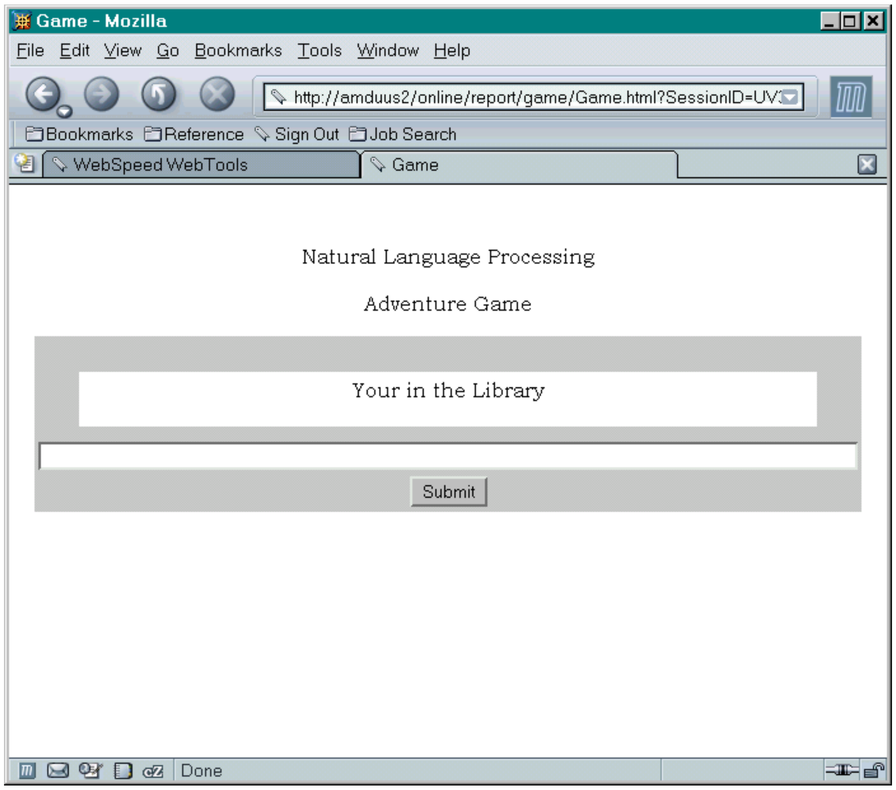
Natural Language Processing

Adventure Game

Your in the South Hall

go to the east

The program will then examine the user's input to determine what the user wants the program to do. In this case, it is travel to the east of the current location the user is located.


Game - Mozilla
File Edit View Go Bookmarks Tools Window Help
http://amduus2/online/report/game/Game.html?SessionID=UV
Bookmarks Reference Sign Out Job Search
WebSpeed WebTools
Game
Submit
Done

Natural Language Processing

Adventure Game

Your in the Library

Here you can see the program has moved the user into the room east of the original room.

