Scott Augé
President
Amduus Information Works, Inc.

sauge@amduus.com
http://www.amduus.com

Dear Reader,

Thanks for picking up this book and giving it a look over! Although it is older, using an older style of coding, I am sure you will still find many of the ideas useful in your applications.

Object Oriented Programming (OOP) makes writing code much more cleaner and simplier. Often in procedural code one has to make a bunch of if statements for certain conditions – something inheritance can ease. Also, it allows routines to something to be more easily packaged up in an object than a group procedures. These are only a few of the benefits!

Of course, I would like to point out areas that Amduus can help your company/organization:

Industries

• Manufacturing (MRP/MRP II/ERP) • Medical Insurance • Services	• Digital Marketing • Real Estate • SaaS Applications	• State and Federal Government • Property Management

Business Analysis

• Software Requirements Analysis • Functional Specifications	• Project Management • Technical Specifications	• Requests For Proposals

Digital Marketing

• Brochure Web Site • Service Requests (Work Orders) • Screencasts (Videos) • Product Returns	• Product Registration • Blogging • Survey Software • Configuration & Quotations	• E-Commerce • Video Brochure • Coupon Generation • Social Networking

Sincerely,

Scott Augé
President
Amduus Information Works, Inc.

Technologies Used By Amduus Information Works, Inc.

<HTML>

Progress | OpenEdge

C/C++

php

SQL

*NIX Shell Scripting

Languages

Programming languages are key to developing software. We develop in languages at the system level, database level, web site level, as well the application level.

HP-UX

LINUX

MS Windows

AIX

solaris

Apple OS X

Operating Systems

No matter the application – the most fundamental layer is the operating system. Amduus supports many types of operating systems from PC to large computers.

PostgreSQL

MySQL

PROGRESS SOFTWARE

IBM DB2

Databases

Any application that stores, retrieves, or deals with large amounts of data requires a database. Different databases offer different price points, functionality, and support levels.

Discussions Of Object Oriented Programming In Progress OpenEdge 10.1x

Scott Auge

Discussions Of
Object Oriented Programming
In Progress OpenEdge 10.1x

By

Scott Auge

Discussions Of Object Oriented Programming In Progress OpenEdge 10.1x

Printed in the United States of America.

Published By:

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

Printing History:
November 2008: First Edition

Introduction

This book is meant for readers who are familiar with the Progress ABL (aka Progress 4GL) programming language. If you are also familiar with object oriented programming – all the better. You do not need to be familiar with OOP to make use of the book.

Progress has a book that comes with their software development product called "Getting Started With Object Oriented Programming." This is a very good book for getting into the nitty-gritty of the object oriented abilities of the language. This book does not do so – it is not an attempt at rewriting something already available (and free for download on their PSDN site!)

What I am trying to do is provide an example oriented book to learning and using the object oriented aspects of the language. One can read the above reference and get a lot of theory on how classes work – but sometimes it's just nice to see some examples and to understand the thinking behind those examples.

Another point I hope to make in this book are ideas on how to go about designing objects. Once you get the paradigm in your head, it is pretty clear (usually) how to go about identifying and naming the methods and properties your object should have. Another thing is how broad does one make an object – should it be reusable? Should it directly update the database every time a method is invoked? Should it store up changes and then commit? These are questions I hope to answer.

To start with we will work with the old stand by data structures such as queues, stacks, vectors, trees, etc. By looking at these commonly known computer science concepts with the new language syntax and abilities, one should be able to pick up quickly what can be done.

Next we start looking at using OO programming as sets of tools to accomplish things.

After that, we look at OO and how it works with the database. We'll start with a class that makes dynamic queries almost as easy to use as a FOR EACH statement. Following we will look at two techniques of performing write[1] operations on the database.

Finally, we start looking at some ideas on how to "think" in an object oriented manner

1 Add, Delete, and Update are considered write operations.

at a more conceptual level. We'll look at the paradigm shift in thinking as well look at some topics like when to inherit from a class or to instance an object.

This book was written with Progress version 10.1c in mind. You can get by with version 10.1a for most of the routines contained in the book with a little bit of tweaking. You will need a minimum of 10.1a to use any of the code and ideas in this book. Object oriented programming constructs are not available in version 10.0 or earlier versions of Progress.

Many if not all of the objects in this text can be used in GUI[2], CHUI[3], or WWW[4] interfaces to your application.

If you wish the source code, please contact Scott Auge at sauge@amduus.com or scottauge@gmail.com.

2 Graphical User Interface, aka Windows.
3 Character User Interface, aka Windows, UNIX terminal connections, bar code readers, etc.
4 World Wide Web interface, aka any thing that uses a browser or web service to interact with your application.

Table of Contents

Why Program In An Object Oriented Manner?

Once you get the taste for OOP you will discover an entirely new way of analyzing and designing software solutions.

Before we start exploring objects, I want to give you ideas to roll around in your head as you read about them. These following paragraphs are both arguments for and a bit of explanation of the benefits of object oriented programming in the ABL.

The world full of many minds and thoughts has pretty much accepted and determined that OOP for highly abstract programming (that is code not “close to the metal”) is the way to go. Even FORTRAN has gone object oriented.

Progress using companies are already asking for the skill. I personally know people who have been passed over on progress jobs interviews because they didn't have OOP experience. Progress versions V6, V7, and V8 are long gone and V9 is on it's way out the door soon. It's in your own interest to keep your skills up to par.

Code re-use is simplified. Take for example an object that implements a stack. By inheriting that object into another object – perhaps an error minder – the stack object immediately becomes part of the error minder with the error minder “adding value” to the stack object with error code translation etc.

Code re-use can be instantiated in parallel. Once again, take the idea of a stack. If one stack is needed – fine for procedural coding. But if n > 1 stacks are needed in an algorithm, then one needs to do n > 1 times the amount of coding. Often this coding becomes awkward with pre-processor definitions and include files.

With OOP, code re-use is simplified by simply creating an instance of the same object as many times as needed.

```
/* Dynamic number of stacks example */

Define temp-table ListOfStacks
  field StackObject as class Progress.Lang.Object.

do I = 1 to 10:
  create ListOfStacks.
  ListOfStacks.StackObject = new Stack().
end.
```

OOP provides a convenient mechanism to collect like functionality together with the ability to address each piece of functionality as needed. One doesn't look for pop.p, push.p, countof.p – one looks for stack.cls and finds those pieces of functionality in there with a simple address of instanceName:functionName() in the source code.

```
Stack:Push("One").
Stack:Push("Two").
put "Stack height: " Stack:CountOf().
```

OOP makes it easier of creating abstraction layers. Not only does it allow you to implement modularization – the modularization is modular! OOP provides a different mindset on how to conceptualize and implement functionality. Developers see an object called Authorization and with that object they can do certain things to the Authorization and learn certain things from the Authorization.

One of the benefits of OOP is the ease of creating data encapsulation. No need to learn about multiple tables. No need to learn about "magic numbers" or "magic codes." No need to know that if one does this in table A one has to do that in table B – the class and it's methods takes care of it. (Of course, the person providing the class will still need the knowledge of the data base – but often the person using the class does not.)

OOP increases programmer functionality – learn the existence of one class and it's methods or learn of multiple fields, tables, procedure files, and their (sometimes funky) needs. The learning curve is smaller and developers get productive earlier.

***Properly defined objects can read almost like a programming language on a certain business concept*.**

```
/* Lets find our authorization.  This is at an abstract  */
/* level - we aren't worried about what table to look in */
/* or any of it's sister tables.                         */

Authorization:FindByNumberOrg("A1111111", "GHHS").
if not Authorization:IsAvailable() then ...

/* We don't even need to check if this exists - the */
/* methods are smart enough to know if it is worth  */
/* the effort or not.                               */

Authorization:CancelAirlineReservation().
Authorization:CancelRentalCar().

/* Lets add an expense to the Authorization */

Expense = new Expense().
Expense:SetType (...).
Expense:SetCost (...).
Expense:SetDate (...).

/* Validations on Sets detail errors with them */
/* if they exist - pop them off to the user.   */

if Expense:HasError() then
  do while Expense:HasError():
    {&OUT} Expense:GetError() skip.
  end.

/* Associate the expense with an authorization */

Authorization:AddExpense(Expense).

/* Has it been signed? */

if Authorization:HasStamp ("SIGNED") then
  Authorization:DoStamp ("APPROVED").
```

Objects are more flexible. Parameters to procedures might need to be changed – with objects one simply adds a new property or method without damaging existing code:

```
/* old way - very kludgy - change mail.p and change */
/* through out the code base!  Else - one more      */
/* wrapper out there to update in maintenance! That */
/* costs money!                                     */

run mail.p (To, Subject, MessageBody).

/* New way with objects! uh - oh - now they want    */
/* BCC! Simply add call SetBCC where needed on a    */
/* class and ignore when not.                       */

Mailer:SetTo(...).
Mailer:SetBCC(...).
Mailer:Subject(...).
Mailer:Body(...).
Mailer:SendByMail(). /* use /usr/bin/mail */

/* Uh oh! This one doesn't need BCC, but could use */
/* attachment and a different underlying transport */
/* mechanism.                                      */

Mailer:SetTo(...).
Mailer:Subject(...).
Mailer:Body(...).
Mailer:AddAttachment (...).
Mailer:SendBySMTP(). /* use SMTP over socket */

/* Just send it with what ever the default transport is */

Mailer:SetTo(...).
Mailer:Subject(...).
Mailer:Body(...).
Mailer:Send().
```

Method overloading[5] allows more flexible arguments to the same method in an object. Yes, you can actually use the same name for a method that takes different arguments. No more `PushInt()`, `PushChar()`, etc. - simply make the concept of `Push()` work with any data you need to.

```
Stack:Push(“One”). /* Push(char) */
Stack:Push(2).     /* Push(int)  */
put “Stack height: “ Stack:CountOf().
```

5 Version 10.1c or better required.

LEARN MORE at

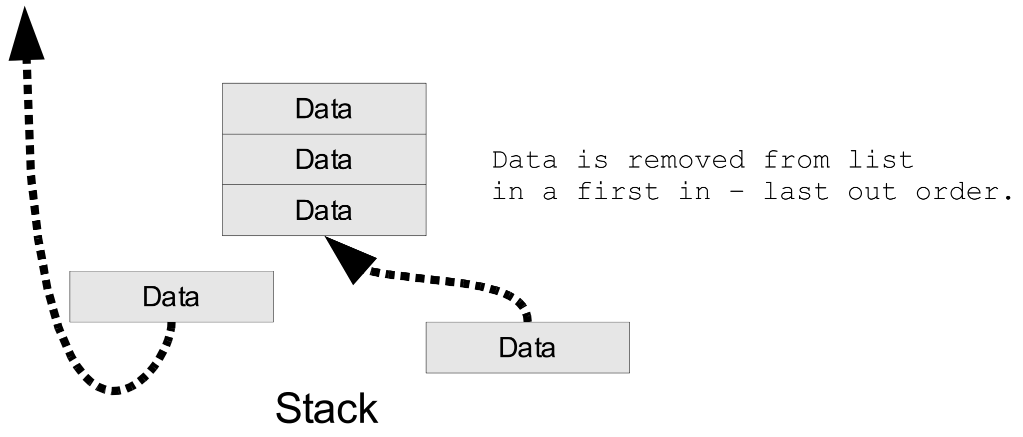
http://www.psdn.com/library/entry.jspa?externalID=4761&categoryID=1917

Examples In The Fundamental Data Structures

When learning computer science, one of the fundamentals learned is the concept of a data structure. A data structure is a collection of data and a collection of operations that are performed on that data. Sounds a lot like an object doesn't it? So lets look at what we know to learn the new.

Stack

One of the most commonly learned data structures (or pattern) is the stack. It allows one to stack up a list of items. It is also known as a LIFO structure – that is, the (l)ast (i)tem into the data structure is the (f)irst (o)ut of the structure. We construct two operations called `PushOnStack()` and `PopOffStack()` to place data into and out of the data structure.



In addition to operations to push the value into the data structure and popping a value out of the data structure, it would be useful to have some informational operations to answer questions like "How many items are in the structure?" or "Is there anything in the structure?" We call these `CountOf()` and `IsEmpty()` respectively.

Another utility operation on the data structure is the ability to clear the stack without having to loop through a series of calls to `PopOffStack()`. We call this operation `ClearStack()`.

When looking at the data structure in terms of a class, we call the operations "methods."

So here are some categories of methods you might want to consider when designing your class in terms of functional collections:

- **Core**. Those methods that are fundamental to the functionality of the class. Examples of this is adding or deleting database records, making decisions, or validating information.

- **Informational**. Those methods that provide some information to the implementing program about what is going on in the object. Examples of these methods are error status and other information pertinent to the running of the object.
- **Utilitarian**. Those methods that are useful in the maintenance of information in the class. Often these are the result of modularization and functional decomposition of the above methods and private[6] to the objects.

Since we have talked about the operational elements of a data structure, now lets talk about the data portion of the data structure. In the code below, we have two main pieces of data – one is a set which we realize with a temp-table. The other is a simple lone value we realize with a variable to help order data coming in and out of the object. Another variable is used to hold the source code revision information – this can be useful to implementors (programmers using the object) or for the ident[7] command to see what revision a piece of r-code really is.

There are various amounts of visibility to these variables from code using this class. The temp table, named `TheStack`, is private. This means only methods within the class can see the data. The variable, Revision, is public. This means that the calling program can access this data. More on that later.

With `TheStack`, we can see making use of a fundamental feature of the ABL within the class – a simple temp-table. One is going to be presented with plenty of opportunities to use what you know to do object oriented programming in the ABL. Our temp table is composed of two pieces of data – the data we want to push and pop off the stack called `Data`, and another that will keep track of what was added when called `OrderNo`.

One of the simple uses for this is a collection of errors that a program may encounter while doing a batch load or a set of validations from a user interface (aka a web page or web service.) A more complicated use might be parsing a string into components. (A stack can indeed be a component within a more complicated data structure such as an arithmetic parser!)

6 More on what "private" means later.

7 A java based ident command is provided for those who do not have a revision control system installed that can use commonly used keywords.

Here is the code for a simple character based stack.

```
/*************************************************************************/
/* Class to conveniently store errors or stacked data                    */
/*************************************************************************/

class Stack:

  define private temp-table TheStack
    field Data as character
    field OrderNo as integer
    index PUKey is unique primary OrderNo ascending.

  define public variable Revision as character
  init "$Revision$"
  no-undo.

  /*********************************************************************/
  /* Place an item onto the stack.                                     */
  /*********************************************************************/

  method public logical PushOnStack (input TheValue as character):

    define variable NextOrderNo as integer no-undo.

    find last TheStack no-lock no-error.

    if available TheStack then
      NextOrderNo = TheStack.OrderNo + 1.
    else
      NextOrderNo = 1.

    create TheStack.

    TheStack.Data = TheValue.
    TheStack.OrderNo = NextOrderNo.

    return true.

  end. /* method */

  /*********************************************************************/
  /* Remove an item from the stack.                                    */
  /*********************************************************************/

  method public character PopOffStack ():
```

```
  define variable TheValue as character no-undo.

  find last TheStack exclusive-lock no-error.

  if not available TheStack then return ?.

  TheValue = TheStack.Data.

  delete TheStack.

  return TheValue.

end. /* method */

/*******************************************************************/
/* Information about the stack.                                    */
/*******************************************************************/

method public logical IsEmpty():

  return not can-find (first TheStack).

end. /* method */

/*******************************************************************/
/* Provide a means to clear the stack without popping all of them.  */
/*******************************************************************/

method public logical ClearStack():

  empty temp-table TheStack.
  return true.

end. /* method */

/*******************************************************************/
/* Information about the stack.                                    */
/*******************************************************************/

method public integer CountOf():

  define variable TheCount as integer init 0 no-undo.

  for each TheStack no-lock:
    TheCount = TheCount + 1.
  end.

  return TheCount.
  end. /* method */

end. /* class */
```

Here is a simple program to use the stack in the Webspeed Script Lab:

```
define variable S as class Stack no-undo.

S = new Stack().

S:PushOnStack ("A").
S:PushOnStack ("B").
S:PushOnStack ("C").

{&OUT} S:CountOf() "<br>".
{&OUT} S:IsEmpty() "<br>".

{&OUT} S:PopOffStack() "<br>".
{&OUT} S:CountOf() "<br>".
{&OUT} S:PopOffStack() "<br>".
{&OUT} S:CountOf() "<br>".
{&OUT} S:PopOffStack() "<br>".
{&OUT} S:CountOf() "<br>".
{&OUT} S:PopOffStack() "<br>". /* ? */
{&OUT} S:CountOf() "<br>".

{&OUT} S:IsEmpty() "<br>".

{&OUT} S:Revision "<br>".

delete object S.
```

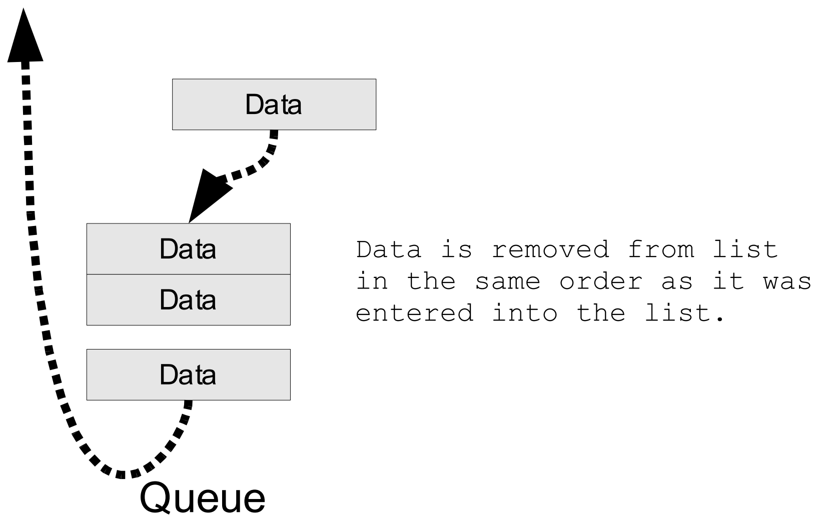
What about those other types of data that can be put on the stack? One simply needs to add a new Data field to the temp table – perhaps `IntData` and a new method called `PushOnStack()` and `PopOffStack()` using arguments of that type.

This is called overloading a method – making the same name work with different types of data. The compiler knows which methods to call because of the name methods and the signature[8] of the arguments being accepted.

8 Signature is the type and order of the types of arguments being sent. This is available only beginning with 10.1c. Previous versions will not compile this kind of code.

Queue

A queue is somewhat opposite of a stack, instead of the first item being the last item out – the first item is the first item out. Think of it as a bank line – the teller can only process so many requests so customers stand in line until it is there turn.


Data
Data
Data
Data
Data is removed from list in the same order as it was entered into the list.
Queue

Old school thought might be – well, I can do the same thing with a temp-table and a find first! And you would be right. Until you needed two of them. Or three of them. And should you need it again you would be coding again.

This is one of the reasons why I chose a queue to illustrate with. It is deceptively easy to make a queue in Progress ABL with a few statements. But hopefully with this code you will see that it is easily re-usable in other code and the same code by simply making an instance of the class.

This means:

- Creating another instance is code re-use. There is no need to customize the code implementing the data structure to be used over and over (aka extents or difficult to use include files) – you just use it.
- It is already tested.

- You save time.
- You save money.

Everything you need to deal with is encapsulated – that is – you have one variable to reference the object by and your data set to work with. So you let the object manage the data for you. No need to remember special fields or values – all you need to focus on is the object reference and how you named it.

Before looking at the code, lets list off some things we want to do: 1) Put items on the queue, 2) take items off the queue, 3) count how many items are on the queue, and 4) make a dump file of the items on the queue (great for testing or debugging.)

You can see below in the code we are interested in two properties – an iterator which helps us order the records stored in a temp table called queue (there we are using the power of the 4GL again!) Following is a constructor it initialize things and the methods that basically do exactly what we named in our list of things above.

```
class amduus.misc.queue:

  define private variable iter as integer no-undo.

  define private temp-table queue
    field order as integer
    field data  as character
    index pukey is primary order.

  /*****************************************************************/
  /* A constructor is run when a class is instantiated into an       */
  /* object.  Any kind of initializations can be done here.          */
  /*****************************************************************/

  constructor public queue():

    iter = 0.

  end. /* constructor */

  /*****************************************************************/
  /* This is a core functionality method - put some data into the    */
  /* queue.                                                          */
  /*****************************************************************/

  method public void enqueue(input string as character):
```

```
  create queue.

  iter = iter + 1.
  queue.order = iter.
  queue.data = string.

end. /* enqueue */

/*******************************************************************/
/* This is a core functionality method - remove the data from the   */
/* queue and give it to the code using the object.                  */
/*******************************************************************/

method public character dequeue ():

  define variable text_out as character no-undo.

  find first queue exclusive-lock no-error.
  if not available queue then return ?.

  text_out = queue.data.

  delete queue.

  return text_out.

end. /* dequeue */

/*******************************************************************/
/* This is a utilitarian method to allow the implementation        */
/* program “reset” the data structure.                             */
/*******************************************************************/

method public void empty ():

  empty temp-table queue.

  iter = 0.

end. /* empty */

/*******************************************************************/
/* This is an informational method to help the implementation code  */
/* know how many items this queue holds.                            */
/*******************************************************************/

method public integer countof():

  define variable counter as integer init 0 no-undo.
```

```
    for each queue no-lock:
      counter = counter + 1.
    end.

    return counter.

  end. /* countof */

  /*******************************************************************/
  /* Another informational method that helps answer the question of   */
  /* if the queue is empty in an easy to comprehend and self-         */
  /* documenting way for the implementation code.                     */
  /*******************************************************************/

  method public logical isempty():

    return not can-find (first queue).

  end. /* isempty */

  /*******************************************************************/
  /* This is a utilitarian method - sometimes ya just want to know    */
  /* what is happening in the structure.  This is a way to see by     */
  /* dumping the data into a file.                                    */
  /*******************************************************************/

  method public void dumpfile(input filename as character):

    output to value (filename).

    for each queue no-lock:
      export queue.
    end.

    output close.

  end. /* dumpfile */

end. /* class */
```

This is an example use of the queue:

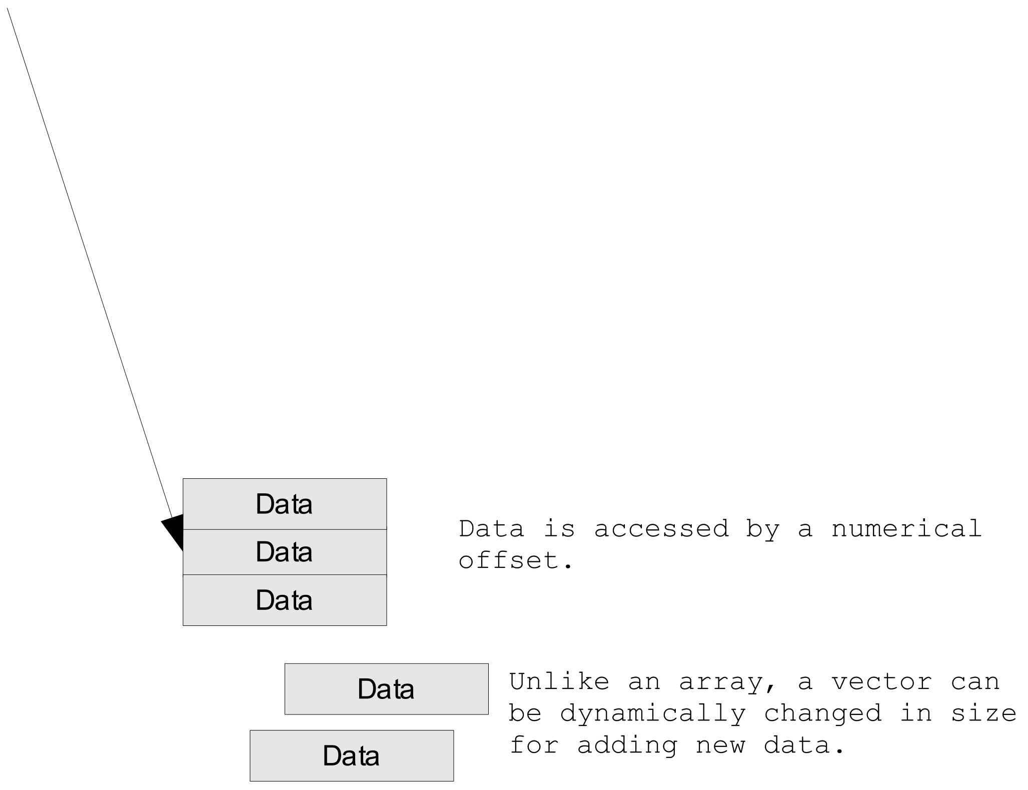
```
define variable myqueue as class amduus.misc.queue no-undo.
define variable out_text as character no-undo.

myqueue = new amduus.misc.queue().
```

```
myqueue:enqueue ("First").

display myqueue:countof().

myqueue:enqueue ("Second").
myqueue:enqueue ("Third").

display myqueue:countof().

display myqueue:dequeue().
display myqueue:dequeue().

display myqueue:countof().

delete object myqueue.
```

Vector

Lets look at another fundamental data structure – the vector. The vector is very much a self regulating array. Think of an extent variable that can extend it's self as needed.

Another construct we use in progress to try and implement vectors is a comma delimited character string and the use of `NUM-ENTRIES()` and `ENTRY()`. Sometimes a string will run out of room (less of a problem with `longchar` now) and the management of the comma's can be tiresome. Often the code is a bit kludgey – but with a class it all becomes quite easy to implement an expanding set of data.


Data
Data
Data
Data is accessed by a numerical offset.
Data
Data
Unlike an array, a vector can be dynamically changed in size for adding new data.

Vector

Since a vector is so close to a string with the operations `NUM-ENTRIES()`, `ENTRY()`, and `CAN-DO()`, this is an excellent time to think about the different view of the world OOP has. With the functions, one thinks in a verb-noun manner. I have these functions (verbs) that I want to operate on this data (noun). OOP is almost the complete opposite. It is more noun-verb thinking – I have this thing (noun) and I want it to do something for me (verb).

