

PROGRESS

The Building Blocks of Progress

Progress Conference Double Issue
Spring 2004 Number 58 & 59

Although Progress Software celebrated its 20th anniversary a few years back, marking 20 years since they started to work on the product, most of us would recognize this year as the 20th year of Progress, since they started selling their product in 1984. What can be said about the past 20 years? Certainly, one of the most instructive insights we can have is to look at these years from several different perspectives: who was in business then, what was the software industry like at that time, what business models have worked over the years?

Digital, Wang, Amdahl, Altos, Atari, Ashton-Tate, Borland, Software Arts, Lotus, Santa

Cruz Operation. There were lots of very well-known companies which were doing very, very well at that time which either do not exist today, or have been entirely absorbed into some other organization. Many people are not aware that Progress is one of the oldest high-tech companies in the 128 corridor. Lots of businesses have come and gone in that time. Certainly, a few such as Oracle, have grown quite large in that time. Informix was absorbed by IBM so its financials are a little tough to tease out. Sybase, another erstwhile competitor, seems to be surviving. Certainly, there are many people in the Progress community who wish the company were bigger. But when all is said and done, both Silicon Valley and Route 128 are a completely different landscape from 20 years ago.

Progress has done very well to survive during these twenty years. And what of the technology changes?

Microsoft was not really much of a player in 1984. And for all their phenomenal growth, how many people actually like their products? Multi-user environments were rare in small businesses. Client-server was not really an existing technology. The web was not even being dreamed of by most computer users. We've seen a lot of technology, a lot of fads, a lot of initiatives come and go during those two decades. Progress has continued to deliver a product which has managed to stick to the fundamentals, and continued to be a high-quality, well-integrated product.

20 Years of Progress

Progress has always taken a different approach to what its

market is than most other database, and indeed, many other software companies. Certainly, Progress has always cultivated its application partners. While at times there were companies, even big ones, with their own in-house applications, most of Progress' business has been from its partners. And while some may move away from Progress, or undertake initiatives in other technical areas, many, if not most, realize that the Progress 4GL is just about the best business development language around, and that the database continues to surprise with its performance, resilience and maintainability. Said differently, the revenue continues to grow in two different perspectives: first, that of the application partners, and second, the international mix of sales. Progress has created a sound business model and stuck to it, albeit rather more conservatively than most other companies in its niche. Its business performance has been stable, and its staying power has been proven.

continued on page 17



Dynamics – Not such a scary word by Gareth Woodhouse

When I first heard the word dynamics my first response was probably the same as yours.... cold sweats, trepidation and fear at what was obviously going to be a real challenge to cram into my head.

However, having conquered my fear and actually looked at what dynamics contains it is actually a sheep in a tiger's skin.

Below I will attempt to explain a simple dynamics example and, hopefully, open a few people's eyes to one of the most useful yet unused functions that Progress GUI has to offer.

The basis of dynamics is the creation of code that can be compiled without any attached databases... no hard coding and no limitations. At first glance that looks like quite a tall order and for the developer a nightmare to develop. However, with the help of about half a dozen commands this is not as hard as you may think.

Firstly - Dynamic Queries

In order for a widget object to display information dynamically you will need a query to supply the records (note - You could use a standard hard coded query. It is not mandatory that a dynamic query be used when dealing with dynamic widgets).

The first step in creating a dynamic query is creating the code the query will run. This can be done as simply as the following:

```
DEFINE VARIABLE ip_tablename AS CHARACTER
NO-UNDO INIT "Customer".
DEFINE VARIABLE l_codestring AS CHARACTER
NO-UNDO.
DEFINE VARIABLE h_Query          AS HANDLE
NO-UNDO.

DELETE WIDGET-POOL "dynpool" NO-ERROR.
CREATE WIDGET-POOL "dynpool" PERSISTENT.

FIND FIRST _FILE
WHERE _FILE._FILE-NAME = ip_tablename
NO-LOCK NO-ERROR.

ASSIGN l_codestring = "FOR EACH " +
ip_tablename + " NO-LOCK."
```

This will give you a simple query that will return records from the Customer table.

Second - Create a buffer for the query results.

```
CREATE BUFFER h_buffer FOR TABLE
_FILE._file-name IN WIDGET-POOL "dynpool".
```

This is so you can manipulate the results (i.e. change the column-labels for display purposes etc).

Now the magic:

The code below creates the query h_query which we defined earlier and then takes the code string you created in the previous step and makes it the code your query will run. No hard coding and very neat. Lastly the query is opened and the result set returned into your buffer.

```
CREATE QUERY h_Query IN WIDGET-POOL
"dynpool".

h_Query:ADD-BUFFER(h_buffer).
h_Query:QUERY-PREPARE(l_codestring).
h_Query:QUERY-OPEN().
```

And there you have it.... you have created a dynamic query. This code can be compiled with no attached databases and has no hard coding. Next we will take our result set and display it in a dynamic Browse.

Firstly you must define and create the browse

```
DEFINE VARIABLE h_browse          AS HANDLE
NO-UNDO.
DEFINE VARIABLE ip_fieldnames    AS CHARACTER
NO-UNDO INIT "Custnum, Country, Name".
DEFINE VARIABLE l_entry          AS INTEGER
NO-UNDO.

CREATE BROWSE h_browse
ASSIGN TITLE =
"Browse for " + STRING(_FILE._FILE-NAME)
FRAME          = FRAME fr_disp:HANDLE
QUERY         = h_Query:HANDLE
X              = 2
Y              = 2
WIDTH         = 96
DOWN          = 16
VISIBLE       = YES
SENSITIVE     = TRUE
READ-ONLY    = NO.
```

Then fill the browse with only the fields you have passed in via an input parameter

```
ASSIGN l_i = 1.

DO l_entry = 1 TO
  NUM-ENTRIES(ip_fieldnames, ","):

  h_col[l_i]= h_browse:ADD-LIKE-COLUMN(
    (_FILE._file-name + "." +
    ENTRY(l_entry, ip_fieldnames, ", "))).

  ASSIGN l_i = l_i + 1.

END.
```

You could of course display all the fields simply by using the `_Field` table instead of passing in values from an Input parameter but this way gives you more control over what is being displayed.

You can manipulate the browse size etc. by amending the "Create browse" statement, but be aware that any amendments must be done before the browse has been

created: once it's been realized and displayed on screen you can't amend it any further.

In just a few lines of code we have created a dynamic query and displayed the result set in a dynamic Browse.

This is just a very simple example of using dynamics and I hope it has opened your eyes to what can be achieved with only a handful of commands....

Gareth Woodhouse

Gareth is an Analyst Programmer for a large insurance company in London. He's been working with Progress for 5 years and started on version 8.3b whilst working for Van Meijel in Holland.

When not developing new applications you can usually find him either watching, playing or talking about football.

He can be reached at
garethwoodhouse@hotmail.com



JARGON™

- **Develop powerful Wireless applications in our new V3 Jargon Writer!**
- **Run wireless apps either online or offline when out of coverage area**
- **Create "bolt-on" solutions to your Progress app in days, not months.**
- **Integrate barcode scanners, printers and other wireless peripherals**
- **Deploy with AppServer or WebSpeed. Save money with AppServer.**
- **Enjoy easy, powerful wireless deployment for under \$150 per handheld.**
- **Leverage your 4GL experience - no need to master Java, C++, or HTML.**
- **Download a FREE Evaluation Copy or contact us for a personal online demo**

JARGON SOFTWARE Inc
 708 North First Street, Suite 432
 Minneapolis, Minnesota 55401
 612 338 1175 | fax 612 338 2974
www.jargonsoft.com (On-line demo!)
info@jargonsoft.com

Go Find Yourself..... by Jop Kluis

Every now and then I get questions about Data Quality projects. Mostly those questions have to do with Business Cases, specific project demands or organizational embedding of data stewardship. However, recently I got a really down-to-earth practical question about data cleansing. This organization was building a corporate data warehouse and had a vast number of data sources which they had to bring together. In many of those sources they held customer information and in order to combine this information into one 'general' record they needed to match the customers in all their sources. The problem was that in each of those sources they used different conventions and they knew that the quality of the data was not really up to standard. "How do we match these sources?"

In order to explain what we did I have simplified the situation for your benefit. In the first diagram (figure 1) you will find five customer records, each from a different source. In my example all these customers are in fact one and the same person: me.

#	Name	Address	City	Phone
1	J. Kluis	Beethoven Street 51	Amsterdam	020 - 2371731
2	Jop Kluis	Beethoven St 51	Amsterdam, NL	+31 20 2371731
3	Kluis J.	Beehtoven Street 51	A'dam	06 51 58 01 81
4	Johannes Kluis	Beethoven Ave	Amstelveen	031 20 2371731
5	Kluis, Jop	Beethoven Street 15	Amsterdam	(031)(0)20 23 71 731

Figure 1

Even though each of the records describes the same person, they use different conventions. (*Johannes* is my given name and *Jop* is the name I use) With the addresses the problem is not limited to the convention. In source #3 there is a transposition of letters (*Beehtoven* instead of *Beethoven*). In source #4 they accidentally used *Avenue* instead of street and in source #5 there is a typo in the street number.

With the cities they used a different convention in source #2 (country code added) and used a common abbreviation in source #3. The city in source #4 is just dead wrong.

And the phone numbers are really a mess. Although they are all correct (#3 is a cell phone) none matches any of the others.

Heuristic Matching

The most common and simplest approach is heuristic. This means that you use a simple find in order to get a match. Let's say I start with data source #1 and look to see if I can find a simple match in one of the other sources. Unfortunately, there is no match on name, no match on phone, no match on address and one match on City. Even if I could have found another *Beethoven Street 15* it would not automatically be a match. Beethoven Street is a common street name in the Netherlands and can be found in over 40 different cities. So a match on address is as such not deterministic. The match on city is of no use either. In Amsterdam there are over 700,000 people, not a few of whom are named Kluis.

And this is just the first record from data source #1. I still have another 280.000 records to go! So this simple approach is not going to help us very much.

The 'finding approach' becomes more helpful if we are able to combine matches between different fields. For example, could we find a record in another source that would match on multiple fields with our record from data source #1? This information becomes more helpful if we know that a match is found on two fields when put together are likely to form one unique address. Finding a match for *Beethoven Street 51* in combination with *Amsterdam* does

help. A match on Name and City is less likely to be unique, especially if the name is a common name and the city is a big one. You can imagine that a match on the combination of a year of birth and a gender is not worth much. So combining fields helps, but this often still proves not to be enough.

