The Progress Electronic Magazine

An Amduus™ Information Works, Inc. Publication

Table of Contents

Publisher's Statement

In this issue I continue with basic data structures in the 4GL. The last issue showed how to simply implement a stack. Some people wondered what such a thing might be useful for. I can tell you one right now – the collection of error messages and the order they were encountered! It would be a good place to store validation errors or the like.

The data structure in this issue is the queue. This is the First In – First Out data structure. It has a lot of useful behaviors for Operations Research and is one of the "fundamental" data structures.

I have put the code out for the stack, queue, and upcoming list objects at http://amduus.com/OpenSrc/SrcLib/OOP in list.zip, stack.zip and queue.zip.

Also included in this issue is a password validation and generation scheme. With Sarbanes-Oxley Act becoming more prominent in the business world, passwords are coming under more scrutiny. This object can help you validate a given password as matching a set of rules. It can also help one generate a password to match a set of rules too. It also is located in the above mentioned link under obj_passwd.zip.

Anyhow, lets get on with the fun!

Scott Auge

sauge at amduus dot com

Password Validation/Generation Object

By Scott Auge

With increased scrutiny on security via the Sarbane-Oxley Act/SAS70[1], developers are finding themselves needing to write algorithms to aid with more secure passwords for applications.

This is accomplished in the article with obj_passwd.p.

We validate on the following attributes of the password:

- Minimum Length
- Minimum Count of Number Characters
- Minimum Count of Alpha Characters
- Minimum Count of Punctuation Characters

The object will default these values upon it's initialization. Check out the routine Init to learn and update the values to your needs. One can also change the defaults by setting the attributes via the SetAttr procedure in the object.

We have the following possible errors on a given password:

- Not achieving the minimum length
- Not achieving the minimum number of letters
- Not achieving the minimum number of numbers
- Not achieving the minimum number of punctuation marks

The error codes can be found in the SetError procedure of obj_passwd.p.

One can also generate passwords with the object through the GeneratePassword procedure.

1 SAS70 is an audit procedure of service organizations. If you are outsourcing, you may want to explore this a little more. If you are providing services to another company, you may want to become familiar with it in case it is called for.

You can download the code from here:

http://amduus.com/OpenSrc/SrcLib/OOP/obj_passwd.zip

Here is an example calling into the object. In general, I use the object manager (objmgr.i). It basically lets me add and subtract persistent procedures and to give them names instead of handles. See a previous E-Zine for more information about this object.

From there, it is pretty simple – we offer up a password to ValidatePassword() and use GetError to determine if an error occurred during validation or not. We are always looking for error code 000 – if another comes up, then the password does not meet the standards as specified in the object.