Lets think of some things that an object can do that perhaps these functions can't – like sort the list of entries? Perhaps capitalize each of the entries? Make sure entries added are unique (ie discard later entries of the same data.) Hopefully you are beginning to see the advantages of creating an object instead of depending on sequences of ABL being used over and over. Use the object and all the power of that object becomes available. Use the ABL and one is kind of stuck with what the ABL provides.

A new thing to learn is notice how the class statement has a period delimited sequence of words before Vector. How this works is you place a file named `Vector.cls` in a directory `/amduus/misc` that hangs off the `PROPATH`. (Of course under UNIX the lettering is case sensitive but not so in Windows.)

So this string of text is telling the compiler in what directory to find a file named `Vector.cls` which contains this source code. The name of the class must match the name of the file.

> *Hopefully in the future, as more open source is created by people using OOABL, the objects can be easily shared. To differentiate one set of class definition files from another, the prefix can be used. For example, the class amduus.misc.Vector would be different from footech.datastructures.Vector. By instantiating with the prefix one knows exactly which version one would be using.*

Here is the code for a vector and some example code implementing it follows:

```
/* Vector - basically an expanding array */

class amduus.misc.Vector:

  define temp-table Vector
  field Data as character
  field Order as integer
  index pukey is primary unique Order ascending.

  /*******************************************************************/
  /* This is an example of an overloaded method - the same name but   */
  /* different arguments.                                             */
  /*******************************************************************/

  /***** 10.1c allows overloading
  method public void SetEntry(input TheData as character):

    define variable LastOrder as integer no-undo.

    find last Vector no-lock.

    LastOrder = if available Vector then (Vector.Order + 1) else (1).

    create Vector.
  Vector.Data = TheData.
  Vector.Order = LastOrder.

end. /* method */
*****/

/*********************************************************************/
/* The method version of adding an entry into the vector.  In a      */
/* static non-expanding extent, it would read like:                  */
/*   Vector[TheIndex] = TheData.                                     */
/* A vector data structure automatically expands as needed.          */
/*********************************************************************/

method public void SetEntry
(input TheIndex as integer,
 input TheData as character
):

  FillIn(TheIndex).

  find Vector exclusive-lock
  where Vector.Order = TheIndex
  no-error.

  Vector.Data = TheData.

end. /* method */

/*********************************************************************/
/* A utilitarian method used by the data structure to automatically */
/* create new blank entries if the implementation code expands past */
/* the last existing entry in vector.                                */
/* Note how it is private so the implementation code cannot call it */
/* directly - only methods in this class can call it.               */
/*********************************************************************/

method private void FillIn (input TheIndex as integer):

  define variable CurIndex as integer no-undo.

  do CurIndex = 1 to TheIndex:

    /* Might be tempted to simply call SetEntry() but think about the */
    /* expense in operations.                                         */

    if not can-find (Vector where Vector.Order = CurIndex) then do:

      create Vector.
      Vector.Order = CurIndex.
      Vector.Data = ?.

    end. /* if */

  end. /* do */

end. /* method */

/*********************************************************************/
/* Obtain the data stored at a given index of the vector.            */
/*********************************************************************/

method public character GetEntry (input TheIndex as integer):

  find Vector no-lock
  where Vector.Order = TheIndex
  no-error.

  if available Vector then return Vector.Data.

  return ?.

end. /* method */

/*********************************************************************/
/* An informational method telling the implementation code how many */
/* items are stored in the vector.                                   */
/*********************************************************************/

method public integer CountOf ():

  define variable Count as integer no-undo.

  for each Vector no-lock:
    Count = Count + 1.
  end.

  return Count.

end. /* method */

/*********************************************************************/
/* An informational method to tell the implementation code what is  */
/* going on inside the data structure.                               */
/*********************************************************************/

method public logical IsEmpty():

  return not can-find (first Vector).
  end. /* method */

  /*******************************************************************/
  /* An utilitarian method to help the implementation code control    */
  /* use of the data structure.                                       */
  /*******************************************************************/

  method public void Reset():

    empty temp-table Vector.

  end. /* method */

  /*******************************************************************/
  /* A utilitarian method to help the implementation programmer if    */
  /* the code is not doing what they are expecting - a debugging      */
  /* tool.                                                            */
  /*******************************************************************/

  method public void SaveToFile (input FileName as character):

    output to value (FileName).

    for each Vector no-lock:

      export Vector.

    end. /* for each */

    output close.

  end. /* method */

end. /* class */
```

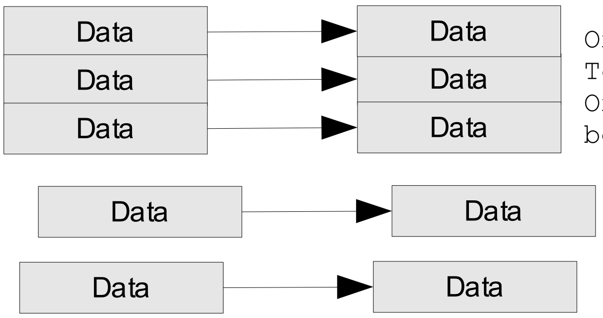
In this example code (I usually include a set of unit test code in every class file so if changes are made the tests are readily available) – we type the T variable as a class with the prefix and then instantiate it with the new operator with the same prefix. From then on it will be using that code base to manipulate it's data.

```
/************************** UNIT TEST ***********************************

define variable T as class amduus.misc.Vector no-undo.

T = new amduus.misc.Vector().
```

```
T:SetEntry (1, "One").
T:SetEntry (3, "Three").

display T:CountOf().
display T:IsEmpty().

display T:GetEntry (1).

delete object T.

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

Hash

A hash is like a vector, but instead of being limited to an integer index offset to find a value, one can use a string to find the value. These are also called “associative arrays” because one value is associated to another value.



```
One set of data is associated
To another set of data in a
One to one form.  The data can
be any type or canonical form.
```

```
Unlike an array, a vector can
be dynamically changed in size
for adding new data.
```

Hash

We could actually use something called “inheritance” to make a vector out of the coding for a hash. But I am going to save that for a whopper of an example to show the flexibility of re-using code.

I have included the code so you can see how it is very similar, yet dissimilar from a vector.

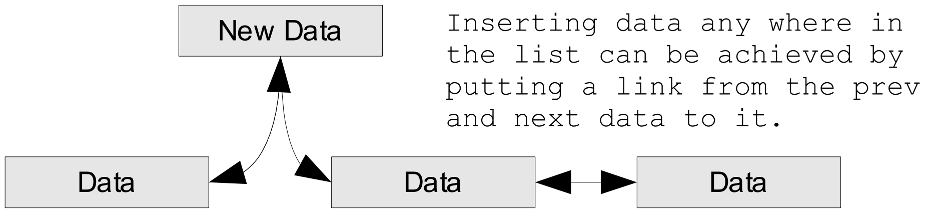
```
/* Hash - basically an associative array */

class amduus.misc.Hash:

  define temp-table Hash
```

```
field Data as character
field Hash as character.

/*******************************************************************/
/* This is part of the core functionality of the hash - accept a    */
/* combination of values and store them.                            */
/*******************************************************************/

method public void AddEntry
(input Hash as character,
 input TheData as character
):

  find Hash exclusive-lock
  where Hash.Hash = Hash
  no-error.

  if not available Hash then do:

    create Hash.
    Hash.Hash = Hash.

  end. /* if */

  Hash.Data = TheData.

end. /* method */

/*******************************************************************/
/* Core functionality - removal of a pair of data in the data       */
/* structure.                                                       */
/*******************************************************************/

method public void DeleteEntry (input Hash as character):

  for each Hash exclusive-lock
  where Hash.Hash = Hash:

    delete Hash.

  end. /* for each */

end. /* method */

/*******************************************************************/
/* Core functionality - given one piece of data, find it's as-      */
/* sociated data.                                                   */
/*******************************************************************/
```

```
method public character SearchEntry(input Hash as character):

  find Hash no-lock
  where Hash.Hash = Hash
  no-error.

  if available Hash then return Hash.Data.

  return ?.

end. /* method */

/*****************************************************************/
/* An informational method - how many items do we have in here?     */
/*****************************************************************/

method public integer CountOf():

  define variable TheCount as integer no-undo.

  for each Hash no-lock:
    TheCount = TheCount + 1.
  end.

  return TheCount.

end. /* method */

/*****************************************************************/
/* An informational method - do we have anything in here?  We pro-  */
/* vide this for self-documenting if statements and other condition */
/* statements.                                                      */
/*****************************************************************/

method public logical IsEmpty():

  return not can-find (first Hash).

end. /* method */

/*****************************************************************/
/* A utilitarian method that allows the implementation code to re-  */
/* set the data structure.                                          */
/*****************************************************************/

method public void Reset ():

  empty temp-table Hash.
  end. /* method */

  /*******************************************************************/
  /* A utilitarian method that allows the programmer using the class  */
  /* to dump data contained within to a file.                         */
  /*******************************************************************/

  method public void SaveToFile (input FileName as character):

    output to value (FileName).

    for each Hash no-lock:

      export Hash.

    end. /* for each */

    output close.

  end. /* method */

end. /* class */

/************************** UNIT TEST **********************************

define variable T as class amduus.misc.Hash no-undo.

T = new amduus.misc.Hash().

T:AddEntry("Scott", "One").
T:AddEntry("Craig", "Two").

display T:IsEmpty().
display T:CountOf().

display T:SearchEntry("Craig").
T:DeleteEntry("Scott").
display T:SearchEntry("Craig").
display T:CountOf().

delete object T.

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

Double Linked List

A double linked list is something even more sophisticated. With it you have the usual ordering fields but they go in both directions to previously existing data. What this means is that you can keep an ordered list of data, but easily insert new data in between existing data – not simply at the beginning or end of the list.


New Data
Inserting data any where in the list can be achieved by putting a link from the prev and next data to it.
Data
Data
Data

Double Linked List

In addition, you will have some iterating methods to move around in the list. Most interesting is the possibility of two types of insert methods (before and after current piece of data.) One has a deletion method that is more complex than most in that it has to tie together multiple pieces of data together while deleting another.

Like the other classes we have discussed, the data portion is kept in a Progress ABL temp-table. Having these different operations on the table kept in a class makes a more robust and fully tested piece of code than trying to do the same over and over in a more procedural environment.

A double linked list is one of the more complicated data structures to code. Hopefully you will notice that many of the methods are composed of literally one statement or a few at most... and then there are those a little larger. The point being though, a very complicated data structure can be created of operations composed of minuscule 4GL code.

As with most classes, you can use different data types than the simple string stored here.

Here is the listing with some example calls in the end.

```
class DualLinkList:

  define private temp-table TheData
    field Data as character
    field NodeNumber as integer
    field PrevNodeNumber as integer
    field NextNodeNumber as integer.

  define private buffer CurrentTheData for TheData.

  /*********************************************************************/
  /* Core functionality - provide a means to read a piece of data in the */
  /* list.                                                               */
  /*********************************************************************/

  method public character GetValue():

    return CurrentTheData.Data.

  end.

  /*********************************************************************/
  /* Informational method - is the list empty?                         */
  /*********************************************************************/

  method public logical IsEmpty():

    return not can-find (first TheData).

  end. /* method */

  /*********************************************************************/
  /* Informational method - does the list already include a value?     */
  /*********************************************************************/

  method public logical DoesContain(input Data as character):

    for each TheData no-lock
      where TheData.Data = Data:

      return true.

    end. /* for each */

    return false.

  end. /* method */

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

```
/* Core functionality - provide a means to remove a piece of data from */
/* the list.                                                           */
/***********************************************************************/

method public logical DeleteCurrent():

  define buffer PrevTheData for TheData.
  define buffer NextTheData for TheData.

  if not available CurrentTheData then return false.

  find current CurrentTheData exclusive-lock.

  /* if there is a previous node bring it up for linking */

  if CurrentTheData.PrevNodeNumber <> ? then
    find PrevTheData exclusive-lock
    where PrevTheData.NodeNumber = CurrentTheData.PrevNodeNumber
    no-error.

  /* if there is a next node bring it up for linking */

  if CurrentTheData.NextNodeNumber <> ? then
    find NextTheData exclusive-lock
    where NextTheData.NodeNumber = CurrentTheData.NextNodeNumber
    no-error.

  /* Link the next data node to the previous node */

  if available NextTheData then
    if available PrevTheData then
      NextTheData.PrevNodeNumber = PrevTheData.NodeNumber.
    else
      NextTheData.PrevNodeNumber = ?.

  /* Link the prev data node to the next node */

  if available PrevTheData then
    if available NextTheData then
      PrevTheData.NextNodeNumber = NextTheData.NodeNumber.
    else
      PrevTheData.NextNodeNumber = ?.

  /* Blow away TheCurrent Data and set to prev or next - whom ever is */
  /* available first.                                                 */

  delete CurrentTheData.

  return true.
```

```
end. /* method */

/**********************************************************************/
/* Informational method - are we currently positioned at the front of  */
/* the list?                                                           */
/**********************************************************************/

method public logical IsFront():

  if not available CurrentTheData then return true.

  return (CurrentTheData.PrevNodeNumber = ?).

end.

/**********************************************************************/
/* Informational method - are we currently positioned at the end of    */
/* the list?                                                           */
/**********************************************************************/

method public logical IsEnd():

  if not available CurrentTheData then return true.

  return (CurrentTheData.NextNodeNumber = ?).

end.

/**********************************************************************/
/* Core functionality - provide the means to add an entry to the list. */
/**********************************************************************/

method public logical InsertAfter (input NewData as character):

  define buffer NextTheData for TheData.
  define buffer NewTheData for TheData.

  create NewTheData.
  NewTheData.Data = NewData.
  NewTheData.NodeNumber = etime + random(1, 1000000).
  NewTheData.NextNodeNumber = ?.
  NewTheData.PrevNodeNumber = ?.

  if available CurrentTheData then
    find NextTheData no-lock
    where NextTheData.NodeNumber = CurrentTheData.NextNodeNumber
    no-error.
  if available CurrentTheData then do:
    NewTheData.PrevNodeNumber = CurrentTheData.NodeNumber.
    CurrentTheData.NextNodeNumber = NewTheData.NodeNumber.
  end.

  if available NextTheData then do:
    NextTheData.PrevNodeNumber = NewTheData.NodeNumber.
    NewTheData.NextNodeNumber = NextTheData.NodeNumber.
  end.

  find CurrentTheData where
  CurrentTheData.NodeNumber = NewTheData.NodeNumber.

end.

/**********************************************************************/
/* Core functionality - provide a means to add data to the list.      */
/**********************************************************************/

method public logical InsertBefore (input NewData as character):

  /* An exercise for the reader! */

end. /* method */

/**********************************************************************/
/* Allow the implementation code to jump to a certain offset in the   */
/* list.  From there data reading, insertion, or deleting can be done. */
/**********************************************************************/

method public logical MoveToEntryN (input N as integer):

  define variable Counter as integer no-undo.

  if N < 1 then return false.

  if not MoveFront() then return false.
  Counter = 1.

  do while true:

    if Counter = N then return true.
    if not MoveNext() then return false.

    Counter = Counter + 1.

  end. /* do */

end. /* method */
```

```
/**********************************************************************/
/* Core functionality - move from the current piece of data to the     */
/* next one for data reading, insertion, deletion or more navigating.  */
/**********************************************************************/

method public logical MoveNext():

  /* Need this intermediate variable because the phrase               */
  /* where CurrentTheData.NodeNumber = CurrentTheData.NextNodeNumber */
  /* does not work.                                                   */

  define variable NodeNumber as integer no-undo.

  if not available CurrentTheData then return false.

  if CurrentTheData.NextNodeNumber = ? then return false.

  NodeNumber = CurrentTheData.NextNodeNumber.

  find CurrentTheData no-lock
  where CurrentTheData.NodeNumber = NodeNumber
  no-error.

  return true.

end. /* method */

/**********************************************************************/
/* Core functionality - move one step over to a previous data item.    */
/**********************************************************************/

method public logical MovePrev():

  /* Need this intermediate variable because the phrase               */
  /* where CurrentTheData.NodeNumber = CurrentTheData.PrevNodeNumber */
  /* does not work.                                                   */

  define variable NodeNumber as integer no-undo.

  if not available CurrentTheData then return false.

  if CurrentTheData.PrevNodeNumber = ? then return false.

  NodeNumber  = CurrentTheData.PrevNodeNumber.

  find CurrentTheData no-lock
  where CurrentTheData.NodeNumber = NodeNumber
  no-error.
  return true.

end. /* method */

/*********************************************************************/
/* Core functionality - jump to the front of the list.              */
/*********************************************************************/

method public logical MoveFront():

  for first CurrentTheData no-lock
    where CurrentTheData.PrevNodeNumber = ?:

    return true.

  end. /* for first */

  return false.

end. /* method */

/*********************************************************************/
/* Core functionality - Jump to the end of the list!                */
/*********************************************************************/

method public logical MoveEnd():

  for first CurrentTheData no-lock
    where CurrentTheData.NextNodeNumber = ?:

    return true.

  end. /* for first */

  return false.

end. /* method */

/*********************************************************************/
/* A utilitarian method that helps with debugging or understanding  */
/* what is going on in the list.                                    */
/*********************************************************************/

method public void DumpDataToFile (input FileName as character):

  output to value(FileName).

  for each TheData:
      export TheData.
    end.

    output close.

  end. /* method */

end. /* class */
```

Here is some example use of the data structure. Hopefully you can see the ease of implementation and clarity of code that OOP provides for this very complex processing.

```
/**************************** UNIT TEST CODE *****************************

PROPATH=PROPATH + ":/export/home/sauge/code".

define variable t as class DualLinkList no-undo.

t = new DualLinkList().

t:InsertAfter("One,").
{&OUT} t:GetValue().

t:InsertAfter("Two,").
{&OUT} t:GetValue().

t:InsertAfter("Three,").
{&OUT} t:GetValue().

t:MoveFront().

t:InsertAfter("One.Two,").
{&OUT} t:GetValue().

t:MoveEnd().
t:InsertAfter("Four").

t:DumpDataToFile ("/tmp/t").

{&OUT} "|".

t:MoveFront().
do while true:
  {&OUT} t:GetValue().
  if t:IsEnd() then leave.
  if not t:MoveNext() then leave.
end.

{&OUT} "|".

if t:MoveToEntryN(2) then
{&OUT} t:GetValue().
else
{&OUT} "Out Bounds".


{&OUT} "|".

if t:MoveToEntryN(7) then
{&OUT} t:GetValue().
else
{&OUT} "Out Bounds".

{&OUT} "|".

t:MoveToEntryN(2).
t:DeleteCurrent().

t:DumpDataToFile ("/tmp/t1").

delete object t.

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

Using Inheritance To Make A Queue

This is where object oriented programming gets interesting. We are going to use inheritance – basically code reuse – to create a new class.

You are familiar with a Queue from a previous section – well, we can actually create a queue by re-using the code from the above dual linked list. In fact, we can use the code above to create a stack too!

Inheritance basically tells the compiler for a given class definition – *also make the code automatically available to this other class*. Note in the class statement line, we use the keyword `inherits` and name the class.

From that point on, we can use all the `public` and `protected` methods found in the `DualLinkList` class in our new Queue1 class as if they were already defined!

That brings code re-use far past cut-n-paste or calling a procedure.

Also I think you will notice, that through code re-use – we focus only on the code needed to implement the queue in the class definition file. *Through code re-use we have defined a queue with far less code than the previous one at the beginning of this section!* With OOP an inheritance, we focus more on the differences that we want to make than trying to re-work the original code into the new code (as often happens in "code reuse.").

```
/* Example queue based on a dual link list */
/* Shows the power of inheritance and code reuse. */

class Queue1 inherits DualLinkList:

  /*********************************************************************/
  /* Provide a Push method to put data into the queue.                 */
  /*********************************************************************/

  method public void Push(input Data as character):

    InsertAfter(Data).

  end. /* method */

  /*********************************************************************/
  /* Provide a pop method to see and remove data from the queue.       */
  /*********************************************************************/

  method public character Pop ():

    define variable Data as character no-undo.

    if IsEmpty() then return ?.
    MoveEnd().
    Data = GetValue().
    DeleteCurrent().
    return Data.

  end. /* method */

end. /* class */


/********************** UNIT TEST CODE ******************************

define variable T as class Queue1 no-undo.
```

```
T = new Queue1().

T:Push("One").
T:Push("Two").
T:Push("Three").
T:DumpDataToFile("/tmp/1").

{&OUT} T:IsEmpty().

{&OUT} T:Pop().
{&OUT} T:Pop().
{&OUT} T:Pop().
{&OUT} T:IsEmpty().

delete object T.

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

Using A Class For A Collection Of Useful Functions

Classes can be composed of useful methods for commonly used functionality. They need not be dedicated to a sole data structure but as a library of useful tools. For example the manipulating strings or file items. (In fact, file systems should be considered a data structure! They are trees!)

Here you will find objects that can be simply reused over and over with different data. The point I am trying to make here, is that one does not need to create a new instance of an object for every use.

File Naming Tools (A singleton)

Here is a very simple object that combines together operations related to information about a file path on a windows or UNIX computer. Sometimes you are given a full path to a file and want to know the prefix, postfix, the simple name of the file, or the directory the file is found in. This is a nice little object to help parse that out for you.

There is something special about this class though – notice the `static` keyword. When you use static methods, variables, and properties one does not have to instantiate the class into an object. You simply start using the class as is. (See the Unit Test Code section for this code below.)

For a set of tools like this, a static class implementation (aka a singleton) will work just fine as it is unlikely you will need multiple copies of this object around.

A gotcha for this is that the code for the static class will remain in memory until the session is closed. Unlike an instantiated object, you cannot delete it from memory. Even if you re-compile the cls file, the new file is not recognized until you re-start the session. The good news is that you will only have one copy in memory at any time. So if you have plans on putting lots of static classes in memory – this is something to consider.

```
class FileTools:

  method public static character postfix (input filename as character):

    return substring (filename, r-index (filename, ".") + 1).

  end.

  method public static character basename (input filename as character):

    return replace (filename, path(filename), "").

  end.

  method public static character prefix (input filename as character):

    return entry (1, basename(filename), ".").

  end.
  method public static character path (input filename as character):

    return substring (filename, 1, r-index(filename, dirdelimiter())).

  end.

  method public static character dirdelimiter():

    if opsys = "UNIX" then return "/".
    else return "~\".

  end.

end.

/********************** UNIT TEST CODE ******************************


define variable FileName as character no-undo.

FileName = "\tmp\this.txt".

display FileTools:PostFix(FileName).
display FileTools:BaseName(FileName).
display FileTools:Prefix(FileName).
display FileTools:Path(FileName).

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

This is a pretty straight forward class – maybe you can extend it to include methods to replace the postfix, rename the base name, change the directory? By adding methods using the `FILE-INFO` handle you can extend it with information about a real file out on the file system. Perhaps you can even add methods to provide directory listings!

> *A way of thinking about object oriented programming is "embrace and extend." Embrace a bit of code and then using the new syntax and paradigm to extend that code into new functionality.*

URL Parser

Sometimes you find yourself needing to make socket communications (a bit on that later) when given a URL. It is becoming more and more common to identify sources of information with a URI beginning with `http:`, `https:`, `file:`, etc. However, the progress tools for socket communication require the pieces to be separate arguments to various items in the socket handle.

Something new to our objects so far is the inclusion of some of some error handling methods. While they are part of the object in this example, you will probably want to make an error class and inherit all new objects from there.

As you look over the error handling methods, you will notice not all of them use the public keyword. One of them uses the private keyword. Lets discuss what these things are doing.

When one marks a method (or variable) as `public` – it means that any code with visibility to the instance of that object can use that method.

When one marks a method (or variable) as `private` – it means only the routines making up the class can use those routines. In the example below – we want only the class to be setting the error condition. The programming using the object has no business working the objects error setting routines and we enforce that by encapsulating it in the class.

There is another keyword not used below but I will discuss it anyways. When one marks a method (or variable) as `protected` – it means only those classes inheriting that class can use the method or variable[9]. So it is the in-between spot of public and private.

> *When creating classes, I have found that unless the property is specifically private, it is worth while to make it protected. When you "embrace and extend" the class into a sub-class you will often find yourself going back and making properties protected so they can be used by the sub-class.*

9 Or temp-table. Often I have found temp-tables used in a class will need to be marked protected as they are important to the workings of any classes inheriting the class containing the temp-table.

This form of error handling is different from the error handling Progress provides. It is oriented to errors that your application will throw when it needs to.

This is a class that we will be making use of in a future class to show you how to instance and delete one object within another.

```
class amduus.web.urlparse:

  define public variable Host as character no-undo.
  define public variable Port as character no-undo.
  define public variable URI as character no-undo.
  define public variable Protocol as character no-undo.
  define public variable ErrMessage as character no-undo.

  /*********************************************************************/
  /* Performs an attempt at parsing URL to it's component parts.  Returns*/
  /* false if it cannot do it.                                          */
  /*********************************************************************/

  method public logical Parse (input ParseThis as character):

    ResetError().

    if not IsValidURL (ParseThis) then do:

      SetError ("001").
      return false.

    end.

    Protocol = entry (1, ParseThis, ":").
    ParseThis = replace (ParseThis, Protocol + "://", "").
    Host = entry  (1, ParseThis, "/").
    URI = replace (ParseThis, Host, "").

    if index (Host, ":") > 0 then do:

      Port = entry (2, Host, ":").
      Host = entry (1, Host, ":").

    end.
    else do:

      case Protocol:
        when "http" then Port = "80".
        when "https" then Port = "443".
        when "ftp" then Port = "21".
      otherwise do:
        SetError ("002").
        Host = ?.
        Protocol = ?.
        Port = ?.
        URI = ?.
        return false.
      end. /* otherwise */

    end. /* case */

  end. /* else */

  return true.

end. /* method Parse */

/**********************************************************************/
/* We are looking to prove form is ppp://machine/uri                  */
/**********************************************************************/

method public logical IsValidURL (input URL as character):

  if index (URL, "://") > 1 then return true.

  return false.

end. /* method */

/**********************************************************************/
/* Means of reseting the error state of the object.                   */
/**********************************************************************/

method public void ResetError ():

  SetError ("000").

end.

/**********************************************************************/
/* Convert error code into error message form and store in public     */
/**********************************************************************/

method private logical SetError (input ErrCode as character):

  case ErrCode:

    when "000" then ErrMessage = ErrCode + ":No Error".
    when "001" then ErrMessage = ErrCode + ":Not valid URL".
```

```
      when "002" then ErrMessage = ErrCode + ":Protocol Unknown".

    end. /* case */

  end. /* method */

  /*********************************************************************/
  /* method to return ErrMessage (some people prefer this.)            */
  /*********************************************************************/

  method public character GetError ():

    return ErrMessage.

  end.

end. /* class */
```

Here is an example on how to use it:

```
define variable Parser as class amduus.web.urlparse no-undo.

Parser = new amduus.web.urlparse().

{&OUT} Parser:Parse("http://localhost:8080/this/workshop") "<br>".

{&OUT} Parser:Host "<br>".
{&OUT} Parser:Port "<br>".
{&OUT} Parser:Protocol "<br>".
{&OUT} Parser:URI "<br>".
{&OUT} Parser:GetError().

delete object Parser.
```

What the above code does, is we instantiate the object with a `new` operation.

