



*The Progress Electronic Magazine*  
An Amduus™ Information Works, Inc. Publication

---

*This document may be freely shared with others without modification. Subscribe for free here:*  
<http://www.amduus.com/online/dev/ezine/EZineHome.html>

*You can find an archive of these E-Zines here:* <http://amduus.com/OpenSrc/FreePublications/>

## Table of Contents

Publish And Subscribe.....	4
What doesn't work!.....	7
More to play with!.....	9
Implementing A Stack With Persistent Procedures.....	10
Publishing Information:.....	22
Other Progress Publications Available:.....	22
Article Submission Information:.....	22

© 2005 Scott Auge, Amduus™ Information Works, Inc., and contributors.

*The information contained in this document represents the current view of the community or Amduus on the issues discussed as of the date of publication. Because the community or Amduus must respond to changing market and technological conditions, it should not be interpreted to be a commitment on the part of the community or Amduus, and the community or Amduus cannot guarantee the accuracy of any information presented. This paper is for informational purposes only. The community or Amduus MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT. Product and company names mentioned herein may be the trademarks of their respective owners.*



## Publisher's Statement

I have been hearing about this “publish” and “subscribe” feature of the 4GL so I figured it was time to figure out what is going on with that!

After learning some about this, I think this feature of the language may be one of THE most important changes yet. Imagine having other portions of your application become aware of something happening in part of the program without a RUN statement. Imagine never having to call other routines to update ancillary data structures or perform ancillary procedures. One simply issues an “event” and the other parts of the application catch that event and do their thing too.

Also included in this issue is part one of basic data structures with the 4GL. I will show how to implement a stack in an object oriented fashion with persistent procedures. This will hopefully aid with learning the use of persistent procedures and how they can be used as objects. For example, with the code available, you can create two or more instances of a persistent procedure and in effect have  $n$  stacks completely separate from each other.

Following issues of the E-Zine will include code for queues and a list object. This list object is so powerful you might never need the use of ENTRY() ever again!

Some readers have also noticed that the E-Zine is becoming less frequent. There are two reasons for that. One is that I have been moving more and more into the PHP/MySQL/PostgreSQL world for Amduus. Reason two is, since I have become a W2 employee of a company, my free time to income ratio is just a tad bit different. As a contractor I use to be able to work less time for more money – leaving extra time for the E-Zine and learning new programming knowledge with Progress.

That said – you all out there can feel free to send some articles into the E-Zine too! Others have done it!

Anyhow, lets get on with the fun!

Scott Auge

sauge at amduus dot com.

## Publish And Subscribe

By Scott Auge

Having a little bit of free time and access to John Sadds excellent books available here: <http://www.progress.com/products/documentation/index.ssp> I picked up on the ideas of “publish and subscribe.” Often we hear how there are no books at the local Barnes & Noble for learning Progress development<sup>1</sup>. Progress has heard our cry and has made a slew of books available on Progress 4GL development.



 <p><b>XCheck</b> Logfile analyzing system checking</p>	 <p><b>Viper</b> Visual Printing and Enhanced Reports</p>	 <p><b>PCase</b> CASE-Extension for the Data Dictionary</p>	 <p><b>QComp</b> Project management compiling, analyzing</p>
--	--	---	---

<ul style="list-style-type: none"> <li>● usable with Windows or UNIX</li> <li>● check activity of NS/DB/WS/http</li> <li>● analyze logfiles of NS/DB/WS</li> <li>● check drive space, space in DB</li> <li>● execute self defined scripts</li> <li>● analyze self defined logfiles</li> <li>● get notified by e-mail, http</li> <li>● or screen output</li> </ul>	<ul style="list-style-type: none"> <li>● uses Windows printer drivers</li> <li>● data processing with 4GL</li> <li>● incl. layout designer (VFD)</li> <li>● stores layouts DB or file based</li> <li>● no runtime licence cost</li> <li>● supports bmp/jpg/wmf images</li> <li>● embedding rtf-texts (font,...)</li> <li>● generates xml output (xslfo)</li> <li>● generates pdf-files (email)</li> <li>● supports WebSpeed /-Client</li> </ul>	<ul style="list-style-type: none"> <li>● view Progress-DB structures</li> <li>● create/update DBs directly</li> <li>● reengineer Progress-DBs</li> <li>● read/write Progress df-files</li> <li>● compare/maintain versions</li> <li>● incl. DB Content Viewer</li> <li>● incl. Open Report Interface</li> <li>● autogenerates references</li> <li>● print resizable ER-Diagrams</li> <li>● report-, structure- or ERD view</li> </ul>	<ul style="list-style-type: none"> <li>● compiles project file lists</li> <li>● includes compiler server</li> <li>● also compiles in char-mode</li> <li>● uses different Progress vers.</li> <li>● compiles for different OS</li> <li>● contains xref-analyze frontend</li> <li>● shows db structure &amp; content</li> <li>● keeps track of project errors</li> </ul>
---	---	---	--



