

# PROGRESSIONS

The Building blocks of progress

Winter 2002 Number 48

---

## Progressions On-line

During our summer conference in Africa, and our fall conferences in Europe, we got a lot of feedback that indicated to us that people would like to be able to receive Progressions electronically. Among other things, it would cut down on the number of trees we use up, as well as reducing the mailing costs. This would substantially reduce subscription costs to our international readers, who pay a premium for "postage". But before we make the leap permanent, we'd like your reaction. Some people have said "Well, we really like to have the piece of paper to hold and to write on". We would beg the indulgence of those (as well as librarians and others who might need a printed copy) to print it out for yourselves. The PDF format should allow you to do so with high quality resolution. If you are a library or other facility, we recommend that you use a heavier stock than regular printer paper. If this is unsatisfactory, we'd like to know. If enough people wish to receive a printed version, we will continue in that mode. We are not a large corporation with a diversity of people to please, but there are minimum costs to creating a printed version; it would not be profitable for us to have a small but significant number of subscribers who want a printed version, and a larger number who want an electronic version. We're here to do what's best for you, and this issue is an experiment. We hope that by reducing our costs that we can pass on the savings to our subscribers, and possibly even increase our circulation.

Please take the time to let us know what you think. If we decide to continue electronically, we will come up with an adjustment to each subscription to compensate for the cost changes that will be the result of this new distribution method.

## Subscription Information

A few of you have written and asked about the number of issues you have received. We'd like to explain a little about how we do things, and what has gone on for us in the last quarter of 2001. Our newsletter has continued to thrive thanks to two sets of people: our contributors and our advertisers. Quite frankly, we did not receive enough contributions for articles to publish a full issue this fall, and we did not feel that any arm-twisting was in order. So how will we deal with this shortfall? We have adjusted everyone's expiration date accordingly. In addition, we don't cut people off at the end of their subscription period anyway.

---

**Thanks to everyone  
for your continued support  
of Progressions**

---

Usually, you get 2-3 extra issues. We send renewal reminders, but we know that you don't always see them or get around to doing anything about them. So what we're saying is that each of you will receive extra issues to make up for the lower number of issues we published in 2001. We are grateful for the opportunity to provide you with this newsletter!

And while we're at it, we would very much like to encourage each of you to become contributors yourselves. Much of the information we provide seems very mundane to the author, but is very interesting to the audience. Take time to think of something interesting you've done over the last few months and let us hear about your ideas. Our editorial staff (John) will work with you to turn it into an article, regardless of your writing skills or language fluency.

In the meantime, thanks to all of you for your continued support, and particularly your understanding as we go through changes to produce this newsletter in a new format.



## Universal Audit Trail Inquiry Program by Paul Guggenheim

In Version 8.1, Progress added features that allowed storing an entire record in a single raw datatype field. Data could be moved to and from a raw field to a record buffer with the raw-transfer statement. They also added the ability to compare buffers with the buffer-compare statement. This statement would compare two buffers and create a comma-delimited list of changed field names for users to examine.

This enabled developers to create a single audit trail table to be used for all tables in an application. No longer would it be necessary to create a separate audit table for a given table by adding userid, date and time to each of these audit tables.

In our Beyond Version 6 and Extreme Courses, we introduce an audit table to the sports and school databases. It consists of the following fields:

Table	Field-Name	Data-Type	Description
audit	audit-id	integer	Unique Index generated by sequence
	audit-user-id	character	
	audit-date	date	
	audit-time	integer	
	audit-type	character	Valid values: Create, Modify and Delete
	audit-table	character	Database table
	old-record	raw	Before Record Image, blank for create
	new-record	raw	After Record Image, blank for delete

Audit trail records are automatically created when the corresponding write trigger fires for each table that has auditing enabled.

The following is a sample write trigger:

```
TRIGGER PROCEDURE FOR WRITE OF salesrep old
salesrepold.

find scontrol no-lock where skey = 1
no-error.
if not available scontrol then do:
  create scontrol.
  assign skey = 1.
end.
```

```
if lookup("salesrep",audit-file-list) > 0
then do:
```

```
  create audit.
  assign audit.audit-type =
  if new salesrep
    then "create"
    else "modify"
  audit.audit-table = "salesrep"
  audit.audit-user-id = userid(ldbname(1))
  audit.audit-date = today
  audit.audit-time = time.

  raw-transfer salesrep to audit.new-record.
  if not new salesrep then
  raw-transfer salesrepold to
  audit.old-record.
end. /* lookup("salesrep",audit-file-list) */
```

A control table exists that allows the end-user to turn off auditing for a particular table by removing that table from the list. The raw-transfer statement is used to transfer the before and after images to the raw fields, old-record and new-record. The audit-id field is assigned a sequence value from the audit-seq sequence during the create trigger not shown.