tst_obj_passwd.p

```
/* Use the obj_password.p to validate a password and to generate a password */

{objmgr.i NEW}

DEFINE VARIABLE cPassWord AS CHARACTER NO-UNDO.
DEFINE VARIABLE cErrCode AS CHARACTER NO-UNDO.
DEFINE VARIABLE cErrMsg AS CHARACTER NO-UNDO.
DEFINE VARIABLE cErrMethod AS CHARACTER NO-UNDO.

OUTPUT TO /tmp/tst_obj_password.log.

RUN OMAdd("Test", "obj_passwd.p").

/* To short */

ASSIGN cPassword = "abc".
RUN ValidatePassword IN OMGH("Test") (cPassword).
RUN GetError IN OMGH("Test") (OUTPUT cErrCode, OUTPUT cErrMsg, OUTPUT
cErrMethod).
PUT UNFORMATTED "cPassword: " cPassword SKIP
                "cErrCode: " cErrCode SKIP
                "cErrMsg: " cErrMsg SKIP
                "cErrMethod: " cErrMethod SKIP(2).

/* Not enough numbers */
```

```
ASSIGN cPassword = "abcdefghi".
RUN ValidatePassword IN OMGH("Test") (cPassword).
RUN GetError IN OMGH("Test") (OUTPUT cErrCode, OUTPUT cErrMsg, OUTPUT
cErrMethod).
PUT UNFORMATTED "cPassword: " cPassword SKIP
                "cErrCode: " cErrCode SKIP
                "cErrMsg: " cErrMsg SKIP
                "cErrMethod: " cErrMethod SKIP(2).

/* Not enough letters */

ASSIGN cPassword = "01234567".
RUN ValidatePassword IN OMGH("Test") (cPassword).
RUN GetError IN OMGH("Test") (OUTPUT cErrCode, OUTPUT cErrMsg, OUTPUT
cErrMethod).
PUT UNFORMATTED "cPassword: " cPassword SKIP
                "cErrCode: " cErrCode SKIP
                "cErrMsg: " cErrMsg SKIP
                "cErrMethod: " cErrMethod SKIP(2).

/* Not ecnough punctuation */

ASSIGN cPassword = "a4s5d6f7ghi".
RUN ValidatePassword IN OMGH("Test") (cPassword).
RUN GetError IN OMGH("Test") (OUTPUT cErrCode, OUTPUT cErrMsg, OUTPUT
cErrMethod).
PUT UNFORMATTED "cPassword: " cPassword SKIP
                "cErrCode: " cErrCode SKIP
                "cErrMsg: " cErrMsg SKIP
                "cErrMethod: " cErrMethod SKIP(2).

/* Should be just fine */

ASSIGN cPassword = "a1-d5f6b7ef".
RUN ValidatePassword IN OMGH("Test") (cPassword).
RUN GetError IN OMGH("Test") (OUTPUT cErrCode, OUTPUT cErrMsg, OUTPUT
cErrMethod).
PUT UNFORMATTED "cPassword: " cPassword SKIP
                "cErrCode: " cErrCode SKIP
                "cErrMsg: " cErrMsg SKIP
```

```
                "cErrMethod: " cErrMethod SKIP(2).

/* Generate a password */

RUN GeneratePassword IN OMGH("Test") (OUTPUT cPassword).
PUT UNFORMATTED "Password From Argument: " cPassword SKIP.

RUN GetAttr IN OMGH("Test") ("Password", OUTPUT cPassword).
PUT UNFORMATTED "Password From Attribute: " cPassword SKIP.

RUN GetError IN OMGH("Test") (OUTPUT cErrCode, OUTPUT cErrMsg, OUTPUT
cErrMethod).
PUT UNFORMATTED "cErrCode: " cErrCode SKIP
                "cErrMsg: " cErrMsg SKIP
                "cErrMethod: " cErrMethod SKIP(2).

/* Generate a password again to make sure we don't get the same one again */

RUN GeneratePassword IN OMGH("Test") (OUTPUT cPassword).
PUT UNFORMATTED "Password From Argument: " cPassword SKIP.

RUN GetAttr IN OMGH("Test") ("Password", OUTPUT cPassword).
PUT UNFORMATTED "Password From Attribute: " cPassword SKIP.

RUN GetError IN OMGH("Test") (OUTPUT cErrCode, OUTPUT cErrMsg, OUTPUT
cErrMethod).
PUT UNFORMATTED "cErrCode: " cErrCode SKIP
                "cErrMsg: " cErrMsg SKIP
                "cErrMethod: " cErrMethod SKIP(2).



RUN OMDel ("Test").

OUTPUT CLOSE.
```

You can actually change the standards in the object by calling the SetAttr() procedure in the object. See the Init() procedure for the names and defaults for the attributes available in this object.

tst_obj_password.log

```
cPassword: abc
cErrCode: 100
cErrMsg: Minimum Length Not Achieved
cErrMethod: ValidatePassword


cPassword: abcdefghi
cErrCode: 102
cErrMsg: Minimum Numbers Not Achieved
cErrMethod: ValidatePassword


cPassword: 01234567
cErrCode: 101
cErrMsg: Minimum Letters Not Achieved
cErrMethod: ValidatePassword


cPassword: a4s5d6f7ghi
cErrCode: 103
cErrMsg: Minimum Punctuation Not Achieved
cErrMethod: ValidatePassword


cPassword: a1-d5f6b7ef
cErrCode: 000
cErrMsg: No Error
cErrMethod: ValidatePassword


Password From Argument: i&qpu5q5ne
Password From Attribute: i&qpu5q5ne
cErrCode: 000
cErrMsg: No Error
cErrMethod: GeneratePassword
```

```
Password From Argument: 09omdkrf'x
Password From Attribute: 09omdkrf'x
cErrCode: 000
cErrMsg: No Error
cErrMethod: GeneratePassword
```

Scott Auge is the founder of Amduus information Works. He has been working with Progress technologies since Version 6. He works with UNIX platforms dealing with integration and web based applications.