**IAP GmbH**  
Moerkenstrasse 9 • D-22767 Hamburg • Germany  
Tel. +49 40 30 68 03 - 0 • Fax +49 40 30 68 03 - 10  
email: [info@tools4progress.com](mailto:info@tools4progress.com)



Information and free testversions at [www.tools4progress.com](http://www.tools4progress.com)

But back to subject... publish and subscribe (pub/sub) is an event throwing and catching mechanism. It is similar in idea to catching a menu click or a button click – only it allows far more flexibility.

That flexibility is, you are allowed to name and define your own events, and the catcher code for that event can be spread out amongst multiple procedures.

As an example, lets say you are working on a sales oriented program. When a sale is made, you need to do somethings like remove items from inventory, perhaps make a

<sup>1</sup>This is one of the reasons why I started the E-Zine – as a means to learn from others and others to learn what I have.

purchase order for items back ordered, create a pull slip for the warehouse, and run a credit card routine to bill their account.

In the past, a developer would have to tie all those pieces together with a RUN statement and know what things need to be run. With an event, things can be more “loosely coupled” as Progress will know what to do.

This code below is the skeleton of throwing an event described above. Of course, we are not going to try and implement all this functionality – just enough to show the convenience of using events to make it happen.

```
RUN sub_inv.p PERSISTENT.      /* Events for inventory management */
RUN sub_warehouse.p PERSISTENT. /* Events for warehouse */
RUN sub_po.p PERSISTENT.      /* Events for purchase orders */
RUN sub_cc.p PERSISTENT.      /* Events for credit card */

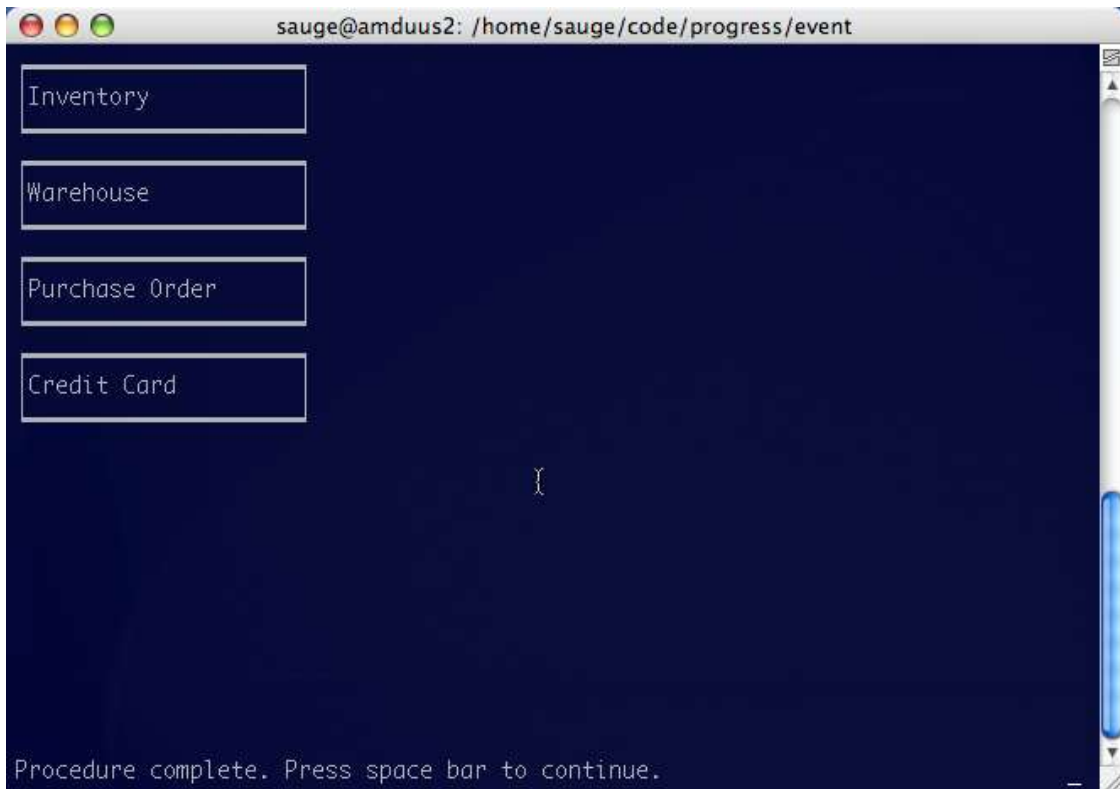
/* ON FINAL SALE BUTTON CLICKED IN SOME WINDOW */
PUBLISH "SaleMade".
```