The features in 8.1 solved the problem of multiple audit tables, reducing it down to a single audit table. However, in order to view the old and new records for a particular table, it was necessary for a developer to write a separate program for each table that was

being audited. This is due (as I am sure you are well aware of) to the fact that Progress needs to know the database table at compile time in Version 8.

In Version 9 (as you have probably learned), we now have dynamic queries, browsers, buffers and temp-tables. I have created a program, auditing.p, that allows the viewing of any data from the audit table, regardless of what table it came from, and I can do it in a single 261 line program.

In figure 1, at the top of the window, the user is allowed to browse through the audit records. As the user selects a row from the top browse, the audit table highlighted generates a new set of field names and values

that are displayed in the middle browse using the tfld temp-table. The user may select either the old or new values for a particular record by using the radio-set "old new".

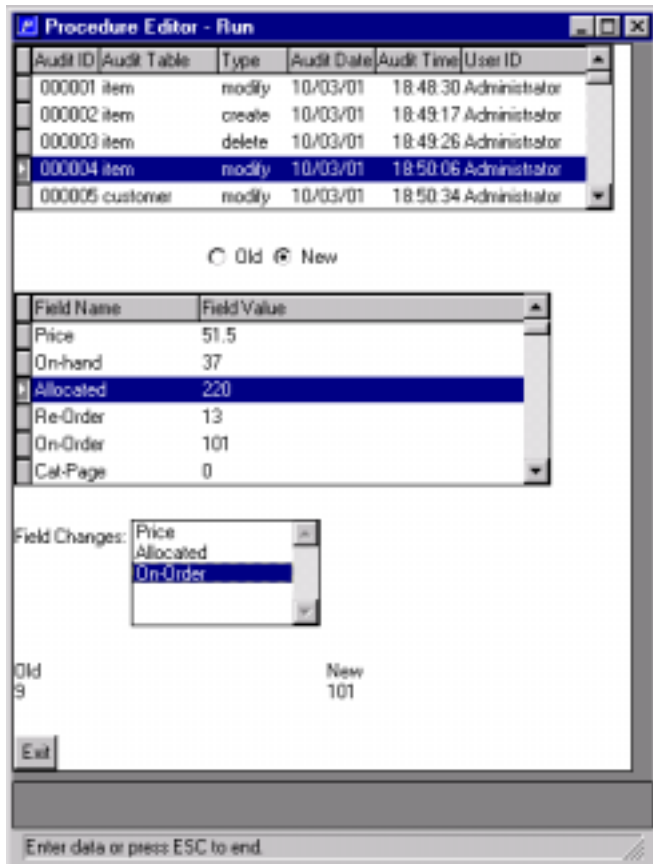


Figure 1

A selection-list of fields that were changed is viewed for modified (type = modify) audit records. The old and the new values are displayed at the bottom for the field selected. Here is how the program works. A static query and browse are used for the user to scroll through audit records. The challenge arises when we want to transfer the raw data from static fields into dynamic temp-tables.

The solution to this is to create a dynamic query and buffer that retrieves the same record that is selected in the static browse. This allows us to use the raw-transfer method to transfer the raw data field into a dynamic temp-table buffer. In the containing procedure, a dynamic buffer and query are created for the audit table. The first record is retrieved using the get-first method which is consistent with the first record in the static browse.

Then we obtain the numeric position of the old-record

and new-record fields in the buffer and store them in ornum (oldrecordnum) and nnum (newrecordnum) respectively.

Next we apply the value-changed event to the audit browse. It stores the audit rowid into the audrowid variable, determines whether to view the widgets for the modify audit type, and then calls three internal procedures getdynaudit, getdifffield and showtable. Getdynaudit keeps the dynamic buffer in sync with the static buffer by using the find-by-rowid method with the audrowid variable. Next, it runs the internal procedure DoRawTransfer for both the old and new records.

The DoRawTransfer procedure creates a temp-table like the one stored in the audit-table field, and then transfers data from the raw field to the temp-table buffer. The getdifffield procedure does two things. It populates the temp-table tfld that is viewed in the middle browse with both old and new field values for a given record, and it populates the difffield selection-list with each changed field. This procedure even handles array fields by checking each array element for differences.

Showtable is then called to reopen the query for temp-table tfld by only showing field values that are either old or new according to the value of the oldnew radio-set. It then runs showvalues which displays the old and new values for the selected changed field.

In summary, Progress Version 9 enables developers to use a single program to extract audit trail information on any table in addition to being able to pull audit information from a single table.

## Paul Guggenheim

Paul has worked with Progress since 1984. He provides public and private Progress training classes and has authored a sales lead tracking program called S.M.A.R.T.S. as well as a custom application development tool called The Sharp Menu System.