Then we use the Parse method in the object to chop up a given URL. We can then obtain those components via the properties that are available, ie Host, Port, Protocol, etc.

Finally we delete the object from memory.

Logging Tools

When developing code, often we want a log of what our algorithms are doing. The progress run time environment provides a means of logging but sometimes we want to use that for something else – like a global logging of what is going on.

This is a class that one can set the log level on, the name of the log file, the ability to put lines in with an EOL or without an EOL if you need to write to the same line repeatedly.

This is an example of a class that interacts with the file system. One of the bonuses of using classes to interact with the file system is one no longer needs to worry about the dreaded five stream limit. Simply make a "stream" class and use that.

```
class amduus.file.logfile:

  define stream mylog.

  define public variable date_format as character no-undo.
  define public variable time_format as character no-undo.
  define public variable loglevel as integer init 0 no-undo.

  define private variable logfilename as character no-undo.


  /*********************************************************************/
  /* Constructor: Helps us decide how things are going to be on start    */
  /*********************************************************************/

  constructor public logfile
  (input filename as character,
   input doappend as logical
  ):

    time_format = "hh:mm:ss".

    /* TODO: In the future, we want "mm-dd-yyyy" or "ddd-yyyy" or
     *       "mmm dd, yyyy" etc.  We need another object to help with
     *       this and to do it in what ever language one wants.
     */

    date_format = if session:date-format = "mdy" then "99-99-9999"
                  else "99-99-9999".

    if doappend then
    output stream mylog to value (filename) append.
  else
    output stream mylog to value (filename).

  logfilename = filename.

end. /* constructor */

/***********************************************************************/
/* When we do the “delete object N” statement, this code is run before */
/* the data and code are taken out of memory.                          */
/***********************************************************************/

destructor public logfile ():

end. /* destructor */

/***********************************************************************/
/* Write to the file with the stamp info and a end-of-line appended.   */
/***********************************************************************/

method public void println
(input level as integer,
 input lineoftext as character
):

  if level <= loglevel then
    put stream mylog unformatted string (today, date_format)
                                 " "
                                 string (time, time_format)
                                 " "
                                 program-name(2)
                                 " "
                                 lineoftext
                                 SKIP.

end.

/***********************************************************************/
/* Write to the file without prepended stamp and the end-of-line.      */
/***********************************************************************/

method public void print
(input level as integer,
 input lineoftext as character
):

  if level <= loglevel then
    put stream mylog unformatted lineoftext.
  end.

  /*********************************************************************/
  /* Close it up!                                                      */
  /*********************************************************************/

  method public void closelog():

    output stream mylog close.

  end.

  /*********************************************************************/
  /* Give the object the ability to clean up it's own logs.           */
  /*********************************************************************/

  method public void deletelog ():

    os-delete value(logfilename).

  end.

end. /* class */
```

And here is a quick example of using the object for logging:

```
define variable h as class amduus.file.logfile  no-undo.

h = new amduus.file.logfile ("/tmp/test.txt", no).
h:loglevel = 1.
h:println (1, "This is a line").
h:println (2, "Should not show").
h:closelog().

pause.

h:deletelog().

delete object h.
```

The pause is in there so you can view the log file before the `deletelog()` method removes it.

Code Generating Object

If one has ever programmed in Webspeed using the embedded method (or even not) – you will find yourself working with multiple languages – the ABL, HTML, and Javascript. Here is an example object that can help with making the programming all ABL based – a set of methods that will generate Javascript as needed when executed.

It's an example of how you can make source code that mixes up OOABL, HTML, and Javascript become more OOABL and less of the other source. Sometimes people get mixed up between what executes on the server and what executes on the client when they first start Webspeed programming.

```
class JavascriptTools:

  /**********************************************************************/
  /* Simply pop up an alert box with an OOABL oriented call.            */
  /**********************************************************************/

  method public static character AlertBox(input TextValue as character):

    define variable Javascript as character no-undo.

    Javascript = '<script language="javascript">'
               + 'alert("' + TextValue + '")</script>'.

    return Javascript.

  end. /* AlertBox */

  /**********************************************************************/
  /* Redirect the browser with an OOABL oriented call.                  */
  /**********************************************************************/

  method public static character Redirect(input NewURL as character):

    define variable Javascript as character no-undo.

    Javascript = '<script language="javascript">'
               + 'location.href="' + NewURL + '";</script>'.

    return Javascript.

  end. /* Redirect */

end. /* class */
```

Here is an example use in an E4GL program (embedded Speedscript.)

```
<html>
<?
JavascriptTools:AlertBox("You have no permission to enter!").
JavascriptTools:Redirect(ParameterManager:GetParameter("HomePage")).
?>
</html>
```

Using this idea can be done for other languages also like postscript, pdf, graphics libraries, and page rendering tools such as troff/groff/latex.

Objects That Are Useful In A Utilitarian Manner

There are many data structures out there. Some are a wee bit more complicated than the usual. Here we explore some tools that are a bit more sophisticated than simple data structures and libraries of useful functions. We start introducing some ideas on how to return errors back to a calling program as well rounding out more sophisticated functionality.

The point being made here, is that objects encapsulating knowledge about how to do something into methods named to do something can make a new or junior programmer functional very quickly. Often *simply using an object is much easier than learning the knowledge about how to make the object.*

XML Parsing – XQuery

A useful object is this XML node walking object XQuery[10]. Often one wants to read in XML documents and access them by the idea of "Give me the value of the first element of the fifth element of the first element in this XML document."

This node walking class is also an example of a basic data structure known as a tree.

For example, you have a configuration file for your application that reads like:

```
<?xml version="1.0" ?>
<Configurations>
  <IncomingRequests>
    <Logging>
      <LogLevel>1</LogLevel>
      <LogDir>/tmp/</LogDir>
    <Logging>
  </IncomingRequests>
  <SalesOrders>
    <Logging>
      <LogLevel>1</LogLevel>
      <LogDir>/tmp/</LogDir>
    <Logging>
  </SalesOrders>
</Configurations>
```

and you want to know the logging level for Sales Order activities.

Instead of parsing the document with awkward loops and 4GL atomic statements for SAX or DOM parsing, you can simply use the `WalkByPath()` method with an argument of `/Configurations[1]/SalesOrders[1]/LogLevel[1]` to retrieve the handle to the node with the data. One could say "In the first configurations tag, for the first Sales Order therein, give me the first Log Level node." The path is very similar to using a directory hierarchy to reach the data you desire to work with. The delimiter is the "/" sign, the tag name is simply the tag name, and the instance of the tag you want is placed between the "`[]`" symbols.

Since it is an object, you write it once and you are good to go for all other kinds of XML documents you may encounter – something you may not encounter using

10 If you are familiar with the XQuery standard you will know this object is very incomplete – but it is the beginnings of an XQuery object as I add to it over time. For now we keep it nice and simple.

specifically DOM/SAX parsing statements.

One basically creates an instance of the object and then loads in XML data with it's `LoadBy*` methods. We have multiple methods to load the object's XML data with so the object is flexible for many types of uses. (This is the power of an object – one object has many ways to do the same thing.)

Obtain the node with the `WalkByPath`() method. Once you have obtained the handle to the object, you can use the `NodeTextValue`() method to help get the TEXT value from that node passed into it and the `NodeAttrValue()` method to obtain any attribute that might be part of the element.

See below the listing for an example use:

```
class XQuery use-widget-pool:

  define private variable XDocument as handle no-undo.
  define public variable XMLPathDelimiter as character init "/" no-undo.
  define public variable ErrMessage as character no-undo.


  constructor public XQuery():


  end.

  /**********************************************************************/
  /* Given an already existing document handle, parse that puppy.       */
  /**********************************************************************/

  method public void LoadByHandle (input XDocumentHandle as handle):

    ResetError().

    if not valid-handle (XDocumentHandle) then
      SetError ("003").
    else
      XDocument = XDocumentHandle.

  end. /* method */

  /**********************************************************************/
  /* Load XML with long data.                                           */
  /**********************************************************************/
```

```
method public logical LoadByLongChar (input Data as longchar):

 define variable LoadOK as logical no-undo.
 define variable Strings as class nu-strings no-undo.

 ResetError().

 if not valid-handle (XDocument) then
   create X-Document XDocument.

 LoadOK = XDocument:Load ("longchar", Data, false) /* no-error */ .

 if not LoadOK then do:

   SetError("001").
   Strings = new nu-strings().
   Strings:long2log (Data).
   delete object Strings.

 end. /* if */

 return LoadOK.

end.

/*********************************************************************/
/* Given a file name, load the XML document we want to parse.         */
/*********************************************************************/

method public logical LoadByFile (input TheFileName as character):

 define variable LoadOK as logical no-undo.

 ResetError().

 if not valid-handle (XDocument) then
   create X-Document XDocument.

 LoadOK = XDocument:Load ("file", TheFileName, false) no-error.

 if not LoadOK then SetError("001").

 return LoadOK.

end. /* method */

/*********************************************************************/
/* Save the document as a file.                                      */
```

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

method public logical SaveAsFile (input TheFilePath as character):

  if valid-handle (XDocument) then do:
    XDocument:Save ("file", TheFilePath).
    return true.
  end.

  return false.

end. /* method */

/**********************************************************************/
/* Provide a means to reach the desired node by walking according to a */
/* path like /node1[child]/node2[child]... /books[1]/telephone[2]...   */
/**********************************************************************/

method public handle WalkByPath (input XMLPath as character):

  define variable NodeRef as handle no-undo.
  define variable SearchTagName as character no-undo.
  define variable SearchTagIter as integer no-undo.
  define variable SearchTagCurrIter as integer no-undo.
  define variable SearchPath as character no-undo.

  ResetError().

  create x-noderef NodeRef.

  /* Given the document, we want the start of the XML */

  XDocument:get-document-element(NodeRef).

  /* We don't want the / in front though it is semantically */
  /* more understandable.                                   */

  if XMLPath BEGINS "/" then XMLPath = substring (XMLPath, 2).

  /* Extract our root node and insure the pathing is cor-   */
  /* rect and matches the XML document.                     */

  SearchPath = entry (1, XMLPath, XMLPathDelimiter).
  SearchTagName = NodeName(SearchPath).
  SearchTagIter = ChildNumber(SearchPath).

  /* On root node, Tag iteration can only be 1 long! */

  if SearchTagIter > 1 then do:
    SetError("002").
    return ?.
  end.

  /* If they are looking for the root node, send it! */
  /* If all we have is root node in path and it does */
  /* not match, then send back unknown.              */

  if index (XMLPath, XMLPathDelimiter) > 0 then
    XMLPath = substring (XMLPath, index (XMLPath, XMLPathDelimiter) + 1).
  else
    XMLPath = "".

  if XMLPath = "" then
    if NodeRef:Name = SearchTagName then
      return NodeRef.  /* Name is same on path */
    else do:
      SetError("002").
      return ?. /* Name not same on path */
    end.

  /* We have more on the XMLPath to work with, we */
  /* need to dive into the children of the node.  */

  return WalkPath (NodeRef, XMLPath).

end. /* method */

/*************************************************************************/
/* Recursively walk the path as specified by the path.                  */
/*************************************************************************/

method public handle WalkPath
(input Node as handle,
 input XMLPath as character
):

  define variable ChildCounter as integer no-undo.
  define variable SearchIter as integer no-undo.
  define variable SearchTagName as character no-undo.
  define variable SearchTagIter as integer no-undo.
  define variable SearchPath as character no-undo.
  define variable ChildNode as handle no-undo.

  create X-NodeRef ChildNode.

  /* We do this for root level */

  SearchPath = entry (1, XMLPath, XMLPathDelimiter).
```

```
    SearchTagName = NodeName(SearchPath).
    SearchTagIter = ChildNumber(SearchPath).
/*
    message "XMLPath " XMLPath.
    message "SearchPath " SearchPath.
    message "SearchTagName " SearchTagName.
    message "SearchTagIter " SearchTagIter.
*/
    do ChildCounter = 1 to Node:Num-Children:
/*
      message "Accessing child " ChildCounter " of " Node:Num-Children "
children.".
*/
      Node:get-child (ChildNode,ChildCounter).
/*
      message "SearchPath " SearchPath.
      message "Child is type " ChildNode:SubType.
      message "NodeTextValue " NodeTextValue(ChildNode).
      message "Child Name " ChildNode:Name.
*/
      if ChildNode:Name = SearchTagName and ChildNode:SubType = "ELEMENT"
then do:
        SearchIter = SearchIter + 1.
       /*  message "Found iteration " SearchIter. */
      end.

      if SearchIter = SearchTagIter then do:
/*
        message "Found my desired iteration. Rewriting XMLPATH".
 */
        if index (XMLPath, XMLPathDelimiter) > 0 then
          XMLPath = substring (XMLPath, index (XMLPath, XMLPathDelimiter) +
1).
        else
          XMLPath = "".
/*
        message "Revised XMLPath " XMLPath.
 */
        if XMLPath <> "" then do:
          /* message "----- Diving into tree ------". */
          return WalkPath (ChildNode, XMLPath).
        end.
        else
          return ChildNode.

      end. /* if SearchIter */

    end. /* do */
```

```
  /* No such tag for our path name! */

  /* message "Firing failure". */
  SetError("002").
  return ?.

end. /* method */

/************************************************************************/
/* Give use the node name of the segment we desire.                    */
/************************************************************************/

method private character NodeName (input Name as character):

  return substring (Name, 1, index (Name, "[") - 1).

end. /* method */

/************************************************************************/
/* Obtain the child number of the path entry the caller desires.       */
/************************************************************************/

method private integer ChildNumber (input Name as character):

  Name = replace (Name, NodeName(Name), "").
  Name = replace (Name, "[", "").
  Name = replace (Name, "]", "").

  return integer(Name).

end. /* method */

/************************************************************************/
/* Return an atrribute on a given node.                                 */
/************************************************************************/

method public character NodeAttrValue
(input NodeRef as handle,
 input AttrName as character
):

  return NodeRef:Get-Attribute(AttrName).

end. /* method */

/************************************************************************/
/* Retrieve the CTEXT area of a node.                                   */
/************************************************************************/
```

```
method public character NodeTextValue (input NodeRef as handle):

  define variable TextNode as handle no-undo.
  define variable Document as handle no-undo.
  define variable Data as character no-undo.
  define variable ChildIter as integer no-undo.

  ResetError().

  if not valid-handle (NodeRef) then return ?.

  if NodeRef:SubType <> "ELEMENT" then return ?.

  create x-noderef TextNode.

  /* Need to walk children because may be comment, etc. */

  ChildIter = 1.
  do while ChildIter <= NodeRef:num-children:
    NodeRef:get-child(TextNode, ChildIter).
    if TextNode:subtype = "TEXT" then leave.
  end.
  Data = TextNode:node-value.

  delete object TextNode.

  return Data.

end. /* method */

/*********************************************************************/
/* Opportunity to find out if an error happened or not.              */
/*********************************************************************/

method public character GetError():

  return ErrMessage.

end. /* method */

/*********************************************************************/
/* Set the error code to human friendly and make available to public.  */
/*********************************************************************/

method private character SetError(input ErrCode as character):

  case ErrCode:

    when "000" then ErrMessage = ErrCode + ":No Error".
```

```
      when "001" then ErrMessage = ErrCode + ":No Such File".
      when "002" then ErrMessage = ErrCode + ":Wrong Path For XML Schema".
      when "003" then ErrMessage = ErrCode + ":Bad XML Document Handle".

    end. /* case */

  end. /* method */

  /*******************************************************************/
  /* Tool for implementing programmer to reset the internal error state  */
  /* of the object.                                                     */
  /*******************************************************************/

  method public void ResetError():

    SetError("000").

  end. /* method */

end. /* class */
```

Here is some example use of the Xquery object.

Here is an example use of it. Lets say we want to access information in the following XML document:

```
<?xml version="1.0" ?>
<root>
  <name record="0x001">
    <firstname>Scott</firstname>
    <lastname>Auge</lastname>
  </name>
  <name record="0x002">
    <firstname>Lorena</firstname>
    <lastname>Brunswick</lastname>
  </name>
</root>
```

We load in an XML file containing some information about people in what amounts to a hierarchical database[11].

Then we navigate to the node containing the information we want with our path. Once receiving it – we use the node as an argument to other methods that look at the text

11 What's old is new again in computing.

and attribute information on the node.

Finally we clean up with the delete object statement.

```
def var t as class XQuery no-undo.
def var x as handle no-undo.

t = new XQuery().

t:LoadByFile("/tmp/test.xml").

x = t:WalkByPath("/root[1]/name[1]/firstname[2]").
display t:ErrMessage with frame a1.
display t:NodeTextValue(x) t:ErrMessage format "x(30)" with frame a.

x = t:WalkByPath("/root[1]/name[1]/lastname[1]").
display t:ErrMessage with frame b1.
display t:NodeTextValue(x)  t:ErrMessage with frame b.

x = t:WalkByPath("/root[1]/name[2]").
display t:ErrMessage with frame c1.
display t:NodeAttrValue(x, "record")  t:ErrMessage with frame c.

delete object t.
```

A few notes. You will notice we have use-widget-pool on the definition of the class. Because we do a lot of dynamic work in the class, we want to manage all that information in the widget pool closets to the class – the class it's self. This is a way to protect yourself from memory leaks. It also means when you destroy your object, you risk destroying data coming out of that object in handles (aka the node reference!)

This is an object that will definitely be in the upcoming Amduus Object Library. It will probably have more advanced routines that will allow the user to use the path not only in absolute terms to find data but to have a search syntax returning lists of nodes matching the query.

Socket Object

There is a bit of housekeeping that needs to be done with socket communications for the ABL. Here is an example of a socket object that encapsulates all the handles, memory pointers, etc. within a single object. The object becomes as easy to use as calling a few methods and sending/receiving the data.

It also encapsulates the knowledge about socket programming so as the developer doesn't really need to know about some of the nitty gritty details. All they need to focus on is which host, port, and what data to send back and forth. The object will not only encapsulate functionality and data (see the public and private tags) – *it also encapsulates knowledge about difficult things* (like network programming which is far different from database programming.)

```
class amduus.network.socket:

  define public variable Revision as character
  initial "$Revision$" no-undo.

  define public variable ErrMessage as character no-undo.
  define private variable hSocket as handle no-undo.
  define private variable Heap as memptr no-undo.
  define public variable CRLF as character no-undo.

  /*********************************************************************/
  /* Constructor.                                                      */
  /*********************************************************************/

  constructor public socket():

     create socket hSocket.
     set-size (Heap) = 1024.

     /* Many protocols like this so simply make it a constant */

     CRLF = chr(13) + chr(10).

  end. /* constructor */

  /*********************************************************************/
  /* Destructor.  Housekeeping here!                                   */
  /*********************************************************************/

  destructor public socket():
```

```
  delete object hSocket.
  set-size(Heap) = 0.

end. /* destructor */

/*********************************************************************/
/* Way for the user to use a Get* for the error state of the object.   */
/*********************************************************************/

method public character GetError():

  return ErrMessage.

end. /* method */

/*********************************************************************/
/* Set the error state of the object - only by the object!           */
/*********************************************************************/

method private void SetError(input ErrCode as character):

  case ErrCode:

    when "000" then ErrMessage = ErrCode + ":No Error".
    when "001" then ErrMessage = ErrCode + ":Already Connected".
    when "002" then ErrMessage = ErrCode + ":Not Connected".
    when "003" then ErrMessage = ErrCode + ":No Data On Socket".

  end. /* case */

end. /* method */

/*********************************************************************/
/* Allow implementor to reset the error state for next operation.      */
/*********************************************************************/

method public void ResetError ():

  SetError ("000").

end. /* method */

/*********************************************************************/
/* Open a connection to the remote system.                           */
/*********************************************************************/

method public logical OpenConnection
(input Host as character,
 input Port as character,
 input UseSSL as logical):

 define variable ConnectionParameter as character no-undo.

 ResetError().

 if IsConnected() then do:

   SetError ("001").
   return false.

 end.

 ConnectionParameter = " -H " + Host
                     + " -S " + Port.

 if UseSSL then ConnectionParameter = ConnectionParameter + " -ssl".

 hSocket:set-socket-option("so-linger", "false").
 hSocket:set-socket-option("so-rcvtimeo", "2").
 hSocket:connect(ConnectionParameter).

 return IsConnected().

end. /* method */

/************************************************************************/
/* Close the connection to the remote server.                           */
/************************************************************************/

method public logical CloseConnection ():

  ResetError().

  if hsocket:connected() then hsocket:disconnect().
  return true.

end. /* method */

/************************************************************************/
/* Determine if we are connected or not.                                */
/************************************************************************/

method public logical IsConnected():

  ResetError().

  return hsocket:connected().
  end. /* IsConnected () */

  /*********************************************************************/
  /* A means to write directly from memory to the tcp cache.           */
  /*********************************************************************/

  method public logical WriteBinary (input Heap as memptr, input l as
integer):

    ResetError().

    if not IsConnected() then do:
      SetError ("002").
      return false.
    end.

    hSocket:write (Heap, 1, l).

  end. /* method */

  /*********************************************************************/
  /* Method to write text to the tcp cache.                            */
  /*********************************************************************/

  method public logical WriteText (input Data as character):

    define variable LocalHeap as memptr no-undo.

    ResetError().

    if not IsConnected() then do:
      SetError ("002").
      return false.
    end.

    set-size(LocalHeap) = length (Data) + 1.

    put-string (LocalHeap, 1) = Data.

    hSocket:write (LocalHeap, 1, length(Data)).

    set-size(LocalHeap) = 0.

  end. /* WriteData */

  /*********************************************************************/
  /* Determine number of bytes waiting on the tcp cache for input.     */
  /*********************************************************************/
```

```
method public integer GetBytesAvailable():

  ResetError().

  if not IsConnected() then do:
    SetError ("002").
    return ?.
  end.

  return hsocket:get-bytes-available().

end.

/*********************************************************************/
/* Some polling implementations will need to pause a certain amount of */
/* of time but the PAUSE statement is really to large a period.  So we */
/* use milliseconds in this one.                                       */
/*********************************************************************/

method public void PauseInMilliseconds
(input TimeInMilliseconds as integer):

  define variable StartTime as int64 no-undo.

  StartTime = etime.

  do while etime - StartTime < TimeInMilliseconds:
  end.

end.

/*********************************************************************/
/* Provide a means to read text oriented data from the socket.  For    */
/* peices of data you will probably have to call this multiple times   */
/* while using GetBytesAvailable() in the implementing program.        */
/*********************************************************************/

method public character ReadText ():

  define variable ReadCache as character no-undo.
  define variable HeapUsed as integer no-undo.

  ResetError().

  if not IsConnected() then do:
    SetError ("002").
    return ?.
  end.
  /* It is possible to have more in the tcp cache than the size of a */
  /* 4gl character variable - so nibble what we need.               */

  HeapUsed = minimum (1024, hsocket:get-bytes-available()).

  if HeapUsed = 0 then do:
    SetError("003").
    return "".
  end.

  hSocket:read (Heap, 1, HeapUsed).

  ReadCache = get-string (Heap, 1, HeapUsed).

  return ReadCache.

end. /* method */

/***********************************************************************/
/* Provide a means to read directly from tcp cache into memory and     */
/* return to the implementing program.                                 */
/***********************************************************************/

method public memptr ReadBinary
(output Heap as memptr,
 output HeapSize as integer
):

  ResetError().

  if not IsConnected() then do:
    SetError ("002").
    return ?.
  end.

  HeapSize = hsocket:get-bytes-available().

  if HeapSize = 0 then do:
    SetError("003").
    return ?.
  end.

  set-size(Heap) = HeapSize.
  hSocket:read (Heap, 1, HeapSize).

  return Heap.

end. /* method */
end. /* class */
```

Here is a simple example of how to use the object to obtain a web page. Don't confuse it for an HTTP client. In the next section, we wrap a couple of objects together (the socket object and the url parser object) to make an HTTP object.

```
define variable SocketTools as class amduus.network.socket no-undo.
define variable CRLF as character no-undo.

message "Starting program".
CRLF = chr(13) + chr(10).

SocketTools = new amduus.network.socket().

SocketTools:OpenConnection ("gaius", "80", no).

{&OUT} "<pre>".
{&OUT} SocketTools:ErrMessage SKIP.
{&OUT} SocketTools:IsConnected() SKIP.

SocketTools:WriteText ("GET / HTTP/1.1" + CRLF).
SocketTools:WriteText ("Host: gaius" + CRLF + CRLF).
{&OUT} SocketTools:ErrMessage skip.

do while SocketTools:IsConnected():

  if SocketTools:GetBytesAvailable() > 1 then
    {&OUT} SocketTools:ReadText().
  else
    pause 1.

end.

{&OUT} "</pre>".

SocketTools:CloseConnection ().

delete object SocketTools.
```

HTTP Object

Here is an example of an object that builds off of other objects. It is an easy to use http client that will dump information into a text file. It can easily be adapted to place the information into a `longchar` too. (Some web pages will blow out a simple character variable.)

I think you will find it delightfully easy to comprehend and it shows the reduction in code lines needed to accomplish something when using objects.

Also it helps identify when an object should be instanced and used compared to simply inheriting the class for that object into the new class. If it is not directly related to the domain of the object – you should use an instance instead of trying to inherit it.

(Note this program is a little fast and loose when it comes to handling errors. As any college professor would say – I leave that exercise to the student. :))

```
class amduus.network.http:

  define private variable obj_socket as class amduus.network.socket no-
undo.

  define public variable error_message as character no-undo.

  define private variable crlf as character no-undo.