Donations

Do you find something useful in the E-Zine now and then? Do you think it holds value for your education in the 4GL and to learn what is out there?

It does take money to produce this electronic magazine. Sure I can cut corners by not needing paper, ink, or postage – but **bandwidth still costs x amount of dollars every month.**

Even with Linux server, hardware does cost money – and **the server is five years old. It's due for an upgrade.**

Yes, I use OpenOffice.org to edit the document on, but **throw in the laptop and that is a few more dollars.**

T**hrow in a developer's Progress license** – well... we all know how much that puppy is! It sure is useful for writing code with! I would like to get on version 10.

All of the above is going to cost about $10,000.00 this year. Bet you didn't know it costs so much for a free publication!

We all know there are not that many publications for Progress based technology. You can help yourself, and others, by donating to "the cause."

Your Name: _____

Organization: _____

___ $5.00 ___ $10.00 ___$20.00

Consider this address good until April 2005:

Scott Auge
222 East Riverside #302
Austin, TX 78704
United States of America

Thanks for your support!
Scott

Implementing A Queue With Persistent Procedures

By Scott Auge

When we go to school to learn computer science or mathematics, we often take that first Comp 101 course studying data structures. If you don't know about data structures, I recommend you hit amazon http://www.amazon.com/exec/obidos/ASIN/0262032937/qid=1106878827/sr=2-1/ref=pd_ka_b_2_1/104-9886269-2431926 for "Introduction to Algorithms" or of course, Volume I of "The Art Of Computer Programming."

A queue is pretty simple to understand – it works very much like a line of people at the bank or grocery store. Queues are also called FIFO which stands for "First In First Out."

This may seem pretty simple, and it is! It is also the base structure that more complex structures and algorithms one might use. Some more complex uses might be in operations research. For example – simulating the activity at your retail store to find the optimum number of lines to have open. Each line in the store is a queue and you can learn from your existing data average lengths of time for each customer and what number of customers are out there for given time periods. Or you can simulate the flow of parts around an assembly system in a factory – each work cell or work station would service a queue of parts coming into it.

This queue code is actually implemented in an object oriented way. We instance the object via the PERSISTENT SET hQueue grammar. If you want to have multiple queues one would RUN the object multiple times. It is also better to use the objmgr.i code found with the code to dynamically start up queues and "name" them.

Lets take a look at a program that runs the queue.

tst_obj_queue.p

```
/* Test queue without object manager code */

DEFINE VARIABLE hQueue AS HANDLE NO-UNDO.
DEFINE VARIABLE cValue AS CHARACTER NO-UNDO.

DEFINE VARIABLE cErrCode AS CHARACTER NO-UNDO.
DEFINE VARIABLE cErrMsg AS CHARACTER NO-UNDO.
DEFINE VARIABLE cErrMeth AS CHARACTER NO-UNDO.

OUTPUT TO /tmp/queue.log.

RUN obj_queue.p PERSISTENT SET hQueue.
```

```
RUN Init IN hQueue ("NoName").

RUN Enqueue IN hQueue ("1").
RUN Enqueue IN hQueue ("2").
RUN Enqueue IN hQueue ("3").
RUN Enqueue IN hQueue ("4").

RUN DumpQueue IN hQueue ("/tmp/sauge").

RUN Dequeue IN hQueue (OUTPUT cValue).
MESSAGE cValue.
PAUSE.

RUN Dequeue IN hQueue (OUTPUT cValue).
MESSAGE cValue.
PAUSE.

RUN Dequeue IN hQueue (OUTPUT cValue).
MESSAGE cValue.
PAUSE.

RUN Dequeue IN hQueue (OUTPUT cValue).
MESSAGE cValue.
PAUSE.

/* Do an extra one to test errors */

RUN Dequeue IN hQueue (OUTPUT cValue).
MESSAGE cValue.
RUN GetError IN hQueue (OUTPUT cErrCode, OUTPUT cErrMsg, OUTPUT
cErrMeth).
MESSAGE cErrCode cErrMsg cErrMeth.

/* Clean up */

DELETE PROCEDURE hQueue.

OUTPUT CLOSE.
```

First we define some variables that aid us with any errors the queue needs to inform us of. There are few errors associated with a queue, so it is easy to determine what kinds of errors one might encounter. We do that running the GetError routine made available by the base object template.