**Paul Guggenheim & Associates, Inc.**  
**1788 Second Street, Suite 201**  
**Highland Park, IL 60035**  
**847.926.9800 Fax: 847.926.9805**  
**paul@pgasmarts.com**

## Hacking The Internet With Progress . . . Leave Your Browser At Home By Ethan Tower

First of all, I have to offer a special thanks to Ken Wilner for his inspiring presentation about SOAP at last summer's Progress Exchange. It piqued my curiosity, got me some free code, and led to the adventures on the Internet which are chronicled in this article. It also reaffirmed my opinion that "Never has so much been made out of so little." So, enjoy.

### HTTP Requests

Web sites sitting out on the Internet act as TCP/IP socket servers listening on port 80. To access a web site, you simply make a client connection to the site. In Progress, it goes something like this:

```
sock:CONNECT( "-H www.yahoo.com -S 80" ).
```

That's it. I always thought that there must be some big complicated protocol known only to the gurus at W3C. No, it is just like making any other TCP/IP socket connection. No secret handshakes, no fancy protocols to figure out. Just make the connection and you are ready to go.

To actually get something useful from the web site, you need to send an HTTP request to the site and gather up the results. The web site won't give you anything back unless you ask for it. I figured that this would be a daunting task, filled with all sorts of secret stuff. Wrong again, grasshopper. Making an HTTP request only requires that you send a simple one line ASCII message to the server. It can be as simple as "GET / HTTP".

In your Progress program, you need to create a request string, convert it to a memptr, send the request out the socket connection, read the response back into a memptr, and then convert the response into human-readable form. Once again, in Progress, it goes something like this:

```
req = "GET / HTTP" + CHR(10) + CHR(10) .
PUT-STRING(reqm,1,LENGTH(req)) = req .
sock:WRITE(reqm,1,GET-SIZE(reqm)) .
sock:READ(rspm,1,
    sock:GET-BYTES-AVAILABLE(),1) .
```

```
rsp = GET-STRING(rspm,1,sock:BYTES-READ) .
```

That's it. You simply send out a request and read in the response. Nothing fancy. The HTTP request is made up of a function followed by the fully qualified path name of the resource you want and, finally, the protocol designator. Don't forget the two carriage returns. You won't get a response without them.

The first example, `hack1.p`, includes a standard subroutine that you can use to make requests to a web site. It also includes a sample program that will make an HTTP request and store the response in a file that you can look at with your favorite editor.

The request "GET / HTTP" will always get you the default web page for a site. At most sites, "GET / index.htm HTTP" will also work. You can also try "OPTIONS / HTTP" to find out what other HTTP functions are supported by the site. This can lead to various interesting experiments, especially if you find a poorly secured site running Microsoft's IIS web server. You can also start probing the web site to track down other static web pages and dynamic services. It's fun to grab those undecipherable URLs displayed in the address area of your web browser, modify them slightly, and see what kind of responses you can get. You will find lots of additional URLs in the responses. Try them out and see where they lead. No browser required.

### SOAP Requests

I am always intimidated by acronyms. When a new one comes out and I don't know what it means, I figure that it is best to not show my ignorance and to simply stay away from the subject. For a long time, I figured that it was best to stay away from acronyms like SOAP, XML, DTD, WSDL and UDDI. Ken Wilner changed my mind.

Ever since Progress started supporting TCP/IP sockets, I had been using them to interface to other systems without knowing how easy it was to access the web. As shown above, accessing the web from Progress is very easy. Likewise, I had been using XML within Progress without knowing how easy it was to use SOAP. Ken showed me how easy it is to use SOAP from Progress. If you can handle the simple HTTP requests described above and know how to use the XML support within Progress, you are ready to use SOAP.

SOAP, Simple Object Access Protocol, is a standard methodology for making function calls and exchanging data over the Internet. With SOAP, you can get stock prices, check the status of a package sent via Federal Express, access the Babelfish translator, etc. Visit [xmethods.com](http://xmethods.com) for a full list. You can also construct your own set of services and use WebSpeed to serve them up. We did. We use XML messages and WebSpeed to provide the dynamic content for our web site, [foxwoods.com](http://foxwoods.com), and for patron information kiosks situated throughout the property.

When you use SOAP, instead of simply sending out an HTTP request, you must append an XML document to the HTTP request. Instead of getting back an HTML document, you get back XML. That simple. Construct a standard HTTP request, append an XML document to it, send it out and read the XML back in. The trick is constructing the XML to meet the SOAP standards and the requirements of the SOAP server. It takes a little getting used to.

Another acronym, WSDL, Web Services Description Language, is used to specify what goes in the SOAP request. You can download these from [xmethods.com](http://xmethods.com). With a little practice and a working example, you should be able to develop all sorts of interesting applications using data that is readily available on the Internet.