The result of a heuristic search becomes more interesting if we search on strings with a 'begins with' approach. With this approach we can calculate and compare a reducing string with the beginning of each field. So we start with comparing *J. Kluis* from data source #1 with the names in the other sources. If we do find a 'full match' this would be a 100% score on this field. In our example we will not find a 'full match'. So we compare *J. Klui*, then *J. Klu*, etc. This way we will find that the record from source #1 has:

Source	Name	Address	City	Phone	Total
#2	12.5%	63.2%	100.0%	0.0%	43.9%
#3	0.0%	15.8%	11.1%	89.5%	8.7%
#4	12.5%	52.6%	45.5%	100.0%	29.6%
#5	0.0%	89.5%	100.0%	0.0%	47.4%

Figure 2

Remember that each of the four records from the other data sources represent the same person as the person in data source #1. You see that each of the records from the other sources has a different 'hit rate'. The idea behind this approach is that these records should come out with a much better hit rate than let us say: *Piet Patser, Geranium Street 54, Dordrecht, 071 – 65 76 19 63*.

By comparing hit rates from combined field matches we introduce an algorithm known as the probability algorithm. With this approach we calculate that it is 47.5% probable that the record from data source #5 is in fact he same person as in data source #1. But this probability approach, although helpful, does not cut it. Each of the example records from the four other data sources has a score below 50%.

Transforming conventions

Now, remember that the difference between the content of the different records is not just an accidental transposition of letters, a mistake or typos. A substantial part of the differences has to do with the difference between the conventions used. In the figures 3 and 4 you can find the conventions/formats of data source #1 and data source #5.

Format data source #1	
name	first initial .space family name
Address	Street or roadname space number
city	name
phone	area code space.space number

Figure 3

Format data source #5	
name	Family name ,space first name
Address	Street or roadname space number
City	name
Phone	(country code) (0) area code minus preceding zero space first two digits of the number space third and fourth digits of the number space

Figure 4

6

In data source #1 the conventions are: we start with the first initial of the given name, followed by a ". ". The family name is preceded by a *space*. In data source # 5 we start with the family name, followed by a ", ". Then a *space* and the first given name. There are some differences in the address and the phone number as well. Now, it is relatively simple to transform the records from each of the data sources to another convention / format. As an example I created a new format that can be seen in figure 5.

New format for all data data sources	
name	Family name space first initial
Address	Street or roadname
number	number
City	name
Phone	country code area code minus preceeding zero number

Figure 5

In this convention not only do we re-order particular elements from the data attributes, but we also give the number from the address a field of its own. On top we stripped the address from extensions like : Street, Avenue, Square, St., Ave, etc. The cities are stripped from the country codes. and the phone numbers are stripped from: *space*, (,), - and preceding zeroes. Then I add the country code to the numbers which do not have such a code at the beginning. I'm sure that each and every one of you can figure out the code necessary to transform the data from the old formats into the new format. When this particular part of our puzzle is solved the data from the different data sources will look like:

#	Name	Address	City	Phone
1	Kluis J	Beethoven	51 Amsterdam	31202371731
2	Kluis J	Beethoven	51 Amsterdam	31202371731
3	Kluis J	Beehtoven	51 A'dam	651580181
4	Kluis J	Beethoven	Amstelveen	31202371731
5	Kluis J	Beethoven	15 Amsterdam	31202371731

Figure 6

We see instantly that we have a much better matching chance then we had before. As a matter of fact, if we would apply the combination of the heuristic algorithm with the reducing string and the probability approach we would find the following values:

Source	Name	Address	number	City	Phone	Total
#2	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
#3	100.0%	33.3%	100.0%	11.1%	0.0%	48.8%
#4	100.0%	100.0%	100.0%	55.6%	100.0%	91.1%
#5	100.0%	100.0%	0.0%	100.0%	100.0%	80.0%

Figure 7

Figure 7 shows that the probability rates have improved dramatically by applying a relatively simple transformation. It is not just that the possible matches get a better rating, but the difference with the unlikely matches gets bigger. Therefore, it becomes easier to identify the candidates. Just an example, suppose we have a record in data source #3 that looks like the following:

3	Hofstede P.	Beethoven Park 88	Amstelveen, NL	06 94 98 63 59
---	-------------	-------------------	----------------	----------------

Figure 8

Using the combination of the heuristic algorithm with the reducing string and the probability calculation this record would score: 0% on name, 52.6% on address, 55.5% on city and 9% on phone number, creating a total score of 28.9%.

This while the record:

3	Kluis J.	Beehtoven Street 51	A'dam	06 51 58 01 81
---	----------	---------------------	-------	----------------

Figure 9

only scored 8.7%. So therefore it looks like Mr Peter Hofstede is a better candidate to be the Mr Jop Kluis then Jop Kluis himself?

Now after the transformation process both records look like:

3	Hofstede P	Beethoven	88	Amstelveen	694986359
3	Kluis J	Beehtoven	51	A'dam	651580181

Figure 10

Now we see that the scores are 31.1% and 48.8%. So now the scores are favorable to the real me.

Deterministic approach

In our examples so far, I calculated the probability factor on the basis of the idea that all fields are equally valuable for my goal. This, however, is far from the truth. A phone number has a far bigger deterministic value than, say, a house number. I know that people share telephone numbers (within a family and within companies) but if I find a match on a phone number I know I'm getting close. My cell phone number (with country code) is unique world wide and I am the only person using that particular phone. And with house numbers? I would not even dare to guess how many people live at # 51 in the Netherlands. So the house number only has a deterministic value in combination with the address and the city. Now, if we would calculate the probability score using the deterministic value of the fields you would see a substantial increase in the reliability of my matching procedures.

Another big improvement is found in what's called the empirical algorithm. With this approach we make use of our experience. We can try to match records on the basis of rules we found in the past. Even the few records we try to match in this article I managed to put in an example. In data source # 3 we found the city *A'dam*. As I mentioned earlier this is, in Holland, a common abbreviation for *Amsterdam*. As is *R'dam* a common abbreviation for *Rotterdam* (EMEA headquarters of Progress). With applying rules like this you should keep in mind cultural differences. My brother is called *Dick*, which is an absolutely normal name over here. While in the US this would often be short for *Richard*. An other sensible thing to do is to put these empirical transition rules into a database and use a general mechanism to read and apply these rules. But who am I to remind you to incorporate flexibility into your product?

Possible enhancements

When your hit rate is still too low to come up with a proper matching proposal that will save you a lot of time, you can enhance the approach. How about trying to sort out transpositions of letter. With an iteration you can switch the two letters on each position and see if you get a better matching result. So in order to match Amstelveen I would have

to try all of the following combinations: *amstelveen*, *mastelveen*, *asmtelveen*, *amtselveen*, *amsetlveen*, *amstleveen*, *amstevleen*, *amsteleven*, *amstelvene*. Please notice that I'm only checking on just one transposition. Now, this will be a costly performance, but will still beat comparing five data sources with each of over 200.000 records manually.

Another good mechanism is based on the tenability of data. Some data has almost limitless tenability and some data loses value over time. People move from time to time. Somebody who lived in Amsterdam last week could be living in Rotterdam today. So an address I received 3 months ago is, statistically, a better source for matching than an address I used 15 years ago. And some data has a better tenability. People never change their date of birth (at least men don't). People hardly ever change their gender, and if so not more than once in a lifetime.

Business case

You see there are a lot of possibilities to match data. Some can become pretty complicated, especially when we start combining different techniques. Therefore please keep in mind the balance between effort and benefit. But then again, we responsible professionals always work on the basis of a sound business case...

Jop Kluis

Jop is a Project Manager with Level Up, a Dutch IT consulting company. Database projects, be it design, data warehousing, data quality or BI, always have had his special interest. Jop welcomes comments and challenges. He can be reached through his email:

Jop_kluis@hotmail.com

Progressions

Leo:	John Campbell
Cancer:	Connie Campbell
Pisces:	Harriet Coates
Aquarius:	Michael Bartlett

Progressions is prepared bi-monthly, which means 6 times a year, not 24.

Now that we are an electronic newsletter, we are offering the same price for everyone, no matter where you live! The annual subscription price is \$60.00 USD.

We accept Visa, Mastercard, and/or American Express (please include your expiration date and whether or not you have enough of a credit line for us to head for Katmandu or simply the border). All checks must be drawn on a US bank; checks and money orders must be in US dollars. In order to protect our contributors, the information in this periodical is copyrighted 1992-2004 White Star Software, Inc., and/or the author. All rights reserved worldwide and to the end of the cosmos.