  /*********************************************************************/
  /* Get our goodies together.                                         */
  /*********************************************************************/

  constructor public http ():

    obj_socket = new amduus.network.socket().

    crlf = chr(13) + chr(10).

  end. /* constructor */


  /*********************************************************************/
  /* Housekeeping.                                                     */
  /*********************************************************************/

  destructor public http():
    delete object obj_socket.

  end.


  /*********************************************************************/
  /* Do a get that will place the data into a file.                    */
  /*********************************************************************/

  method public logical get_text_to_file
  (input url as character,
   input filename as character
  ):


    define variable urltool as class amduus.network.urlparse no-undo.

    urltool = new amduus.network.urlparse().
    urltool:Parse(url).

    obj_socket:OpenConnection(urltool:Host, urltool:Port, no).

    obj_socket:WriteText("GET " + urltool:URI + " HTTP/1.1" + crlf).
    obj_socket:WriteText("Host: " + urltool:Host + crlf).
    obj_socket:WriteText(crlf).

    output to value (filename).

    do while obj_socket:IsConnected():

      if obj_socket:GetBytesAvailable() > 1 then
        put unformatted obj_socket:ReadText().
      else
        pause 1.

    end.

    output close.

    obj_socket:CloseConnection().

    delete object urltool.

  end.

end.
```

Here are some experiments to play with the code. Since the URL parser is used often, one could actually instance one in the constructor. That would be a speed increase as it wouldn't need to be instanced each time the `get_text_to_file()` method was called. Also one could create a `get_text_to_long()` method that would simply keep adding the received data to a long character value and return it to what ever programming was using the http client.

Here is an example use:

```
define variable obj_http as class amduus.network.http no-undo.

obj_http = new amduus.network.http().
if obj_http:get_text_to_file ("http://gaius/", "/tmp/page.txt") = false
then
  {&OUT} "An error occurred.".
else
  {&OUT} "OK".

delete object obj_http.
```

Some IP Tools

Sometimes you will find yourself dealing with security from a “Who can connect?” aspect. This is a quick little object that contains methods to help with breaking up IP addresses and a match tool one can run IP masks through to determine if the IP matches.

When writing tools like this – it is really important to put some forethought into the class. You will notice that a lot of code uses IPv4 as part of it's name. This is because we all know there is an IPv6 already out and coming into accepted use. One can simply add the parsing routines for this new protocol to the object and still have the same object available for what ever is being done with either protocol.

```
class iptools:

  define public variable ipv4classa as character no-undo.
  define public variable ipv4classb as character no-undo.
  define public variable ipv4classc as character no-undo.
  define public variable ipv4classd as character no-undo.

  method public void DisassembleIPv4Address (input IP as character):

    ipv4classa = CanonicalForm(entry(1, IP, ".")).
    ipv4classb = CanonicalForm(entry(2, IP, ".")).
    ipv4classc = CanonicalForm(entry(3, IP, ".")).
    ipv4classd = CanonicalForm(entry(4, IP, ".")).

  end. /* method */

  method public character IdentifyIPVersion (input IP as character):

    if index(IP, ".") > 0 then return "V4".
    if index(IP, ":") > 0 then return "V6".

  end. /* method */

  method public character CanonicalForm (input IPClass as character):

    return string(integer(IPClass), "999").

  end.

  /* Do the numbers match an exact number or * per class? */

  method public logical DoesMatchMask
  (input IP as character,
   input Mask as character):

   define variable IsMatch as logical no-undo.
   define variable NetworkClass as character no-undo.

   IsMatch = no.

   DisassembleIPv4Address(IP).

   NetworkClass = entry(1, Mask, ".").
   IsMatch = (NetworkClass = "*" or CanonicalForm (NetworkClass) =
ipv4classa).

   NetworkClass = entry(2, Mask, ".").
   IsMatch = ((NetworkClass = "*" or CanonicalForm (NetworkClass) =
ipv4classb) and IsMatch).

   NetworkClass = entry(3, Mask, ".").
   IsMatch = ((NetworkClass = "*" or CanonicalForm (NetworkClass) =
ipv4classc) and IsMatch).

   NetworkClass = entry(4, Mask, ".").
   IsMatch = ((NetworkClass = "*" or CanonicalForm (NetworkClass) =
ipv4classd) and IsMatch).

   return IsMatch.

  end. /* method */

end. /* method */
```

Here are some example uses of the object:

```
define variable I as class iptools no-undo.

I = new iptools().

{&OUT} I:IdentifyIPVersion ("127.0.0.1") " | ".
I:DisassembleIPv4Address("127.0.0.1").
{&OUT} I:ipv4classa " | ".
{&OUT} I:ipv4classb " | ".
{&OUT} I:ipv4classc " | ".
{&OUT} I:ipv4classd " | ".

{&OUT} I:DoesMatchMask("127.0.1.1", "127.*.*.1").

delete object I.
```

Serialization – Saving an object

When one wants to save a variable or temp-table, often one simply writes it to a file. But then, how does one actually save an object?

Pretty much the same way. Only instead of holding the values of one variable, we construct a method to saving the values of all the data portions of the object instance in the method.

In order to do this, we use this class which will construct an XML document for us. The implementing programmer can then save off the XML document file or place it into a database record. When that instance of the object is needed at a later time – it can be easily recalled.

> *This class is used by the class that will implement the serialization – this class can't do the serialization it's self for a sub-class that implements it. It is a tool for the programmer to provide a serialization service in the class.*

Of particular interest is one could even save it off as a long char and pass it over to another computer, un-serialize the object and that object would be a mirror of what was happening or set up on the originating system. This is a way to make objects in a distributed application system[12].

If the object is simple and does not use any database entries, one can simply save all the properties that compose the object. If the object does use the database, then one has two choices. One may want to store the key information to re-finding the records when the object is instantiated again with the serialized data; or the current data in the object is more important than the data in the database and so the data must be stored for un-serialization.

Below is a tool for helping to construct the XML to store the serialized version of the data into and to obtain the un-serialized data out of. In usual use, for any objects you need to serialize or unserialize, you will want to inherit from this class.

12 Applications that are distributed across two or more hardware systems.

```
/* Objects that need to be serialized and unserialized will    */
/* find this useful.                                           */

/* Note the use of the %1% placeholder - we build by replacing */
/* this macro with the serialized data and then reinserting    */
/* the macro at the end of the newly serialized data.          */

&GLOBAL-DEFINE PLACEHOLDER "%1%"

class Serializer:

  define protected variable SerializedData as longchar no-undo.
  define private variable XQuery as class XQuery no-undo.

  /*******************************************************************/
  /* Initialize our object upon instantiating.                       */
  /*******************************************************************/

  constructor public Serializer ():

    Reset().

    XQuery = new XQuery().

  end. /* constructor */

  /*******************************************************************/
  /* When deleting, we need to get rid of our XQuery object.         */
  /*******************************************************************/

  destructor public Serializer ():

    delete object XQuery.

  end. /* destructor */

  /*******************************************************************/
  /* Reset the object for re-use.                                    */
  /*******************************************************************/

  method public void Reset():

    SerializedData = "<?xml version=~"1.0~" ?>"
                   + "<serialized>" + {&PLACEHOLDER} + "</serialized>".

  end. /* method */

  /*******************************************************************/
  /* Return the data in fully serialized form.                       */
  /*******************************************************************/

  method public longchar GetSerialized():

    return replace (SerializedData, {&PLACEHOLDER}, "").

  end. /* method */

  /*******************************************************************/
  /* Add data to the serialized data.                                */
  /*******************************************************************/

  method public void AddSerializedData
  (input DataName as character,
   input DataValue as character,
   input DataType as character
  ):

    define variable NewSerialized as longchar no-undo.

    NewSerialized = '<' + DataName
                  + ' type="' + DataType + '">'
                  + DataValue
                  + '</' + DataName + '>'
                  + {&PLACEHOLDER}.

    SerializedData = replace (SerializedData, {&PLACEHOLDER},
NewSerialized).

  end. /* method */

  /*******************************************************************/
  /* Allow the caller to set the serialized data for use with        */
  /* GetSerializedData.                                              */
  /*******************************************************************/

  method public void SetSerialized (input Data as longchar):

    SerializedData = Data.

  end. /* method */

  /*******************************************************************/
  /* After using SetSerialized, one can parse it with this.  Do not  */
  /* use until after use of GetSerialized because the internal rep   */
  /* will have the placeholder in it screwing the XML load.          */
  /*******************************************************************/

  method public logical GetSerializedData
  (input DataName as character,
   output DataType as character,
   output DataValue as character
  ):

    define variable NodeRef as handle no-undo.

    create x-noderef NodeRef.

    XQuery:LoadByLongChar(SerializedData).
    NodeRef = XQuery:WalkByPath("/serialized[1]/" + DataName + "[1]").
    if NodeRef = ? then return false.
    DataValue = XQuery:NodeTextValue(NodeRef).
    DataType = XQuery:NodeAttrValue(NodeRef, "type").
    return true.

  end.  /* method */

end. /* class */
```

Here is some example code used for testing the Serialization object. In real use – this code wouldn't be so procedural as below but would be contained in methods called `Serialize()` and `Unserialize()`. Serialize could return a `longchar` that contains the XML to be stored or transported. Unserialize would accept the `longchar` and chop it up into the variable's values. Unfortunately – since classes can have such a diverse set of data you will need to write `Serialize()` and `Unserialize()` methods specific to the function.

In this test, the "object" has three properties – `This`, `That`, and `CurrentNumber`. We use the `AddSerializedData()` function to add these data to the serialization construct (basically an XML document.)

When we are ready to store the serialization of the object, we use the `GetSerialized()` function to obtain the XML document containing all the data.

Now you have an object that you wish to restore to its previous value. Simply instance that object and call method `Unserialize()` that you would write custom to that object.

The `Unserialize()` method would use the Serializer class to reset the variables. First one would obtain the XML from a file or database record. Then use the `SetSerialized()` function to load the XML saved with the `GetSerialized()` into the object. From there, you can use the

`GetSerializedData()` method to pull back the variables as you need.

Below is some example code:

```
/* Place this in a method */

define variable t as class Serializer no-undo.

t = new Serializer().

t:AddSerializedData("This", "Scott", "character").
t:AddSerializedData("That", "Craig", "character").
t:AddSerializedData("CurrentNumber", "1", "integer").

/* Prep for yanking values out */

t:SetSerialized(t:GetSerialized()).

define variable DataVal as character.
define variable DataType as character.

t:GetSerializedData("That", output DataType, output DataVal).
display "That" DataType DataVal with frame b.
down with frame b.
t:GetSerializedData("CurrentNumber", output DataType, output DataVal).
display "CurrentNumber" DataType DataVal with frame b.


delete object t.

Example Data

<serialized><This type="character">Scott</This><That
type="character">Craig</That><CurrentNumber
type="integer">1</CurrentNumber></serialized>
```

You can get much more sophisticated than above – you will notice that the type is available should you want to use a dynamic temp table in some way.

If you need to serialize a table, then I would create an entry like "`tablename.length`" such as "`Customers.length`" to note the number of entries. Then use names like `tablename.fieldname.n` for each piece of data in the table – so for row 1 one would name the data "`Customer.Name.1`" so it is easier to understand how much data you will be pulling back.

Objects That Interact With The Database

We've seen some objects that are pretty independent of transactional interaction with a database. Here we explore some simple objects that do interact with the database.

Dynamic Query Object

Anyone familiar with my DynTookKit programming include made available at:

http://www.oehive.org/amduus/

should recognize this as the class version of the functionality.

In short, this class makes the use of dynamic queries simple to use. One simply instantiates the class, throw it the `FOR` statement you wish it to work with, navigate with some iterating methods and pull the results out.

So this is an example of using an object to read from the database. It does have some routines for writing to the database but I leave those to the reader to experiment with.

Following is the code and an example for using it.

```
/*****************************************************************************/
/* This is a re-write from dyntoolkit.i to a class oriented dynamic query    */
/* manager.                                                                  */
/*****************************************************************************/

CLASS DynToolKit:

  DEFINE PUBLIC VARIABLE RCSVersion_dyntoolkit_i AS CHARACTER
  INIT "$Id: dyntoolkit.i,v 1.9 2006/10/20 04:50:09 sauge Exp sauge $" NO-UNDO.

  /*****************************************************************************/
  /* When the application runs against one database, it might be worth it to   */
  /* set this preprocessor to NO to prevent additional code running that does  */
  /* not need to run.  Leaving it as YES will not effect single DB applic-     */
  /* actions - merely that it will run through some code that it doesn't need  */
  /* to.  I would leave it as YES, but I know there are performance junkies out*/
  /* there. See documentation for more information.                            */
  /*****************************************************************************/

  DEFINE PUBLIC VARIABLE UseMultiDBs AS LOGICAL INIT YES NO-UNDO.

  /*****************************************************************************/
  /* When using an Oracle DB, sometimes the schema holder pops up in the list. */
  /* It brings everything to a grinding halt with a 916 error.  Choose to ig-  */
  /* nor this DB(s).                                                           */
  /*****************************************************************************/

  DEFINE PUBLIC VARIABLE IgnoreDBNames AS CHARACTER NO-UNDO.

  /*****************************************************************************/
  /* We keep a list of our dynmically created objects in this temp-table so    */
  /* we can dynamically clean house.  One of the bad things though - is that   */
  /* unless this table is made global - the METHOD PUBLICs only run in the scope of
*/
  /* of this table.                                                            */
  /****************************************************************************/

  DEFINE TEMP-TABLE ttDynToolKit
    FIELD QryHndl AS HANDLE     /* Use the table with multiple queries   */
    FIELD TblHndl AS HANDLE     /* How we reach the buffers of the query */
    FIELD TblName AS CHARACTER. /* For the getvalue METHOD PUBLICs.           */

  /****************************************************************************/
  /* Use these for error reporting                                             */
  /****************************************************************************/

  DEFINE PUBLIC VARIABLE cDyn_ErrCode AS CHARACTER NO-UNDO.
  DEFINE PUBLIC VARIABLE cDyn_ErrMsg  AS CHARACTER NO-UNDO.

  /****************************************************************************/
  /* Determine the tables available in the given query.                        */
  /****************************************************************************/

  METHOD PUBLIC CHARACTER dyn_gettables (INPUT cQry AS CHARACTER):

    DEFINE VARIABLE iIter        AS INTEGER NO-UNDO.
    DEFINE VARIABLE iIterMax     AS INTEGER NO-UNDO.

    DEFINE VARIABLE cToken       AS CHARACTER NO-UNDO.

    DEFINE VARIABLE cTblList     AS CHARACTER INIT "" NO-UNDO.

    DEFINE VARIABLE iDBSeq       AS INTEGER NO-UNDO.
    DEFINE VARIABLE lIsTable     AS LOGICAL NO-UNDO.

    /************************************************/
    /* Determine the number of tokens in our query */
    /************************************************/

    ASSIGN iIterMax = NUM-ENTRIES(cQry, " ").

    /************************************************/
    /* Check which tokens are files in the DB.     */
    /************************************************/

    /***************************************************/
    /* This code runs best on multi DB apps.          */
    /* SGA: Inspired by Dayne May daynem @ linx.com.au*/
    /***************************************************/

    IF UseMultiDBs THEN DO:

      TOKEN_LOOP:
      DO iIter = 1 TO iIterMax:

        ASSIGN cToken = ENTRY(iIter, cQry, " ").
    DB_LOOP:
    DO iDBSeq = 1 TO NUM-DBS:

      IF CAN-DO(IgnoreDBNames, LDBNAME ( iDBSeq )) THEN NEXT.

      CREATE ALIAS TEMPDB FOR DATABASE VALUE ( LDBNAME ( iDBSeq ) ).

      /*****************************************************/
      /* Because CREATE ALIAS statement doesn't take affect */
      /* for the current compilation, split out the FIND.   */
      /*****************************************************/

      RUN dyn_findinschema.p
        (INPUT  cToken,
         OUTPUT lIsTable).


      IF lIsTable AND NOT CAN-DO(cTblList, cToken) THEN DO:
        ASSIGN cTblList = cTblList + "," + cToken.
        NEXT TOKEN_LOOP.
      END.

      /* Useful for debugging on multi databases
      ELSE DO:
        MESSAGE "Could Not Find " cToken.
      END.
      */

    END. /* DO iDBSeq = 1 TO NUM-DBS */

  END. /* DO iIter = 1 TO iIterMax */

END. /* IF UseMultiDBs */

/*********************************************/
/* This code runs best on single DB apps.      */
/*********************************************/

ELSE DO:

  DO iIter = 1 TO iIterMax:

    ASSIGN cToken = ENTRY(iIter, cQry, " ").

    IF CAN-FIND(FIRST _File WHERE _File._File-Name = cToken) THEN DO:

      IF NOT CAN-DO(cTblList, cToken) THEN ASSIGN cTblList = cTblList + "," +
cToken.

    END. /* IF CAN-FIND() */

  END. /* DO iIter = 1 TO iIterMax */

END. /* ELSE IF UseMultiDBs */
```

```
  /**********************************************/
  /* We always end up with a closing , from      */
  /* above so prune that out.                    */
  /**********************************************/

  IF cTblList > "" THEN ASSIGN cTblList = SUBSTRING(cTblList, 2).

  RETURN cTblList.

END. /* METHOD PUBLIC GetTables */


/**************************************************************************/
/* Open up a dynamic query and return a handle to that query.             */
/**************************************************************************/

METHOD PUBLIC HANDLE dyn_open (INPUT cQry AS CHARACTER):

  DEFINE VARIABLE cBufferList AS CHARACTER NO-UNDO.
  DEFINE VARIABLE cBufferName AS CHARACTER NO-UNDO.

  DEFINE VARIABLE iIter       AS INTEGER NO-UNDO.
  DEFINE VARIABLE iMaxIter    AS INTEGER NO-UNDO.

  DEFINE VARIABLE hQryHndl    AS HANDLE NO-UNDO.

  DEFINE VARIABLE hTblHndl    AS HANDLE NO-UNDO.

  DEFINE VARIABLE lStatus     AS LOGICAL NO-UNDO.

  /*****************************************************/
  /* Prep our error variables in case something goes bad. */
  /*****************************************************/

  ASSIGN cDyn_ErrCode = "000"
         cDyn_ErrMsg = "No Error:" + cQry.

  /*****************************************************/
  /* Create a query and do that memory stuff.          */
  /*****************************************************/

  CREATE QUERY hQryHndl.

  /*****************************************************/
  /* Determine buffers needed for our query.           */
  /*****************************************************/

  ASSIGN cBufferList = dyn_gettables(cQry)
         iIter = 0.
      iMaxIter = NUM-ENTRIES(cBufferList).

  /* MESSAGE "cBufferList = " cBufferList. */

  IF cBufferList = "" THEN DO:
```

```
  ASSIGN cDyn_ErrCode = "102"
         cDyn_ErrMsg = "Could Not Determine Tables:" + cQry.

  RETURN ?.

END. /* IF NOT lStatus */

/*****************************************************/
/* Allocate buffer space and "remember" them.         */
/*****************************************************/

DO iIter = 1 TO iMaxIter:

  ASSIGN cBufferName = ENTRY(iIter, cBufferList).

  /* MESSAGE "cBufferName = " cBufferName. */

  CREATE BUFFER hTblHndl FOR TABLE cBufferName.

  CREATE ttDynToolKit.

  ASSIGN ttDynToolKit.QryHndl = hQryHndl
         ttDynToolKit.TblHndl = hTblHndl
            ttDynToolKit.TblName = cBufferName.

END. /* FOR iIter = */

/*****************************************************/
/* Lets assign our buffers to the query               */
/*****************************************************/

FOR EACH ttDynToolKit NO-LOCK
WHERE ttDynToolKit.QryHndl = hQryHndl:

  hQryHndl:ADD-BUFFER(ttDynToolKit.TblHndl).

END. /* FOR EACH */

/*****************************************************/
/* Let's open er up.                                  */
/*****************************************************/

ASSIGN lStatus = hQryHndl:QUERY-PREPARE(cQry) NO-ERROR.

IF NOT lStatus THEN DO:

  ASSIGN cDyn_ErrCode = "100"
         cDyn_ErrMsg = "Could Not Prepare:" + cQry.

  RETURN ?.

END. /* IF NOT lStatus */

ASSIGN lStatus = hQryHndl:QUERY-OPEN() NO-ERROR.
```

```
  IF NOT lStatus THEN DO:

    ASSIGN cDyn_ErrCode = "101"
           cDyn_ErrMsg = "Could Not Open:" + cQry.

    RETURN ?.

  END. /* IF NOT lStatus */

  /*******************************************************/
  /* Return a handle to the goods.                       */
  /*******************************************************/

  RETURN hQryHndl.

END. /* METHOD PUBLIC dyn_open */

/****************************************************************************/
/* Delete all the buffers and then the query (or what ever order!)          */
/* If you do not call this - YOU WILL HAVE MEMORY LEAKS.                    */
/****************************************************************************/

METHOD PUBLIC LOGICAL dyn_close (INPUT hQryHndl AS HANDLE):

  hQryHndl:QUERY-CLOSE().

  FOR EACH ttDynToolKit EXCLUSIVE-LOCK
  WHERE ttDynToolKit.QryHndl = hQryHndl:

    DELETE OBJECT ttDynToolKit.TblHndl.

  END. /* FOR EACH */

  DELETE OBJECT hQryHndl.

END. /* METHOD PUBLIC dyn_close */

/****************************************************************************/
/* Like named wrapped around get next method.                               */
/****************************************************************************/

METHOD PUBLIC LOGICAL dyn_next (INPUT hQryHndl AS HANDLE):

  hQryHndl:GET-NEXT.

END. /* METHOD PUBLIC dyn_next */

/****************************************************************************/
/* Like named wrapped around get prev method.                               */
/****************************************************************************/

METHOD PUBLIC LOGICAL dyn_prev (INPUT hQryHndl AS HANDLE):

  hQryHndl:GET-PREV.
```

```
END. /* METHOD PUBLIC dyn_prev */

/****************************************************************************/
/* Slip to the end of the result set.                                      */
/****************************************************************************/

METHOD PUBLIC LOGICAL dyn_last (INPUT hQryHndl AS HANDLE):

  hQryHndl:GET-LAST.

END.

/****************************************************************************/
/* Slip to the beginning of the result set.                                */
/****************************************************************************/

METHOD PUBLIC LOGICAL dyn_first (INPUT hQryHndl AS HANDLE):

  hQryHndl:GET-FIRST.

END.

/****************************************************************************/
/* Determine if we are at the end or before start of the query.            */
/****************************************************************************/

METHOD PUBLIC LOGICAL dyn_qoe (INPUT hQryHndl AS HANDLE):

  RETURN hQryHndl:QUERY-OFF-END.

END.

/****************************************************************************/
/* Pull a string version of the data off the field buffer.                 */
/* The TblFld is meant to be called as table.field like in usual 4GL       */
/* Right now this doesn't handle same table different DBs.                 */
/****************************************************************************/

METHOD PUBLIC CHARACTER dyn_getvalue (INPUT hQryHndl AS HANDLE,
                                      INPUT cTblFld AS CHARACTER):

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

  FIND ttDynToolKit EXCLUSIVE-LOCK
  WHERE ttDynToolKit.QryHndl = hQryHndl
    AND ttDynToolKit.TblName = ENTRY(1, cTblFld, ".").

  IF NOT AVAILABLE ttDynToolKit THEN RETURN ?.

  ASSIGN hFldHndl = ttDynToolKit.TblHndl:BUFFER-FIELD(ENTRY(2, cTblFld, ".")).

  RETURN STRING(hFldHndl:BUFFER-VALUE).
```

```
END. /* METHOD PUBLIC dyn_getvalue () */

/*****************************************************************************/
/* Pull a RAW version of the data off the field buffer.                      */
/* The TblFld is meant to be called as table.field like in usual 4GL         */
/* Right now this doesn't handle same table different DBs.                   */
/*****************************************************************************/

METHOD PUBLIC RAW dyn_getvalue_raw (INPUT hQryHndl AS HANDLE,
                                    INPUT cTblFld AS CHARACTER):

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

  FIND ttDynToolKit EXCLUSIVE-LOCK
  WHERE ttDynToolKit.QryHndl = hQryHndl
    AND ttDynToolKit.TblName = ENTRY(1, cTblFld, ".").

  IF NOT AVAILABLE ttDynToolKit THEN RETURN ?.

  ASSIGN hFldHndl = ttDynToolKit.TblHndl:BUFFER-FIELD(ENTRY(2, cTblFld, ".")).

  RETURN hFldHndl:BUFFER-VALUE.

END. /* METHOD PUBLIC dyn_getvalue_raw () */

/*****************************************************************************/
/* Pull a ROWID of the record off the field buffer.                   */
/* The TblFld is meant to be called as table.field like in usual 4GL         */
/* Right now this doesn't handle same table different DBs.                   */
/*****************************************************************************/

METHOD PUBLIC ROWID dyn_getvalue_rowid (INPUT hQryHndl AS HANDLE,
                                        INPUT cTblName AS CHARACTER):

  DEFINE VARIABLE hTblHndl AS HANDLE NO-UNDO.

  FIND ttDynToolKit EXCLUSIVE-LOCK
  WHERE ttDynToolKit.QryHndl = hQryHndl
    AND ttDynToolKit.TblName = cTblName.

  IF NOT AVAILABLE ttDynToolKit THEN RETURN ?.

  hTblHndl = ttDynToolKit.TblHndl.

  RETURN hTblHndl:ROWID.

END. /* METHOD PUBLIC dyn_getvalue_rowid () */

/*****************************************************************************/
/* Pull a RECID of the record off the field buffer.                   */
/* The TblFld is meant to be called as table.field like in usual 4GL         */
/* Right now this doesn't handle same table different DBs.                   */
/*****************************************************************************/
```

```
METHOD PUBLIC RECID dyn_getvalue_recid (INPUT hQryHndl AS HANDLE,
                                        INPUT cTblName AS CHARACTER):

  DEFINE VARIABLE hTblHndl AS HANDLE NO-UNDO.