The second example, `hack2.p`, uses SOAP to check the current price of a stock. This program uses the "Delayed Stock Quote" service provided by [xmethods](http://xmethods.com). I suggest that you download the WSDL for the service and compare it to the sample program. After a couple of drinks, it should all become perfectly clear. I was also able to get the "Babelfish Translation Service" and "FedEx Tracker" to work. Good luck and have some fun.

## Ethan Tower

Ethan is currently the Senior Application Architect for Gaming Systems at Mashantucket Pequot Tribal Nation, aka Foxwoods Resort Casino, the world's largest casino. A well-scarred Progress warrior, he is the principle designer and programmer of Progress-based casino management systems used throughout the world. When not testing the outer limits of Progress, he can be found sailing on Fisher's Island Sound. He can be reached at [etower@mptn.org](mailto:etower@mptn.org).

## Progressions

**Lightening White:**

**John Campbell**

**Turquoise Blue:**

**Connie Campbell**

**Royal Purple:**

**Harriet Coates**

**Electric Blue:**

**Michael Bartlett**

Progressions is published bi-monthly, which means 6 times a year, not 24. The cost of the subscription depends upon where you live.

USA is \$70.00 ('cause the government lets us send it bulk mail --slower but cheaper). Canada is \$80.00 (something a-boot postal agreements). International is \$90.00 because you get it first (because we send it first class, otherwise you might not see it in your lifetime).

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 1992-2002 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**

**970.96.3545 Voice 970.963.3548 Fax**

## 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](mailto:progressions@wss.com) or call Connie or Michael at 970.963.3545.

## The Rules Ain't Sacred! by Jop Kluis

It's a well known fact that this magazine is read only by the best of the best in the Progress community. Therefore, I am convinced that everyone of you has tried to build and deploy Progress based source code generators. I will not get into the discussion about the added value of this kind of case tool, but I always encourage every less experienced software engineer to build one if only as part of his or her training. You can learn a lot from building generators.

Not so long ago a trainee came to me with a question about how to solve a particular situation in his generator. While explaining I caught myself almost making a very common mistake about databases. Then I found I had a hard time explaining. For this there can be several causes. It could be that I'm not the gifted coach I think I am, it could be that the trainee was less intelligent as we would like him to be, but personally I think it has to do with the fact that you can design beautiful databases without being aware of this misconception. In fact most databases are designed based on this misconception, and it is not incorrect. In those cases the design is correct, but based on the wrong rules. So fair warning. This is not an article about source code generators, but an article about database design from a fairly theoretical angle. Worse yet: I will try to prove that a construct that is considered to be incorrect has some advantages over the more common solution.

### Constraints

In order to make my point, I first have to explain a bit about the principle of *constraint minimisation*. Constraint minimisation is not a skill or a trick, but a way to validate a database design as being the easiest to maintain. Now, what are constraints? Every (database) design comes with constraints. Constraints are used to enforce the needs within a design. For instance the demand that for every order there should be at least one order-line. On the other hand, constraints are used to prohibit situations we would like to avoid. Example: We do not want orders for which the customer is not known in the database. This means that if we want to delete a customer, we should delete his orders as well. But we would not like to delete customers with back-orders or customers who have not paid all their invoices, etc. Constraints, constraints, constraints....

Now there are two sources of constraints: Functional ones and technical ones. The functional ones should represent all the demands the users have on the system. The technical ones are there to make the whole thing work. For instance data integrity is assured by technical constraints.

Briefly there are two different places to implement your constraints. A constraint can be either enforced by the database management system or by the application(s) using the database. Or shortly: database- and program-constraints. And both of these constraints can fulfill functional or technical needs.

A short example: A company has decided that for any order where the total-order-amount is less than \$500, shipping costs will be charged. Orders over \$500 are free of shipping charges (as long as the delivery address is in the country of origin). This is a business rule and this will result in functional constraints. But there are several ways to implement the enforcement of the constraints. For example, you can do this through the database management system. For instance at the write trigger of an order-line: calculate the total-order-amount and write this to the order-header. Unfortunately this will result in a calculation by each write-trigger. If you are entering an order with 12 order-lines, the system will calculate the shipping charges 12 times. And what happens if someone deletes an order-line?

On the other hand you can implement the enforcement of the constraint in a program. This way you can control the calculation better. You only perform the calculation of the shipping charges when the user is finished adding the order-lines. However, this does not do the trick if one of the customers starts sending his order in an electronic way. Fortunately, techniques are available which enable us to write the needed source only once. We just have to figure out on which places the triggers should be implemented.