What we aim to do is show how catching a “click” on some “final sale” button could be implemented to do all these things. An event called “SaleMade” is thrown and we aim to see that event captured in the sub\_\*.p programs listed above.

Let's see what happens when we run that program<sup>2</sup>:

---

<sup>2</sup> This window may look a little funky, because I use an Apple OS X machine to ssh into a Linux computer that actually is running Progress.



```
sauge@amduus2: /home/sauge/code/progress/event
Inventory
Warehouse
Purchase Order
Credit Card
}

Procedure complete. Press space bar to continue.
```

Wow! Obviously some stuff happened when we threw that event called “SaleMade” into the works. Let's see what the programs looked like that made that happen:

sub\_cc.p

```
SUBSCRIBE TO "SaleMade" ANYWHERE RUN-PROCEDURE "a" NO-ERROR.

PROCEDURE A:
  disp "Credit Card" FORMAT "x(20)".
END.
```

sub\_inv.p

```
SUBSCRIBE TO "SaleMade" ANYWHERE RUN-PROCEDURE "a" NO-ERROR.

PROCEDURE A:
  disp "Inventory" FORMAT "x(20)".
```

```
END.
```

sub\_po.p

```
SUBSCRIBE TO "SaleMade" ANYWHERE RUN-PROCEDURE "a" NO-ERROR.  
  
PROCEDURE A:  
    disp "Inventory" FORMAT "x(20)".  
END.
```

sub\_warehouse.p

```
SUBSCRIBE TO "SaleMade" ANYWHERE RUN-PROCEDURE "a" NO-ERROR.  
  
PROCEDURE A:  
    disp "Warehouse" FORMAT "x(20)".  
END.
```

You might notice I used the same routine name, "A," to catch the events with in my SUBSCRIBE statement. You do not need to match the same routine name in all the SUBSCRIBE statements.