Next we run the object creating an instance in memory accessible via the hQueue handle. Remember the data storage for each instance of the object is separate from any other's. So if you instance two or more objects, they all need unique names between them and any data you Enqueue or Dequeue from one instance has no effect on the data in the other instances. Think of them as two lines at the grocery store.

Once we have an object available to us, we start running the Enqueue procedure available in the object to add our data into it. Notice we identify which object we wish to manipulate with the `IN hQueue` phrase of the statement. We enqueue the following data in the following order: 1 2 3 4.

Remember that a queue is First In First Out. So unlike the stack we talked about in issue 40, the first item out will be 1 as that was the first item put into the queue. The act of dequeuing an item also removes the item from data storage. This makes the last – 1 item(s) available for dequeuing. You can see the results in queue.log below:

queue.log

```
[~/code/progress/objects]$ cat /
tmp/queue.log
1
2
3
4
?
100 Dequeue on empty queue Dequeue
```

You can see that some of our return values from the stack are the unknown denoted by the "?" symbol. The program is designed to return a ? when there is nothing more to dequeue off the queue.

One may think this is good programming style – but it is not. To truly set up an error detection system, one should have code specifically for that. After all, what if one of the values you wanted to enqueue and dequeue off the queue was an unknown? Then you would not know if the queue was empty or if the valid value came off of it.

To aid in determining if something went wrong with the actions on the queue, one can check the GetError routine made available by the base object template. We have an error code of 100 with an error message "Dequeue on empty Queue" to aid the programmer with error conditions that have happened during an operation.

In our implementation of a queue, we use the template for an object available at http://amduus.com/4glwiki/index.php?pagename=Template%20for%20object%20oriented%20programming as our base code.

We then added the following routines:

Enqueue – allows an implementor to put a value onto the queue.

Dequeue – allows an implementor to remove a value from the queue

IsEmpty – allows the implementor to query the object if it is storing anything

Count – allows the implementor to query the object for how many items remain on the stack.

If you read the code below, you will also find some additional routines that could be helpful.

We also added the following possible error conditions that might occur:

```
100 – Dequeue on empty Queue
```

Because really this is the only error that could happen on a queue.

There is also the base object's error code 001 which states GetAttr was called for an attribute that wasn't available. This is related to the object and not really to the queue code.

Below is the code for implementing the queue[2]. It really is quite simple – we use a temp table to store our entries and the Enqueue and Dequeue routines to put and remove entries from the table. This is where a database 4GL language really shines as in 3GL languages one usually needs to do memory manipulations. In the 4GL they become table manipulations.

obj_queue.p

```
/*********************************************************************/
/* This program is meant to run as an object.                        */
/* It is a simple program to demonstrate:                            */
/*   -- "Private" attributes available to it                         */
/*   -- "Public" methods available to other programs.                */
/*   -- A good template for future objects with Constructor and      */
/*      Destructor routines.                                         */
/*********************************************************************/

FUNCTION cGetAttr RETURNS CHARACTER (INPUT cAttrName AS CHARACTER) FORWARD.

/* ----------------------- Begin Attributes List ----------------------*/
```