### Minimisation

Being experienced developers we all know that for every problem there are as many solutions as there are developers on the project. So which solution is the best? I do not know, but this is where constraint minimisation comes in. It is obvious that you can not minimise the functional constraints. We are talking about technical constraints here. If there are several solutions to a problem (and there always are) try to find the one with

the minimal technical constraints. This not only saves you time implementing, but it will simplify the system in a way that will result in a higher maintainability. A simple example:

There is a transport company with a bunch of trucks. They want to keep track of who is the current driver of a particular truck. They also tell us that some trucks have more than one regular driver. These demands can result in two different solutions:

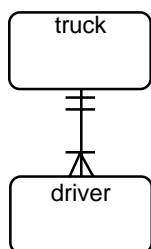


Figure A

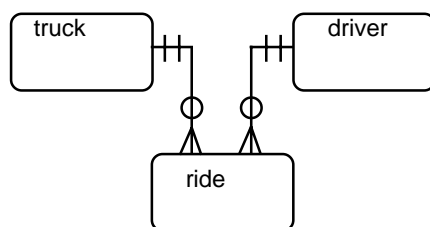


Figure B

Now, the example in Figure A does exactly what the company asked me. However Figure B looks much more flexible to us. And flexibility is what our customers want. On the other hand, if we look at the constraints, we get a different perspective. In Figure A we will find at least the following constraints: The truck-key in *truck*  $\diamond$  null and unique; the driver-key in *driver*  $\diamond$  null and unique; the truck-key in *driver* should exist in the truck table. 3 constraints.

To make example B work we need: The truck-key in *truck*  $\diamond$  null and unique; the driver-key in *driver*  $\diamond$  null and unique; the truck-key in *ride* should exist in the *truck* table; the driver-key in *ride* should exist in the *driver*-table; for each entry in driver there should be exactly one entry in ride. 5 constraints and no additional functionality. I admit it is a silly example, and I would seriously consider implementing situation B. But still, it is something to think about.

### Alternative key

Back to the original problem. The software engineer trying to build a generator wanted (actually I told him to) implement referential checks as well. For instance: In a program to enter order-lines on a order there should be a check that the article number entered in the order-line is an existing article-number, get the record and automatically display the article description. Now, with

a generator you do not want to have to tell the generator everything it has to do. The generator should recognise the need for this referential check by itself and generate the needed source-code without manual interference. The question to the engineer put to me was, "How can a generator recognise these needed referential checks in the data model?" And I almost answered, "Take the record you want to maintain and see if you can find other tables of which the primary key is a field (or a combination of fields) in your record." Now, what's wrong with this answer? The answer itself is correct, but not complete. This answer suggests that referential integrity can only be maintained by referring to primary keys. It's correct to state that that is one way, but you can also use the alternative-keys. Or to take this further, in some cases the use of alternative keys for referential integrity will reduce the amount of technical constraints. Now, what is an alternative key? You have alternative keys if a table has more than one candidate key. Now, what is a candidate key? Each column (or combination of columns) in a table that can uniquely identify a record is a candidate key. They are candidates to become the primary key, but only one key can be the primary key. The other keys that can uniquely identify a record are so called alternative keys.

### Reference to an alternative key

An alternative key is a unique identifier to a record that you can use for reference as well as the primary key. In some books such keys (unique identifiers used for referential integrity) are called referential keys. Now, the mistake I (almost) made was based on the misconception that a referential key should always be the primary key. Yes, a primary key will do fine, but alternative keys will do just as well. Or in some cases even better. A small example

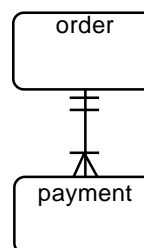


Figure C

The situation: We have orders, invoices and payments. Each order will - eventually - result in one invoice. But, for each invoice there can be several partial payments (see Figure C).

Now, the order table will look something like:

ORDER(ORDER-NUMBER, CLIENT-NUMBER, INVOICE-NUMBER,.....etc) . Here the primary key is obviously the ORDER-NUMBER, as not every order has an INVOICE-NUMBER (not invoiced yet) and the CLIENT-NUMBER will not do unless we expect every client to use our services only once. The payment table will look something like: PAYMENT(ORDER-NUMBER, PAYMENT-SEQUENCE, DATE, AMOUNT....etc) (see Figure D below)

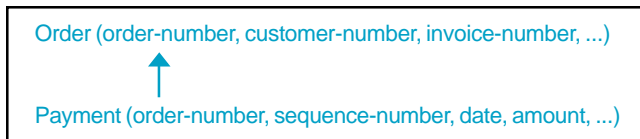


Figure D

In this case (although it is against a lot of conventions) I think that a nice alternative would be to make the primary key in payment not out of ORDER-NUMBER combined with PAYMENT-SEQUENCE, but out of INVOICE-NUMBER combined with PAYMENT-SEQUENCE (see Figure E).