  FIND ttDynToolKit EXCLUSIVE-LOCK
  WHERE ttDynToolKit.QryHndl = hQryHndl
    AND ttDynToolKit.TblName = cTblName.

  IF NOT AVAILABLE ttDynToolKit THEN RETURN ?.

  hTblHndl = ttDynToolKit.TblHndl.

  RETURN hTblHndl:RECID.

END. /* METHOD PUBLIC dyn_getvalue_recid () */

/****************************************************************************/
/* Given a table.field, determine the field type.                           */
/* The TblFld is meant to be called as table.field like in usual 4GL        */
/* Right now this doesn't handle same table different DBs.                  */
/****************************************************************************/

METHOD PUBLIC CHARACTER dyn_fieldtype (INPUT hQryHndl AS HANDLE,
                                       INPUT cTblFld AS CHARACTER):

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

  FIND ttDynToolKit EXCLUSIVE-LOCK
  WHERE ttDynToolKit.QryHndl = hQryHndl
    AND ttDynToolKit.TblName = ENTRY(1, cTblFld, ".").

  IF NOT AVAILABLE ttDynToolKit THEN RETURN ?.

  ASSIGN hFldHndl = ttDynToolKit.TblHndl:BUFFER-FIELD(ENTRY(2, cTblFld, ".")).

  RETURN hFldHndl:DATA-TYPE.

END. /* METHOD PUBLIC dyn_fieldtype () */

/****************************************************************************/
/* Given a table.field, determine the field type.                           */
/* The TblFld is meant to be called as table.field like in usual 4GL        */
/* Right now this doesn't handle same table different DBs.                  */
/* WARNING: THIS IS NOT TESTED.                                             */
/****************************************************************************/

METHOD PUBLIC HANDLE dyn_fieldhdl (INPUT hQryHndl AS HANDLE,
                                   INPUT cTblFld AS CHARACTER):

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

  FIND ttDynToolKit EXCLUSIVE-LOCK
```

```
    WHERE ttDynToolKit.QryHndl = hQryHndl
      AND ttDynToolKit.TblName = ENTRY(1, cTblFld, ".").

    IF NOT AVAILABLE ttDynToolKit THEN RETURN ?.

    ASSIGN hFldHndl = ttDynToolKit.TblHndl:BUFFER-FIELD(ENTRY(2, cTblFld, ".")).

    RETURN hFldHndl.

  END. /* METHOD PUBLIC dyn_fieldhdl () */

  /***************************************************************************/
  /* Given a query and table name, return the buffer table for the table.      */
  /***************************************************************************/

  METHOD PUBLIC HANDLE dyn_tablehdl (INPUT hQryHndl AS HANDLE,
                                    INPUT cTableName AS CHARACTER):

    FIND ttDynToolKit NO-LOCK
    WHERE ttDynToolkit.QryHndl = hQryHndl
    AND ttDynToolkit.TblName = cTableName
    NO-ERROR.

    IF NOT AVAILABLE ttDynToolKit THEN RETURN ?.

    RETURN ttDynToolKit.TblHndl.

  END. /* METHOD PUBLIC dyn_tblhndl */

  /***************************************************************************/
  /* Provide a means of returning the number of results in a query.            */
  /* Running this on 9.1C and getting zero even though there is a result set   */
  /* greater than zero.                                                        */
  /***************************************************************************/

  METHOD PUBLIC INTEGER dyn_numresults (INPUT hQryHndl AS HANDLE):

    DEFINE VARIABLE iNum AS INTEGER NO-UNDO.

    ASSIGN iNum =  hQryHndl:NUM-RESULTS.

    RETURN iNum.

  END. /* METHOD PUBLIC dyn_numresults */

  /***************************************************************************/
  /* Given a query handle, build up the ttDynToolKit table from it.  Useful    */
  /* for when a query handle is passed into an external procedure and one      */
  /* wants to use the tool kit's METHOD PUBLICs.
*/
  /***************************************************************************/

  METHOD PUBLIC LOGICAL dyn_qryinfo (INPUT hQryHndl AS HANDLE):

    DEFINE VARIABLE iCurBufSeq AS INTEGER NO-UNDO.
    /*****************************************************/
    /* Clean up the ttDynToolKit table of this query so */.
    /* we don't get duplicates.                         */
    /*****************************************************/

    FOR EACH ttDynToolKit EXCLUSIVE-LOCK
    WHERE ttDynToolKit.QryHndl = hQryHndl:

      DELETE ttDynToolKit.

    END.

    /*****************************************************/
    /* Rebuild the table from the info available in the */
    /* dynamic objects.                                 */
    /*****************************************************/

    DO iCurBufSeq = 1 TO hQryHndl:NUM-BUFFERS:

      CREATE ttDynToolKit.

      ASSIGN ttDynToolkit.QryHndl = hQryHndl
             ttDynToolKit.TblHndl = hQryHndl:GET-BUFFER-HANDLE(iCurBufSeq)
                ttDynToolKit.TblName = ttDynToolKit.TblHndl:TABLE.

    END. /* DO iCurBufSeq = 1 TO hQryHndl:NUM-BUFFERS */

  END. /* METHOD PUBLIC dyn_qryinfo () */

  /*****************************************************************************/
  /* Simple dump routine for the table.                                      */
  /*****************************************************************************/

  METHOD PUBLIC LOGICAL dyn_dump (INPUT cFileName AS CHARACTER):

    OUTPUT TO VALUE (cFileName).

    FOR EACH ttDynToolKit EXCLUSIVE-LOCK:

      EXPORT INT(ttDynToolkit.QryHndl)  INT(ttDynToolKit.TblHndl)
ttDynToolKit.TblName.

    END. /* FOR EACH ttDynToolKit */

    OUTPUT CLOSE.

  END.

  /*****************************************************************************/
  /* Given a table name, determine the number of fields on it.               */
  /*****************************************************************************/

  METHOD PUBLIC INTEGER dyn_numfields (INPUT hQryHndl AS HANDLE,
                                       INPUT cTableName AS CHARACTER):
```

```
  FIND ttDynToolKit NO-LOCK
      WHERE ttDynToolkit.QryHndl = hQryHndl
        AND ttDynToolkit.TblName = cTableName

  NO-ERROR.

  IF NOT AVAILABLE ttDynToolKit THEN RETURN ?.

  RETURN ttDynToolKit.TblHndl:NUM-FIELDS.

END. /* METHOD PUBLIC dyn_numfields */

/*****************************************************************************/
/* Given a query handle, how many tables are in the query   .                */
/*****************************************************************************/

METHOD PUBLIC INTEGER dyn_numtables (INPUT hQryHndl AS HANDLE):

  DEFINE VARIABLE iCnt AS INTEGER INIT 0 NO-UNDO.

      FOR EACH ttDynToolKit NO-LOCK
      WHERE ttDynToolkit.QryHndl = hQryHndl:

        ASSIGN iCnt = iCnt + 1.

     END.

      RETURN iCnt.

END. /* METHOD PUBLIC dyn_numtables () */

/*****************************************************************************/
/* Given a query handle, what are the table names          .                 */
/*****************************************************************************/

METHOD PUBLIC CHARACTER dyn_listtables (INPUT hQryHndl AS HANDLE):

  DEFINE VARIABLE cList AS CHARACTER INIT "" NO-UNDO.

      FOR EACH ttDynToolKit NO-LOCK
      WHERE ttDynToolkit.QryHndl = hQryHndl:

        ASSIGN cList = cList + ttDynToolKit.TblName + ",".

     END.

      ASSIGN cList = SUBSTRING(cList, 1, LENGTH(cList) - 1).

      RETURN cList.

END. /* METHOD PUBLIC dyn_numtables () */

/*****************************************************************************/
/* Given a query handle, what are the table names          .                 */
```

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

METHOD PUBLIC CHARACTER dyn_listfields (INPUT hQryHndl AS HANDLE,
                                        INPUT cTableName AS CHARACTER):

  DEFINE VARIABLE iCntFields AS INTEGER NO-UNDO.
     DEFINE VARIABLE iCurField  AS INTEGER NO-UNDO.
  DEFINE VARIABLE cList AS CHARACTER INIT "" NO-UNDO.
     DEFINE VARIABLE hField AS HANDLE NO-UNDO.

     ASSIGN iCntFields = dyn_numfields (hQryHndl, cTableName).
     IF iCntFields = ? THEN RETURN ?.

     FIND ttDynToolKit NO-LOCK
     WHERE ttDynToolkit.QryHndl = hQryHndl
       AND ttDynToolKit.TblName = cTableName.


     DO iCurField = 1 TO iCntFields:

       ASSIGN hField = ttDynToolKit.TblHndl:BUFFER-FIELD(iCurField).

       ASSIGN cList = cList + hField:Name + ",".

    END.

     ASSIGN cList = SUBSTRING(cList, 1, LENGTH(cList) - 1).

     RETURN cList.

END. /* METHOD PUBLIC dyn_numtables () */

/******************************************************************************/
/******************************************************************************/
/*                                PLEASE READ                                 */
/*                                                                            */
/*                                  ______                                    */
/*                               .-"      "-.                                 */
/*                              /            \                                */
/*                  _          |              |          _                    */
/*                 ( \         |,  .-.  .-.  ,|         / )                   */
/*                  > "=._     | )(__/  \__)( |     _.=" <                    */
/*                 (_/"=._"=._ |/     /\     \| _.="_.="\_)                   */
/*                        "=._ (_     ^^     _)"_.="                          */
/*                            "=\__|IIIIII|__/="                              */
/*                           _.="| \IIIIII/ |"=._                             */
/*                 _     _.="_.="\          /"=._"=._     _                   */
/*                ( \_.="_.="     `--------`     "=._"=._/ )                  */
/*                 > _.="  DO NOT EDIT FRIVOLOUSLY!  "=._ <                   */
/*                (_/                                    \_)                  */
/*                                                                            */
/* WARNING: ASSIGNING A FIELD USED TO ORDER THE QUERY WILL HOSE THE QUERY.   */
/*                                                                            */
/******************************************************************************/
/******************************************************************************/
```

```
/****************************************************************************/
/* Progress has no LOGICAL METHOD PUBLIC.  Here we set up a way to translate */
/* CHAR representations of logicals to a actual progress data type of       */
/* LOGICAL.                                                                 */
/****************************************************************************/

METHOD PUBLIC LOGICAL set_logical (INPUT cText AS CHARACTER):

  IF cText = ? THEN RETURN ?.

  IF CAN-DO("Y,YES,TRUE", cText) THEN RETURN TRUE.

  RETURN FALSE.

END. /* METHOD PUBLIC SET_LOGICAL */

/****************************************************************************/
/* Allow the setting of any type values via a string source.  No error      */
/* checking - assume the programmer has a clue.                             */
/****************************************************************************/

METHOD PUBLIC LOGICAL dyn_set (INPUT hQryHndl AS HANDLE,
                               INPUT cTblFld AS CHARACTER,
                                                    INPUT cText AS CHARACTER):

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

  FIND ttDynToolKit EXCLUSIVE-LOCK
  WHERE ttDynToolKit.QryHndl = hQryHndl
    AND ttDynToolKit.TblName = ENTRY(1, cTblFld, ".").

  IF NOT AVAILABLE ttDynToolKit THEN RETURN FALSE.

  ASSIGN hFldHndl = ttDynToolKit.TblHndl:BUFFER-FIELD(ENTRY(2, cTblFld, ".")).

  CASE hFldHndl:DATA-TYPE:

    WHEN "CHARACTER" THEN ASSIGN hFldHndl:BUFFER-VALUE = cText.

    WHEN "LOGICAL" THEN ASSIGN hFldHndl:BUFFER-VALUE = SET_LOGICAL(cText).

    WHEN "DATE" THEN ASSIGN hFldHndl:BUFFER-VALUE = DATE(cText).

    WHEN "INTEGER" THEN ASSIGN hFldHndl:BUFFER-VALUE = INTEGER(cText).

    WHEN "DECIMAL" THEN ASSIGN hFldHndl:BUFFER-VALUE = DECIMAL(cText).

  END. /* CASE */

  RETURN TRUE.
```

```
END. /* METHOD PUBLIC dyn_setc() */

/******************************************************************************/
/* Allow the setting of character type values.  No error checking - assume   */
/* the programmer has a clue.                                                 */
/******************************************************************************/

METHOD PUBLIC LOGICAL dyn_setc (INPUT hQryHndl AS HANDLE,
                                INPUT cTblFld AS CHARACTER,
                                                     INPUT cText AS CHARACTER):

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

  FIND ttDynToolKit EXCLUSIVE-LOCK
  WHERE ttDynToolKit.QryHndl = hQryHndl
    AND ttDynToolKit.TblName = ENTRY(1, cTblFld, ".").

  IF NOT AVAILABLE ttDynToolKit THEN RETURN FALSE.

  ASSIGN hFldHndl = ttDynToolKit.TblHndl:BUFFER-FIELD(ENTRY(2, cTblFld, ".")).

  ASSIGN hFldHndl:BUFFER-VALUE = cText.

  RETURN TRUE.

END. /* METHOD PUBLIC dyn_setc() */

/******************************************************************************/
/* Allow the setting of character type values.  No error checking - assume   */
/* the programmer has a clue.                                                 */
/******************************************************************************/

METHOD PUBLIC LOGICAL dyn_seti (INPUT hQryHndl AS HANDLE,
                                INPUT cTblFld AS CHARACTER,
                                                     INPUT iVal AS INTEGER):

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

  FIND ttDynToolKit EXCLUSIVE-LOCK
  WHERE ttDynToolKit.QryHndl = hQryHndl
    AND ttDynToolKit.TblName = ENTRY(1, cTblFld, ".").

  IF NOT AVAILABLE ttDynToolKit THEN RETURN FALSE.

  ASSIGN hFldHndl = ttDynToolKit.TblHndl:BUFFER-FIELD(ENTRY(2, cTblFld, ".")).

  ASSIGN hFldHndl:BUFFER-VALUE = iVal.

  RETURN TRUE.

END. /* METHOD PUBLIC dyn_setc() */

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

```
/* Allow the setting of character type values.  No error checking - assume   */
/* the programmer has a clue.                                                 */
/*****************************************************************************/

METHOD PUBLIC LOGICAL dyn_setf (INPUT hQryHndl AS HANDLE,
                                INPUT cTblFld AS CHARACTER,
                                                      INPUT fVal AS DECIMAL):

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

  FIND ttDynToolKit EXCLUSIVE-LOCK
  WHERE ttDynToolKit.QryHndl = hQryHndl
    AND ttDynToolKit.TblName = ENTRY(1, cTblFld, ".").

  IF NOT AVAILABLE ttDynToolKit THEN RETURN FALSE.

  ASSIGN hFldHndl = ttDynToolKit.TblHndl:BUFFER-FIELD(ENTRY(2, cTblFld, ".")).

  ASSIGN hFldHndl:BUFFER-VALUE = fVal.

  RETURN TRUE.

END. /* METHOD PUBLIC dyn_setc() */

/*****************************************************************************/
/* Allow the setting of character type values.  No error checking - assume   */
/* the programmer has a clue.                                                 */
/*****************************************************************************/

METHOD PUBLIC LOGICAL dyn_setl (INPUT hQryHndl AS HANDLE,
                                INPUT cTblFld AS CHARACTER,
                                                      INPUT lVal AS LOGICAL):

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

  FIND ttDynToolKit EXCLUSIVE-LOCK
  WHERE ttDynToolKit.QryHndl = hQryHndl
    AND ttDynToolKit.TblName = ENTRY(1, cTblFld, ".").

  IF NOT AVAILABLE ttDynToolKit THEN RETURN FALSE.

  ASSIGN hFldHndl = ttDynToolKit.TblHndl:BUFFER-FIELD(ENTRY(2, cTblFld, ".")).

  ASSIGN hFldHndl:BUFFER-VALUE = lVal.

  RETURN TRUE.

END. /* METHOD PUBLIC dyn_setc() */

/*****************************************************************************/
/* Allow the setting of character type values.  No error checking - assume   */
/* the programmer has a clue.                                                 */
/*****************************************************************************/
```

```
  METHOD PUBLIC LOGICAL dyn_setd  (INPUT hQryHndl AS HANDLE,
                                   INPUT cTblFld AS CHARACTER,
                                                         INPUT dVal AS DATE):

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

    FIND ttDynToolKit EXCLUSIVE-LOCK
    WHERE ttDynToolKit.QryHndl = hQryHndl
      AND ttDynToolKit.TblName = ENTRY(1, cTblFld, ".").

    IF NOT AVAILABLE ttDynToolKit THEN RETURN FALSE.

    ASSIGN hFldHndl = ttDynToolKit.TblHndl:BUFFER-FIELD(ENTRY(2, cTblFld, ".")).

    ASSIGN hFldHndl:BUFFER-VALUE = dVal.

    RETURN TRUE.

  END. /* METHOD PUBLIC dyn_setc() */

END. /* CLASS */
```

In order for the tool to work, it also makes use of this procedure[13] `dyn_findinschema.p`:

```
/***************************************************************************/
/* ONLY CALL THIS PROGRAM FROM dyntoolkit.i WHICH CREATES THE ALIAS        */
/* TEMPDB.                                                                 */
/* ONLY CALL THIS PROGRAM FROM DynToolKit.cls WHICH CREATES THE ALIAS      */
/* TEMPDB.                                                                 */
/* SGA: Donated by Dayne May daynem @ linx.com.au                          */
/* How to compile this.                                                    */
/*    Connect to a DB with -ld tempdb  (Any DB should do)                  */
/*    Compile and save r-code                                              */
/***************************************************************************/

DEFINE VARIABLE RCS AS CHARACTER INIT "$Id: dyn_findinschema.p,v 1.3 2006/10/20
05:01:44 sauge Exp sauge $" NO-UNDO.

DEFINE INPUT  PARAMETER pcTable AS CHARACTER.
DEFINE OUTPUT PARAMETER plOK AS LOGICAL.

FIND FIRST tempdb._file
    WHERE tempdb._file._file-name = pcTable
    NO-LOCK NO-ERROR.

ASSIGN plOK = AVAILABLE tempdb._file.
```

13 There is a revised version of this code found in the appendix. It is less logical database name oriented for compilation.

Using the class is quite simple. Initialize a variable of type `DynToolKit` to hold the class handle.

There are a couple of properties of interest – `UseMultiDBs` which is a logical of whether the class is being used on one or more databases. If you are using multiple databases – there is no choice – it must be `YES/TRUE`. If not, then there is a choice, it's just that the algorithm will have a few more steps in it to handle potential multiple databases than required.

Some databases can be connected as a name for the data servers for non-Progress databases. There are connections that you may want to skip the name of and use the schema holder name for that database. This is the purpose of the `IgnoreDBNames` property.

Next one uses the `dyn_open()` method to start the query. One simply puts in a query like one would for any dynamic query. The class will take care of determining the buffers needed, etc.

The `dyn_open()` method will return a handle for use on other methods in the class. The class can actually handle multiple queries so you do not need an object per query. This is an example where the class can be re-used without re-instancing. Since the functionality is so basic, it was determined this would be the right way to go.

There are a couple of other properties that will alert if there is a problem: `cDyn_ErrCode` and `cDyn_ErrMsg`. One can check the values to determine if the query has a problem and what kind it is.

To actually reach the data, one can use the navigation methods of `dyn_first()`, `dyn_next()`, `dyn_prev()`, and `dyn_last()` to navigate in the query's result set.

To pull the data out, one can use `dyn_getvalue()` method or one of it's type specific cousins.

Finally use `dyn_close()` to close the query and free up the associated dynamically created elements.

Reading the code above, I am sure you will discover quite a bit more functionality available in the class.

```
DEFINE VARIABLE h AS HANDLE NO-UNDO.
DEFINE VARIABLE DynToolKit AS CLASS DynToolKit NO-UNDO.

DynToolKit = new DynToolKit().

/*****************************************************************/
/* Test One : A good query in DB 1                               */
/*****************************************************************/

DynToolKit:UseMultiDBs = YES.
DynToolKit:IgnoreDBNames = "oracle".

ASSIGN h = DynToolKit:dyn_open("FOR EACH Customer NO-LOCK where
Customer.CustomerID = 251").

DISPLAY DynToolKit:cDyn_ErrCode DynToolKit:cDyn_ErrMsg FORMAT "x(30)".

DISPLAY h:NUM-RESULTS COLUMN-LABEL "NumResults".

REPEAT:

  DynToolKit:dyn_next(h).
  IF DynToolKit:dyn_qoe(h) THEN LEAVE.

  DISPLAY DynToolKit:dyn_getvalue(h, "Customer.Name")
          DynToolKit:dyn_getvalue(h, "Customer.CreditLimit")
          DynToolKit:dyn_getvalue(h, "Customer.IsActive")
          DynToolKit:dyn_getvalue(h, "Customer.Phone").

END.

DynToolKit:dyn_close(h).
```

Iterating Collection Object

One of the nicest uses of classes is the ability for them to provide a structure for scrolling data. If you are a GUI programmer, a browser can do things nicely – but if you are a web programmer (either HTML, AJAX, or Web Services interfaces) you can't conveniently use such a thing. If you are an old timer, you will remember the many attempts at an include file that would magically create a browse based on scrolling frame and an editing-loop. and other fun toys in the language like that.

This is something where OOP really stands out as this iteration data structure is clear and concise in it's use and implementation.

The class provides methods to collate information into one string. This string goes into a scratch table in the database and the data is named with a `DataSetName` so on multiple uses it can be found again[14].

Then one merely sets the paging size parameter sent to the constructor and the class will automatically chop up pages for you for use with the `NextPage()`, `PrevPage()` and other iterating based methods on the data set.

Read the class code below for some additional functionality available and the example use of the code following the listing.

```
class scroller:

  define public variable DataSetName as character no-undo.
  define public variable PageLength as integer no-undo.
  define public variable CurrentPage as integer no-undo.

  define temp-table DataRetrieved
    field RowNumber as integer
    field Data as character.

  /* Get 142 errors if not private buffer */
  define buffer gScratch for Scratch.

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

14 For example, if you are programming with a web interface or a state-less connection, the data set name would be placed into a session data structure so it can be referenced again on sub-sequent hits on the web app.

```
constructor public scroller(input DataName as character,
                            input PageLen as integer):

  define variable S as class strings no-undo.

  if DataName = ? then do:
    S = new strings().
    DataName = S:MakeID3(20).
    delete object S.
  end. /* if DataName = ? */

  DataSetName = DataName.
  PageLength = PageLen.
  CurrentPage = 1.

end. /* constructor */

/************************************************************************/

method public void ClearData():

  for each scratch no-lock
    where scratch.dataname = DataSetName:

    delete Scratch.

  end. /* for each */

end. /* method */

/************************************************************************/

method public integer NumberOfPages():

  define variable NumberOfPages as decimal.

  NumberOfPages = NumberOfRows() / PageLength.

  if integer(NumberOfPages) < NumberOfPages then
    NumberOfPages = NumberOfPages + 1.

  return integer(NumberOfPages).

end. /* method */

/************************************************************************/

method private integer NumberOfRows():
    define variable Counter as integer init 0 no-undo.

    for each scratch no-lock
      where scratch.dataname = DataSetName:

      Counter = Counter + 1.

    end. /* for each */

    return Counter.

  end. /* method */

  /**********************************************************************/

  method public logical IsNextPage():

    return CurrentPage < NumberOfPages().

  end. /* method */

  /**********************************************************************/

  method public logical IsPrevPage():

    return CurrentPage > 1.

  end. /* method */

  /**********************************************************************/

  method public logical GetPageData (output table DataRetrieved):

    define variable CurrentRow as integer init 1 no-undo.
    define variable OffSetRow as integer no-undo.

    if NumberOfRows() = 0 then return false.

    empty temp-table DataRetrieved.

    OffSetRow = max(1, (CurrentPage - 1 ) * PageLength).
    if CurrentPage > 1 then OffSetRow = OffSetRow + 1.  /* Math dealing
with 0 */

    do while CurrentRow <= PageLength:

      find gScratch no-lock
      where gScratch.K1 = string(OffSetRow)
      and gScratch.DataName = DataSetName
    no-error.

    if not available gScratch then leave.

    create DataRetrieved.

    DataRetrieved.RowNumber = CurrentRow.
    DataRetrieved.Data = gScratch.Data1.

    OffSetRow = OffSetRow + 1.
    CurrentRow = CurrentRow + 1.


  end. /* do */

  return true.

end. /* method */

/************************************************************************/

method public void NextPage():

  CurrentPage = min(CurrentPage + 1, NumberOfPages()).

end. /* method */

/************************************************************************/

method public void PrevPage():

  CurrentPage = max(CurrentPage - 1, 1).

end. /* method */

/************************************************************************/

method public void FirstPage():

  CurrentPage = 1.

end. /* method */

/************************************************************************/

method public void LastPage():

  CurrentPage = NumberOfPages().
  end. /* method */

  /*********************************************************************/

  method public void AddRow(input RowData as character):

    create gScratch.

    gScratch.K1 = string(NumberOfRows() + 1).

    gScratch.Data1 = RowData.
    gScratch.DataName = DataSetName.
    gScratch.CreateDate = TODAY.
    gScratch.CreateTime = TIME.
    gScratch.StructureName = "scroller.cls".

  end. /* method */

end. /* class */
```

The scratch table for the database can be found in the appendix of this book.

Here is an example use.

First we clear out the entire database table. Really one should use `ClearData()` but that is for use of a named set of entries. The testing code likes to start with a clear table.

Next we define a temp table that matches the temp-table definition within the object. *This is a breaking of the rules regarding encapsulation.* I think this is one of those few cases where it can be allowed. Why? It is very very unlikely the structure will change since it is designed to handle a dynamic structure of data within it.

We instantiate an object from the class and start feeding it data to work with.

From there we start navigating around in the collection of data with the navigation methods in a manner like a browser.