Progressions

PO Box 250

Carbondale, CO 81623

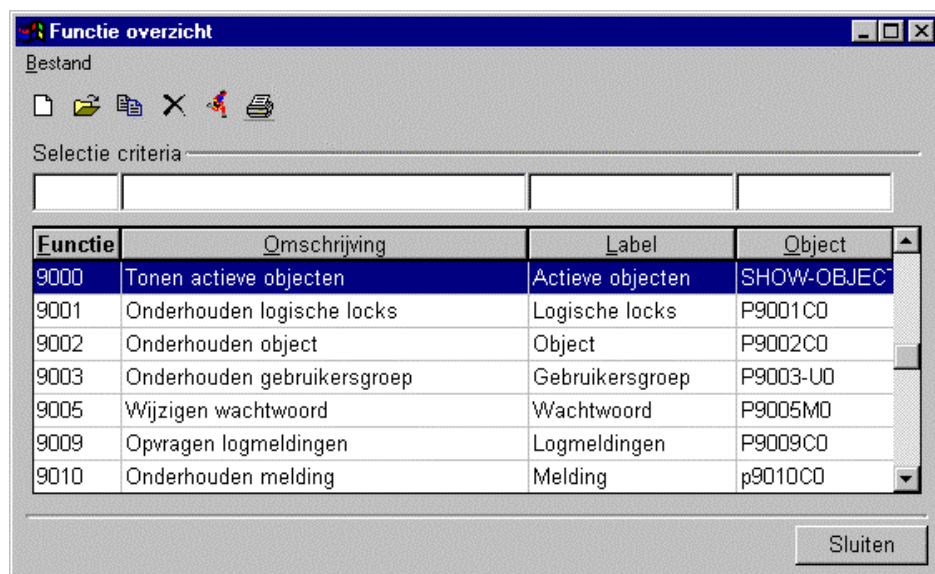
970.963.3545 Voice

970.963.3548 Fax

An Alternative to Publish and Subscribe by John Kattestaart

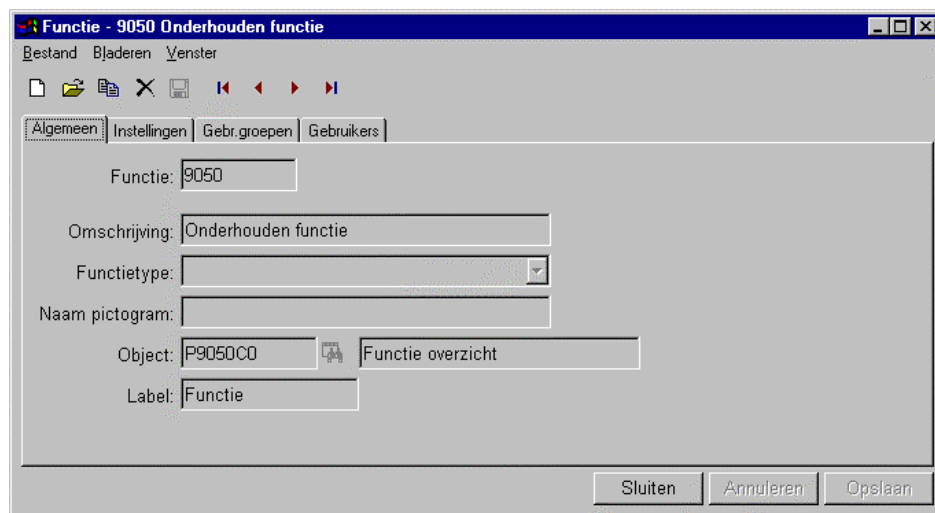
Some years ago Progress Software Corporation (PSC) introduced the Publish and Subscribe method in the 4GL. Using these a form of messaging could be implemented in your application. By subscribing to a particular event you could react to the published event. Technically a sub-procedure was run in response to the event. This is a nice way to inform one or more programs with a single statement, because you don't have to administer the programs involved. So no handles are needed. It's always awkward to deal with all kinds of handles, which make your program code look quite technical. One major goal of writing programs is trying to keep up with the functional description. So Publish and Subscribe could be a step in this direction. So why an alternative to this Publish and Subscribe dogma?

When writing an application we found that most of the time you know to which program you are sending the message. For instance: when you select another record in the browse field you want to inform the detail screen your record has been changed so that it refreshes the screen with this new record. That's why we introduced the addressed sending of messages. Because we divided our application in objects we named this messaging system: O2O – Object-to-Object.



P9050C0

BRW-FUNC



P9050U0

P9050U0A

10

How does this work? At first we will take a closer look at our application window. A standard window application is divided into several objects:

A controlling program with a menu, toolbar and buttons, with a navigate object to browse the records in the database. This program can also contain an update object (which can be in a separated window or on the same window). This update object can contain one or more tabs. Each with its own object. See the example above.

For communication between the objects we introduced a special object and called it EXCHANGE. This because it is equivalent to the Microsoft Exchange: it exchanges messages between objects. The process of exchanging messages we call 'SendMail'.

So one object sends mail to another object by the exchange object. As this is managed by **exchange**, the sender becomes the parent and the recipient becomes the child. So **exchange** registers a relation between the objects. This is used when objects are finished.

Exchange takes care of starting objects. It registers the handles of all programs and their relations. Because **exchange** is aware of all the relations it terminates all the children when a parent is closed. Through this mechanism a message can be delivered to another object.

Suppose a program is started in a menu-program the menu sends a 'START'-message to the program. Written like:

```
{lib/sendmail.i
  &ToObject = 'P9050C0'
  &Subject  = 'START'
}
```

P stands for program (there are also batch programs, starting with 'B')

9050 is the number of the program (9050 = maintenance of function

C stands for controlling program (U = update); 0 is the first program for this number

Exchange picks up this message and checks to see if there is already a relationship defined for these objects. Because it's the first mail sent this will not be true.

Before the message can be delivered the object needs to be loaded. The object 'P9050C0' is defined in an 'Objc' (Object) table. The 'P9050C0'- record contains the path to the corresponding procedure which is run. Each program has a public procedure which is named "pbReceiveMail" and has the following parameters:

ipFromObjectPh – the handle of the object that places the send mail
ipSubjectCd – the subject of the message
ipMailinfoTx – a text that can be sent along
opReturnValueTx – an output text that is send back

Normally the "ipFromObjectPh" is not used. In the O2O-core there are several programs that use this info. For instance: the **exchange** object self uses it to store the relation-from.

In each "pbReceiveMail" procedure the Subjects are examined and the corresponding actions are taken. The START-subject in this case causes the program to activate itself. One of the actions taken is the navigate object. In the code there is a navigate object which is defined:

```
{cls/control.i
  &Table           = Func
  &NavigateObject = 'BRW-FUNC'
  &ChildObject    = 'P9050U0'
}
```

The BRW-FUNC object is started as well. This object shows a browser with filters where users can select records. As you can see that the code also defines a child object. The child object is a viewer object with one or more tabs where the data is displayed and can be edited. After this object is started the following relations are defined:

```
ROOT -> MENU (menu)
MENU -> P9050C0 (control)
P9050C0 -> BRW-FUNC (browser)
P9050C0 -> P9050U0 (update)
P9050U0 -> P9050U0A (tab)
```

If another record is selected a number of messages are sent:

Of course this is a simplified model. Actually more events occur and more objects are involved.

From	To	Subject	MailInfo	ReturnValue
BRW-FUNC	P9050C0	SELECTION-CHANGED		
P9050C0	P9050U0	SELECTION-CHANGED		
P9050U0	P9050C0	GET-ROWID	Func	
P9050C0	BRW-FUNC	GET-ROWID	Func	Rowid from the selected record
P9050U0	P9050U0A	REFRESH-FRAME		
P9050U0A	P9050U0	GET-ROWID	Func	Rowid of the current record

For instance there is a TRIGGER-object which handles all the UI-things and on field-level there is a type of SMART-FIELD-objects involved, which takes care of all intelligent UI-characteristics as COMBO-BOXes, LIST-ITEMs etc.

Summary

Object-to-Object(O2O) is a bit like Publish and Subscribe for interacting with other programs (objects), without having to deal with handles. The advantage of O2O is that messages are sent to specific objects. There is a loose coupling between objects and objects can be easily reused in or replaced by other objects. You don't have to worry about starting and closing objects. A simple 'START' message will start the object, defined in the Objc-table and a 'CLOSE' message will destroy the object.

Children are automatically closed. There are also multi-parent objects, which will stay in memory until every parent is closed. This object will only have one instance.

The functionality of a program is very easily discovered by examining the "pbReceiveMail" routine because this is the place where the messages arrive and actions are taken on the given subject.

O2O has proven to be a stable development principle and we're using it in all of our projects with great success.

John Kattestaart

John joined the Delias company in Arnhem (Netherlands) five years ago. O2O wasn't a bridge too far for him and he quickly became an impassioned O2O evangelist. He has ten years of experience with the Progress 4GL. Before his introduction to Progress, he developed applications for hospital laboratories in good old Fortran IV on a DEC PDP-11.

He can be reached at: jkattestaart@delias.nl

New Book!

Coding Smart

by Michael Lonski, is a new book on learning and using Progress SmartObjects in real world applications. This practical guide will help you quickly understand and write for GUI, WebClient or AppServer. Order a copy on our website or pick one up in our booth at the Progress Conference in Las Vegas.

AllegroConsultants.com

Customizing ADM2 Classes by Michael Lonski

There are many changes in the structure of ADM2 versus its ancestor, the version 8 ADM, often now called ADM1. One important difference can be seen in all of the ready-made hooks that allow developers to customize SmartObject behavior. More importantly, this is done without forcing these developers into staying static on a specific version of Progress or having to laboriously redo all of the customizations each time their Progress version is patched or updated.

The full source code for the method libraries, SmartObject classes, templates and other pieces of ADM2 code are found in the *src/adm2* directory under the Progress install. If nothing else, this lets the developer look at what is going on under the hood with the various SmartObjects being used. But the real gems as far as we're concerned are in the *src/adm2/custom* directory. The problem is that these gems are given to us in raw form. It is our need to add specific class customizations that drives us to polish them into a more useable form.

As shown in other sections of Coding Smart, the method libraries are the key to any SmartObject master file. This include file makes sure any given SmartObject master has the access to the appropriate class files. The hook we are looking for is where the method library includes that class' customization file. If we use the viewer class as an example, that hook is the include reference of the *src/adm2/custom/viewercustom.i* file, as shown below.

```
/* _ADM-CODE-BLOCK-START _CUSTOM
_INCLUDED-LIB-CUSTOM CUSTOM */
{src/adm2/custom/viewercustom.i}
/* _ADM-CODE-BLOCK-END */
```

Figure 1 - Include Of Viewer Customization File

Now a quick scan of the *viewercustom.i* file can seem disappointing at first, but let's pick it apart and understand exactly what we've got here. Most of the file consists of comments and preprocessor code used by the AppBuilder to interpret the file. The heart and soul of this customization file lies at the very bottom.

```
/* Uncomment the custom super procedure to
run */
```

```
/* RUN start-super-proc
("adm2/custom/viewercustom.p":U) .*/
```

Figure 2 – Commented Call To run Custom SO Class

Unaltered, the *viewercustom.i* file doesn't really do anything. It is the act of un-commenting the **RUN** statement that enables a developer to add the new layer of functionality.

But wait, if you change this include file and add your customizations, won't you risk having them overwritten the next time you patch or upgrade your Progress install? That would most likely be the case if you were foolhardy enough to make your changes in the Progress install directory. The intent here is for the developer to instead copy any needed files over to a matching directory layout under their application's root directory. Don't do this for all of the files in the custom directory. Files should only be copied as needed for specific customizations. Once you've made the copy, that local copy can then be changed to solve your current customization need. Don't worry. The originals aren't exactly going anywhere. Just make sure you don't confuse things by copying over more than just the files needed for the customization that you are currently working on.

So, returning to our example above, you would copy the original versions of the two pieces of the viewer class customization process, *viewercustom.i* and *viewercustom.p*, from their location in *\$DLC/src/adm2/custom* over to the *\$MYAPP/src/adm2/custom* directory. Then you simply remove the comments around the RUN statement in your new copy of *viewercustom.i* so that it looks something like below.

```
/** run the customized class file */
RUN start-super-proc
("adm2/custom/viewercustom.p":U) .
```

Figure 3 - Working Call To Run Custom SO Class

Now it becomes a simple matter of opening the local copy of *viewercustom.p* in the AppBuilder and adding any desired internal procedures and user-defined functions. If one of these internal entries has the same signature as an entry in the matching SmartObject class, it will overload that entry with your own variation of the code. If the entry is something completely new, then you have just added brand new functionality to that SmartObject class.

Still confused about how it all works? Let's break down the chain of events so that it becomes a little more understandable.