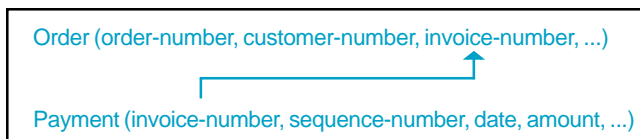


Figure E

Although INVOICE-NUMBER is not a real candidate-key (not every order is invoiced, therefore not every order is uniquely identified by an INVOICE-NUMBER), this will do nicely. Actually, if you would use constraint minimisation (as I always do) to compare these two options you will see that the second option is even better. If I would enter a payment with the first option I would have to make sure that the ORDER-NUMBER in the new payment record refers to an existing order, and that the INVOICE-NUMBER in this order  $\neq$  null. With the second option I get the same result by only checking that the INVOICE-NUMBER in the new payment record is an existing INVOICE-NUMBER in the order table. Voila! One constraint less. You see, sometimes it pays off to question your actions. However, I agree it is arguable whether it is efficient to

figure out these situations in a large database design, unless you train yourself to think like this. But then again there is a lot more you can do to train yourself in regards to database design.

## Jop Kluis

Jop has been around for some time now. Actually he recently spoke at an international Progress meeting where he, for the first time, was not the rookie. Over the years he has lost his programming skills as he focuses on project management and design. As a typical Dutchman he loves to point out mistakes to others. That probably explains his interest in database design. Not only a lot of errors in applications find their origin in the database design, but designing a database enables you to make a very large number of beautiful mistakes.

At the moment Jop is Project manager with Caesar Propago, one of the leading Progress consulting partners in the Netherlands. Email: [jop\\_kluis@hotmail.com](mailto:jop_kluis@hotmail.com)



## Code On-Line

We have made the code easier to access. Simply visit [www.wss.com](http://www.wss.com). Choose "**Publications**" in the top menu, and "**Code**" in the lower menu.

You can obtain code from any issue back to number 28!! No more fussing around with passwords. Don't you love it??

## Putting Super Procedures to Use at the Session Level by John D. Johnson

While many Progress developers have employed the capabilities of Super Procedures since their introduction in Version 9.0, few have taken advantage of the ability to associate a Super Procedure with the SESSION handle. In this article, I will discuss several uses for such an implementation as employed by some of my clients.

### Storage of Properties Needed Throughout an Application

Many applications rely upon data obtained during the login process to control processing throughout the application. In the "Procedural World", we used to use GLOBAL SHARED variables to hold this data and make it available to all procedures within the application, but the use of shared resources is not a good practice in an "Event-Driven World". Therefore, we now need a place to store such properties and a simple method for setting and retrieving their values. A SESSION Super Procedure provides an easy solution to this problem.

For example, in one recent ADM application the Company the user wished to work with is determined at the time of login. Every query within that user's session, therefore, needs to be filtered to display only the data associated with the selected Company. But that means that we need to determine the value of the "Current Company" property prior to opening each query. For this, we used a simple Super Procedure attached to the SESSION handle.

Figure A shows a simplified listing (all of the Structured Procedure code has been removed) of the Super Procedure code. The Definitions Section contains the definition of the variable used to hold the Company Property – the value of the selected Company. The remainder of the procedure consists of 'get' and 'set' functions to manipulate this property. The setCompInfo function takes two INPUT parameters – one for Company Key and the other for Company Name – and assigns them to the cCompInfo variable as a CHR(3)-delimited list. This function is called from the login program. The getCompInfo function can then be called from anywhere within the application and returns the value of the cCompInfo variable. Figure B illustrates how this function is used by a SmartDataObject to

configure its query within a 'local' initializeObject procedure. Figure C illustrates how the getCompInfo function is also used within a 'local' initializeObject procedure in a SmartWindow to update the Title of the window.

```

/***** Definitions *****/
DEFINE VARIABLE cCompInfo AS CHAR NO-UNDO.

/***** Function Prototypes *****/
FUNCTION getCompInfo RETURNS CHARACTER
PRIVATE
( ) FORWARD.

FUNCTION setCompInfo RETURNS LOGICAL
( piCompNum AS INT,
  pcCompName AS CHAR ) FORWARD.

/***** Function Implementations *****/
FUNCTION getCompInfo RETURNS CHARACTER
PRIVATE
( ) :
/*-----
Purpose: To RETURN the value of the
Company Info Property
Notes:
-----*/

RETURN cCompInfo.

END FUNCTION.

FUNCTION setCompInfo RETURNS LOGICAL
( piCompNum AS INT,
  pcCompName AS CHAR ) :
/*-----
Purpose: To set the value of the
Company Info Property
Notes:
-----*/

ASSIGN cCompInfo = STRING(piCompNum) +
CHR(3) + pcCompName.
RETURN TRUE.

END FUNCTION.