Then we delete the data. Note that deleting the object DOES NOT delete the data. This would be useless if on a stateless connection we once again needed that data.

```
for each scratch exclusive-lock: delete scratch. end.
```

```
define variable S as class scroller no-undo.
define variable i as integer no-undo.

  define temp-table DataRetrieved
    field RowNumber as integer
    field Data as character.


S = new scroller(?, 10).


do i = 1 to 45:

  S:AddRow ("Test Row " + string (i)).

end.

S:FirstPage().
display S:NumberOfPages().

display "----".
S:GetPageData (output table DataRetrieved).
for each DataRetrieved:
display DataRetrieved.RowNumber DataRetrieved.Data format "x(40)".
end.

display "----".
S:NextPage().
S:GetPageData (output table DataRetrieved).
for each DataRetrieved:
display DataRetrieved.RowNumber DataRetrieved.Data format "x(40)".
end.

display "----".
S:NextPage().
S:GetPageData (output table DataRetrieved).
for each DataRetrieved:
display DataRetrieved.RowNumber DataRetrieved.Data format "x(40)".
end.

display "----".
S:NextPage().
S:GetPageData (output table DataRetrieved).
for each DataRetrieved:
display DataRetrieved.RowNumber DataRetrieved.Data format "x(40)".
end.

display "----".
S:NextPage().
```

```
S:GetPageData (output table DataRetrieved).
for each DataRetrieved:
display DataRetrieved.RowNumber DataRetrieved.Data format "x(40)".
end.

display "----".
S:PrevPage().
S:GetPageData (output table DataRetrieved).
for each DataRetrieved:
display DataRetrieved.RowNumber DataRetrieved.Data format "x(40)".
end.

display "Last Page ----".
S:LastPage().
S:GetPageData (output table DataRetrieved).
for each DataRetrieved:
display DataRetrieved.RowNumber DataRetrieved.Data format "x(40)".
end.


delete object S.
```

This is a very basic iterator. By making some changes, one can actually serialize the data and make this a generic scroller for all types of information beyond a string.

In addition, hopefully you can see how the data can be fed out in XML/HTML for AJAX use as well as traditional page programming with a hyper link detailing the action being `NextPage()` or `PrevPage()`. All one needs to do is remember two pieces of information either in the client or in the web session table: `DataSetName` and `CurrentPageNumber`.

(Remember not to repopulate the table after the first time unless you need to reset the data – otherwise you are doing work already done. The purpose of the `DataSetHandle` is to remember populations from selecting and joining data out one or more tables.)

Writing and Deleting From The Database

One of the basic sub-systems in almost any application is the ability to adjust parameters of operation. These might be log levels, where logs are stored, email addresses, snippets of HTML, etc.

Below is a simple class that interacts with a database table called `SysParameter` in read, write, delete, and update manner. As you read the code, you will see that one interacts with the database just as any other body of ABL code using the usual database manipulation statements.

Objects used in this manner act exactly like subroutines or blocks of code in the ABL.

```
class ParameterMngr:

  define public variable SystemEnvironment as character no-undo.

  /*********************************************************************/
  /* Some parameters might be different based on the environment the app */
  /* is running under (like dev, test, or production.)  We use a var to  */
  /* to help identify the proper parameter for the given environment.    */
  /*********************************************************************/

  constructor public ParameterMngr():

    SystemEnvironment = OS-GETENV("APPLICATION_ENVIRONMENT").

  end. /* constructor */

  /*********************************************************************/
  /* Unused.                                                             */
  /*********************************************************************/

  destructor public ParameterMngr():

  end. /* destructor */

  /*********************************************************************/
  /* Create and update a parameter. If the ParmComment is ? no change is */
  /* made to the comment.                                                */
  /*********************************************************************/

  method public logical SetParameterValue (input ParmName as character,
                                           input ParmValue as character,
                                           input ParmComment as character):

    find SysParameter exclusive-lock
    where SysParameter.ParameterName = ParmName
    no-error.

    if not available SysParameter then do:

      create SysParameter.
      SysParameter.ParameterName = ParmName.

    end. /* if not available SysParameter */

    SysParameter.Data = ParmValue.

    /* Don't overwrite any comments that might be there already */

    if ParmComment <> ? then SysParameter.Comment = ParmComment.


  end. /* SetParameterValue */

  /**********************************************************************/
  /* Provide the value to a given parameter.                            */
  /**********************************************************************/

  method public character GetParameterValue (input ParmName as character):

    find SysParameter no-lock
    where SysParameter.ParameterName = ParmName
    no-error.

    if not available SysParameter then return ?.

    return SysParameter.Data.

  end. /* GetParameterValue */

  /**********************************************************************/
  /* Provide the comment to a named parameter.                          */
  /**********************************************************************/

  method public character GetParameterComment
 (input ParmName as character):

    find SysParameter no-lock
    where SysParameter.ParameterName = ParmName
    no-error.
    if not available SysParameter then return ?.

    return SysParameter.Comment.

  end. /* GetParameterComment */

  /*********************************************************************/
  /* Delete a given parameter name.                                    */
  /*********************************************************************/

  method public logical DeleteParameter (input ParmName as character):

    find SysParameter exclusive-lock
    where SysParameter.ParameterName = ParmName
    no-error.

    if not available SysParameter then return false.

    delete SysParameter.

    return true.

  end. /* DeleteParameter */

end.  /* class */
```

As you read the code, you may have thought to yourself this is a candidate for use with static methods. Depending on it's use – it very well might be.

Some very simple examples of it's use. One of the common questions is if “I define the class reference as a no-undo variable – does that mean the database actions are no-undo?” The answer to that is “No. The database interactions are not undone unless the use of the class is done in a block of code that undoes database transactions.

```
define variable ParmMngr as class ParameterMngr no-undo.

ParmMngr = new ParameterMngr().
ParmMngr:SetParameterValue(“amduus.html.home.company”,
                           “Amduus Information Works, Inc.”,
                           “Web App Home Page Welcome”).
display GetParameterValue(“amduus.html.home.company”).
delete object ParmMngr.
```

One of the more interesting things that can be done with parameters is making them specific to a customer used within the system. This often happens in web applications for user personalization and or customer organization personalization.

Set up the names to use `custom.parameter...` formatting and with some modification of the `GetParameterValue()` method one can "drop down" along the parameter for those that are global to the user, the customer, or the system. For example, a user with an id of `k6453` might have `k6543.html.pagecolor` while the default should the user not customized their set up is `html.pagecolor`.

Remembering that we can add values to objects without being dependent on parameter arguments, we could easily add a property/public variable called `WorkingUserID` that is set when the object is created. Perhaps one might even want a `WorkingCustomerCompany` property/public variable for SaaS apps sharing databases amongst different customers.

Interacting With The Database With Your Own Commit

One of the things we know about objects is the ability to encapsulate data. Can one encapsulate transactions?

The answer – to a degree – is yes. When the implementing developer using the object uses the `Set*()` and `Get*()` methods to manipulate data on the object – these do not need to be on the database record buffer it's self but a temp-table defined as like the database record.

When the object is constructed or use of a `FindBy*()` method is called – this temp-table can be buffer-copy to from the database records by the object.

When the implementing code calls the objects `Commit()` method – the temp-table can be validated and if good, is buffer-copy to the database record. Even better, when the `Rollback()` method is called – one can simply forget all about the changes made on the temp-table record.

Of course, if you `Commit()` then roll backing of the database change will need to be done with the ABL statement `UNDO`.

Here is a small example program[15] using this technique.

```
/* Warning: This is generated code. */
class DiscussionManager:

  define temp-table TheRecord like Discussion.
  define buffer WorkingDiscussion for Discussion.
  define private variable IsNew as logical no-undo.
  define private variable ErrorCode as character no-undo.


  /*********************************************************************/
  /* Constructor                                                       */
  /*********************************************************************/

  constructor public DiscussionManager():

  end. /* constructor */
```

15 I have a program that will generate most of the code to make an object specific to a table. Contact me if you wish to have a copy of it.

```
/*************************************************************************/
/* Destructor                                                            */
/*************************************************************************/

destructor public DiscussionManager():

end. /* destructor */



/*************************************************************************/
/* Commit changes                                                        */
/*************************************************************************/

method public logical Commit():

  if IsNew then create WorkingDiscussion.

  buffer-copy TheRecord to WorkingDiscussion.

  ReleaseToOthers().

end. /* method */



/*************************************************************************/
/* Release changes                                                       */
/*************************************************************************/

method public logical ReleaseToOthers():

  release WorkingDiscussion.

end. /* method */



/*************************************************************************/
/* Rollback changes                                                      */
/* A RollBack() after a Commit() is pretty much useless.  Use UNDO.      */
/*************************************************************************/

method public logical RollBack():
  empty temp-table TheRecord.
  IsNew = false.

  ReleaseToOthers().

end. /* method */



/*********************************************************************/
/* Set/Get for CreateDate */
/*********************************************************************/

method public void SetCreatedate(input TheValue as date):

  SetError("000").

  /* Ensure the record is available from a FindBy method */

  if not available TheRecord then do:

    SetError("001").
    return.

  end. /* if */

  /* All is OK.  Set the value. */

  TheRecord.Createdate = TheValue.

end. /* SetCreateDate */


method public date GetCreatedate():

  SetError("000").

  /* Ensure the record is available from a FindBy method */

  if not available TheRecord then do:

    SetError("001").
    return ?.

  end. /* if */

  /* All is OK.  Return the value. */
  return TheRecord.Createdate.

end. /* GetCreateDate */


/*********************************************************************/
/* Set/Get for CreateTime */
/*********************************************************************/

method public void SetCreatetime(input TheValue as integer):

  SetError("000").

  /* Ensure the record is available from a FindBy method */

  if not available TheRecord then do:

    SetError("001").
    return.

  end. /* if */

  /* All is OK.  Set the value. */

  TheRecord.Createtime = TheValue.

end. /* SetCreateTime */


method public integer GetCreatetime():

  SetError("000").

  /* Ensure the record is available from a FindBy method */

  if not available TheRecord then do:

    SetError("001").
    return ?.

  end. /* if */

  /* All is OK.  Return the value. */

  return TheRecord.Createtime.

end. /* GetCreateTime */
```

```
/***********************************************************************/
/* Set/Get for DiscussionID */
/***********************************************************************/

method public void SetDiscussionid(input TheValue as character):

  SetError("000").

  /* Ensure the record is available from a FindBy method */

  if not available TheRecord then do:

    SetError("001").
    return.

  end. /* if */

  /* All is OK.  Set the value. */

  TheRecord.Discussionid = TheValue.

end. /* SetDiscussionID */


method public character GetDiscussionid():

  SetError("000").

  /* Ensure the record is available from a FindBy method */

  if not available TheRecord then do:

    SetError("001").
    return ?.

  end. /* if */

  /* All is OK.  Return the value. */

  return TheRecord.Discussionid.

end. /* GetDiscussionID */


/***********************************************************************/
/* Set/Get for FromIMUserID */
/***********************************************************************/
```

```
method public void SetFromimuserid(input TheValue as character):

  SetError("000").

  /* Ensure the record is available from a FindBy method */

  if not available TheRecord then do:

    SetError("001").
    return.

  end. /* if */

  /* All is OK.  Set the value. */

  TheRecord.Fromimuserid = TheValue.

end. /* SetFromIMUserID */


method public character GetFromimuserid():

  SetError("000").

  /* Ensure the record is available from a FindBy method */

  if not available TheRecord then do:

    SetError("001").
    return ?.

  end. /* if */

  /* All is OK.  Return the value. */

  return TheRecord.Fromimuserid.

end. /* GetFromIMUserID */


/************************************************************************/
/* Set/Get for MessageText */
/************************************************************************/

method public void SetMessagetext(input TheValue as character):

  SetError("000").

  /* Ensure the record is available from a FindBy method */
  if not available TheRecord then do:

    SetError("001").
    return.

  end. /* if */

  /* All is OK.  Set the value. */

  TheRecord.Messagetext = TheValue.

end. /* SetMessageText */


method public character GetMessagetext():

  SetError("000").

  /* Ensure the record is available from a FindBy method */

  if not available TheRecord then do:

    SetError("001").
    return ?.

  end. /* if */

  /* All is OK.  Return the value. */

  return TheRecord.Messagetext.

end. /* GetMessageText */


/*************************************************************************/
/* Set/Get for RoomID */
/*************************************************************************/

method public void SetRoomid(input TheValue as character):

  SetError("000").

  /* Ensure the record is available from a FindBy method */

  if not available TheRecord then do:

    SetError("001").
    return.
  end. /* if */

  /* All is OK.  Set the value. */

  TheRecord.Roomid = TheValue.

end. /* SetRoomID */


method public character GetRoomid():

  SetError("000").

  /* Ensure the record is available from a FindBy method */

  if not available TheRecord then do:

    SetError("001").
    return ?.

  end. /* if */

  /* All is OK.  Return the value. */

  return TheRecord.Roomid.

end. /* GetRoomID */


/*********************************************************************/
/* Set/Get for ToIMUserID */
/*********************************************************************/

method public void SetToimuserid(input TheValue as character):

  SetError("000").

  /* Ensure the record is available from a FindBy method */

  if not available TheRecord then do:

    SetError("001").
    return.

  end. /* if */

  /* All is OK.  Set the value. */
  TheRecord.Toimuserid = TheValue.

end. /* SetToIMUserID */


method public character GetToimuserid():

  SetError("000").

  /* Ensure the record is available from a FindBy method */

  if not available TheRecord then do:

    SetError("001").
    return ?.

  end. /* if */

  /* All is OK.  Return the value. */

  return TheRecord.Toimuserid.

end. /* GetToIMUserID */


/***********************************************************************/
/* Provide a means to query any errors occurred.                       */
/***********************************************************************/

method public character GetError():

  return ErrorCode.

end. /* GetError */


/***********************************************************************/
/* Provide a means to set the error code and message.                  */
/***********************************************************************/

method private character SetError(input ErrorNumber as character):

  case ErrorNumber:

    when "000" then ErrorCode = ErrorNumber + ":No Error".
    when "001" then ErrorCode = ErrorNumber + ":No Record".

  end. /* case */
  end. /* SetError */


  /*********************************************************************/
  /* Provide a means to reset the error code and message.              */
  /*********************************************************************/

  method private void ResetError():

    SetError("000").

  end. /* ResetError */


  /*********************************************************************/
  /* Provide a FindBy*() to update a record with or to query values by.  */
  /*********************************************************************/

  method public logical FindByDiscussionid
    (
    input Discussionid as character
    ):

    find WorkingDiscussion no-lock
    where WorkingDiscussion.Discussionid = Discussionid
    no-error.


    if available WorkingDiscussion then do:
      IsNew = false.
      create TheRecord.
      buffer-copy WorkingDiscussion to TheRecord.
    end. /* if available */

    return available WorkingDiscussion .

  end. /* method */




  /*********************************************************************/
  /* Provide a FindBy*() to update a record with or to query values by.  */
  /*********************************************************************/

  method public logical
FindByRoomidFromimuseridToimuseridCreatedateCreatetime
    (
    input Roomid as character,
  input Fromimuserid as character,
  input Toimuserid as character,
  input Createdate as date,
  input Createtime as integer
  ):

  find WorkingDiscussion no-lock
  where WorkingDiscussion.Roomid = Roomid
    and WorkingDiscussion.Fromimuserid = Fromimuserid
    and WorkingDiscussion.Toimuserid = Toimuserid
    and WorkingDiscussion.Createdate = Createdate
    and WorkingDiscussion.Createtime = Createtime
  no-error.


  if available WorkingDiscussion then do:
    IsNew = false.
    create TheRecord.
    buffer-copy WorkingDiscussion to TheRecord.
  end. /* if available */

  return available WorkingDiscussion .

end. /* method */


/**********************************************************************/
/* After using a FindBy*() and want to update - use this.             */
/**********************************************************************/

method public logical LockRecord ():

  SetError("000").

  /* Ensure the record is available from a FindBy method */

  if not available WorkingDiscussion then do:

    SetError("001").
    return false.

  end. /* if */

  find current WorkingDiscussion exclusive-lock no-error.

  return available WorkingDiscussion.

end. /* method */
```

```
  /***********************************************************************/
  /* Provide a means to create a new record with.                        */
  /***********************************************************************/

  method public logical CreateRecord():

    create TheRecord.

    IsNew = true.
    return available TheRecord.

  end. /* method */


end. /* class */
```

Here is a small test program to give an idea how to use the object as generated.

```
define variable N as class DiscussionManager no-undo.
N = new DiscussionManager().

N:CreateRecord().
N:SetRoomID("Test").
N:Commit().

N:CreateRecord().
N:SetRoomID("Test2").
N:Rollback().

delete object N.
```

With this basic structure you can add additional tables, methods, validations, error handling, and other object interactions as you need. Often, a set of three or four tables make up some like collection of data. Creating an object that handles the interactions on those tables often works nicely so the implementing programmer need not understand the tables “magic” numbers, key fields, etc.

More Advanced Subjects

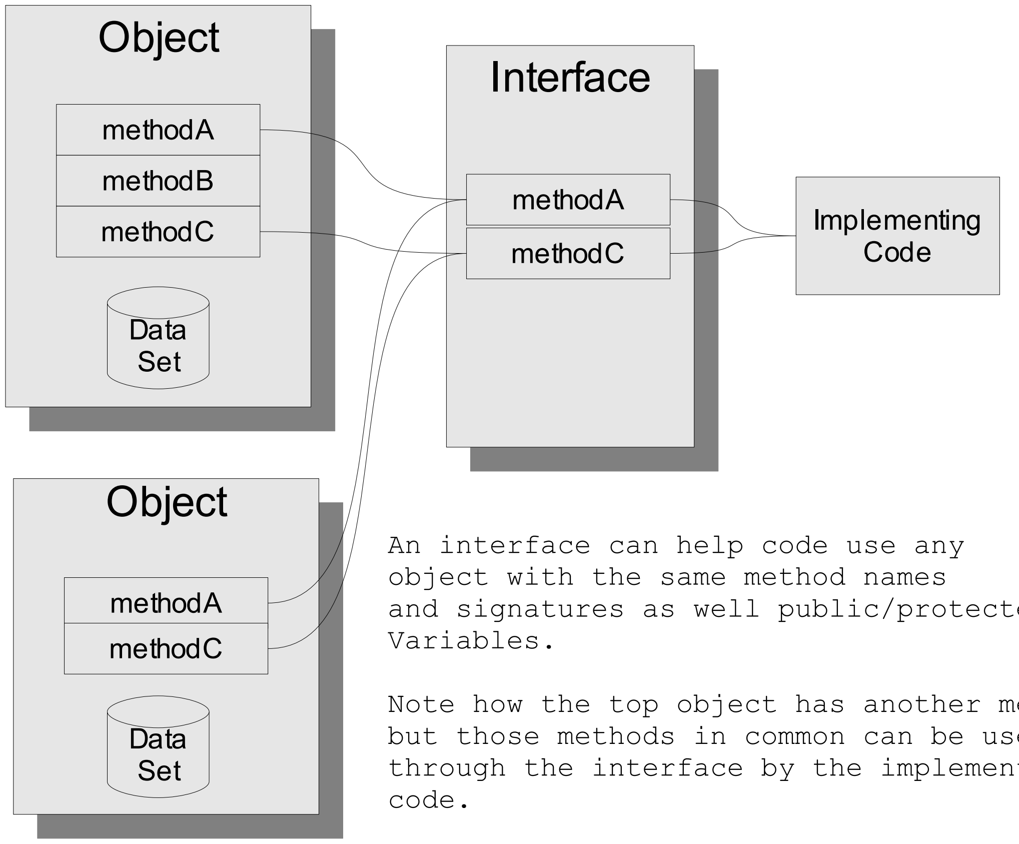
There are going to be times when you do not know what kind of object you will need at run time. Just as Progress provides the `RUN VALUE()` construct for it's procedural methodology, the object oriented programming methodology has something similar.

The Concept of a Factory

There are times when you are going to need to dynamically instantiate an object but at run-time you won't know which class it will be of. Think of it as a `run value(myprogram)` type of technique.

This is the purpose of a factory.

In order to start, you will need to create an `interface` – that is, a definition of what methods the classes you might use must provide.



```
An interface can help code use any
object with the same method names
and signatures as well public/protected
Variables.

Note how the top object has another method
but those methods in common can be used
through the interface by the implementing
code.
```

Here is an example made up strictly of methods. One can include protected temp tables and variables/properties in the definition too.

```
interface Iterator:

  method public void GoNext().
  method public void GoPrev().
  method public void GoFirst().
  method public void GoLast().
  method public integer CountOf().
  method public logical IsEmpty().

end.
```

Next, you need a set of objects that provide these pieces of functionality. Here we make use of an object that will walk through the table in the Vector object.

Finally, you create a factory object. Note the big magic is in the `Generate()` method. Note how it's class is `Iterator` which isn't really a class definition but an interface to a class we defined above. ***Any class can be returned long enough it has those methods (and their signatures) as part of it's definition.***

```
/* This is a bit like the run value(program) idea in the proc- */
/* dural use of the language.  This shows how to instance an   */
/* object holder defined by an interface (Iterator.cls) with   */
/* different types of objects that will guarantee those        */
/* methods are available to it.                                */

class IteratorFactory:

  /* We might want to tell the user if something went wrong. */

  define public variable ErrorMessage as character init "Ready" no-undo.

  /* Each time you call this, you will be generating a new instance */
  /* so be careful about cleaning up with delete object statements. */

  method public class Iterator Generate(input TypeOfObject as character):

    /* Create a handle for each type of Iterator we might send back */

    define variable N as class VectorIterator no-undo.

    ResetErrorMessage().

    /* Decide the iterator type, instance, and return it's handle  */
    /* to be used by the invoker of the class.                     */

    case TypeOfObject:

      when "Vector" then do:

        ErrorMessage = "Created VectorIterator".
        N = new VectorIterator().
        return N.

      end. /* when */

      otherwise do:

        ErrorMessage = "I don't know what to do!".
        return ?.

      end. /* otherwise */

    end. /* case */

  end. /* method */

  method private void ResetErrorMessage():

    ErrorMessage = "Ready".

  end. /* method */

  method public character GetErrorMessage():

    return ErrorMessage.

  end. /* method */

end. /* class */
```

Here is an example use of the factory object. Note we use two different variables to hold each instance (N and N1) because since they are dynamically created they will need to be dynamically deleted.

```
define variable N as class Iterator no-undo.
define variable N1 as class Iterator no-undo.
define variable F as class IteratorFactory no-undo.

F = new IteratorFactory().
```

```
display F:GetErrorMessage().

N = F:Generate("Vector").
display F:GetErrorMessage().

N1 = F:Generate("Hash").
display F:GetErrorMessage().

N:GoFirst().

delete object F.
delete object N.
delete object N1.
```

The point plainly made, is that if you have an interface definition that matches all of some of the definition of any class in terms of methods and data, one can dynamically create any of those classes and refer to it with the interface.

Dynamic Creation Of Objects With Dynamic-New

Note: This section requires the use of Progress 10.1c or better.

This is a slight variant on the factory idea that is built into the language. One sets up a character variable with the name of the class you wish to instance and give it to a reference that can handle that instance.

For some control, you may want to do a `search()` of sorts on the class name (remember it might be prefixed with `path.to.class` and postfixed with `.cls`) to make sure such a beast actually exists.

The example above with the factory does some work on handling unexpected classes where this will use the Progress error system.

A benefit of this method is for example specific objects being created for a specific customer. (This is common in SaaS thinking.) For example, one may have a class named `SalesOrder.cls`. This would be the super-class all the customer specific sub-classes would inherit from. So it could be:

```
ClassName =  SourcePrefix(CurrentCustomerID) + "SalesOrder".
```

Then one can dynamically create the class to use based on what customer is being used. (SourcePrefix might return a prefix or a “” based on a table of entries.)

Here is an example of using `dynamic-new` and how it works when *not* using an `interface` to abstract with. In this case, the hierarchy becomes more important when trying to create a class and the class variable used.

*This is why OOP has the mantra **Always Program To The Interface**.*

Using an interface is so much easier than trying to keep track of inheritances and which classes have which methods available when dynamically mixing and matching classes.

But, lets go down that path anyhow.

The variable `TheClass` holds the actual class name (not it's .cls file name.) But we plan on putting the B instance into a variable of class A. What happens can be of interest.

```
define variable T as class A no-undo.
define variable TheClass as character no-undo.
TheClass = "B".
T = dynamic-new TheClass ().
display T:FireThis() format "x(40)".
delete object T.
```

In this example, the classes are defined as so:

```
class A:

  method public character FireThis():
    return "A is firing".

  end.

end.
```

Here is B which over-rides the `FireThis()` method from A with it's own.

```
class B inherits A:

  method public character FireThisB():
    return "B is firing".

  end.

  method public override character FireThis():

    return "B is firing".

  end.

end.
```

When executed, one gets:

```
B is firing
```

This is because B has a method of the same signature as in A, one can dynamically create B and use those methods (common to A.)

If one tries to use a method in B (such as `FireThisB()`) while using an A reference type, it says A doesn't know what to make of it. This makes sense.

Hence, one can hold an object reference of any derived classes (sub-class) in a reference to a base class (super-class) dynamically. Just don't expect to be able to use any methods not found in the base (super) class.

Lets switch things around, ie using a derived reference to use a base class. Here we will use a sub-class to hold a reference to a super class.

```
define variable T as class B no-undo.
define variable TheClass as character no-undo.
TheClass = "A".
T = dynamic-new TheClass ().
display T:FireThis() format "x(40)".
delete object T.
```

Gives ya:

```
A is firing
```

Hopefully this isn't a big surprise because the instance is an A class and not a B class. We just happen to hold an A in a B which is allowed because B is inherited from an A. Everything in A is guaranteed to be available in the B.

So basically if you plan on creating a dynamic object with a reference familiar with all the possible inherited objects, one can dynamically create any one of those inherited objects and use the top-most derived reference to reach into there.

To be safe, if you have C ← B ← A where A is the top most superclass and you plan on putting a reference to an A or B or C into some object reference variable – make the type of that variable a C (the farthest sub-class.)

So, just to be clear. Above, we are using a B reference to play with an A instance and so A's methods are what are firing.

Below, we are using a B to make a B, and so B's methods are what are firing.

```
define variable T as class B no-undo.
define variable TheClass as character no-undo.
TheClass = "B".
T = dynamic-new TheClass ().
display T:FireThis() format "x(40)".
delete object T.
```

Flipping Object's Class Types With Dynamic-Cast

In the previous section, we saw a lot of using a super-class or a sub-class object reference type used with an instance of the object along it's inheritance path.

Now the question is – I have this object reference so how do I flip it from an instance of a sub-class into an instance of a super-class?