### ***What doesn't work!***

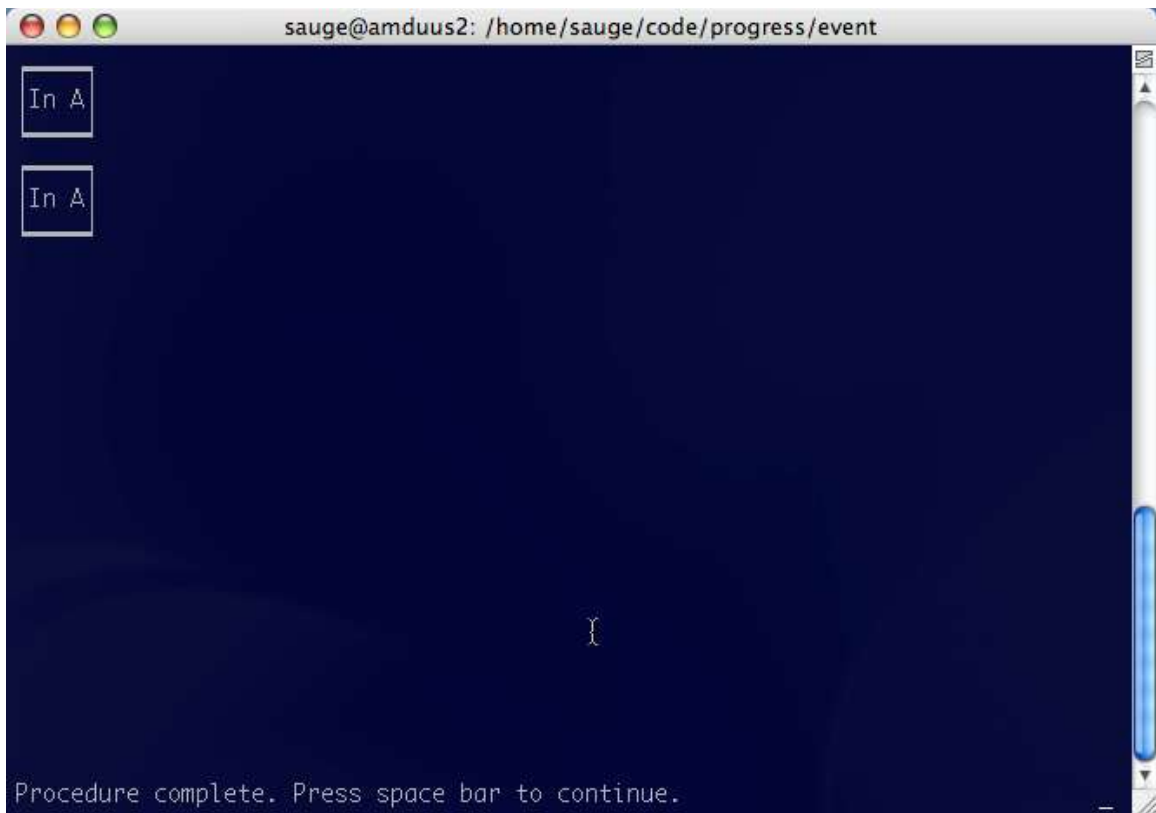
I tried this program:

```
DEFINE VARIABLE hSub AS HANDLE NO-UNDO.  
DEFINE VARIABLE hSub1 AS HANDLE NO-UNDO.  
  
RUN subscribers.p PERSISTENT SET hSub.  
RUN subscribers.p PERSISTENT SET hSub1.  
  
PUBLISH "Event_A".  
  
DELETE PROCEDURE hSub.  
DELETE PROCEDURE hSub1.
```

Where subscribers.p reads as:

```
SUBSCRIBE TO "Event_A" ANYWHERE RUN-PROCEDURE "a" NO-ERROR.  
SUBSCRIBE TO "Event_A" ANYWHERE RUN-PROCEDURE "b" NO-ERROR.  
  
PROCEDURE A:  
  disp "In A" with frame a.  
END.  
  
PROCEDURE B:  
  disp "In B" with frame b.  
END.
```

And received this:



```
saug@amduus2: /home/saug/code/progress/event  
In A  
In A  
|  
Procedure complete. Press space bar to continue.
```



We got two “In A” messages because the persistent procedure subscriber.p was instanced twice. So each instance will catch the same type of event.

As you can see, we didn't get an “In B” message – so it appears, once a persistent procedure has caught an event and executed it, it doesn't use the other SUBSCRIBE statement at all to reach PROCEDURE B, even though no compile error is thrown.

Just a little bit of behavior to be aware of.

### ***More to play with!***

The PUBLISH statement can even include parameters! So the example code above might be adapted to include an INPUT parameter containing the sales order number – or some means to explain to the catching routines.

The SUBSCRIBE statement can actually be limited to a particular handle of a procedure too. This way if an event is thrown in one procedure, you can prevent it from being caught from other procedures. If you need to do this, I would recommend looking at sorting out the multiple events into different names.

Also, you do not need to put the SUBSCRIBE statements in a different persistent procedure – they can be used in the same source file as the PUBLISH statement. I just would think that would turn into a big mess so I recommend better modularity than what that would allow you.

John Sadd's freely available book mentioned in the link above has more example code showing the use of events to make other things happen in the same window form for input. It shows it's use in a way that is more microcosm to a window than with the idea of a “business event” as I have shown in this article.

*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.*

### *Advertisement*

Service Express is golden and ready for use. Below find Service Express configured for an apartment management system, though it is flexible enough to be used by help desks in nearly any kind of industry for smaller businesses.

#### **SAVE MONEY! SAVE STRESS!**

- Allow your external customers to manage and create their tickets.



- Internal users manage all tickets.

- Web based – use Internet Explorer, Mozilla, Safari, or Opera.

- Easy to use, easy to understand.

- Configurable statuses (workflow)

- Configurable priorities

- Configurable HTML areas for your look and feel.