2Remember this code is download-able also, so you do not need to cut and paste from this document.

```
/* Example attribute specific to the object */

/* Used for object management */
DEFINE VARIABLE cgObjName          as character NO-UNDO.


/* ------------------------- End Attributes List ----------------------*/

/* -------------------------- Begin Methods List ----------------------*/

/*************************************************************************/
/* This is the "destructor" for the routine.  It should be called be   */
/* fore deleting the handle to the instance of this object.            */
/*************************************************************************/

PROCEDURE Destroy:

END.

/*************************************************************************/
/* If other "constructors" are needed, they can be put in.  This one   */
/* is called Init.  Unlike C++, it will need to be run manually.       */
/* If you are using ObjMgr.i, it will be run automatically.            */
/* If you need to make other Inits, place them here and name beginning */
/* "Init" : InitByRowID or InitBySalesOrderNumber.                     */
/*************************************************************************/

PROCEDURE Init:

  DEFINE INPUT PARAMETER cName AS CHARACTER NO-UNDO.

  ASSIGN cgObjName = cName.

  RUN SetError IN THIS-PROCEDURE ("000", "Init").

  RUN ClearQueue IN THIS-PROCEDURE.

END.

/*************************************************************************/
/* All procedures need a way to describe their errors back to the cal- */
/* ler.                                                                */
/*************************************************************************/

PROCEDURE GetError:

  DEFINE OUTPUT PARAMETER cErrCode AS CHARACTER NO-UNDO.
  DEFINE OUTPUT PARAMETER cErrMsg  AS CHARACTER NO-UNDO.
  DEFINE OUTPUT PARAMETER cErrMethod AS CHARACTER NO-UNDO.

  ASSIGN
  cErrCode = cGetAttr("ErrCode")
  cErrMsg  = cGetAttr("ErrMsg")
  cErrMethod = cGetAttr("ErrMethod").
```

```
END. /* PROCEDURE GetError */

/*********************************************************************/
/* This is the central point where internal procedures can communicate */
/* their problems to the procedure as a whole.                       */
/* It's main purpose is to convert codes into human readable form in   */
/* cgObjErrMsg as well as populate the procedures globally available   */
/* vars.                                                             */
/*********************************************************************/

PROCEDURE SetError:

  DEFINE INPUT PARAMETER cErrorCode AS CHARACTER NO-UNDO.
  DEFINE INPUT PARAMETER cMethodName AS CHARACTER NO-UNDO.

  CASE cErrorCode:

    WHEN "000" THEN DO:
      RUN SetAttr IN THIS-PROCEDURE ("ErrCode", cErrorCode).
      RUN SetAttr IN THIS-PROCEDURE ("ErrMsg", "No Error").
      RUN SetAttr IN THIS-PROCEDURE ("ErrMethod", cMethodName).
    END.

    WHEN "100" THEN DO:
      RUN SetAttr IN THIS-PROCEDURE ("ErrCode", cErrorCode).
      RUN SetAttr IN THIS-PROCEDURE ("ErrMsg", "Dequeue on empty queue").
      RUN SetAttr IN THIS-PROCEDURE ("ErrMethod", cMethodName).
    END.

    OTHERWISE DO:
      RUN SetAttr IN THIS-PROCEDURE ("ErrCode", cErrorCode).
      RUN SetAttr IN THIS-PROCEDURE ("ErrMsg", ?).
      RUN SetAttr IN THIS-PROCEDURE ("ErrMethod", cMethodName).
    END.

  END. /* CASE */

END. /* PROCEDURE SetError */

/*********************************************************************/
/* These are methods to perform activities on the data controlled by   */
/* the object.                                                       */
/* Attributes are actually stored in a temp-table, that way we can add */
/* new ones easily, and not need to create more Set/Get routines for   */
/* for each attribute.  The bad news is, we need to cast to and from   */
/* character for the data to pass back and fourth.  If this is a pron- */
/* lem, then create some Set/Get routines for those values.          */
/*********************************************************************/

DEFINE TEMP-TABLE ttAttributes
  FIELD AttrName AS CHARACTER
  FIELD AttrValue AS CHARACTER
  INDEX key1 AttrName ASCENDING.

/*********************************************************************/
```

```
/* We don't make this a function, because we need to FORWARD declare   */
/* in the code using this object, and this name is going to be pret-   */
/* ty popular causing conflict with other objects with a GetAttr.      */