1. SmartObject master file executes
2. SO master file includes the method library
3. Method library adds the normal SO class to the local SUPER stack
4. Method library includes the class customization include file
5. Class customization include file adds the custom SO class file to the local SUPER stack
6. Custom SO class file contains internal entries that overload internal procedures and user-defined functions in the normal SO class

Thus we see how the thighbone is connected to the knee bone, which is connected to the shinbone, and so on. Since the custom class file is added to the SUPER stack last, the SUPER stack's LIFO behavior means that we'll look in the custom class file before checking the default class file for any internal entries. Simply recompile any master files that use the affected method library and your changes will now work.

Whenever you patch or upgrade your version of Progress, only the **\$DLC** directory is affected. The modified copies of the customization files in your application directory will be untouched. Once again, all you need to do is recompile your procedures after you have applied a Progress patch. This will ensure that you receive the benefit of any changes in the normal SmartObject classes without overwriting any customizations particular to your application. Of course, there's no guarantee that some other change to a portion of the ADM2 made by the patch won't break one of your customizations. You'll still need to do at least some minimal testing but at least you won't have to recode everything.

Michael Lonski

A founding member of Allegro Consultants, Mike Lonski's expertise in Progress started in 1986 with version 3. Mike has worked with projects in locations ranging from the Pentagon and the US Capitol Building to manufacturing shops all over the North American continent. He is Allegro's primary trainer and a frequent speaker at user groups, regional, national and international conferences. Mike's "Coding Smart", a how-to book on working in ADM2, is already an international hit in the Progress community.

mlonski@allegroconsultants.com

Cross Tab Report Revisited by Paul Guggenheim

Consider the old cross tab report. The temp-table is constructed with a composite key of two fields followed by a statistics field of either an amount or a count.

One key field's values would appear in columns across the top and the other key field's values would appear in rows going down the left side of the report. The key field containing the most values would be the one for the rows of the report. The remaining key's values would go into each column. Because of the limited width of a report page, the programmer had to determine which key values to select for each column or how to group them together meaningfully.

This meant that the programmer had to hard code the values in the columns leaving the rows available for the other key values. This type of report didn't offer a lot of flexibility since a user might want to select a different set of columns each time.

You may be aware of setting labels for interactive programs with side-label frames using the label attribute. However, did you know you can also set the label attribute for the columns of a report? This feature means that the user can pick which field value is desired for each column.

Here is an example illustrating this feature. Suppose that in the sports2000 database we want to see the relationship between items and sales reps. It is simple matter to read the order, orderline and item records to calculate the order value for each sales rep and item combination. It turns out that there are 55 items and 10 sales reps. Because there are so many more items than sales reps, it is logical to assign a given item to each row.

Ten sales reps are too many to be displayed across each column of the report. For horizontal spacing considerations, we will limit the number of sales rep columns to five. Furthermore, let's prompt the user for which 5 sales reps they would like to see on the report using a multiple selection-list. We will then assign the proper label for each sales rep column using the label attribute.

In sritem3.p, a variable called srary is defined with 5 extents, representing the 5 columns in the report for each

14

sales rep. A frame must be defined with the srary variable included in the frame before the label attribute can be assigned. In addition, the scope of the frame must be to the procedure block. This is to prevent the frame from advancing with each iteration of the for each temp-table tsritem block.

The selected sales reps are assigned to the array elements and then blank-filled so that the sales rep code is right adjusted in the column. Even though the srary variable is defined as integer and the label will normally appear right adjusted, the label attribute assignment treats it as left adjusted, making the blank fill necessary. You may be wondering why I didn't do a loop for the array elements and assign this with an integer value for the array element. Unfortunately, Progress doesn't allow you to do this while setting attributes for array elements.

Next, the temp-table tsritem is read and sorted by sales rep and then by item name. In the first for each, the item name is displayed and the srary is displayed showing all zeros in each column.

Next, the total amounts for each sales rep across all item names are displayed using the lookup of the salesrep against the selected screen-values to determine the proper array element. This value is added to the corresponding totary variable element used at the bottom of the page. As mentioned previously, the frame does not advance down since it is scoped to the procedure block.

At the last-of-a particular item name, the total amount for that item (totitem) across the five sales reps is displayed in the last column and the down statement is issued advancing the frame.

Finally, when the last record of the whole report is read, the grand total for each sales rep across each item is displayed at the bottom along with the report grand total.

I hope this helps you produce some nice cross tab reports. For the next issue, I will try to create a cross tab generator, capable of generating a cross tab report for any set of data in the database.

```
/* sritem3.p - cross tab report with custom
top labels. */

def var i as int.
def var srary as int extent 5
    format ">, >>>, >>>9".
def var totary as int extent 5
    format ">, >>>, >>>9".
def var totitem as int label "Total".
def var selsalesrep as char format "x(3)"
    label "Salesrep"
    view-as selection-list inner-chars 8
    inner-lines 10 multiple scrollbar-vertical.

def temp-table tsritem
field tsalesrep like salesrep.salesrep
field titemnum like item.itemnum
field titemname like item.itemname label
    "Item" format "x(20)"
field tamount like orderline.extendedprice
    format ">>>, >>>9"
index srin is unique
tsalesrep
titemnum
.

define frame sr
selsalesrep
with 1 down.

form titemname srary totitem
with stream-io frame body width 120 down.

for each salesrep:
    selsalesrep:add-last(salesrep.salesrep).
end. /* for each salesrep */

update selsalesrep with frame sr.
if num-entries(selsalesrep:screen-value)
ne 5 then do:
    message "The number of sales reps selected
    must be equal to 5"
    view-as alert-box error.
    undo, retry.
end. /* num-entries() ne 5 */
else hide frame sr.

/* build temp-table for report */
for each order
    where
lookup(salesrep, selsalesrep:screen-value)
gt 0,
    each orderline of order,
    each item of orderline:

find tsritem where
tsritem.tsalesrep = order.salesrep and
tsritem.titemnum = item.itemnum
no-error.
```

Item	BBB	DOS	HXM	KIK	RDR	Total
Sailboat	380	6,566	6,426	114	851	14,337
Ski Bindings	3,435	559	2,000	1,932	904	8,830
Ski boots	11,858	8,168	3,104	5,862	1,089	30,081
Ski Gloves	938	869	1,366	1,564	518	5,255
Ski Hat	599	307	410	256	175	1,747
Ski Poles	55	2,178	1,123	0	1,341	4,697
Ski Wax - Red	212	171	200	56	7	646
Sled	306	398	619	896	3,029	5,248
Snorkel	112	829	695	451	419	2,506
Snow Shoes	12,319	9,180	4,388	14,378	2,340	42,605
Soccer ball	161	209	93	489	430	1,382
Surfboard	134	1,641	249	1,097	763	3,884
Swim Goggles	1,800	1,361	342	958	635	5,096
Swimming Trunks	368	434	339	0	136	1,277
Tennis Balls	6	59	51	220	238	574
Tennis Racquet	3,386	755	1,509	5,483	581	11,714
Tennis Shorts	1,108	252	855	428	1,315	3,958
Tent	68	1,938	162	0	1,578	3,746
Volleyball	1,166	1,071	182	467	92	2,978
Water Polo Ball	595	502	2,798	614	1,017	5,526
Water Polo Net	1,539	900	608	580	450	4,077
Wet Suit	20,700	15,019	1,350	12,240	12,319	61,628
Wind Surfer	1,039	308	1,306	0	673	3,326
Total	86,578	121,206	86,966	87,392	65,313	447,455

```
if not available tsritem then do:
```

```
  create tsritem.
  assign tsritem.tsalesrep = order.salesrep
    tsritem.titemnum = item.itemnum
    tsritem.titemname = item.itemname.
  end. /* not available tsritem */
  assign tamount = round(extendedprice,0).
end. /* for each */
```

```
/* must hard code array element
```

```
  when using attributes. */
  srary[1]:label = fill(" ",8 -
length(entry(1,selsalesrep:screen-value)))
    + entry(1,selsalesrep:screen-value) .
  srary[2]:label = fill(" ",8 -
length(entry(2,selsalesrep:screen-value)))
    + entry(2,selsalesrep:screen-value) .
  srary[3]:label = fill(" ",8 -
length(entry(3,selsalesrep:screen-value)))
    + entry(3,selsalesrep:screen-value) .
  srary[4]:label = fill(" ",8 -
length(entry(4,selsalesrep:screen-value)))
    + entry(4,selsalesrep:screen-value) .
  srary[5]:label = fill(" ",8 -
length(entry(5,selsalesrep:screen-value)))
    + entry(5,selsalesrep:screen-value) .
```

```
default-window:width = 132.
```

```
output to terminal paged.
```

```
for each tsritem
```

```
  break by titemname
  with frame body:
```

```
  accumulate tamount (total by titemname).
```

```
  if first-of(titemname) then
    display titemname srary.
```

```
  assign
  totary[lookup(tsalesrep,
selsalesrep:screen-value)] = tamount +
  totary[lookup(tsalesrep,
    selsalesrep:screen-value)].
  display tamount @
  srary[lookup(
    tsalesrep,selsalesrep:screen-value)].
```

```
  if last-of(titemname) then do:
    display accum total by titemname
    tamount @ totitem.
    down.
  end. /* last-of(titemname) */
```


continued from front page

There is always a lot of talk about market share, and about all kinds of financial indicators. But fundamentally, what counts to keep a company around? Good fiscal management, good technical direction, good business model. But there's another side to the picture, as well. Take two of the top performers in this industry: Microsoft and Oracle. Ask a dozen people what they think of when they consider dealing with the company, or ask them what they think of the CEOs. It appears that phenomenal growth may come at the price of the "soul" of the company. Those who would rather that Progress be a multi-billion dollar company might want to give a lot of thought to what that might mean for their own longevity in the Progress "world". When we look at the number of people who were developing applications and doing consulting in the Progress world in the 84-85 time frame, a huge, huge percentage of them are still in business. If they were completely dissatisfied with the technical environment, with the company policies, with the attitudes, would they stick it out for that long? It's not likely. The same is true for the application partners. If Progress weren't a workable milieu, they would have bolted as well. Progress isn't a household word, unfortunately, and never has been, so there must be something else which keeps people "in the fold". Let's attribute it to great technical content, good business management, and a decent "soul" to the company and in the community.

It's been a lot of fun for those who've been around quite a while. We've all learned a huge amount, we've made untold numbers of friends; the web and the conferences have brought us together, we've watched each other and Progress and its partners change over time. It's so easy to wish that things were some other way in so many aspects of our lives, particularly when things are tough going. But overall, Progress has been a good partner to almost all of us, and we at White Star, for one, would like to wish both those at the Progress Company, as well as everyone we've made friends with, everyone we've met or done business with, a superb, prosperous, enjoyable pair of decades ahead.