ParseCommand.p

So how does this word stuff work? Below is the code that acts as a “verb dispatcher” so to speak. It’s purpose is to examine the input the user presented to the application and figure out what parts of the input are unimportant. Then it will figure out which program module to send the remaining words to in order for that verb to be actuated on.

```
DEF INPUT PARAMETER cSessionID AS CHARACTER NO-UNDO.

DEF VAR cCommand AS CHARACTER NO-UNDO.
DEF VAR iCurWord AS INTEGER NO-UNDO.
DEF VAR iMaxWord AS INTEGER NO-UNDO.

DEF VAR cVerb    AS CHARACTER NO-UNDO.
DEF VAR cError   AS CHARACTER NO-UNDO.

/* Figure out the verb routine that should work on the command */

DEF NEW SHARED TEMP-TABLE WordList
  FIELD Order AS INTEGER
  FIELD Word  AS CHARACTER
  FIELD WordType AS CHARACTER
  INDEX pukey IS PRIMARY Order.

/* First we break up the sentence presented to us by the user */
/* We remember the order for grammar later on.                */

FIND WebState NO-LOCK
WHERE WebState.SessionID = cSessionID
  AND WebState.Category = "FeedBack"
  AND WebState.Name = "Command"
NO-ERROR.

ASSIGN cCommand = WebState.Data.

DO iCurWord = 1 TO NUM-ENTRIES(cCommand, " "):

  CREATE WordList.

  ASSIGN
  WordList.Order = iCurWord
  WordList.Word  = ENTRY (iCurWord, cCommand, " ").

END.

/* Determine the word types for each word given.              */

FOR EACH WordList:

  FIND WordType NO-LOCK
  WHERE WordType.Word = WordList.Word
  NO-ERROR.

  IF NOT AVAILABLE WordType THEN
    ASSIGN WordList.WordType = "_NULL".
  ELSE
    ASSIGN WordList.WordType = WordType.WordType.

END. /* FOR EACH WordList */

/* Next we eliminate those words that are of little interest to our grammar */
/* parser, that is articles, prepositions, etc.                             */
/* _NULL is a special word for this system.  It should not be in the uni-   */
/* verse of vocabulary for the user.                                        */

ASSIGN cVerb = "_NULL".
FOR EACH WordList:

  IF WordList.WordType = "ARTICLE" THEN DELETE WordList.
  IF WordList.WordType = "_NULL" THEN DELETE WordList.
  IF WordList.WordType = "PREPOSITION" THEN DELETE WordList.
  IF WordList.WordType = "VERB" THEN ASSIGN cVerb = WordList.Word.

END. /* FOR EACH WordList */

/* Now determine the grammar construct that is available in the phrase and */
/* determine if it is something we can work with.  We do this by iden-     */
/* tifying the verb and it's associated routine to handle that "action."   */

FIND Verb NO-LOCK
WHERE Verb.Word = cVerb
NO-ERROR.

IF AVAILABLE Verb THEN
  RUN VALUE(Verb.Program) (INPUT cSessionID).
ELSE
  RUN WriteState.p (INPUT cSessionID,
                    INPUT "FeedBack",
                    INPUT "Message",
                    INPUT "I don't understand you",
                    OUTPUT cError).
```

If the program cannot figure out what the user is talking about, it will simply reply back "I don't understand you."

Now, as a note to people using this application or one's like it, users should be trained in the vocabulary of the application. If they are unaware of the words the program is aware of, they will grow frustrated with it's "stupidity" when they use words it is unfamiliar with. If the user's are knowledgeable of the words the program is familiar with, they will interact with the program much more smoothly.

In future issues of this e-zine, we will explore modifying this code to handle conjunctions – in particular the word "and." This way the user can say "Open the door and go east" or "Pick up the torch and the ring." (Hint, note the first example phrase is two verb-noun combinations. We just break them into two commands and execute their verbs programs sequentially. The second is a verb noun noun combination – one could rewrite it into "verb noun and verb noun" as in "pick up the torch and pick up the ring" – then apply the same algorithm of breaking up the command into two commands and executing sequentially. Perhaps someone out there will want to take a shot at this?)