```

Figure A

```

PROCEDURE initializeObject :
/*-----
Purpose:      Super Override
Parameters:
Notes:       In ddept.w SDO
-----*/
DEFINE VARIABLE cCompNum AS CHAR NO-UNDO.
DEFINE VARIABLE lSuccess AS LOGI NO-UNDO.

/* Code placed here will execute PRIOR to
standard behavior. */
ASSIGN
cCompNum = ENTRY(1,
DYNAMIC-FUNCTION('getCompInfo'), CHR(3))
lSuccess = DYNAMIC-FUNCTION(
'assignQuerySelection':U,
INPUT "Company",
INPUT cCompNum,
INPUT "").

RUN SUPER.

/* Code placed here will execute AFTER
standard behavior. */

END PROCEDURE.

```

Figure B

```

PROCEDURE initializeObject :
/*-----
Purpose:      Super Override
Parameters:
Notes:       In SmartWindow to update Window
Title
-----*/
DEFINE VARIABLE cTitle AS CHAR NO-UNDO.
DEFINE VARIABLE cCompInfo AS CHAR NO-UNDO.
/* Code placed here will execute PRIOR to
standard behavior. */
ASSIGN cTitle = CURRENT-WINDOW:TITLE
cCompInfo = DYNAMIC-FUNCTION(
'getCompInfo')
cTitle = cTitle + " - " +
ENTRY(1, cCompInfo, CHR(3)) +
" " + ENTRY(2, cCompInfo, CHR(3))
CURRENT-WINDOW:TITLE = cTitle.

RUN SUPER.

/* Code placed here will execute AFTER
standard behavior. */

END PROCEDURE.

```

Figure C.

### Providing Context-Sensitive Help

Another client needed a simple method for providing context-sensitive help throughout the application. In some cases, this help would take the form of a simple message telling the user something about the field. In other situations, the developer wished to provide the user with a pop-up browser containing all valid values for the field. Once again, a SESSION Super Procedure provided the solution.

The schema was modified to add a 'HelpControl' table. This table was keyed upon the field name and contained two other pertinent fields. If the value of the HelpProg field is "Text", then the value of the MsgText field is displayed with a MESSAGE statement. Otherwise, the value of the HelpProg field is used to RUN the lookup program.

To implement the help function, the SESSION Super Procedure – shown in Figure D – contains an 'ON F1 ANYWHERE' trigger that uses the SELF:NAME property to determine the identity of the current field. It then runs the helpControl procedure – passing it the name of the current field. The cFieldValue variable is set within the helpControl procedure if the user selects a record from a lookup program. If this variable is set when control returns to the trigger, the value is then assigned to the current field.

```

/***** Definitions *****/
DEFINE VARIABLE cFieldValue AS CHAR NO-UNDO.

ON F1 ANYWHERE
DO:
ASSIGN cFieldValue = "".
RUN helpControl (input SELF:NAME).
IF cFieldValue NE "" then
ASSIGN SELF:SCREEN-VALUE = cFieldValue.
END.

ON F10 ANYWHERE
DO:
RUN 'w-submitBugs.w'.
END.

```



- 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](http://www.jargonsoft.com) (On-line demo!)  
[info@jargonsoft.com](mailto:info@jargonsoft.com)

All brand and product names are the property of their respective owners.



## White Star Software



### Products – Great Stuff for the New Millennium

- TailorPro – Custom application screens and business rules without source code!
- Product – An open source dictionary reporting tool

### Training – By the Guy Who Wrote Making Good Progress

- Get to Know Progress – a thorough introduction for programmers new to Progress
- SpeedyWeb – Teaches your 4GL programmers how to implement in WebSpeed
- Smart Coding – Using ADM2/Smart Objects to develop applications

### Services – We're Here to Serve You

- Get High on Progress – high performance, high availability, high reliability with our system Guru
- Get with it – Move your application to GUI, Web or Web client with our conversion tools
- You Name It – Consulting (mfg/pro mxp, you name it), development, contract programming

### Publications – Learn What There Is to Know About Progress

- Making Good Progress – The Progress Encyclopedia – everything you want to know
- Code Smarter – Reusability, dynamic widgets, super procedures – the best of Progress
- Disaster Prevention – How to design and maintain a program to protect your system

Creating a Shared Vision of Excellence in the Progress Community

[www.wss.com](http://www.wss.com)

```

/***** Internal Procedures *****/
PROCEDURE helpControl :
/*
Purpose: To return a help message or run a
lookup program, depending upon the value
of the HelpProg field.
Parameters: INPUT the name of the current
field on the screen.
Notes:
*/
DEF INPUT PARAM cFieldName AS CHAR NO-UNDO.

FIND HelpControl WHERE
    HelpControl.HelpField = cFieldName
    NO-LOCK NO-ERROR.
IF NOT AVAILABLE HelpControl THEN
    MESSAGE
    "No On-Line Help is available for the "
    cFieldName
    " field." VIEW-AS ALERT-BOX INFO.
ELSE DO:
    IF HelpControl.HelpProg = "Text" THEN
        MESSAGE HelpControl.MsgText
        VIEW-AS ALERT-BOX INFO.
    ELSE
        RUN VALUE(HelpControl.HelpProg)
        (OUTPUT cFieldValue).
END.
END PROCEDURE.