App Technologies

Web Business Application Technologies

Custom Web Solutions

powered by

AppPro

Web Business Application
Development Tools

www.apptechnologies.com

800.861.4988

Attention all Writers Published & Unpublished Desirous of Fame & Fortune Progressions needs you!

We are accepting articles of from one to five pages in length. All published articles entitle their writers to:

- 1). A one year free subscription to Progressions
- 2). An invitation to a private Progressions party at the annual Progress Conference
- 3). A special Progressions gift
- 4). Fame and fortune (not guaranteed, but hopeful; may not be applicable in your state)

Please forward articles (Word format a plus) to: progressions@wss.com or call Connie or Michael at 970.963.3545.

4GL Payment Processing by Dan Yost

Electronic payment processing is a complicated business. Processors, networks, acquirers, issuers, associations, banks, protocols, fees, surcharges, discount rates, interchange, software providers, compatibility—the list of issues and potential complicating factors is almost endless, and it can make one's head spin.

Progress-based businesses are already leading the way into the technology-rich 21st century. Progress users are accustomed to powerful IT solutions that run with speed, efficiency, and accuracy—the standard is high. Such high-power solutions are possible thanks to Progress technology, but why do so few Progress applications make use of quality integrated payment processing functionality? There are a multitude of answers, many ending with phrases like "there isn't enough time" or "we're too busy." Most Progress-based companies have seen the results of this dilemma, and the results can be quite costly.

Money can be, and often is, lost via electronic payment processing (or the lack thereof) in three ways: by sustained problems with business practices, merchant account mistakes, and software slips, as we call them. Given that this is a technical journal we'll focus on the software slips and leave the more direct policy- and finance-related categories for a different venue.

Software Slips

Slow, insecure, inefficient, and generally poor software constructs are unfortunately the norm for many Progress-based companies when it comes to payment processing. Several categories comprise the landscape of errors that Progress customers can overcome.

1) Manual Madness

The first "software slip" is actually the absence of any software at all. And, not too surprisingly, this is the most common of all the software-related mistakes made by Progress-based businesses. The companies usually have a manual card swipe reader—the kind found on the countertops of some retail merchants, restaurants, and so forth. Even mail order/telephone order/Internet order businesses will occasionally employ such a device. The Progress business then utilizes either a full-time employee or, more commonly, diverts another employee

on occasion to manually run emailed, faxed, other otherwise acquired credit card transactions through the terminal. This employee manually enters the transaction information into some kind of Progress session, be it GUI, CHUI, etc. Of course, it is often true that by transaction run time the customer is already long gone and must be contacted if there is a problem with the transaction. Such business practices usually underestimate the cost of wasted employee time and overhead, significantly increased error rates, data propagation and availability delay, transaction misqualifications/underqualifications (watch that merchant account statement!) and many other similar factors. The "sneakernet" (a.k.a. "swivelnet") is actually quite expensive.

2) Kludge Tape

Next in the software debacle is the prevalent use of a collection of kludge "solutions" to handle payment processing. The dictionary defines the slang term "kludge" as:

1. A system, especially a computer system, that is constituted of poorly matched elements or of elements originally intended for other applications.

2. A clumsy or inelegant solution to a problem.

(Source: The American Heritage® Dictionary of the English Language, Fourth Edition).

The dominance of kludges in the Progress world is remarkable. Most often, Progress-based businesses "duct tape" (Kludge Tape) a collection of non-Progress third party elements together into one "solution," which is certainly "clumsy or inelegant," and then hope that the mess will work in the long run once they get it "up and running." Usually, Kludge Tape applications exist in a company because of the perceived cost of a more sophisticated solution (either hard cost or implementation cost). And, of course, the true cost of the Kludge Tape solution is typically underestimated. Kludge Tape solutions also typically involve extra hardware, albeit relatively cheap hardware.

3) Wheel Waste

A follow-up to Kludge Tape is referred to as Wheel Waste. Simply put, Progress shops will spend time (read: money) developing enough understanding of the payment processing industry to make the Kludge Tape work and to have a prayer of reasonably judging between non-Progress payment processing technology vendors. The wheel is subsequently reinvented, yet again.

4) Contortion Distortion

Some Progress-based businesses will actually mold their business practices and processes around pre-existing hard-coded payment processing solutions. Money that could be saved by streamlining is lost when, for example, realtime payment processing capabilities are not added at the points within an application where they make the most sense, such as at order entry time. The legacy solutions usually do not port easily, so a sales staff member must go through contortions just to bill a credit card instead of having immediate access as needed, not only to transactions in general, but the right transaction types for the point within the sales/production cycle that the staffer is in. The hard-coded nature of the legacy solutions proves to be quite costly in moving forward, if nothing else than from a lack of ability to adapt to changing process and application needs. Obviously, this is typically not an official company policy decision, but rather "just happens" in the trenches, so to speak. It is also common for Progress shops to believe (perhaps after being misled) that the current merchant account will not work with other software, and hence the contortions get worse as folks try to "get by" with what they have.

5) Web Woes

Everyone is now doing business on the web, or so it seems. Naturally, then, Progress-based businesses require payment acquisition services on their websites. In a large number of cases, however, the company may be Progress-based in the enterprise but use an off-the-shelf or one-size-fits-all non-Progress shopping cart system in order to accept electronic payments. At the end of the day, the problem is that one size generally doesn't fit all.

6) Surcharge City

Though very closely related to Kludge Tape, the surcharge issue merits additional consideration. That is because some Progress-based merchants actually lose money hand-over-fist each month in the form of merchant account-imposed surcharges due to non-compliant or incomplete software. Amazingly, a large number of these merchants don't realize that it is happening.

7) Time Troubles

Finally, Progress-based businesses are simply busy. Overpacked schedules abound and, unfortunately, many companies cut corners (or do nothing—see above) because of the perceived difficulty of an elegant Progress-based solution.

Progress-based businesses lose extraordinary amounts of money due to faulty assumptions about electronic payment processing and resulting bad processing practices. The financial loss is triple-pronged; company policy problems, merchant account mistakes, and software slips combine for significant revenue losses. We've focused here on some sample software-related issues only, but these mistakes alone can cause large annual expenses for Progress-based merchants—expenses that simply are not necessary. What's more troubling is that these expensive practices are extremely common in the Progress world. The electronic payment processing industry continues to grow rapidly, and Progress technology is right there to benefit its users. With Progress properly applied, the results are beyond objection—more money for merchants, more speed, service and satisfaction for customers, and fewer headaches for IT professionals. Nobody likes a headache.

Dan Yost

Dan Yost (dan@tri8.com) is President of Tri-8, Inc. of Stillwater, Oklahoma, a technology firm founded in 1984. Tri-8 is the Progress community's advocate in the Electronic Transactions Association, a collective organization of payment processing industry players including Visa USA and MasterCard. Dan lives in Stillwater with his beautiful wife and daughter, and their dog Kitty. Yes, the dog's name is Kitty. Kind of like a boy named Sue.



Melding V9 Character Progress "Wait-For" with Legacy "Choose" by Theodore J. Duke

One of my clients has an application that is still going strong after twelve years. For reasons too painful to explain here, the application was still in V6 character mode on an IBM RS 6000 until early 2003 when it was converted to V9 character and moved to a new Linux server.

How it all started

All of my other clients had long since converted to V7, V8, then V9 during the intervening years, so it took a bit of head scratching now and then for me to remember how to not use V9 code capabilities, in a character application. The conversion from V6 character to V9 character was relatively painless and the application is now humming along on a Linux server in V9, albeit still in character mode.

Going back to the origins, I was able to give the users some semblance of a modern 1990s application by providing what the client liked to call "ring menus." For example, in a maintenance window, the majority of the screen comprised a display of fields from the record in a separate frame at the top of the screen. At the bottom of the screen, there would be a small, centered frame, three rows high with un-labeled options; for example, something like:

```
Exit  Edit  Delete  AddNew  History
```

Fig. 1—A "ring menu" for options

The options are in an array variable and are selected by a "choose field ..." statement. The user would highlight the desired option, then press Return. (Auto-return was rejected by the client.) This was probably implemented as soon as the "choose" statement became available in Progress although, frankly, I don't remember which pre-V6 version of Progress was originally used. After the V9 conversion, there were so many new options that I really wanted to bring a somewhat reluctant client to consider using them in newer programs as well as in some older ones when modifications are needed.

"If it ain't broke, don't fix it," I was notified. Primarily, this meant two things:

- Don't rewrite any existing code that's working.
- Continue to use the ring-menu style for any new maintenance screens, to avoid confusing the users.

Certainly, these were not unreasonable requests.

Businesses not remaining static, a new requirement arose, one that was more complex than other parts of the application. For this particular table, the client wanted to be able to review some related, new child records for selection, then proceed on to manipulate those records in their own maintenance screen with multiple options.

I should add that all of the scrollable browsers in the legacy application are still based on a lengthy include file (ugh!) that I wrote many years ago in Version 3 Progress. These browses allow users to view records and to choose one if desired. But, they do not lend themselves to being integrated into a screen with other displays and they are quite cumbersome to implement compared to the V9 query and browse statements.

At the client's request, I wrote and tested a second maintenance screen with its ring menu for prototype access to multiple options. I also created a prototype screen in which the child records of a parent are displayed in a browser at the left and a few fields are displayed at the right. Not rocket science, just good V9 capabilities. As the user scrolls through the browser, the value-changed trigger fires and displays some fields outside the browser, Something like:

LastName	FirstName	Details
Albertson	William	Name: Mr Herman Barkley
Aaron	Henry	Addr1: 46 Monroe Dr
		Addr2: Apt B
Conklin	Osgood	City/St: Reston VA 20190
RETURN = Select highlighted record		
F4 = Cancel/Exit		

Fig. 2—A character display frame with embedded V9 browse for record selection

Of course, Fig. 2 is oversimplified, but it shows the general idea. From here on in this article, screens will be referred to as:

- **Parent Maintenance.** Legacy maintenance/display of parent records, blocking for user input with a choose statement (Fig. 1.)
- **Child Browse.** Viewer/selector of child

records, blocking for user input with a wait-for statement (Fig. 2).

· **Child Maintenance.** Legacy-style maintenance/display of child records, blocking for user input with a choose statement (not illustrated, but similar to Fig. 1).

The general idea was for program flow as follows:

1. User begins with Parent Maintenance, which allows selection of a parent record and displays the options. A new option, "ChildMaint" would be added to the ring menu, Fig. 1. It was intended to run the new, Child Browse procedure of Fig. 2.