Travel.p

Once ParseCommand.p has figured out that the verb involves moving the player around on the map, it will call the Travel.p program. This is because we as programmers specifically informed ParseCommand.p via the verb table that the occurrence of that word should send execution into this program (see Ancillary Code section below.)

```
/* Handle movement of the player around the map of the game world. */

DEF INPUT PARAMETER cSessionID AS CHARACTER NO-UNDO.

DEF VAR iCurrentRoom AS INTEGER NO-UNDO.
DEF VAR iNextRoom    AS INTEGER NO-UNDO.
DEF VAR cDirection   AS CHARACTER NO-UNDO.
DEF VAR cMessage     AS CHARACTER NO-UNDO.
DEF VAR cCommand     AS CHARACTER NO-UNDO.
DEF VAR cError       AS CHARACTER NO-UNDO.

/* Find the player's current room */

FIND WebState NO-LOCK
WHERE WebState.SessionID = cSessionID
  AND WebState.Category  = "Map"
  AND WebState.Name      = "CurrentRoom"
NO-ERROR.

ASSIGN iCurrentRoom = INT(WebState.Data).
```

```
/* Determine where the player desires to go */
/* Little fast and loose with the grammar here.  We simply look */
/* for keywords since we have passed muster with the grammer    */
/* check.  That rule is [travel verb] [direction]               */

FIND WebState NO-LOCK
WHERE WebState.SessionID = cSessionID
  AND WebState.Category  = "FeedBack"
  AND WebState.Name      = "Command"
NO-ERROR.

ASSIGN cCommand = WebState.Data.

IF INDEX (cCommand, "North") <> 0 THEN ASSIGN cDirection = "North".
ELSE IF INDEX (cCommand, "South") <> 0 THEN ASSIGN cDirection = "South".
ELSE IF INDEX (cCommand, "East") <> 0 THEN ASSIGN cDirection = "East".
ELSE IF INDEX (cCommand, "West") <> 0 THEN ASSIGN cDirection = "West".

/* Determine if the player is going someplace they can't go */
/* If so, then tell them so, else, set the current room to the next room */

FIND RoomMap NO-LOCK
WHERE RoomMap.RoomNumber = iCurrentRoom
NO-ERROR.

ASSIGN cMessage = ""
       iNextRoom = iCurrentRoom.

CASE cDirection:

  WHEN "North" THEN DO:

    IF RoomMap.North = 0 THEN
      ASSIGN cMessage = "You cannot move in that direction!".
    ELSE
      ASSIGN iNextRoom = RoomMap.North.

  END. /* WHEN "North" THEN */

  WHEN "South" THEN DO:

    IF RoomMap.South = 0 THEN
      ASSIGN cMessage = "You cannot move in that direction!".
    ELSE
      ASSIGN iNextRoom = RoomMap.South.

  END. /* WHEN "South" THEN */

  WHEN "East" THEN DO:

    IF RoomMap.East = 0 THEN
      ASSIGN cMessage = "You cannot move in that direction!".
    ELSE
      ASSIGN iNextRoom = RoomMap.East.

  END. /* WHEN "East" THEN */

  WHEN "West" THEN DO:

    IF RoomMap.West = 0 THEN
      ASSIGN cMessage = "You cannot move in that direction!".
    ELSE
```

```
      ASSIGN iNextRoom = RoomMap.West.

  END. /* WHEN "West" THEN */

END. /* CASE cDirection: */

/* If we changed rooms, then update the room state */

IF iCurrentRoom <> iNextRoom THEN
  RUN WriteState.p (INPUT cSessionID,
                    INPUT "Map",
                    INPUT "CurrentRoom",
                    INPUT STRING(iNextRoom),
                    OUTPUT cError).

/* Update the message state */

RUN WriteState.p (INPUT cSessionID,
                  INPUT "FeedBack",
                  INPUT "Message",
                  INPUT cMessage,
                  OUTPUT cError).
```

You will see it is pretty simple actually. Very little grammar is used to figure out where the user wants to go. This method tries to use the meanings of the words that appear in order to manipulate the data structures representing the world to the user. The code simply looks for specific expected words/nouns and then updates the data structures used by Game.html to determine which room information should appear on the screen – in effect moving the player into a different room.

Notice this code is unprepared for such commands as “Move south and then east.” (But, as spoken of above, ParseCommand.p probably will be ready for that as in future versions as it would take that expression and turn it into “move south and move east.”

Also note that it is not prepared for the user to say “Move to the library.” In order for such a thing to happen, one would need to uniquely name each room in the RoomMap, and then search for RoomMap based not on direction, but on a RoomMap.Name and then update PlayerCurrentRoom to that room number. Perhaps a future enhancement to the code by the readership?

Ancillary Code

Below is some ancillary code to the application. It basically assists the “game master” in prepping the game for use. We explore the scripts to use, the program listing and some sample data to help prep the game for use once it has been compiled and configured.

<u>setenv.ksh</u>

```
export DLC=/usr/dlc
export PROPATH=.
export PFARG="-pf /db/amduus/parm/amduus.pf"

export WORDTYPE=./wordtype.d
export VERBPRG=./verbprg.d
```

LoadData.ksh

```
. setenv.ksh

$DLC/bin/_progres -b -p LoadData.p $PFARG
```

LoadData.p

```
DEF VAR cCurrentFileName AS CHARACTER NO-UNDO.

/*********************************************************************/
/* Load word types                                                   */
/*********************************************************************/

ASSIGN cCurrentFileName = OS-GETENV ("WORDTYPE").

INPUT FROM VALUE(cCurrentFileName).

FOR EACH WordType EXCLUSIVE-LOCK:
  DELETE WordType.
END.

REPEAT:

  CREATE WordType.
  IMPORT WordType.

END.

INPUT CLOSE.

/*********************************************************************/
/* Load verb to program data                                         */
/*********************************************************************/

ASSIGN cCurrentFileName = OS-GETENV ("VERBPRG").

INPUT FROM VALUE(cCurrentFileName).

FOR EACH Verb EXCLUSIVE-LOCK:
  DELETE Verb.
END.

REPEAT:
  CREATE Verb.
  IMPORT Verb.

END.

INPUT CLOSE.
```