- For more information contact Scott Auge at [sauge@amduus.com](mailto:sauge@amduus.com) or see



Service Express is available in three ways:

1. As an Application Service from Amduus Information Works, Inc.
2. As a leased application on an Amduus provided machine.
3. As a licensed program for use on your machine.

## Implementing A Stack 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](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 stack is pretty simple to understand – one pushes values on top of each other, much like putting words on pieces of paper and then putting the papers on top of each other. This act of putting papers down on top of each other is called "pushing". Each paper represents an entry in the stack. Then to remove the papers, one takes off the top paper

first and works their way down. This action is called “popping.” Stacks are also called LIFO which stands for “Last In First Out.”

This may seem pretty simple, and it is! It is also the base structure that more complex structures and algorithms might use. Lets take a look at a program that runs the stack.

tst\_obj\_stack.p

```
{objmgr.i NEW}

DEF VAR cTemp AS CHARACTER NO-UNDO.
DEF VAR cErrCode AS CHARACTER NO-UNDO.
DEF VAR cErrMsg AS CHARACTER NO-UNDO.
DEF VAR cErrMethod AS CHARACTER NO-UNDO.

OUTPUT TO stack.log.

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

RUN Push IN OMGH("Test") (INPUT "a").
RUN Push IN OMGH("Test") (INPUT "b").
RUN Push IN OMGH("Test") (INPUT "c").
RUN Push IN OMGH("Test") (INPUT "d").

RUN Pop IN OMGH("Test") (OUTPUT cTemp).
MESSAGE "cTemp = " cTemp.
RUN Pop IN OMGH("Test") (OUTPUT cTemp).
MESSAGE "cTemp = " cTemp.
RUN Pop IN OMGH("Test") (OUTPUT cTemp).
MESSAGE "cTemp = " cTemp.
RUN Pop IN OMGH("Test") (OUTPUT cTemp).
MESSAGE "cTemp = " cTemp.
RUN Pop IN OMGH("Test") (OUTPUT cTemp).
MESSAGE "cTemp = " cTemp.
RUN Pop IN OMGH("Test") (OUTPUT cTemp).
MESSAGE "cTemp = " cTemp.

RUN GetError IN OMGH("Test") (OUTPUT cErrCode, OUTPUT cErrMsg, OUTPUT
cErrMethod).

MESSAGE cErrCode.
MESSAGE cErrMsg.
MESSAGE cErrMethod.

RUN OMDel ("Test").
```

OUTPUT CLOSE.

### OneStep Charge

#### *Premier Credit Card Processing for the 4GL*

- Integration in 10 minutes
- Realtime authorizations in 2 seconds
- Pure Progress
- Only Requires V9 or higher
- Fully-documented API
- NO drop files
- NO plain -text hazards
- Certified with all major processors such as VITAL, Nova, Paymentech, NDC, FHMS
- Tri-8-sponsored merchant accounts (optional) can save literally thousands per month

<http://OneStepCharge.com>  
[oscinfo@onestepcharge.com](mailto:oscinfo@onestepcharge.com)  
 866.461.TRI8



First we define some variables that aid us with any errors the stack needs to inform us of. There are few errors associated with a stack, 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 use the object manager's OMAAdd to instance the object "Test" who is composed of the code and data storage from obj\_stack.p. 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 Push or Pop from one instance has no effect on the data in the other instances. Think of them as two piles of paper or stacks of plates.

The object manager's OMAAdd procedure will also call into the Init procedure of the stack object automatically. More on this later on.

Once we have an object available to us, we start running the Push procedure available in the object to add our data into it. Notice we identify which object we wish to manipulate with the IN OMGH("Test") phrase of the statement. Push the following data in the following order: a b c d.

Remember that a stack is Last In First Out. So the last item in is a “d” so that should be the first item out of the stack when we “pop” an item from the stack. The act of popping an item also removes the item from data storage. This makes the last – 1 item available for popping. You can see the results in stack.log below:

stack.log

```
cTemp = d
cTemp = c
cTemp = b
cTemp = a
cTemp = ?
cTemp = ?
100
Pop On Empty Stack
Pop
```

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 pop off the stack.

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 push and pop off the stack was an unknown? Then you would not know if the stack was empty or if the valid value came off of it.

To aid in determining if something went wrong with the actions on the stack, one can check the GetError routine made available by the base object template. We have an error code of 100 with an error message “Pop On Empty Stack” to aid the programmer with error conditions that have happened during an operation.