2. While running Child Maintenance, the user can scroll through the embedded browse, Fig. 2. F4 will exit to Fig. 1. The Return key, or F1, will invoke the Child Maintenance screen

3. The Child Maintenance ring-menu options will be similar to those of the Parent Maintenance screen in Fig. 1, except that they will address the child records, not the parent record.

The three procedure flow with respect to blocking was choose to wait-for to choose, which, unfortunately, does not work.

Trouble rears its ugly head

Each of the procedures had been tested individually by commenting out the parameters and replacing them with specific finds of the required parent record. For example, in the Child Maintenance procedure, I found one parent record and one child of that parent, then tested all the options for that record. For final testing, I chained all the procedures together using parameter buffers to pass the appropriate records back and forth and commenced what I thought would be the final testing. The sequence was:

1. Run Parent Maintenance. Find a parent record. Choose the Child Maintenance option from the ring menu and run the Child Browse procedure, passing it the buffer parameter for the currently selected parent record. Success.
2. Select a child record in the Child Browser (Fig. 2.) and press Return to bring up the child maintenance

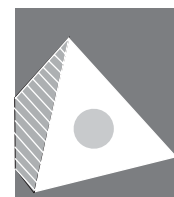
Users want web-enabled apps. Customers and vendors want XML.

If you don't have a web-enabled application, your users will think of you as aging "green-screen" or clunky client-server. If you still push files back and forth using flat file and EDI, you are missing out on the flexibility and cost-savings of XML. No matter which version or type of Progress you have, we can show you how to extend your application to

- ◆ Deliver data to anyone with a web browser no matter where they are
- ◆ Breath new life into your app with new, exciting web screens
- ◆ Accept and send data to other systems via XML

You don't have to be stuck in a ChUI or GUI only world. Give us a call and we'll show you how to deliver functionality that looks great, is easy to use, and is much easier to support.

AllegroConsultants.com
Knowledgeable - Reliable - Affordable



screen. Initial success. Then *unexpected failure*.

3. The Child Maintenance procedure, which had previously been tested successfully in stand-alone mode, displayed the child record, but none of the ring-menu options (choose field statement) functioned. Even the exit option failed to react. It required a stop (Control-C) to get out of the frame.

The solution turned out to be a forehead slapper. I should have anticipated that, but I had been so frustrated at the unexpected failure that at first, I failed to notice the message that followed a Control-C exit: "Window with wait-for is no longer visible."

What I had inadvertently done was mix old-style user blocking (choose, update, etc., in a repeat loop) with new-style user blocking (wait-for). The two are highly incompatible, especially so in character mode. What I was dealing with was a sequence of:

1. Parent Maintenance: Block for input with Repeat/Choose (no wait-for); then run Child Browse (which has a wait-for).

2. Child Browse: Block for input with a wait-for statement; attempt to run Child Maintenance on demand.

3. Child Maintenance: Block for input with Repeat/Choose. (Doesn't work. The reason it doesn't work is that there is still an active, conflicting wait-for hanging around from the un-terminated Child Browse procedure.)

A work-around was needed

In most of my GUI Progress work, I had carefully avoided using update and choose except for an occasional dialog-box. Even when working with add-on functionality for users who had used frame updates in GUI applications, the judicious use of dialog-boxes generally overcame the problem. But a quick attempt to put the third procedure's choose statement inside a dialog-box showed that to be not workable as a solution to the problem. Clearly, these different blocking methods had to be separated.

Bingo!

The solution came to me at about 3:00 am. I was of course, asleep at the time, but somehow the idea resurrected itself during my first morning coffee. The concept was to place a non-wait-for procedure between

the three procedures as a sort of traffic cop. The ability of Parent Maintenance (choose) to successfully call Child Browse (wait-for) implied that the problem was not insurmountable. I devised an intermediate procedure that I will refer to as "Intermediate Procedure," and the program flow was changed to the following:

1. Parent Maintenance: If there are no child records, the option to run Child Maintenance is effectively disabled. Otherwise, Parent Maintenance calls the new Intermediate Procedure which has no user interface. The parent passes the currently displayed parent record to the Intermediate Procedure in a parameter buffer.

2. Intermediate Procedure: This procedure runs the Child Browse, passing it the parent record's buffer in a buffer parameter. The Intermediate Procedure has two parameter buffers: one for parent and one for child. The child buffer has no available records unless one is returned by the Child Browse procedure. How that is managed is explained in item 3, below.

3. Child Browse: This procedure has two defined parameter buffers, one for the parent record which is populated by the Parent Maintenance procedure and passed through to the child Browse by the Intermediate Procedure and another for the child record.

The child parameter buffer is populated, or not, according to how the user exits: F1, Return, or F4. Because the browse and query use the defined parameter buffer for the child table, that parameter buffer always contains a child record; namely, whatever record is highlighted in the browse.

A trigger on the F4 key (End-Error) releases that record. In any case, control flow returns to the repeat block in the Intermediate Procedure and, importantly, the wait-for of the Child Browse procedure is terminated.

4. When flow returns to the Intermediate Procedure, there either is a record in the child table's parameter buffer, or there isn't—no, if user exited with F4; otherwise, yes. The available() function detects which, and if no child record is available, the Intermediated Procedure is exited, returning control to the Parent Maintenance procedure. Otherwise, the Child Browse procedure is executed again and the user can elect to run Child Maintenance, or not.

5. When the user exits the Child Maintenance screen, the Intermediate Procedure's repeat block iterates and again runs the Child Browse procedure where the user can continue to work with child records or, eventually exit by using F4.

Pseudo code for the intermediate procedure is given, below:

```
def param buffer pbParent for Parent

/*- There is no display interface in this
procedure. It receives a parent record in
the param buffer, -*/

RPT-1: repeat:

    run ChildBrowse.p(
        buffer pbParent,
        buffer pbChild).

/*- If no child record is available, exit
the loop and return to whatever program
called this one.-*/
```

```
if not available(pbChild)
then leave RPT-1.

/*- (Else) run the ChildMaint procedure
which has ring-menu options. -*/

run ChildMaint.p(buffer pgChild) .

/* RPT-1 */ end.
```

Ted Duke

Ted lives in Reston, Virginia and has been using Progress since early 1985, shortly after founding his full-time consulting business, TJD Enterprises, Inc. His one-man company (he wears all the hats from President to Janitor) specializes in Progress database design and coding, but he has had extensive experience in editing and publishing. He is currently planning the release of a new Progress GUI Visual Tutorial Series of publications "For Reasonably Intelligent People (tm)".

V10 is more powerful, faster and more fun. Don't miss out.

Even the best programs could use some extra speed, take advantage of newer 4GL statements and widgets, better maintainability with SmartObjects, functions and modularized code. Our clients rely on us to:

- ◆ Mentor programmers in the latest 4GL statements and tools
- ◆ Split user interface code from business logic code
- ◆ Move to WebSpeed, WebClient and AppServer
- ◆ Train their DBA's how to install and troubleshoot the new Progress services

Don't let your maintenance dollars go to waste! Call us, learn how to better use what you've got, have more fun programming in your favorite language, and impress your boss and users with how far you can take "that old Progress app".

AllegroConsultants.com
Knowledgeable - Reliable - Affordable



Shell scripting for Progress by Jan Postema

Paul Koufalis announced our joint undertaking, a few months ago, to write a booklet about Shell scripting for Progress. In this volume of Progressions Paul says a word or two about a particular kind of script. In this article I would therefore like to explain a bit – beforehand – about the concept behind the scripts we have been working on. It may also be helpful in making Paul's examples more comprehensible to those less familiar with shell scripting.

Items & Types

Basically, what it comes down to is our understanding of databases, brokers and the like as configuration items. For each item we can define certain properties, such as the name, the kind of Progress process (i.e. a database), how it is started, the options with which it is started, etc. Furthermore, we can draw up a list of the actions to be executed upon that item; most common amongst them are starting and stopping. But for a database we could also think of making a backup. What we often see is that additional scripts are made for each new item and each new action to be undertaken. Adam Backman has set up a set of scripts that use a registry which comes very near our own concept. We could have used a similar registry for every Progress process we have identified but in doing so we would also be obliged to maintain different registry files. Most important, however, in not following this example, is the fact that the field order is fixed.

We have chosen, therefore, the more flexible solution of the **item.properties** file. We departed from the principle that all configuration item related information be stored in a configuration file. We had to choose between individual files per item and one configuration file in which the variables for all items should be stored. In the end we resorted to one file. We choose the form as in the example below.

```
[BEGIN.DB.athens]
ITEM_NAME=athens
ITEM_HOST=castor
ITEM_USER=root
ITEM_TYPE=DB
ITEM_VERSION="9.1D"
ITEM_PF=athens
ITEM_WDOG="yes"
ITEM_APW=2
```

```
ITEM_BIW=4
ITEM_PORT=sv_athens
ITEM_DBNAME=athens
ITEM_DBPATH=$TOP/db/athens
ITEM_DBOPTIONS=" -n 10 -Mn 1 -H castor -S
sv_athens -N TCP"
ITEM_DLC=/usr/progress91d
ITEM_DUMPDIR=$TOP/usr/general/dump
ITEM_DUMPFILE=athens.df
ITEM_DUMPUID=
ITEM_DUMPPWD=
ITEM_BACKUP_DEVICE=/dev/rmt0
ITEM_BACKUP_TYPE="on"
ITEM_BACKUP_INTERVAL=7
ITEM_BIFILE=$ITEM_DBPATH/athens.bi
ITEM_LGFILE=$ITEM_DBPATH/athens.lg
[END.DB.athens]
```

You can choose any kind of variable to store in the properties file for they are read by calling the appropriate function that reads them from the properties file and stores them in a temporary file. That file is then called by the script using the dot-method: `.<tempfile>`. Variables have to be known to at least parts of the scripts.

Each configuration item is entered into the file and marked by a BEGIN and END using the format:
`[{BEGIN}{END}.<item type>.<item name>].`

Two things are crucial to the framework: the item name and the item type. Without them it does not work, for the scripts are named using the type of the items, e.g.: DB.start, DB.Status, WS.start, WS.kill. So far we have identified the following types of items.