verbprg.d

```
"GO" "game/verb/Travel.p"
"TRAVEL" "game/verb/Travel.p"
"RUN" "game/verb/Travel.p"
"WALK" "game/verb/Travel.p"
```

wordtype.d

```
"GO" "VERB"
"TO" "PREPOSITION"
"THE" "ARTICLE"
"TRAVEL" "VERB"
"MOVE" "VERB"
"RUN" "VERB"
"WALK" "VERB"
"NORTH" "NOUN"
"SOUTH" "NOUN"
"EAST" "NOUN"
"WEST" "NOUN"
```

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: Simple application logging code

This code can be used on GUI, CHUI, and WWW clients. It should work on both UNIX and Windows operating systems, though testing on Windows operating systems has not been done.

StartLog.i

This is the starting point for the logging. It should be placed at the beginning of the program file so that Log.i can be used later on in the program. It should always precede Log.i use.

First it defines a shared stream. This stream will be used to write information to the log file. One can determine if it is a NEW stream or a shared stream for use in sub programs. It really doesn't matter if you repeatedly use NEW or not. If you do not use NEW, you should use "" as a place holder.

Next it uses a function called MyPID to look up the PID of the process currently processing the program. This is useful for matching execution flow in the log file with multiple programs writing to it.

Following, the routine looks up the parameter records that store the log level and log file location. Since you may want to have multiple logs for different programs, it offers the ability to programmatically define which parameter records should be used.

Amduus Information Works, Inc. also provides documentation services! One of the things I have noticed throughout my contracting career is that companies with developed software always seem to be missing or weak on user documentation, administration documentation, and programmer documentation. Amduus can help you with this!

Finally, it opens the log file and places the log level into the file. All other messages are placed into the log file via the Log.i include file.

```
DEF {1} SHARED STREAM LogStream.

DEF VAR cLogFile   AS CHARACTER NO-UNDO.
DEF VAR cLogLevel  AS CHARACTER NO-UNDO.
DEF VAR cMyPID     AS CHARACTER NO-UNDO.

/* Pretty much for UNIX only, assumes $PPID has parent PID */

FUNCTION MyPID RETURNS CHARACTER:

  DEF VAR cPID AS CHARACTER NO-UNDO.

  /* So windows won't blow up */

  IF OPSYS = "UNIX" THEN DO:

    INPUT THRU echo $PPID.
    IMPORT UNFORMATTED cPID.
    INPUT CLOSE.
  END.
  ELSE
    ASSIGN cPID = "Windows".

  RETURN cPID.

END.

ASSIGN cMyPID = MyPID().

FIND Parms NO-LOCK
WHERE Parms.Application = "{2}"
  AND Parms.GroupName   = "{3}"
  AND Parms.ParmName    = "LogLevel"
NO-ERROR.

ASSIGN cLogLevel = Parms.ParmValue.

FIND Parms NO-LOCK
WHERE Parms.Application = "{2}"
  AND Parms.GroupName   = "{3}"
  AND Parms.ParmName    = "LogFile"
NO-ERROR.

ASSIGN cLogFile = Parms.ParmValue.

IF "{1}" = "NEW" THEN
  OUTPUT STREAM LogStream TO VALUE(cLogFile) APPEND.

{Log.i 1 "'Log Level = ' cLogLevel"}
```

Log.i

This file will place a message into the log file defined in the parameters. There is an IF to determine if the log message should be written to the log file or not based on the log level. Generally the higher the log level, the more detailed the log file will be.

Information placed into the log include the date, the time, the PID of the process writing to the file (multiple processes may be writing to the file), the program name currently under execution, and finally the programmer’s message.

```
/* LogLevel is {1}, Message is {2} */

IF INT("{1}") <= INT(cLogLevel) THEN
  PUT STREAM LogStream UNFORMATTED TODAY " " STRING(TIME,"HH:MM:SS") "
" cMyPID " " PROGRAM-NAME(1) " " {2} "~n".
```

EndLog.i

This file simply closes the stream. As far as the coder is concerned, it is the the wrap up of the logging function within the application.

```
OUTPUT STREAM LogStream CLOSE.
```

Sample use of code

In this system, there are three log levels. Level 0 means no logging should be done. Level 1 is for informational logging. Finally level 2 is debugging logging – the most messages about operation of the program.