Or – I have this instance of a class tucked away into the reference handle of a super class and I want to get back to my "real" class. (This happens a lot on temp-tables where a field is of type `Progress.Lang.Object`.)

This is where the `dynamic-cast` function comes in handy.

Take for example this most likely used scenario. There are a bunch of objects created and stored in a temp-table.

```
define temp-table SalesLinesOfOrder
  field ObjRef as Progress.Lang.Object[16]
  field ObjType as character.
```

And we have some code that creates some objects on the fly – perhaps used in Sales Order Lines – and stores them into the temp-table.

```
for each SalesOrderLine of SalesOrder no-lock:

  create SalesLinesOfOrder.
  SalesLinesOfOrder.ObjRef = new SalesLines(SalesOrder.OrderNumber,
                                             SalesOrderLine.LineNumber
                                            ).
  SalesLinesOfOrder.ObjType = "SalesLines".

end.
```

Now (usually in some other program) we actually want to go and use a method on those objects. But wait – the objects are at a reference which is the lowest of the low super-class (`Progress.Lang.Object`). These references have no idea what these instances are capable of doing.

Hence, we need to cast them to the right class and then make use of their methods.

16 Progress.Lang.Object can store any object of any class the user might create. This way you do not need to set the field to a specific type. We store the specific type in ObjType for use later on.

```
define variable SalesLine as class SalesLines no-undo.

for each SalesLinesOfOrder no-lock:

  SalesLine= dynamic-cast (SalesLinesOfOrder.ObjRef,
                           SalesLinesOfoRder.ObjType
                          ).
  SalesLine:ComputeTax().
  SalesLine:Commit().

end.
```

There is a weakness in the language where one cannot dynamically cast and use the results directly. So while it might be nice to do:

```
dynamic-cast (SalesLinesOfOrder.ObjRef,
SalesLinesOfoRder.ObjType):ComputeTaxes()
```

One cannot do so. Hopefully in a future version they will allow this to happen.

Error Handling

There are three ways of handling errors. One is your own error handling object, which will be discussed here.

The other is the Progress supplied error handling techniques which are discussed in their book "Error Handling" found with their product documentation (version 10.1c). The book reviews the legacy error handling methodology as well the new `Progress.Lang.*` objects available for error handling – as well `CATCH/FINAL` processing. It is very interesting reading, available for free, and I recommend taking a look at it.

The third way is having the error routines in your class which were exampled in previous discussions. Sometimes you want this – sometimes you don't. Sometimes you will use an error collection object to collect errors from the various classes you end up using.

If you find there can be one or multiple errors that you need to record at a time, you will want to create your own error handling sub-system. Examples of such subsystems include web services or web pages. One doesn't want the user to jump through multiple iterations on a web page or multiple soap faults on a web service to handle errors when all the errors can be collected sent back to the user in a convenient manner.

What we want our sub system to do is 1) Record multiple errors. 2) Not undo any errors that are recorded. 3) Provide a means of reading each error out to an implementing program. 4) Provide a means to match an error number to an error message.

To achieve this we will inherit a `Progress.Lang.AppError` class and add to it with errors oriented methods.

First we are going to make the error to message conversion database oriented. This way different languages can be used for the errors or the errors can be customized by the user. To do this, we place the errors into the `SysParameter` table that is used by the `ParameterMngr` class:

```
create SysParameter.
SysParameter.ParameterName = "error.10".
SysParameter.Data = "No Invoice".
SysParameter.Comment = "Wrong Invoice Number".

create SysParameter.
SysParameter.ParameterName = "error.20".
SysParameter.Data = "No Customer".
SysParameter.Comment = "Wrong Customer Number".
```

Next here is the `ErrorCollection` class we can use to collect a bunch of errors:

```
class ErrorCollection inherits Progress.Lang.AppError:

  define private variable ParmMngr as class ParameterMngr no-undo.

  constructor ErrorCollection (input ipErrNumber as character):

    ParmMngr = new ParameterMngr().

    /* Calling our over-ridden AddMessage below */

    AddMessage (ipErrNumber).

  end. /* constructor */

  constructor ErrorCollection ():

    ParmMngr = new ParameterMngr().

  end. /* constructor */

  destructor ErrorCollection():

    delete object ParmMngr.

  end. /* destructor */

  /*********************************************************/
  /* Over-ride the Progress.Lang.AppError AddMessage() method */
  /* with ours.                                            */
  /* Because the super-class expects an integer - our error   */
  /* numbers must be integer.                              */
  /*********************************************************/

  method public logical AddMessage (input ipErrNumber as character):

    /* Running AddMessage() in Progress.Lang.AppError */

    super:AddMessage ( ParmMngr:GetParameterValue("error." + ipErrNumber),
                       integer(ipErrNumber)).
  end. /* method */

end. /* class */
```

The program above looks pretty short and appears to be missing things like the number of messages stored and a means to reach the messages stored. But because we are inheriting from the `Progress.Lang.AppError` class, that code has already been written for us! *We are re-using that code* – specifically the `NumMessages` property and the `GetMessage()` method which are a part of the inherited class[17].

Lets see how easy it is to use:

```
define variable H as class ErrorCollection no-undo.

H = new ErrorCollection().

H:AddMessage ("10").
H:AddMessage ("20").

display H:NumMessages.
display H:GetMessage(1).
display H:GetMessage(2).

delete object H.
```

Pretty simple eh?

17 Actually – some of these are inherited from Progress.Lang.AppError from it's inherited class Progress.Lang.ProError.

OBJECT DESIGN PRINCIPLES

There is a lot of theory out there for designing of objects that sometimes gets a little abstract. (The whole idea of course.)

But this section looks at some common questions you will want to consider with your object designs.

The philosophical difference

The main difference between object oriented programming and procedural coding is a switch in perspective. Procedural coding tends to be more verb – noun. That is, some activity (verb) made upon data (noun). OOP tends to be completely the opposite: noun - verb. There is this thing (noun) that one asks to provide some sort of activity (verb).

This may seem subtle but allows for a much more domain knowledge oriented analysis of what should be coded.

In OOP – one focuses on the entity – the what or the who. Common in programming is a collection of users. One can create a manager – a who – that does certain things with users.

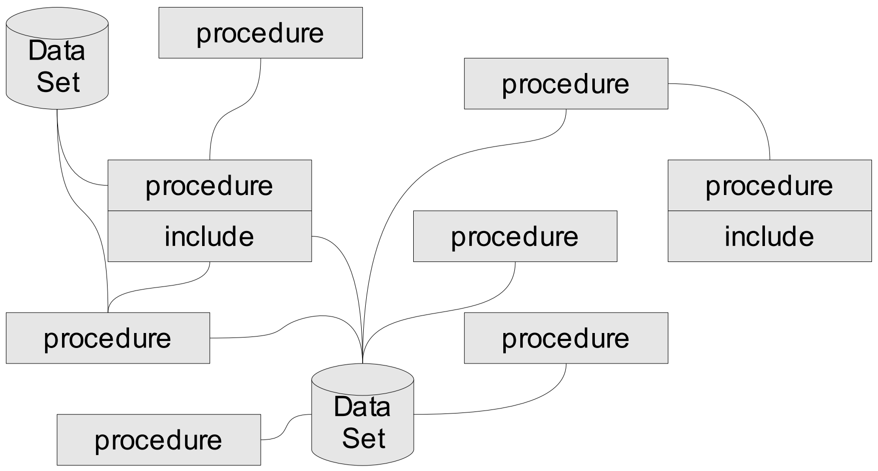
A sales order – a what - is common in any business and with an object. One asks it to not only know about the sales order's data – but it will also provide actions one does on the sales order. Think of the object as a person who does knows the data and provides actions on the data with other parts of the system.

Objects Lend To Better Code Organization

Verbs to nouns are very much one to one. After a bit of time, in large complex coding – procedural programing lends it's self to an "entropy" of structure. How routines relate and interact with each other becomes less and less obvious. Often, in order to use one subroutine, one must find data that is not relevant to the caller simply to provide the needed parameter to the called subroutine. This pollutes the purpose of the routines and makes them expand in complexity.

Another problem is a "leakage" of one domain of activities into another. This is the famous problem of "fixing one thing that breaks another." This often happens in subroutines that have expanded beyond their original scope. What looks like a good simple fix becomes a problem because the *context* of how the subroutine is used might be different based on the callers using it and *their intents*.

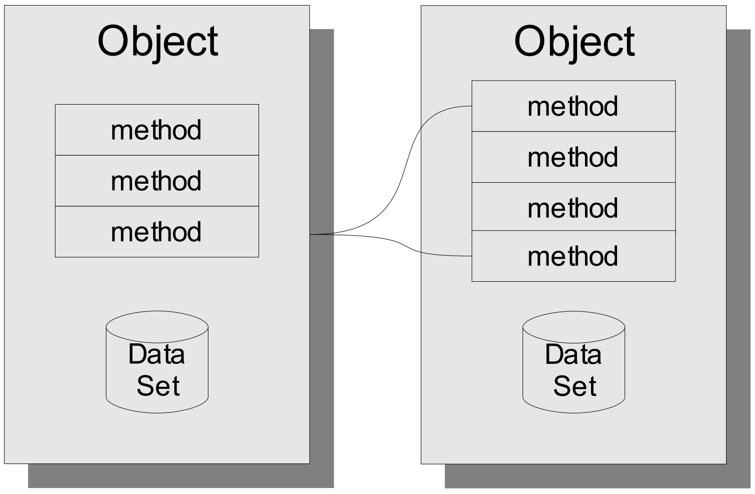
There is a measurement tool called "software cohesion" that helps identify just how much scoped has creeped "outside their boxes." I recommend googling the term and becoming familiar with the concept. Having a good cohesion factor means easier maintenance, cheaper maintenance, and faster maintenance.



“Hodge Podge” of procedures and
their data sets.

OOP is more of a “holistic” approach. There is a set of code and a set of data and the object owns these things. Because OOP involves data encapsulation – a lot of pieces are “built into” the object.

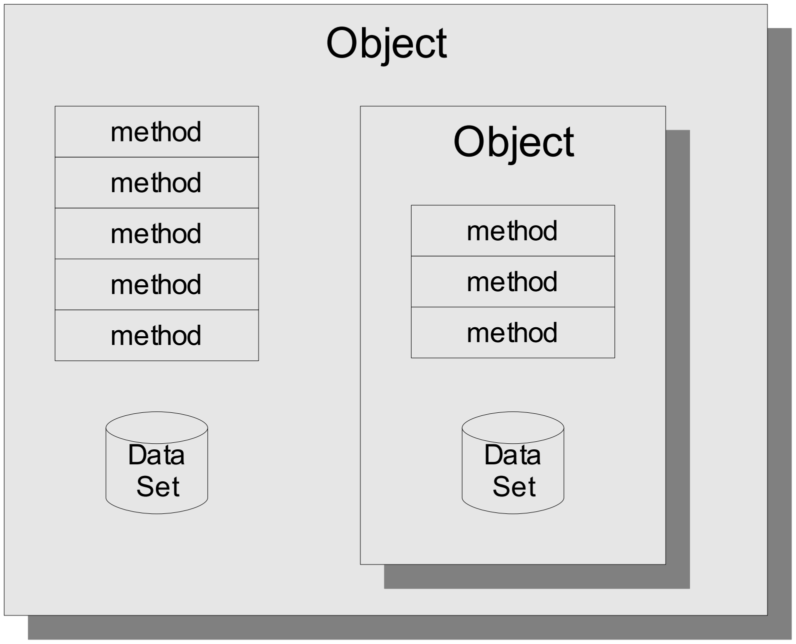
At times there is data related to activities that cause parameters to procedures to expand. Soon programming involves finding data and passing more and more data along to procedures. If this is not done, then the use of shared variables come into use and those have a whole other set of problems.



Activities and data are stored in a way they are all associated to each other and "own" the verbs and nouns of it's domain of action.

In OOP, one asks an object that fully knows about it's domain of operation (both in terms of data and action on that data) to do something within it's domain. If you need to inform it about additional information – the object will have a means to do that regardless of the outside programming asking it to do something.

Often one will want to make a special condition. Perhaps one has coded for an entity in an object but another entity like it has just a couple more special features about it. One can "embrace and extend" the old object into the new object focusing not on the base features of it but on the differences between the base object and the expanded object.



Inheritance continues to keep domains Segregated but allows expansion.

When the programmer is working on the new extension of the object – they won't need to risk changes to the original object for it's special purpose. Nor will they even need to deal with the base objects code and data – their focus is on the differences and new features of activities and data.

Class Naming Conventions

Often I have found the following convenient:

Collections of like methods that don't really provide a business logic are called `*Tools` as in `FileTools`. `FileTools` contains items such as `GetPath()`, `GetPostfix()`, etc.

An object that manages a collection of anything I call a `Manager`. For example, an object that handles a collection of users might be called `UserManager`.

An object that manages one specific set of data is named for that data in business parlance such as `SalesOrder`. This object would be oriented to manipulating one specific item and that is a sales order for the customer.

When instancing an object, I tend to put `Obj` in front of it – as in `ObjSalesOrder`. This helps separate the static use of classes separate from the dynamic instantiation of objects defined by classes.

Method Naming Conventions

You will want your methods to describe what they do. For example, if they are meant to set some value within the object, then it should be prefixed with `Set*()`. And it is often useful to pull the value back with a `Get*()`. Sometimes computation needs to be done on the value for the `Get*()` just as there may need to be some done for a `Set*()`. An example might be `GetAccountNumberAsInteger()` or `GetAccountNumberAsCharacter()`.

If you want to initialize the object in some manner, I tend to use `FindBy*()` - as in `FindByAccountNumber(AccountNumber)`. There may be an inclination of using an index as part of the name of the `FindBy*()` - and you may not have a choice on that. However, the `FindBy*()` should really be based on the conceptualization of how things are understood at a human level. For example, the constructor will take care of the context of the data such as this:

```
ObjAccountManager = new AccountManager(InstanceID, SessionID).
```

We segregate all the data by a value in the record known as an `InstanceID` to identify one company's data from another. But, this really has nothing to do with finding an account in the context of the company – they have no idea that other companies exist in the database nor are they concerned about a web session identifier.

They just know they should get the account by the account number and so when the program attaches the user interface to the object it should be straight forward:

```
ObjAccountManager:FindByAccountNumber(TheNumber).
```

So, in the constructor we try to define all the context the object might work under and let the methods be focused on the exacting set of data that the programmer is really interested in dealing with at the time of the call.

When to inherit and when to instance within an object

This can be a thorny question and sometimes there is no right answer.

If one plans on adding additional data and methods to an object whom you plan on being able to access it's methods directly, the answer is easily inheritance. You are simply building up on a base of functionality.

Sometimes it is more fuzzy. For example below, the object makes use of a lot of the stack's methods – but has some wrappers around it to make it more oriented to the proper use of the object. This could go both ways – either inheriting or instancing a stack object within the object and making use of the instance.

In this case, I chose to inherit.

```
/* Class to track and transform error codes back to web service */
/* users.  This is the place to centralize the errors.          */

class amduus.misc.errors inherits amduus.misc.stack:

  /*********************************************************************/
  /* Translate an error code into a message and store it.             */
  /* Some routines will not be using this per se to store errors, but  */
  /* need a tool to translate an error code into a message.  This method */
  /* will do this.                                                     */
  /*********************************************************************/

  method public character TranslateCode (input ErrorCode as character):

    define variable ErrorMsg as character no-undo.

    case ErrorCode:

      when "000" then ErrorMsg = ErrorCode + ":No Error".
      when "001" then ErrorMsg = ErrorCode + ":No Such Traveler".
      when "015" then ErrorMsg = ErrorCode + ":This Cannot Be Blank".
      when "V05" then ErrorMsg = ErrorCode + ":Required Information Missing!".

      otherwise ErrorMsg = ErrorCode + ":No Description".

    end. /* case */

    return ErrorMsg.

  end.

  /*********************************************************************/
  /* Given a code, it will translate it and store in this object's data   */
  /* store for use with HasError() and GetError().                     */
  /*********************************************************************/
  method public logical SetError(input ErrorCode as character):

    PushOnStack (TranslateCode(ErrorCode)).

  end.

  /*********************************************************************/
  /* Send back one error, do not remove from this object's storage area.  */
  /*********************************************************************/

  method public character PeekError():

    return PeekOnStack().

  end. /* method */

  /*********************************************************************/
  /* Sometimes there are warning, info, and fatal stop errors in the      */
  /* this provides a means to determine if such a thing is in there.      */
  /*********************************************************************/

  method public logical ContainsErrorCodeBeginning
  (input ErrorCodeBegins as character):

    return can-find (first TheStack where TheStack.Data begins ErrorCodeBegins).

  end.

  /*********************************************************************/
  /* Send back one error, remove from this object's storage area.         */
  /*********************************************************************/

  method public character GetError ():

    return PopOffStack().

  end. /* method */

  /*********************************************************************/
  /* Determines if any error codes reside on the object yet.  Good for a  */
  /*    repeat while S:IsError()                                           */
  /* loop for error handling.                                             */
  /*********************************************************************/

  method public logical HasError():

    return not IsEmpty().

  end. /* method */

end. /* class */
```

```
/*************** UNIT TEST *****************************

define variable T as class amduus.misc.errors no-undo.

T = new amduus.misc.errors().

T:SetError ("000").
T:SetError ("003").
T:SetError ("101").
T:SetError ("J12").

display "Peeking" with frame a.

display T:PeekError() format "x(70)".

if T:ContainsErrorCodeBeginning ("002") then
  display "yes on 002".
else
  display "no on 002".

if T:ContainsErrorCodeBeginning ("003") then
  display "yes on 003".
else
  display "no on 003".

display "Popping" with frame b.
repeat while T:HasError():
  display T:GetError() format "x(70)".
end.

delete object T.


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

The stack object has value added to it by converting a code number into a code number and error message as well as methods that are error related.

Now take for example an object that does something completely different than errors yet one might want to record any errors that occur with it. The object is far removed in concept from a stack or from an error tool and so it is better to simply make an instance of an object within the object and use that.

Adding constants for use with an object

A lot of times an object will be configurable – for example – in the `socket.cls` object, there is a `CRLF` constant that is a public variable. This is because in many protocols – HTTP Message, SMTP, POP3, IMAP, etc., a command or line of data ends with a CRLF.

As an example, this line on a pop3 object would retrieve a particular message:

```
POPConnection:WriteText ("RTRV " + MsgNumber + POPConnection:CRLF).
```

Another example is the `socket.cls` object has an `IsSSL` constant on it. This way when a connection is made, one can say if the connection should be SSL or not:

```
ObjSocketManager:OpenConnection(Host,
                                PortNumber,
                                not ObjSocketManager:IsSSL).
```

Hopefully upon reading the code one will realize this is a non-SSL socket connection. This helps with the object being all inclusive in the types of uses for it.

Adding enumerations for use with an object

Other times you may want to enumerate the type of data that can be placed into an argument of an object. One can use constants for that like:

```
define private variable EnumJobState as character
init "START,STOP,PAUSE,DONE,PROCESSING" no-undo.
define public variable JobState_START as character init "START" no-undo.
define public variable JobState_STOP as character init "STOP" no-undo.
```

Note `EnumJobState` is a private variable and simply used to check if a value coming in is a proper value. The other variables are named in such a way that they can be used as enumerations. We use variables because we don't want the developer using the object researching into what are valid values for their own variables or pre-processor definitions. Also, it gives an abstraction layer where if we want to change the values around within the object they can be done so without interrupting the other code.

Then defining a `Set*` method might be:

```
method public logical SetJobState(input JobState as character):
  if not can-do(EnumJobState, JobState) then do:
    SetError (“300”).
    return false.
  end.

  ...

end.
```

And using the method would be something like:

```
Jobs:SetJobState(Jobs:JobState_STOP).
```

This gives you some bonuses:

- Developer never needs to guess or understand what the internal representation of that value might be – so no research into setting variables or `GLOBAL-DEFINE`.
- The code is self-documenting.
- The object is there to complete what it needs of it's self.
- Abstraction layer in the values used by the object. The programmer knows to use the variables – not the values which can be changed in a new version of the object.

Should I make my app purely object oriented?

That is really up to the developer. It is OK to mix and match – especially when introducing an OO change to an existing code base.

Business Object Representations

Objects really shine when used in the capacity of a business or high level abstraction object. While many people new to object oriented programming focus on user widgets, the best deep magic for object oriented programming is embedded knowledge and information about business rules and process within an object. If a programmer learns how to use the object – they are empowered in an incredible manner!

Why Business Objects?

There are many benefits to programming in objects – some of which are described below.

No need to have knowledge of the database structure.

When properly formed objects are available to a developer, they can focus on using the tools the object provides rather than knowing arcane abbreviations and relations between tables.

Hopefully, when the object is designed well, all this knowledge is not required of the developer . They merely know to fill in some blanks in the object and the rest is auto-magically done.

No need to have knowledge about "magic values" and database quirks.

In my travels as a contracting programmer, I have encountered many systems. All of them have contained the so called "magic values." These are values that land into a field and there is no real heads or tails to what they are for.

For example, a field might contain "01" or "AA01" - these are actual values I have encountered. They don't really say much do they?

Objects often can deal with these values because this knowledge is embedded in the object instead of in the programmer's mind. (Unless of course it is the programmer maintaining the object.)

No need to have knowledge about sub-objects.

Often developers can become easily familiar with "top level" objects. That is when there is work to be done – often developers integrate the top level objects into data loads, web services, etc.

The objects often make use of lower level objects and the developer does not need to know about these in order to be productive.

Ease of adding additional properties.

One of the biggest problems with procedural programming is when one discovers a change requires a change in parameters or data going into a procedure.

Since objects can simply have another variable/property associated with them and they are immediately available (but not required to be handled by) other code.

Ease of adding additional methods.

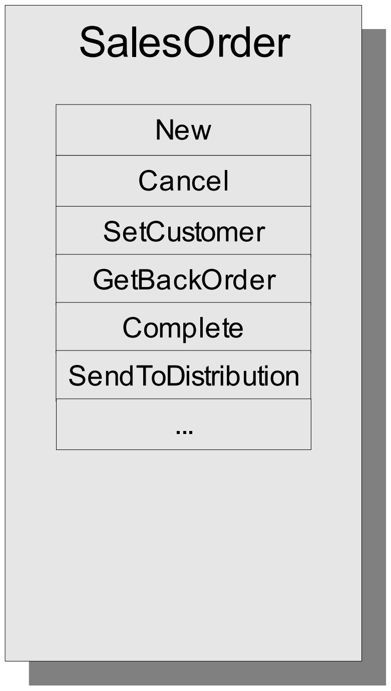
The same can be said of adding methods. For example, one might find a need to send information via a different route. A mail object might use a simple mail command, a smtpmail command, or a SMTP connection. Making any one of these methods gives the implementing programmer another option of how to send mail.

In a procedural world, one would need to send in all the information for the mail as well as a flag for which transport method to use. In an object, the method IS the transport object flag.

Most importantly – focus on the purpose, not the implementation!

Often OOP programming allows the programmer to work with data in a different manner. One merely calls `SalesOrder:FindByOrderNumber(100)` and all the data in the tables and such can be made available to the developer. There is no need to know of tricky relations or the like – one line does it all as far as the developer is concerned.

That is where the 4GL showed it's power in years previous with statements that can do a lot with little work. The 3GL languages made up for this with the addition of OOP to their grammar – allowing them to tie a lot of power into a few invocations of methods which in the end were “business statements.”



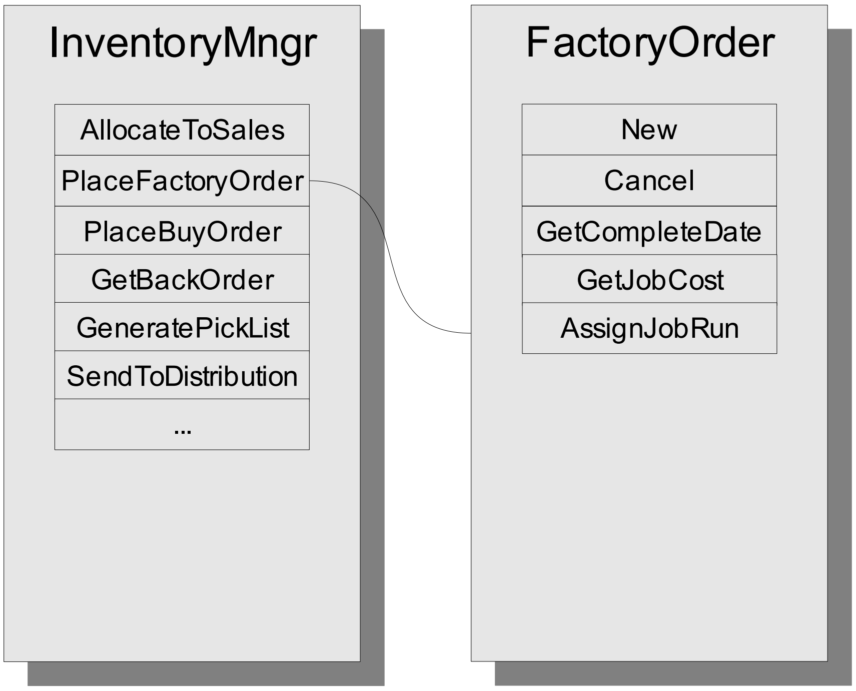
Elaborating further, *objects allow one to create "business statements" instead of programming statements.* For example:

```
SalesOrder:FindByOrderNumber(100).
SalesOrder:Cancel().
```

Take into consideration – even in the 4GL – how many statements would be required of the programmer to cancel a sales order. However, the *objects are very much like creating a language of operations and data that the developer can use to work data* in the process of the business.

Business Objects can beget or use other Business Objects

When using business objects, one can actually create other business objects already prepped with the data needed for working. As an example below – the programmer can ask the InventoryMngr object – an object for manipulating inventory in the company – to create a factory order object. The factory order object could present methods (messages) that the InventoryMngr could use to manipulate a factory order for a specific part/kit.



As developers become familiar with the various objects they can become more powerful in their work. The above shows how the `InventoryMngr` could place a factory order – here it is in coding:

```
/* Obtain a factory order for needed inventory */

FactoryOrder = InventoryManager:PlaceFactoryOrder(input PartNumber).