DB	Database
DS	Data Server
AS	AppServer
AD	AdminServer
NS	NameServer
MQ	Sonic adapter
WS	Webspeed Transaction broker
RP	Reports
BC	Batch process
AP	Application
ENV	Environment

From this it can be seen that entire applications can be understood as items as well. Suppose you would have to start an application which needs three databases, an appserver, a Webspeed broker and an Adminserver. You could then store that information in the item.properties file, though slightly differently. The most comprehensive way would be to use arrays, e.g. as in:

```
Set -A ITEM_APPL athens sparta thebes AS1 WS1
Adminserver_Greece
Set -A TYPE_APPL DB DB DB AS WS AD
```

The scripts for this item type would, in our example, be called: AP.start (or stop etc.). The script then loops through the array and calls the other scripts pertaining to the particular types; the types are stored in the TYPE_APPL variable in the same order as the item names in ITEM_APPL. So, **maint.sh -i ERP -a start -t AP** would call a script AP.start which could contain the following code:

```
iIdx=0
while [[ ${iIdx} -lt ${#ITEM_APPL[*]} ]]
do
    ${TYPE_APPL[${iIdx}]} /
    ${TYPE_APPL[${iIdx}]} . ${ITEM_APPL[${iIdx}]} -
    start
    iIdx=$((iIdx+1))
done
```

How to extract configuration information

To extract the necessary information from the configuration file we use a short AWK-function as below. The information is then written to a temporary file that is read from the calling script.

```
function frmwrk_ItemInfo {
awk '/^\[BEGIN\.'"$2"'\'.'"$1"'\'\/\//
^\[END\.'"$2"'\'.'"$1"'\'\/ {
    if (match ($0, "^\[BE-
GIN\.'"$2"'\'.'"$1"'\'") > 0) {
        print $0
    }
}' item_properties >$2$1Tmp
}
```

The idea is that functions like this one are assembled in a library script, i.e. **frmwrk.shlib**. Of course it is totally up to you to choose another name. This script is then called by every script, again with the dot-method.

Note that this function works on AIX 4.3, but did not quite work on Solaris 9.

On modularity

1.1.1 One shell or more

For much the same reason as with the configuration files a choice had to be made between writing a number of functions or to create different scripts for each Progress related command we want to execute. Using separate scripts makes it easier to call them from an application such as Fathom. That also means that scripts run in a separate environment. They could be called with

the "dot"-method: **. DB.start \${ITEM_NAME} \${ITEM_TYPE}**. However, that causes the script to run in the same shell as the script that calls it. Therefore, since they are meant to function as a modular system, scripts are called without the dot: **DB.start \${ITEM_NAME} \${ITEM_TYPE}**. That way every script runs in its own shell.

In exiting, the script then returns control to the calling script. It also makes it easier to use scripts from within scheduling tools such as Tivoli. The result is a modular system, enabling all scripts to call particular actions to be performed. The **frmwrk.shlib** script is used by all other scripts. This way of working is not unique of course; Adam Backman used the same method when he wrote his utilities.

1.1.2 Environments

In order to make it possible for scripts to be used standalone, it is necessary to read a standard set of variables. We stored those in the **frmwrk.env** file, which is read.

```
GEN_BIN=/usr/local/scripts/alg/bin
GEN_CONF=/usr/local/scripts/alg/conf
GEN_LOG=/usr/local/scripts/alg/log
GEN_LIB=/usr/local/scripts/alg/lib
```

The scripts can then be called using, for example the GEN_BIN variable: GEN_BIN/ITEM_TYPE/ITEM_TYPE.start

Again, one can make that file as such, but it is also possible to incorporate the information in the **item.properties** file. It suffices to add a new type, say ENV, and give it a name, e.g.:

```
[BEGIN.ENV.frmwrk]
GEN_BIN=/usr/local/scripts/alg/bin
GEN_CONF=/usr/local/scripts/alg/conf
GEN_LOG=/usr/local/scripts/alg/log
GEN_LIB=/usr/local/scripts/alg/lib
[END.ENV.frmwrk]
```

That way it is possible to make an environment file for every host one has to deal with. It is even possible to omit, for example, the ITEM_HOST variable in each configuration item and add it to the environment entry (if you have only one server). In those cases where you would want to deviate from the standard, you simply read the standard environment variables and afterwards read the configuration information with a renewed entry of the particular variable you wish to give a deviating content.

Maint.sh: starting it all

The general idea behind the use of scripts is that the sequence is started with a script called **maint.sh**. It checks the parameters being given and performs a number of other functions, such as initiating logging and closing the logging at the end.

Since we start our scripts using one startup script – **maint.sh** – we also provide a number of startup parameters, being:

- Ø The configuration item on which we want to perform a certain action

- Ø The action, which has to be known to the script of course

- Ø The kind of Progress process involved. Practice has taught us that frequently the same name is used for databases and AppServers, for example, which might lead to the following entries in **item.properties**:

```
§ [BEGIN.DB.athens]
§ [BEGIN.AS.athens]
```

A number of methods for parsing parameters are available. In the scripting solutions presented in this document, the order in which the parameters are given is not important, reducing the risk of error when calling them. In the example below the command could look like this: **maint.sh athens start**.

```
grep "BEGIN" item.properties >tmpFile
COMMAND="set -A ITEM_OPT_ARRAY" # ITEM_OPT
contains the name of the configuration item
while read cLine
do
    cTmpVar=${cLine##*.}
    COMMAND=${COMMAND}" ${cTmpVar%%}" "
done <tmpFile
rm tmpFile 2>&1

${COMMAND}
set -A ITEM_ACT_ARRAY start licences checklog
stop status truncate_bi kill dump load versie
set -A ITEM_SRT_ARRAY DB AS NS WS MQ

cActFnd=FALSE
cOptFnd=FALSE
cSrtFnd=FALSE

if [[ $# -lt 2 ]]
then
    iErrorStatus = 1
else
    # If there are exactly two parameters, the
    script assumes that all items have to be
    treated
```

```
if [[ $# -eq 2 ]]
then
    cOptFnd=TRUE
    ITEM_OPT="ALL"
fi
fi

for cPar in $@
do
    if [[ ${cActFnd} = FALSE ]]
    then
        iActIdx=0
        while [[ ${iActIdx} -lt
        ${#ITEM_ACT_ARRAY[*]} ]]
        do
            if [[ ${ITEM_ACT_ARRAY[$iActIdx]} =
            $cPar ]]
            then
                ITEM_ACT=${cPar}
                iActIdx=$(( ${#ITEM_ACT_ARRAY[*]} +1))
                cActFnd=TRUE
            else
                iActIdx=$(( ${iActIdx} +1))
            fi
        done
    fi

    if [[ ${cSrtFnd} = FALSE ]]
    then
        iActIdx=0
        while [[ ${iActIdx} -lt
        ${#ITEM_SRT_ARRAY[*]} ]]
        do
            if [[ ${ITEM_SRT_ARRAY[$iActIdx]} =
            $cPar ]]
            then
                ITEM_SRT=${cPar}
                iActIdx=$(( ${#ITEM_SRT_ARRAY
                [*]} +1))
                cSrtFnd=TRUE
            else
                iActIdx=$(( ${iActIdx} +1))
            fi
        done
    fi

    if [[ $cOptFnd = FALSE ]]
    then
        iActIdx=0
        while [[ ${iActIdx} -lt
        ${#ITEM_OPT_ARRAY[*]} ]]
        do
            if [[ ${ITEM_OPT_ARRAY[$iActIdx]} =
            $cPar ]]
            then
                ITEM_OPT=${cPar}
                iActIdx=$(( ${#ITEM_OPT_ARRAY
                [*]} +1))
                cOptFnd=TRUE
            else
                iActIdx=$(( ${iActIdx} +1))
            fi
        done
    fi
fi
```

```

        fi
    done
fi
done

```

The way things are done above, it is possible to omit the name of the item. In that case all items are addressed. However, if we use `getopts` the script can be made more elegant, for the `for`-loop could look like this:

```

while getopts i:a:t: cOpt 2>/dev/null
do
    case ${cOpt} in
        i) Log -i "Item name is ${OPTARG}."
            ITEM_NAME=${OPTARG};;
        a) Log -i "Action is ${OPTARG}."
            ITEM_ACT=${OPTARG};;
        t) Log -i "Item type is ${OPTARG}."
            ITEM_TYPE=${OPTARG};;
        \?) Log -e "Wrong parameters; execution
            is aborted."
            exit 1
    esac
done

```

In this case the command would look like: **maint.sh -i athens -a start -t DB**, where `-i` indicates the name of the item, `-a` the action to be executed and `-t` the type of the item. But what if you want to start everything? Treat it as an application! You can even mix individual databases and applications in such a definition.

It is possible, however, to start a script without using **maint.sh**. For a number of actions to be performed, functions have been defined and written. They are saved in **frmwrk.shlib**. Using scripts stand alone might be useful when for example. Fathom is used. In this case a limited number of parameters have to be given: the configuration item name and the action. To that end every script starts with a general section, which is common to all.

Conclusion

The work in this article is the basis for the framework methodology used by Paul Koufal's `killprosession` script and the forthcoming 'Shell Scripting for Progress' booklet. In our view this way of setting up shell scripting is more efficient than "just writing a new script". This does not mean that you will never again have to write a script, of course; but you will not have to do it because someone adds a new database or appserver. It requires some thinking before you start writing the script and, of course, every environment has its own peculiarities you have to

cope with but it pays afterwards; more so since you can expand the use of this concept to other environments as well. We rarely work in a pure Progress environment and it may be interesting to know that we used it on Netscape as well.

Jan Postema

Jan has been working with Progress since 1989 as a programmer and a DBA, mainly in a production plant. After a brief interlude he again returned to Progress during 2002 and 2003 in a subsidiary of the Dutch ministry of Justice. Jan can be reached at

j.postema@pinkroccade.com

"WebSpeed Complete"

The New Book By Geoff Crawford

Frustrated by the lack of
WebSpeed documentation?

Wish there was a 238 page how-to written by a
top WebSpeed expert that comes with a CD?

- ♦ Administration
- ♦ Web servers
- ♦ Basic Programming
- ♦ Credit Cards
- ♦ Pop-Ups
- ♦ PDF's
- ♦ XML
- ♦ Security

Order On-line: \$49.95

http://www.innov8cs.com/products_books.asp

Innov8 Computer Solutions, LLC

711 Route 10, Suite 204

Randolph NJ 07869

(973) 361-4224

Killing a Progress Session by Paul Koufalis

Introduction

When it comes to the subject of killing a local, shared memory Progress session, there is more myth and hearsay out there than actual fact. Kill -9 always works. No no!! Never use kill -9!! Kill -8 is the way to go! Oh my, you use kill -8? You should disconnect the user from the database first. No, never disconnect the user from the database!

Finally, after all the discussions and arguments, Tom Bascom decided to put the facts to paper in an article entitled "Traps and Kills." In fact, before you continue reading, open up your favourite internet browser and head on over to http://www.greenfieldtech.com/articles/traps_and_kills.shtml and read that article.