Simply the routine examines an email box for a specific kind of attachment in an email message and then extracts that attachment into a file. I won't give to much detail, it mostly is here to illustrate how to use the logging routines.

```
/*
 * Written by Scott Auge scott_auge@yahoo.com sauge@amduus.com
 * Copyright (c) 2001 Amduus Information Works, Inc.  www.amduus.com
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *      This product includes software developed by Amduus Information Works
 *      Inc. and its contributors.
 * 4. Neither the name of Amduus Information Works, Inc. nor the names of
 *    its contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY AMDUUS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AMDUUS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

/* Modified for court system.  Will pop out the latest cjde.xml file */
/* available in the mail server.  Connects to the given server, de-  */
/* termines the final peice of mail that is a CJDE file, pulls it    */
/* it down from the server, runs munpack on it to change from mime   */
/* to the file, places the file in the expected location.            */


{pop3api.i}
{filepath.i}

DEF VAR hServer   AS HANDLE NO-UNDO.
DEF VAR cMessage  AS CHARACTER NO-UNDO.
DEF VAR cFileName AS CHARACTER NO-UNDO.

DEF VAR cServerIP         AS CHARACTER NO-UNDO.
DEF VAR cUser             AS CHARACTER NO-UNDO.
DEF VAR cPassword         AS CHARACTER NO-UNDO.
DEF VAR cFinalFileName    AS CHARACTER NO-UNDO.
DEF VAR cMessageNumber    AS CHARACTER NO-UNDO.
DEF VAR cMessageList      AS CHARACTER NO-UNDO.
DEF VAR iCurMessageNumber AS INTEGER NO-UNDO.
DEF VAR iMaxMessageNumber AS INTEGER NO-UNDO.
DEF VAR lFoundIt          AS LOGICAL NO-UNDO.
DEF VAR cResult           AS CHARACTER NO-UNDO.

{StartLog.i NEW UJIS Misc}

{Log.i 2 "'Entering File:PopCJDE.p'"}

/* Determine the information available from parameters. */

/* Where we connect into */

FIND Parms NO-LOCK
WHERE Parms.Application = "MailService"
```

```
  AND Parms.GroupName   = "MailServer"
  AND Parms.ParmName    = "PopIP"
NO-ERROR.

ASSIGN cServerIP = Parms.ParmValue.

{Log.i 1 "'MailServer Is ' cServerIP"}

/* Who we are */

FIND Parms NO-LOCK
WHERE Parms.Application = "MailService"
  AND Parms.GroupName   = "MailServer"
  AND Parms.ParmName    = "PopLogin"
NO-ERROR.

ASSIGN cUser = Parms.ParmValue.

{Log.i 1 "'User Is ' cUser"}

/* What the password is */

FIND Parms NO-LOCK
WHERE Parms.Application = "MailService"
  AND Parms.GroupName   = "MailServer"
  AND Parms.ParmName    = "PopPassword"
NO-ERROR.

ASSIGN cPassword = Parms.ParmValue.

{Log.i 1 "'Password Is ' cPassword"}

/* Final destination of the file */

FIND Parms NO-LOCK
WHERE Parms.Application = "UJISMigration"
  AND Parms.GroupName   = "ReadCJDE"
  AND Parms.ParmName    = "FileName"
NO-ERROR.

ASSIGN cFinalFileName = Parms.ParmValue.

{Log.i 1 "'Destination Is ' cFinalFileName"}

/* Now we start processing on the server */

RUN ConnectToServer (INPUT cServerIP, OUTPUT hServer, OUTPUT cMessage).

{Log.i 2 "'ConnectToServer:' TRIM(cMessage)"}

RUN Login (INPUT hServer, INPUT cUser, INPUT cPassword, OUTPUT cMessage).
```

```
{Log.i 2 "'Login:' cMessage"}

/* We need to identify how many messages are on the server.  We cannot */
/* assume there is only one.  We will also need to look at the sub-    */
/* ject of each message cuz sooner or later it is gonna get spammed.   */

/* Find out our messages */

RUN ListMessages (INPUT hServer, OUTPUT cMessageList).

{Log.i 2 "'ListMessages:' TRIM(cMessage)"}

/* Pull out the final message and determine if it is a CJDE message */
/* We know this by looking for cjde.xml in the body of the message  */
/* (like the name of the mime attachment in the message.)           */

ASSIGN iMaxMessageNumber = NUM-ENTRIES(cMessageList)
       lFoundIt = NO.

{Log.i 2 "'Message Count:' iMaxMessageNumber"}

DO iCurMessageNumber = iMaxMessageNumber TO 1 BY -1:

  RUN RetrieveMessage (
    INPUT hServer,
    INPUT STRING(iCurMessageNumber),
    OUTPUT cFileName).

  {Log.i 2 "'RetrieveMessage:' iCurMessageNumber cFileName"}

  RUN TypeMailMsg.p (INPUT cFileName, OUTPUT cResult).

  IF cResult = "CJDEData" THEN DO:
    ASSIGN lFoundIt = YES.
    LEAVE.
  END.

  {Log.i 1 "'Deleting ' cFileName"}

  OS-DELETE VALUE(cFileName).

END. /* DO iCurMessageNumber = iMaxMessageNumber TO 1 BY -1 */


/* We are done with the server.  Perform housekeeping on the connection */
/* and bail.                                                            */

RUN Logout (INPUT hServer, OUTPUT cMessage).
{Log.i 2 "'Logout:' TRIM(cMessage)"}
RUN DisconnectFromServer (INPUT hServer).
```

```
{Log.i 2 "'DisconnectFromServer:' TRIM(cMessage)"}

/* De-mime the file.  Throw the results in the temporary directory */

IF lFoundIt THEN DO:

  {Log.i 1 "'Unpacking ' cFileName ' into ' SESSION:TEMP-DIRECTORY"}

  OS-COMMAND munpack -C VALUE(SESSION:TEMP-DIRECTORY) VALUE(cFileName) 1>
/dev/null 2>/dev/null.

  /* Move the file into the expected location according to parameter.  */

  {Log.i 1 "'Moving '  SESSION:TEMP-DIRECTORY 'cjde.xml' ' to '
cFinalFileName"}

  OS-COPY VALUE( SESSION:TEMP-DIRECTORY + "cjde.xml") VALUE(cFinalFileName).

  {Log.i 1 "'Deleting ' cFileName"}

  OS-DELETE VALUE(cFileName).

END. /* IF lFoundIt */

{Log.i 2 "'Leaving File:PopCJDE.p'"}

{EndLog.i}
```