/* Have the factory order find an existing run or create a new */
/* run to create this part.                                     */

FactoryOrder:AssignJobRun().
```

A caveat to OOP though, is one must do a lot of thinking ahead. Sometimes a developer (myself included) tends to allow scope creep to get into an object. Often programming goes along and then one realizes that some amount of functionality should have been placed into a different object.

Appendix: Tips And Tricks

- Keep the revision number of a program within a property of the class. This way programs will know what version they are working with if need be. This also benefits in deployment as one can `ident` the r-code and determine what versions of what code made it into the binary.
- Use a format for your error messages and codes. A recommended format might be `ddd:s` where d is a digit and s is a string. This way 1) there is an easy to process number via programming, 2) a message that can be read by a human, 3) the colon acts as a convenient split character for `ENTRY()`.
- If you can, keep re-using an object for other collections of data, such as having a `FindBy*()` methods, do so. Instantiating and destroying objects is expensive. If you write your object such that they are re-usable do so and you will be happy.
- Have a `Debug` property as part of your object. Then you can switch on debugging via a parameter or the like or have a special flag in a parameter or file that your object looks at to determine if it should turn debugging mode on for it's self.

Appendix – Ident for source code identification

Often source code revision systems use the `$Id$` and `$Revision$` keywords to store information in a source code listing so people reading it (or compiling it) know what version they are using.

One can embed these keywords by using a variable init'ed to the keyword. When the revision system provides source the keywords will be substituted in the init of the character variable and when compiled by the Progress compiler the value will be transferred to the r-code.

See the preceding class coding examples for a Revision variable and how this works for the revision control system.

I provide this in java because it is generally available on all systems for free.

Here is the bash script that calls out to the compiled `javaident.class` program to look up these keywords in progress r-code files.

```
#!/bin/bash

java javaident $@
```

The java code that provides javaident:

```
//
// Not all systems have ident and not all systems have C++
// on them anymore but it appears more and more have java
// so make an ident that works with PVCS keywords in bin-
// ary files.
// Should also work on subversion, RCS, CVS keyword Id
// Contact for updates at
//   scottauge@gmail.com scott_auge@yahoo.com sauge@amduus.com
//

import java.io.*;

class javaident {

  public static void main (String Args[]) throws IOException {

    //
  // Determine if we have anything to work with else tell
  // caller what we need
  //

  if (Args.length < 1) {

    System.out.println ("Provide files to search in as arguments.");
    System.exit (1);

  } // no args

  //
  // Walk the file list processing each file sent in
  //

  for (int i = 0; i < Args.length; i++)
    LookForKeywordsInFile (Args[i]);

} // main ()

public static void LookForKeywordsInFile (String FileName) {

  int  byteFromFile = 0;
  char charFromFile;
  boolean InDelimiter = false;
  boolean FoundKeyword = false;

  try {

    System.out.println (FileName + ":");

    FileInputStream str = new FileInputStream (FileName);

    StringBuffer PotentialKeyword = new StringBuffer();

    while (true) {

      byteFromFile = str.read();
      if (byteFromFile == -1) break;

      charFromFile = (char) byteFromFile;

      Character T1 = new Character (charFromFile);

      // First time we encounter a $ while not in a delimter

      if (charFromFile == '$' && !InDelimiter) {
        InDelimiter = true;
        PotentialKeyword = new StringBuffer();
```

```
          PotentialKeyword.append(charFromFile);
          continue;
        }

        // Found a closing $ to an actual keyword

        if (InDelimiter && FoundKeyword == true && charFromFile == '$') {
          FoundKeyword = false;
          PotentialKeyword.append(charFromFile);
          System.out.println (PotentialKeyword);
          continue;
        }

        if (InDelimiter) {

          PotentialKeyword.append(charFromFile);

          String T = new String (PotentialKeyword);

          if (!FoundKeyword && InDelimiter && T.equals("$Id")) FoundKeyword
= true;
          if (!FoundKeyword && InDelimiter && T.equals("$Revision"))
FoundKeyword = true;

          if (!FoundKeyword && InDelimiter && T.length() > 9) {
            InDelimiter = false;
            PotentialKeyword = new StringBuffer();
          }

        } // if InDelimiter

      } // while (true)

      str.close();

     } catch (FileNotFoundException c) {
      System.out.println ("Couldn't find file " + FileName);
      return;
     }

    catch (IOException c) {
      System.out.println ("IOException for file " + FileName);
      return;
    }


  } // LookForKeywordsInFile ()

} // class javaident
```

Appendix: Class Generator With Commit

```
/* Code generator for an object next to a table */

define input parameter CodeFile as character no-undo.
define input parameter TableName as character no-undo.

define variable FindByName as character no-undo.
define variable FindByParameters as character no-undo.
define variable FindByWhere as character no-undo.

function Capitalize returns character (input TheWord as character):

  TheWord = lower(TheWord).

  TheWord = upper(substring (TheWord, 1, 1))
          + substring (TheWord, 2).

  return TheWord.

end.

function RemoveDash returns character (input TheWord as character):

  return replace (TheWord, "-", "").

end.

message "Running".

find _file no-lock
where _file._file-name = TableName
no-error.

if not available _file then return.

TableName = Capitalize(trim(_file._file-name)).

CodeFile = CodeFile + Capitalize(TableName) + "Manager.cls".

output to value (CodeFile).

put unformatted  skip.
put unformatted  "/* Warning: This is generated code. */" skip.
put unformatted  skip.

put unformatted  "class " Capitalize(TableName)  "Manager:" skip.
put unformatted skip(1).
put unformatted "  define temp-table TheRecord like " trim(_file._file-
name) "." skip.
put unformatted "  define buffer Working" trim(_file._file-name) " for "
trim(_file._file-name) "." skip.
put unformatted "  define private variable IsNew as logical no-undo." skip.
put unformatted "  define private variable ErrorCode as character no-undo."
skip.

put unformatted  skip(2).
put unformatted "
/**********************************************************************/"
skip.
put unformatted "  /* Constructor
*/" skip.
put unformatted "
/**********************************************************************/"
skip.
put unformatted skip (1).
put unformatted "  constructor public " TableName "Manager():"  skip(1).
put unformatted "  end. /* constructor */" skip(2).

put unformatted  skip(2).
put unformatted "
/**********************************************************************/"
skip.
put unformatted "  /* Destructor
*/" skip.
put unformatted "
/**********************************************************************/"
skip.
put unformatted skip (1).
put unformatted "  destructor public " TableName "Manager():"  skip(1).
put unformatted "  end. /* destructor */" skip(2).

put unformatted  skip(2).
put unformatted "
/**********************************************************************/"
skip.
put unformatted "  /* Commit changes
*/" skip.
put unformatted "
/**********************************************************************/"
skip.
put unformatted skip (1).
put unformatted "  method public logical Commit():"  skip(1).
put unformatted "    if IsNew then create Working" TableName "." skip(1).
put unformatted "    buffer-copy TheRecord to Working" TableName "."
skip(1).
put unformatted "    ReleaseToOthers()." skip(1).
put unformatted "  end. /* method */" skip(2).

put unformatted  skip(2).
put unformatted "
/*********************************************************************/"
skip.
put unformatted "  /* Release changes
*/" skip.
put unformatted "
/*********************************************************************/"
skip.
put unformatted skip (1).
put unformatted "  method public logical ReleaseToOthers():"  skip(1).
put unformatted "    release Working" TableName "." skip(1).
put unformatted "  end. /* method */" skip(2).

put unformatted  skip(2).
put unformatted  "
/*********************************************************************/"
skip.
put unformatted  "  /* Rollback changes
*/" skip.
put unformatted  "  /* A RollBack() after a Commit() is pretty much
useless.  Use UNDO.    */" skip.
put unformatted  "
/*********************************************************************/"
skip.
put unformatted skip (1).
put unformatted "  method public logical RollBack():"  skip(1).
put unformatted "    empty temp-table TheRecord." skip.
put unformatted "    IsNew = false." skip(1).
put unformatted "    ReleaseToOthers()." skip(1).
put unformatted "  end. /* method */" skip(2).

for each _field of _file no-lock:

  put unformatted  skip(2).
  put unformatted  "
/*********************************************************************/"
skip.
  put unformatted  "  /* Set/Get for " _field._field-name " */" skip.
  put unformatted  "
/*********************************************************************/"
skip.
  put unformatted skip (1).
  put unformatted  "  method public void Set"
Capitalize(trim(_field._field-name)) "(input TheValue as "
trim(_field._data-type) "):" skip.
  put unformatted  skip(1).
  put unformatted  "    SetError(~"000~")." skip(1).
  put unformatted  "    /* Ensure the record is available from a FindBy
method */" skip (1).
  put unformatted  "    if not available TheRecord then do:" skip(1).
  put unformatted  "      SetError(~"001~")." skip.
  put unformatted  "      return." skip(1).
  put unformatted  "    end. /* if */" skip(1).
  put unformatted  "    /* All is OK.  Set the value. */" skip (1).
  put unformatted  "    TheRecord." Capitalize(trim(_field._field-name)) "
= TheValue." skip.
  put unformatted  skip(1).
  put unformatted  "  end. /* Set" trim(_field._field-name) " */" skip.
  put unformatted  skip(2).
  put unformatted  "  method public " trim(_field._data-type) " Get"
Capitalize(trim(_field._field-name)) "():" skip.
  put unformatted  skip(1).
  put unformatted  "    SetError(~"000~")." skip(1).
  put unformatted  "    /* Ensure the record is available from a FindBy
method */" skip (1).
  put unformatted  "    if not available TheRecord then do:" skip(1).
  put unformatted  "      SetError(~"001~")." skip.
  put unformatted  "      return ?." skip(1).
  put unformatted  "    end. /* if */" skip(1).
  put unformatted  "    /* All is OK.  Return the value. */" skip (1).
  put unformatted  "    return TheRecord." Capitalize(trim(_field._field-
name)) "." skip.
  put unformatted  skip(1).
  put unformatted  "  end. /* Get" trim(_field._field-name) " */" skip.

end. /* for each */

put unformatted  skip(2).
put unformatted  "
/*********************************************************************/"
skip.
put unformatted  "  /* Provide a means to query any errors occurred.
*/" skip.
put unformatted  "
/*********************************************************************/"
skip.
put unformatted skip (1).
put unformatted "  method public character GetError():" skip(1).
put unformatted "    return ErrorCode." skip(1).
put unformatted "  end. /* GetError */" skip.

put unformatted  skip(2).
put unformatted  "
/*********************************************************************/"
skip.
```

```
put unformatted  "  /* Provide a means to set the error code and message.
*/" skip.
put unformatted  "
/*********************************************************************/"
skip.
put unformatted skip (1).
put unformatted "  method private character SetError(input ErrorNumber as
character):" skip(1).
put unformatted "    case ErrorNumber:" skip(1).
put unformatted "      when ~"000~" then ErrorCode = ErrorNumber + ~":No
Error~"." skip.
put unformatted "      when ~"001~" then ErrorCode = ErrorNumber + ~":No
Record~"." skip(1).
put unformatted "    end. /* case */" skip (1).
put unformatted "  end. /* SetError */" skip.

put unformatted  skip(2).
put unformatted  "
/*********************************************************************/"
skip.
put unformatted  "  /* Provide a means to reset the error code and message.
*/" skip.
put unformatted  "
/*********************************************************************/"
skip.
put unformatted skip (1).
put unformatted "  method private void ResetError():" skip(1).
put unformatted "    SetError(~"000~")." skip(1).
put unformatted "  end. /* ResetError */" skip.


for each _Index no-lock
  where _Index._File-RecID = recid(_File)
    /* and _Index._Unique = true */
    :

  FindByName = "".
  FindByParameters = "  (~n".
  FindByWhere = "".

  for each _Index-Field OF _Index no-lock,
      each _Field OF _Index-Field no-lock:

    FindByName = FindByName + RemoveDash(Capitalize(_Field._Field-Name)).

    if FindByWhere = "" then
      FindByWhere = "    where Working" + TableName + "." +
Capitalize(_Field._Field-Name)
                + " = " + Capitalize(_Field._Field-Name) + "~n".
    else
      FindByWhere = FindByWhere
                  + "      and Working" + TableName + "." +
Capitalize(_Field._Field-Name)
                  + " = " + Capitalize(_Field._Field-Name) + "~n".

    FindByParameters = FindByParameters
                       + "    input " + Capitalize(_Field._Field-Name) + " as
" + trim(_Field._Data-type) + ",~n".

  end. /* for each _Index-Field */

  FindByParameters = substring (FindByParameters, 1,
length(FindByParameters) - 2) + "~n    ):~n".

  put unformatted  skip(2).
  put unformatted  "
/*****************************************************************/"
skip.
  put unformatted  "  /* Provide a FindBy*() to update a record with or to
query values by.  */" skip.
  put unformatted  "
/*****************************************************************/"
skip.
  put unformatted skip (1).
  put unformatted "  method public logical FindBy" FindByName skip.
  put unformatted "  " FindByParameters skip.
  put unformatted "    find Working" TableName " no-lock " skip.
  put unformatted FindByWhere .
  put unformatted "    no-error." skip (2).
  put unformatted "    if available Working" TableName " then do:" skip.
  put unformatted "      IsNew = false." skip.
  put unformatted "      create TheRecord." skip.
  put unformatted "      buffer-copy Working" TableName " to TheRecord."
skip.
  put unformatted "    end. /* if available */" skip(1).
  put unformatted "    return available Working" TableName " ." skip(1).
  put unformatted "  end. /* method */" skip(2).

end. /* for each _Index */

put unformatted  "
/*****************************************************************/"
skip.
put unformatted  "  /* After using a FindBy*() and want to update - use
this.             */" skip.
put unformatted  "
/*****************************************************************/"
skip.
```

```
put unformatted skip (1).
put unformatted  "  method public logical LockRecord ():" skip(1).
put unformatted  "    SetError(~"000~")." skip(1).
put unformatted  "    /* Ensure the record is available from a FindBy
method */" skip (1).
put unformatted  "    if not available Working" TableName " then do:"
skip(1).
put unformatted  "      SetError(~"001~")." skip.
put unformatted  "      return false." skip(1).
put unformatted  "    end. /* if */" skip(1).
put unformatted  "    find current Working" TableName " exclusive-lock no-
error." skip(1).
put unformatted  "    return available Working" TableName "." skip(1).
put unformatted  "  end. /* method */" skip.

put unformatted  skip(2).
put unformatted  "
/*********************************************************************/"
skip.
put unformatted  "  /* Provide a means to create a new record with.
*/" skip.
put unformatted  "
/*********************************************************************/"
skip.
put unformatted skip (1).
put unformatted "  method public logical CreateRecord():"  skip(1).
put unformatted "    create TheRecord." skip(1).
put unformatted "    IsNew = true." skip.
put unformatted "    return available TheRecord." skip(1).
put unformatted "  end. /* method */" skip(2).

put unformatted  "end. /* class */" skip(2).

put unformatted "/*************************** UNIT TEST CODE
**********************" skip(2).
put unformatted " define variable N as class " TableName "Manager no-undo."
skip.
put unformatted " N = new " TableName "Manager()." skip(1).
put unformatted " delete object N." skip(2).
put unformatted
"******************************************************************/"
skip.

output close.



/****************************** UNIT TEST CODE
***********************************
```

```
run e:\code\ClassGeneratorWithCommit.p ("e:\code\", "Discussion").

compile e:\code\DiscussionManager.cls.

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

Appendix – Revised dyn_findinschema.p Program

This is for use with the Dynamic Toolkit. To fit it in you will probably need to rework the code around the call a little bit.

```
/*************************************************************************
  PROCEDURE             : utils/db/isdbtable.p

  CHANGES MADE AND DATE:
  Date      Version      Description

  11/08/06  LINX#daynem  Initial Version.
  03/10/06  daynem       Rework to use dynamic query rather than change DB.
************************************************************************/

DEFINE INPUT  PARAMETER icTableName          AS CHARACTER NO-UNDO.
DEFINE OUTPUT PARAMETER olIsTable            AS LOGICAL NO-UNDO.
DEFINE OUTPUT PARAMETER oiDatabaseNumber     AS INTEGER NO-UNDO.

DEFINE VARIABLE lhFileBuffer    AS HANDLE NO-UNDO.
DEFINE VARIABLE lhQuery         AS HANDLE NO-UNDO.

DEFINE VARIABLE lcQuery         AS CHARACTER NO-UNDO.

DEFINE VARIABLE lcDbName        AS CHARACTER NO-UNDO.

CREATE WIDGET-POOL "ISDBTABLE".

DB_LOOP:
DO oiDatabaseNumber = 1 TO NUM-DBS:

    lcDbName = LDBNAME ( oiDatabaseNumber ).

    lcQuery =
        "FOR EACH " + lcDbName + "._file NO-LOCK      " +
        "   WHERE " + lcDbName + "._file._file-name = " + QUOTER
( icTableName ).

    CREATE BUFFER lhFileBuffer
        FOR TABLE ( lcDbName + "._file" )
        IN WIDGET-POOL "ISDBTABLE".

    CREATE QUERY lhQuery
        IN WIDGET-POOL "ISDBTABLE".
    lhQuery:ADD-BUFFER ( lhFileBuffer ).
    lhQuery:QUERY-PREPARE ( lcQuery ).
    lhQuery:QUERY-OPEN ( ).
    lhQuery:GET-FIRST ( ).

    IF NOT lhQuery:QUERY-OFF-END THEN
    DO:
        olIsTable = TRUE.
        LEAVE DB_LOOP.
    END.

END.

IF NOT olIsTable THEN
    oiDatabaseNumber = ?.

DELETE WIDGET-POOL "ISDBTABLE".
```

Appendix – Scratch Table Data Dictionary Listing

```
============================================================================
============================= Table: Scratch ===============================

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


Table                             Table Field Index Table
Name                              Flags Count Count Label
-------------------------------- ----- ----- ----- -----------------------
Scratch                                   14     2 Scratch

    Dump Name: scratch
  Description: Transitory data collection.
 Storage Area: Schema Area


============================= FIELD SUMMARY =============================
============================= Table: Scratch ============================

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

Order Field Name                       Data Type   Flags
----- -------------------------------- ----------- -----
   10 CreateDate                       date        i
   20 CreateTime                       deci-2      i
   30 CreationProgram                  char
   40 DataName                         char        i
   50 StructureName                    char
   90 Data1                            char
  100 Data2                            char
  110 Data3                            char
  120 K1                               char
  130 K2                               char
  140 K3                               char

Field Name                        Format
-------------------------------- -----------------------------
CreateDate                        99/99/99
CreateTime                        ->>,>>9.99
CreationProgram                   x(8)
DataName                          x(8)
StructureName                     x(8)
Data1                             x(8)
Data2                             x(8)
Data3                             x(8)
K1                                x(8)
K2                                x(8)
K3                                x(8)

Field Name                        Initial
-------------------------------- -----------------------------
CreateDate                        ?
```

```
CreateTime                      0
CreationProgram
DataName
StructureName
Data1
Data2
Data3
K1
K2
K3

Field Name                     Label                  Column Label
------------------------------ ---------------------- ----------------------
CreateDate                     CreateDate             CreateDate
CreateTime                     CreateTime             CreateTime
CreationProgram                CreationProgram        CreationProgram
DataName                       DataName               DataName
StructureName                  StructureName          StructureName
Data1                          Data1                  Data1
Data2                          Data2                  Data2
Data3                          Data3                  Data3
K1                             ?                      ?
K2                             ?                      ?
K3                             ?                      ?


============================= INDEX SUMMARY =============================
============================= Table: Scratch ============================

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

Flags Index Name                       Cnt Field Name
----- -------------------------------- --- ---------------------------------
      key1                               2 + CreateDate
                                           + CreateTime

p     key5                               1 + DataName

** Index Name: key1
 Storage Area: Schema Area
** Index Name: key5
 Storage Area: Schema Area


============================= FIELD DETAILS =============================
============================= Table: Scratch ============================

** Field Name: CreateDate
  Description: Date alert created.

** Field Name: CreateTime
  Description: Time alert created.

** Field Name: CreationProgram
  Description: Name of program that threw the alert.
```

```
** Field Name: DataName
  Description: Name of the data set.

** Field Name: StructureName
  Description: Identify the type of data stored.

** Field Name: Data1
  Description: Data Field.

** Field Name: Data2
  Description: Data Field.

** Field Name: Data3
  Description: Data Field.
```

APPENDIX: DISCUSSION TABLE DATA DICTIONARY LISTING

```
10/10/08 12:14:49       PROGRESS Report
Database: forum (PROGRESS)



============================================================================
============================= Table: Discussion ==========================

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


Table                            Table Field Index Table
Name                             Flags Count Count Label
-------------------------------- ----- ----- ----- -------------------------
Discussion                                   7     2 Discussion

    Dump Name: discussion
  Description: Contains the discussion records
 Storage Area: Schema Area


============================= FIELD SUMMARY =============================
============================= Table: Discussion ==========================

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

Order Field Name                         Data Type   Flags
----- -------------------------------- ----------- -----
   10 DiscussionID                     char        i
   20 RoomID                           char        i
   30 FromIMUserID                     char        i
   40 ToIMUserID                       char        i
   50 MessageText                      char
   60 CreateDate                       date        i
   70 CreateTime                       inte        i

Field Name                       Format
-------------------------------- -----------------------------
DiscussionID                     x(70)
RoomID                           x(70)
FromIMUserID                     x(70)
ToIMUserID                       x(70)
MessageText                      x(80)
CreateDate                       99/99/99
CreateTime                       ->,>>>,>>9
```

```
Field Name                       Initial
-------------------------------- -----------------------------
DiscussionID
RoomID
FromIMUserID
ToIMUserID
MessageText
CreateDate                       TODAY
CreateTime                       0

Field Name                     Label                  Column Label
------------------------------ ----------------------
----------------------
DiscussionID                   DiscussionID           DiscussionID
RoomID                         RoomID                 RoomID
FromIMUserID                   FromIMUserID           FromIMUserID
ToIMUserID                     ToIMUserID             ToIMUserID
MessageText                    MessageText            MessageText
CreateDate                     CreateDate             CreateDate
CreateTime                     CreateTime             CreateTime


============================= INDEX SUMMARY =============================
============================= Table: Discussion =========================

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

Flags Index Name                       Cnt Field Name
----- -------------------------------- ---
--------------------------------
p     DiscussionID                       1 + DiscussionID

      RoomFromToDateTime                 5 + RoomID
                                           + FromIMUserID
                                           + ToIMUserID
                                           + CreateDate
                                           + CreateTime

** Index Name: DiscussionID
 Storage Area: Schema Area
** Index Name: RoomFromToDateTime
 Storage Area: Schema Area


============================= FIELD DETAILS =============================
============================= Table: Discussion =========================
** Field Name: DiscussionID
  Description: Unique identifier for the record

** Field Name: RoomID
  Description: Which room does this discussion take place in

** Field Name: FromIMUserID
  Description: Who is the discussion from?

** Field Name: ToIMUserID
  Description: Who is this discussion meant to be read by?

** Field Name: MessageText
  Description: Text of the message between users.

** Field Name: CreateDate
  Description: Date message was created

** Field Name: CreateTime
  Description: Time message was created.
```

APPENDIX: SYSPARAMETER TABLE DATA DICTIONARY LISTING

```
10/10/08 13:28:13       PROGRESS Report
Database: forum (PROGRESS)



=========================================================================
============================= Table: SysParameter =======================

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


Table                             Table Field Index Table
Name                              Flags Count Count Label
--------------------------------- ----- ----- ----- -------------------------
SysParameter                                3     1 SysParameter

    Dump Name: sysparameter
  Description: Hold parameters of operation
 Storage Area: Schema Area


============================= FIELD SUMMARY =============================
============================= Table: SysParameter =======================

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

Order Field Name                        Data Type   Flags
----- --------------------------------- ----------- -----
   10 ParameterName                     char        i
   20 Data                              char
   30 Comment                           char

Field Name                        Format
--------------------------------- -----------------------------
ParameterName                     x(40)
Data                              x(8)
Comment                           x(8)

Field Name                        Initial
--------------------------------- -----------------------------
ParameterName
Data
Comment
```

```
Field Name                     Label                  Column Label
------------------------------ ----------------------
----------------------
ParameterName                  ParameterName          ParameterName
Data                           Data                   Data
Comment                        Comment                Comment


============================= INDEX SUMMARY =============================
============================= Table: SysParameter =======================

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

Flags Index Name                       Cnt Field Name
----- -------------------------------- ---
---------------------------------
p     ParameterName                      1 + ParameterName

** Index Name: ParameterName
 Storage Area: Schema Area


============================= FIELD DETAILS =============================
============================= Table: SysParameter =======================

** Field Name: ParameterName
  Description: Name of the parameter.

** Field Name: Data
  Description: Setting for the parameter

** Field Name: Comment
  Description: Some description of the parameter's purpose.
```

Appendix: Other Useful Toolkits & Information

PDF Include

PDF Include is a great open source software package written completely in the ABL language for operation system independence. It is a set of APIs programmers can use to create and manipulate PDF files.

http://sourceforge.net/projects/pdf-inc

freeframework.org

The Free Framework project provides a set of code you can use in your Webspeed application for increased functionality. It is worth looking over to see if it has anything of interest to your work.

http://www.oehive.org/

OE Hive

OE Hive is a collection of papers and open source code for Progress ABL programmers. There is a lot of good stuff on here.

http://www.oehive.org/

peg.com

PEG (Progress Email Group) is the grand daddy of groups of Progress developers and users supporting each other regarding all kinds of issues on a mailing list. In addition to running the mailing list, the site has papers and source code one can make use of in your work.

http://peg.com/

Progress Talk

Progress talk is the web version of peg.com run by another group.

http://progresstalk.com/

Progress PSDN

Progress Software Developers Network is a good source hosted by Progress by a community of developers for white papers, code, pod casts, and screen casts.

http://www.psdn.com/library/index.jspa

About The Author

Scott Augé founded and is president of Amduus Information Works, Inc. He has worked with Progress technologies since version 6 - over ten years.

Working primarily with UNIX based systems he has written enterprise class applications supporting tens of thousands of users to small systems supporting tens of users. His domain experience includes manufacturing, e-commerce, judicial/law enforcement, travel and service management industries.

He can be reached at scottauge@gmail.com or sauge@amduus.com.