/*************************************************************************/

PROCEDURE GetAttr:

  DEFINE INPUT PARAMETER cName   AS CHARACTER NO-UNDO.
  DEFINE OUTPUT PARAMETER cValue AS CHARACTER NO-UNDO.

  FIND ttAttributes NO-LOCK
  WHERE ttAttributes.AttrName = cName
  NO-ERROR.

  IF NOT AVAILABLE ttAttributes THEN DO:

    /*******************************************************/
    /* Note that sometimes it is OK for an attribute to be ? */
    /* so be sure to remember to check the error if the at-  */
    /* tribute wasn't found or really is ?.  Sometimes it    */
    /* it works out it does not matter, sometimes it does    */
    /* matter.                                               */
    /*******************************************************/

    RUN SetError IN THIS-PROCEDURE ("001", "GetAttr").
    ASSIGN cValue = ?.
    RETURN.

  END. /* IF NOT AVAILABLE ttAttributes */

  ASSIGN cValue = ttAttributes.AttrValue.

END.  /* PROCEDURE GetAttr */

/*************************************************************************/
/* However it IS useful to have a GetAttr function for use in THIS-    */
/* PROCEDURE within the internal procedures available.                 */
/*************************************************************************/

FUNCTION cGetAttr RETURNS CHARACTER (INPUT cAttrName AS CHARACTER):

  DEFINE VARIABLE cAttrValue AS CHARACTER NO-UNDO.

  RUN GetAttr IN THIS-PROCEDURE (INPUT cAttrName, OUTPUT cAttrValue).

  RETURN cAttrValue.

END. /* FUNCTION cGetAttr() */

/*************************************************************************/
/* If the attribute already exists, we overwrite, not error out...     */
/*************************************************************************/

PROCEDURE SetAttr:
```

```
  DEFINE INPUT PARAMETER cName   AS CHARACTER NO-UNDO.
  DEFINE INPUT PARAMETER cValue AS CHARACTER NO-UNDO.

  FIND ttAttributes NO-LOCK
  WHERE ttAttributes.AttrName = cName
  NO-ERROR.

  IF NOT AVAILABLE ttAttributes THEN CREATE ttAttributes.

  ASSIGN ttAttributes.AttrName = cName
         ttAttributes.AttrValue = cValue.

  /* Some attributes are dependent on other attributes, handle those */
  /* here.                                                           */

  RUN DependentAttr (cName, cValue).

END. /* PROCEDURE SetAttr */

/*********************************************************************/
/* Some attributes are dependent on the values of other attributes.  */
/* We keep them in sync with this code here.                         */
/*********************************************************************/

PROCEDURE DependentAttr:

  DEFINE INPUT PARAMETER cName   AS CHARACTER NO-UNDO.
  DEFINE INPUT PARAMETER cValue AS CHARACTER NO-UNDO.

END. /* PROCEDURE DependentAttr */

/*********************************************************************/
/* This is a way to quickly transfer record information into attri-  */
/* butes.  The attribute name is tablename_fieldname.                */
/* Example Use:                                                      */
/* FIND FIRST Person NO-LOCK.                                        */
/* ASSIGN hBuffer = BUFFER Person:Handle.                            */
/* RUN Record2Attr (hBuffer).                                        */
/*********************************************************************/

PROCEDURE Record2Attr:

  DEFINE INPUT PARAMETER hBuffer AS HANDLE NO-UNDO.

  DEFINE VARIABLE hField     AS HANDLE NO-UNDO.
  DEFINE VARIABLE i          AS INTEGER NO-UNDO.

  DO i = 1 TO hBuffer:Num-Fields:

    ASSIGN hField = hBuffer:Buffer-Field(i).

    RUN SetAttr IN THIS-PROCEDURE (hBuffer:Name + "_" + hField:Name,
hField:String-Value).

  END.
```

```
END.

/*********************************************************************/
/* Scan the attributes table for entries beginning with the table name */
/* and apply their values to the field named in the second part of the */
/* attribute name.                                                   */
/* Note this doesn't manage multiple buffers of the same name very     */
/* well.  If you need n records from the same table, name the buffers  */
/* seperately.                                                       */
/*********************************************************************/

PROCEDURE Attr2Record:

  DEFINE INPUT PARAMETER hBuffer AS HANDLE NO-UNDO.

  DEFINE VARIABLE hField     AS HANDLE NO-UNDO.
  DEFINE VARIABLE cFieldName AS CHARACTER NO-UNDO.

  FOR EACH ttAttributes NO-LOCK