After reading the article, I decided to take the subject of killing shared memory Progress sessions a step further: I have attempted to write *the* definitive killprosession script. The purpose of this script is to provide a tool for both Progress experts and newbies that easily and safely kills a Progress session. Have I succeeded? Probably not. But I am definitely a whole lot closer than I was a few months ago.

For those of you who have not yet read Tom's article, let me summarize a few of the points he makes about killing a Progress session:

- Tell the user and over-zealous UNIX systems administrator to step away from their keyboards.
- Kill -2
- Kill -1
- Monitor and wait

The Real-Life Problem

From painful experience I have come to the realization that a systems administrator who is not a Progress DBA has considerable difficulty with the "monitor and wait" part of the above list. He wants to kill kill kill!!!!!! Kill -2 did not work and kill -9 is strictly forbidden so kill -8 it is. But guess what? Kill -8 can bring down a database just as readily as a kill -9. Besides, the non-DBA administrator knows nothing of VST's and shared memory latches. Even if we were to ask him to monitor, he would not know where to begin.

The solution? Script everything: all the killing, all the monitoring and waiting, everything is scripted. The administrator gets a live play by play of what's going on and at the end the process is either dead or at the very least the administrator knows to just let it finish whatever it is doing.

The Procedure

In a nutshell:

1. Kill -2, kill -1, wait
2. Monitor the database VST's
3. Attempt to disconnect the user from the database
4. Wait and monitor some more
5. Check for open files
6. Check for attached shared memory segments

Sections of the script are shown below and the complete script is available at www.progresswiz.com. Note that this script has been tested with Progress 9.1D08, AIX 5.2 and the Korn shell and that there are no guarantees that it will behave adequately in your environment. In other words, use at your own risk.

First, a kill -2 and a kill -1 are sent to the process. If the process is well-behaved and responding, this should be sufficient to terminate it. The script then waits 10 seconds to give the process a chance to clean up and exit and then check to see if it is still there.

```
kill -2 $PID
# "Note" is a function that echoes with a
date and time stamp
Note "Kill -2 executed. Wait 5 sec"
sleep 5

# Send a HANGUP to the Progress session
kill -1 $PID
Note "Kill -1 executed. Wait 10 sec"
sleep 10
```

If the process still exists, the script begins the monitoring phase. The function "ConnectedDB" scans all the databases defined in the item.properties file (see Jan Postema's article *Shell Scripting for Progress*) and lists to which databases and with what user number the user is connected.

```
# Ex DBList = "/db01/sports/sports 47\n/db02/
whse/whse 35"
#
# ConnectedDB does not currently work with
batch processes
```

```

DBList='ConnectedDB $User $Tty'
echo "$DBList" | while read ThisUser ThisDb
do
  # The WaitProSession function returns:
  # 0: User no longer connected
  # 1 Process not using any CPU or IO
  # 2: User still active. May still be
  # rolling back tx.
  WaitProSession $ThisDb $ThisUser
  if [[ $? = 2 ]]
  then
    # If the user is active DON'T TOUCH!
    Note "User is active - USE $0 again - DO
NOT MANUALLY KILL!"
    exit 0
  fi
done

```

The heart of this section of code is the "WaitProSession" function and the killprosession.p program that it calls:

```

WaitProSession () {
  Note "Waiting for end of session for DB
$1"

  _progres -b $1 -param "MONITOR $2" -p
$RSYS/killprosession.p | \
  while read Line
  do
    # Parse output from killprosession.p
and act accordingly
    # See actual code at
www.progresswiz.com

    ...
  done
}

```

The killprosession program receives the Progress user number as a parameter and monitors its `_CONNECT` record. It is too long to be quoted here but is available on my web site. What is important is the output of killprosession.p. It gives the systems administrator and/or Progress DBA all the information he needs to understand exactly what the process is doing in its attempt to terminate:

```

2004/04/19-15:22:22 Waiting for end of ses-
sion for DB /home/koup/aitest/db1/pktest

```

```

INFO      : User disconnect already initiated
INFO      : Process in transaction 140563
INFO      : Process NOT attempting to roll back
a transaction
INFO      : Old Latch List:
INFO      : New Latch List:  2 7 9 12 13 20
INFO      : Database I/O last 10 seconds:

```

```

395571
INFO      : CPU Time last 10 seconds: 23 secs

***** SLEEP 10 seconds *****

INFO      : User disconnect already initiated
INFO      : Process in transaction 140563
CRITICAL: Process rolling back transaction -
do not kill!
INFO      : Old Latch List:  2 3 5 7 9 10 12 13
15 17 20 21 25 26 27 28
INFO      : New Latch List:  2 3 5 7 9 10 12 13
15 17 20 21 25 26 27 28
INFO      : Database I/O in last 10 seconds:
95432
INFO      : CPU Time in last 10 seconds: 8 secs

***** SLEEP 10 seconds *****

```

The information above is crucial. It tells the administrator whether or not the process successfully received the kill signal and has begun its exit process. It also tells the administrator, with a "CRITICAL" prefix, if the process is rolling back a transaction. It monitors whether or not the process is using up any CPU cycles and/or is doing any disk I/O and, finally, lists latches that the process is holding in an attempt to point out the existence of a long-held latch.

With the above information streaming across the screen, the administrator cannot arbitrarily send kill after kill to the process. The killprosession script tells him exactly what was done, how it has affected the process and what the process is doing in response. The above loop will not exit until the process has been disconnected from the database or there is no more activity. In the latter case, the administrator is warned that the process was not successfully disconnected.

If the disconnect flag was not raised, the next step is to directly disconnect the user from the database using the `proshut -C` disconnect command. The script then repeats the "WaitProSession" function, again waiting for the process to disconnect itself from the database.

If after all these attempts the process is still alive, the script checks to see what files the process still has open using the `lsf` (LiSt Open Files) command. Since it is impossible to programmatically parse the output of `lsf`, the script asks the user to manually verify the listing. The output of the `lsf` command shows that no database files are open:

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE/OFF	NODE	NAME
_progres	347206	root	cwd	VDIR	61,5	1024	2130019	/home (/dev/lvhome)
_progres	347206	root	0u	VCHR	30,92	0t41486	41186	/dev/pts/92
_progres	347206	root	1u	VCHR	30,92	0t41486	41186	/dev/pts/92
_progres	347206	root	2u	VCHR	30,92	0t41486	41186	/dev/pts/92
_progres	347206	root	3r	VREG	61,2	1026108	16394	/opt/rona (/dev/lvopt rona)
_progres	347206	root	4r	VREG	61,2	865553	16439	/opt/rona (/dev/lvopt rona)
_progres	347206	root	5u	VREG	61,5	0	2130076	/home (/dev/lvhome)

Finally, the process checks for attached shared memory segments with the `svmon` command and searches for the matching database:

```
AttchSHM=$(svmon -P $PID | grep "shared
memory segment" | awk '{print $1}')
Note "Attached shared memory
segments:\n$AttchSHM"
```

```
Note "Listing all matching database shared
memory segments:"
```

```
ps -ef -o pid= -o args= | grep _mprosrv |
grep -v m1 | grep -v m3 | grep -v grep |
while read Pid Process Param1 Param2 Param3
Param4
do
  SharedMem=$(svmon -P $Pid | grep "shared
memory segment" | awk '{print $1}')
  if [[ $AttchSHM = *${SharedMem}* ]]
  then
    echo $SharedMem $Param2
  fi
done
```

Had there been attached databases, the output would have resembled:

```
2004/04/29-11:53:43 Attached shared memory
segments:
43a61
1c9aa
2004/04/29-11:53:43 Listing all matching
database shared memory segments:
1c9aa /dbdvp/bc/rona-bc
43a61 /dbdvp/tlkt/dpTlKt40
```

If the process does not have any open database files nor has any attached database shared memory segments then it is very likely that it is in no way attached to a running database. Best solution: leave it alone. Of course, waiting is probably not an option otherwise you never would have attempted to kill the process in the first place. The only other answer is `kill -8` or `kill -9`. This should terminate the process. No cleanup will be done but there is almost no chance that you will bring down a database doing it.

Conclusion

While I agree that the best response to a hung Progress session is to just leave it alone, I understand that this is rarely possible in the real world. Hopefully, the above *almost* definitive killprosession script will help systems administrators in Progress environments safely terminate hung sessions. And remember, test in your environment before using this or any script in production.

Paul Koufalis

Paul began his Progress career at Le Groupe Cogicom in 1994 and has been working on his own as a Progress DBA and implementation specialist since 1999. He is currently working on a book with Jan Postema called "Shell Scripting for Progress" which should be available soon.

Paul can be reached at pk@progresswiz.com



GOT HEADACHES?

TailorPro is the headache solution for you!

Do Any of these Describe You?

- ◆ You have a new Progress-based package you're implementing but want some customizations.
- ◆ The users of your existing application are asking for numerous modifications.
- ◆ You want to upgrade; you've made customizations; you don't want to spend a fortune "going back" to vanilla code but you don't want to lose what you've implemented.

Do You Need to:

- ◆ Add or remove some fields in an existing screen, change some labels or defaults, or even rewrite the screen entirely...
- ◆ Add your own tracking screens in the middle of the application flow...
- ◆ Put your own messages, validation or extended business rules into the application...
- ◆ Add fields or indexes to existing tables...
- ◆ Get better rollup information, and get it in real-time...

TailorPro is the solution to all these problems, and many more. It is a Progress-based application which allows you to insert your own application logic anywhere in your current programs. It enables you to add fields not tracked by your application, and perform data entry to them in the "middle" of existing screens. No source code or DB changes required!

TailorPro Saves You Money by:

- ◆ Cutting operating costs in your company by making your users more efficient.
- ◆ Dramatically reducing the implementation costs for custom application components.
- ◆ Eliminating the future costs of upgrading a customized application.

Look into the *TailorPro* features on our web demo: www.wss.com/tailorpro

White Star Software

voice: 970.963.3545

tailorpro@wss.com

fax: 970.963.3548



Table of Contents

Patience with small details makes perfect a large work, like the universe

-Rumi

Dynamics – Not such a scary word	Gareth Woodhouse	2
Go Find Yourself	Jop Kluis	4
An Alternative to Publish and Subscribe	John Kattestaart	9
Customizing ADM2 Classes	Michael Lonski	12
Cross Tab Report Revisited	Paul Guggenheim	13
4GL Payment Processing	Dan Yost	18
Melding V9 Character Progress "Wait-For" with Legacy "Choose"	Theodore J. Duke	20
Shell scripting for Progress	Jan Postema	24
Killing a Progress Session	Paul Koufalis	28

Progress Conference Double Issue

Las Vegas, Nevada

June 2004