*In our implementation of a stack, 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:*

*Push – allows an implementor to put a value onto the stack*

### Michelle's Web Design Services

<http://www.floridagoldens.com/web.htm>

Contact Email: [rtbionic@yahoo.com](mailto:rtbionic@yahoo.com)

I'm just one person so you'll be getting my personal touch. I specialize in simple, easy to navigate clean looking websites that load fast and peak interest.

My prices are very affordable, perfect for small businesses or quick projects.

We can start from the ground up from selecting the right domain name and finding you a host.

If you already have these things then all you need is a design. Email me at the above address and give me an idea of what you think you might like.

I'd love to hear YOUR ideas.

Sincerely,

Michelle

*Pop* – allows an implementor to remove a value from the stack

*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.

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

#### 100 – Pop On Empty Stack

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

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 stack code.

Below is the code for implementing the stack. It really is quite simple – we use a temp table to store our entries and the Pop and Push 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\_stack.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
/* Destructer routines.
*****/

{objmgr.i} /* May be deleted if not using object manager */

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

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

/* 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").

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 "001" THEN DO:
    RUN SetAttr IN THIS-PROCEDURE ("ErrCode", cErrorCode).
    RUN SetAttr IN THIS-PROCEDURE ("ErrMsg", "No Such Attribute").
    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", "Pop On Empty Stack").
    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 */

/*****
/* This is a special set of routines that allow the object's user to */
/* reach the tables directly. Convenient for ASSIGNS and the like so */
/* one is not continously calling a Set* routine for each and every */
/* field.                                                                */
/* WARNING: This CAN BE ABUSED.                                          */
/* WARNING: Sets of Like records may be a problem.                     */
/*   ATTN: Buffers can go both ways. This can be good. This can be */
/*         bad.                                                          */
*****/
PROCEDURE GetBuffers:

  DEFINE PARAMETER BUFFER YourRecord FOR YourTable.

  BUFFER-COPY ObjBuffer YourRecord.

END. /* PROCEDURE GetBuffers */

PROCEDURE SetBuffers:

  DEFINE PARAMETER BUFFER YourRecord FOR YourTable.

  BUFFER-COPY YourRecord ObjBuffer.

END.
*/

/* ----- Begin stacking code ----- */

/* Here we are using an index to keep the entries ordered */
/* This temp table is local to this instance only - multiple stacks */
/* can be used.                                          */

DEFINE TEMP-TABLE TheStack
  FIELD Order AS INTEGER
  FIELD AnEntry AS CHARACTER
  INDEX pukey Order.

PROCEDURE Push:

  DEF INPUT PARAMETER cEntry AS CHARACTER NO-UNDO.

  DEF VAR iEntry AS INTEGER NO-UNDO.
```

```
FIND LAST TheStack NO-LOCK NO-ERROR.

IF NOT AVAILABLE TheStack THEN ASSIGN iEntry = 1.
ELSE ASSIGN iEntry = TheStack.Order + 1.

CREATE TheStack.
ASSIGN TheStack.AnEntry = cEntry
      TheStack.Order = iEntry.

END. /* PROCEDURE Push */

PROCEDURE Pop:

  DEF OUTPUT PARAMETER cEntry AS CHARACTER NO-UNDO.

  FIND LAST TheStack NO-LOCK NO-ERROR.

  IF AVAILABLE TheStack THEN DO:
    ASSIGN cEntry = TheStack.AnEntry.
    DELETE TheStack.
  END.
  ELSE DO:
    ASSIGN cEntry = ?.
    RUN SetError ("100", "Pop").
  END.

END. /* PROCEDURE Pop */

PROCEDURE IsEmpty:

  DEF OUTPUT PARAMETER lReturn AS LOGICAL NO-UNDO.

  FIND FIRST TheStack NO-LOCK NO-ERROR.

  ASSIGN lReturn = AVAILABLE TheStack.

END. /* PROCEDURE IsEmpty */

PROCEDURE Count:

  DEF OUTPUT PARAMETER iCount AS INTEGER NO-UNDO.

  ASSIGN iCount = 0.
  FOR EACH TheStack NO-LOCK:
    ASSIGN iCount = iCount + 1.
  END.

END. /* PROCEDURE Count */

/* ----- 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](mailto: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 similar 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<sup>3</sup> 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.

Send your articles to [sauge@amduus.com](mailto:sauge@amduus.com)! Thanks!

---

<sup>3</sup> 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.