  WHERE ttAttributes.AttrName BEGINS hBuffer:Name + "_":

    /*************************************************************/
    /* Luckily, it appears that buffer-value types automatically...  */
    /* It will puke on bad data sent - such as text for an int, etc. */
    /*************************************************************/

    ASSIGN hfield = hBuffer:Buffer-Field(ENTRY(2, ttAttributes.AttrName, "_"))
           hfield:Buffer-Value = ttAttributes.AttrValue.

  END. /* FOR EACH ttAttributes */

END. /* PROCEDURE Attr2Record */


/*********************************************************************/
/* Useful for debugging. Note we output to a file so we don't get any  */
/* wrong display device type errors when the r-code is shared between  */
/* different interfaces.                                             */
/*********************************************************************/

PROCEDURE AttrDebug:

  DEFINE INPUT PARAMETER cFileName AS CHARACTER NO-UNDO.

  OUTPUT TO VALUE(cFileName).

  FOR EACH ttAttributes NO-LOCK:

    PUT UNFORMATTED ttAttributes.AttrName "=" ttAttributes.AttrValue SKIP.

  END. /* FOR EACH ttAttributes */

  OUTPUT CLOSE.
```

```
END. /* PROCEDURE AttrDebug */

/*************************************************************************/
/* Clear out all attributes that have been set.  It is better to set   */
/* an attribute to ? and code for that; but sometimes one needs this.  */
/*************************************************************************/

PROCEDURE ClearAttr:

  FOR EACH ttAttributes:

    DELETE ttAttributes.

  END. /* FOR EACH ttAttributes */

END.  /* PROCEDURE ClearAttr */

/* ---------------------- Object Specific ----------------------- */

DEFINE TEMP-TABLE Queue
  FIELD Order AS INTEGER
  FIELD Data AS CHARACTER
  INDEX pukey Order ASCENDING.

PROCEDURE Enqueue:

  DEFINE INPUT PARAMETER cValue AS CHARACTER NO-UNDO.

  DEFINE VARIABLE iLastOrder AS INTEGER NO-UNDO.

  DEFINE BUFFER Cur_Queue FOR Queue.

  FIND LAST Queue NO-LOCK NO-ERROR.

  IF AVAILABLE Queue THEN ASSIGN iLastOrder = Queue.Order + 1.
  ELSE ASSIGN iLastOrder = 1.

  CREATE Cur_Queue.

  ASSIGN Cur_Queue.Order = iLastOrder
         Cur_Queue.Data = cValue.

END. /* PROCEDURE Enqueue */

PROCEDURE Dequeue:

  DEFINE OUTPUT PARAMETER cValue AS CHARACTER NO-UNDO.

  DEFINE BUFFER Cur_Queue FOR Queue.

  FIND FIRST Cur_Queue EXCLUSIVE-LOCK NO-ERROR.

  IF AVAILABLE Cur_Queue THEN DO:
    ASSIGN cValue = Cur_Queue.Data.
    DELETE Cur_Queue.
```

```
  END.
  ELSE DO:
    ASSIGN cValue = ?.
    RUN SetError IN THIS-PROCEDURE ("100", "Dequeue").
  END.

END. /* PROCEDURE Dequeue */

PROCEDURE Count:

  DEFINE OUTPUT PARAMETER iCount AS INTEGER NO-UNDO.

  DEFINE BUFFER Cur_Queue FOR Queue.

  ASSIGN iCount = 0.

  FOR EACH Cur_Queue NO-LOCK:

    ASSIGN iCount = iCount + 1.

  END.

END. /* PROCEDURE Count */

PROCEDURE IsEmpty:

  DEFINE OUTPUT PARAMETER lIsEmpty AS LOGICAL NO-UNDO.

  DEFINE BUFFER Cur_Queue FOR Queue.

  FIND FIRST Cur_Queue NO-LOCK NO-ERROR.

  lIsEmpty = AVAILABLE Cur_Queue.

END. /* PROCEDURE IsEmpty */

PROCEDURE DumpQueue:

  DEFINE INPUT PARAMETER cFileName AS CHARACTER NO-UNDO.

  OUTPUT TO VALUE (cFileName).

  FOR EACH Queue NO-LOCK:

    EXPORT Queue.

  END.

  OUTPUT CLOSE.

END. /* PROCEDURE DumpQueue */

PROCEDURE ClearQueue:

  FOR EACH Queue EXCLUSIVE-LOCK:
```

```
    DELETE Queue.
  END.

END. /* PROCEDURE ClearQueue */

/* -------------------------- End Methods List ------------------------*/
```

Scott Auge is the founder of Amduus information Works. He has been working with Progress technologies since Version 6. He works with UNIX platforms dealing with integration and web based applications.

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 http://www.peg.com .

There is a web ring of sites associated with Progress programming and consultants available at http://i.webring.com/hub?ring=prodev&id=38&hub .

White Star Software publishes a commercial document called "Progressions." It is simular to this document but with different content. More information can be found at http://wss.com/. White Star also publishes Progress Programming books!

Article Submission Information:

Please submit your article in OpenOffice[3] 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.

3 OpenOffice is a freely available Office Suite for Windows, Apple, and *NIX based operating systems. You can download it at http://openoffice.org. This document is edited on OpenOffice.

Send your articles to sauge@amduus.com! Thanks!