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.

Amduus Information Works, Inc. assists in the publication of this document by providing an internet connection and web site for redistribution:

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

Other Progress Publications Available:

This document focuses on the programming of Progress applications. If you wish to read more business oriented articles about Progress, be sure to see the Profile's magazine put out by Progress software:

http://www.progress.com/profiles/

There are other documents/links available at www.peg.com.

Products Available From Amduus:

Amduus Information Works, Inc. is a Progress reseller and ASPen partner. We primarily develop UNIX/Linux based applications for the web. We also perform integration of Progress applications through such languages and tools as MQ Series, C, and C++.

Amduus provides support for the following applications: Blue Diamond, Denkh, Denkh HTML Reporter, Red Arrow Portal (CMS), Survey Express and other software.

Amduus is looking for consultants who might want to promote the use of our tools at user groups and companies they might work in. Send some information to sauge@amduus.com to let me know you are out there!

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.

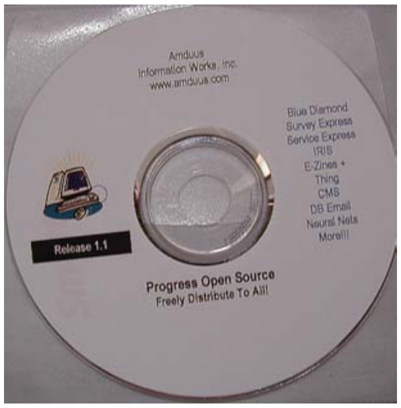
Looking for technical articles, *marketing Progress* articles, articles about books relevant to programming/software industry, white papers, etc.

Order Form for Progress Open Source CD-ROM

COUPON 001A

This is an offer for the CD-ROM at lower than list savings!
This is a great way to support the E-Zine too!

Mail this form to:
Amduus Information Works, Inc.
1818 Briarwood
Flint, MI 48507



Please send ______ copies of the Open Source CD-ROM at $25.00 per disk to:

Name ____
Company ____
Address ____
City ____
State ____________________ Country ____________________
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)
- Denkh HTML Reporter – web based report writer
- 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
- Denkh – create PDF file reports for Webspeed/UNIX CHUI!
- More!