```

Figure D.

One interesting addition that the client made to this Super Procedure is apparent in the 'ON F10 ANYWHERE' Trigger. This trigger provide the users with the capability of logging an application bug report at any time without leaving the application.

### Starting the Session Super Procedure

The easiest method for instantiating the SESSION Super Procedure is to include it as part of the login process. Figure E shows the startup program for the client using the context-sensitive help process. This program is the first one called when the user starts the application. After running the login screen – c-login.w, it checks that a valid USERID has been set. If it has, the startup program instantiates the errorhandler.p program and adds it as a Super Procedure to the SESSION handle. This allows all code within the errorhandler.p program to be accessed from anywhere within the application without the need to obtain the handle of the procedure.

```

RUN c-login.w.

IF USERID("labelapp") NE "" THEN DO:
    DEF VAR hHdl AS HANDLE NO-UNDO.
    RUN 'errorHandler.p' PERSISTENT SET
hHdl.
    SESSION:ADD-SUPER-PROCEDURE(hHdl).
    RUN w-main.w.
    END.
ELSE
    MESSAGE "Blank userids are not allowed."
    VIEW-AS ALERT-BOX ERROR.

QUIT.

```

Figure E.

While many developers have learned to take advantage of Super Procedures that are associated with specific programs within their applications, few have thought to look at the special advantages offered by associating a Super Procedure with the SESSION handle. The ability to start a Super Procedure at the time of login and never have to worry about it afterwards provides a valuable tool to the developer, while also providing enhanced functionality and performance to the user.

## John D. Johnson

John is the author of the **Roadmap To The SmartObjects** book and spends much of his time roaming the world doing such things as Progress training, code tuning, replication, and project management. But he's never too busy for new clients.

**J. D. Johnson & Associates**  
**Voice: 972-733-1422**  
**Fax: 972-733-1580**  
**jdj@jdjohnson.com**



## WHITE STAR BOOKS AND PUBLICATIONS



**MAKING GOOD PROGRESS:** An absolutely comprehensive book on Progress. Almost 1,000 pages and 35 chapters on everything you need to know about the application development process --from design to tuning.

\$130 (Priority ship included)

\$144 USD Canada (air ship)

\$160 USD International (air ship)

**DISASTER PREVENTION:** A booklet by Adam Backman on what you need to know to guarantee your system's integrity. Insights on RAID, replication, on-line back ups and much more. There's a lot more to backing up your system than just putting a tape in the drive.

\$24 US

\$27 USD Canada/International

**HIGH PERFORMANCE REPORTS:** This guide will lead you through the steps of analyzing user requests and designing reports that will be efficient and maintainable; how to analyze and rework slow reports to speed them up.

\$23 US

\$26 USD Canada/International

**CODE SMARTER, PROGRAMMING FOR THE FUTURE:** All of the features for "Future Proof" programming and how to use them: the preprocessor, internal procedures, dynamic functions, publish/subscribe, super procedures, dynamic buffers, widgets and browsers . . .

\$26 US

\$29 USD Canada/International

**WHAT DO I DO NOW THAT I HAVE V7/V8?** An introduction to the principles of event-driven programming and the many new features and concepts of V7/V8, this booklet will get you up to speed with the preprocessor, schema and UI triggers, internal procedures, the query command and new issues in scoping and transactions.

\$25 US

\$28 USD Canada/International

**PROGRESSIONS:** A technical journal by Progress programmers, Progressions covers a broad range of subjects for all levels of expertise. It features articles (with code) on such topics as report tricks, database design, undocumented commands, editorials, product reviews, as well as articles on Versions through 8.2 for both GUI and character mode. 1 year subscription is 6 issues. Back issues are available: US. \$7.00 each Canada/International US \$8.00.

\$70.00 US

\$80 USD Canada

\$90 USD International

**VERSION 8/9 FOR \$89:** Can't find Version 8/9 information? It's right at your fingertips! We have 11 back issues of Progressions—our V8/V9 Package. Each issue contains one or more articles on useful Version 8/9 topics: insights, tips, the editor, dynamic menus, launching windows apps, triggers, benchmarks, strip menus, application security, designing a GUI interface, internal procedures and much more. Progressions V8/V9 Package -11 issues

US \$89 USD

\$99 USD Canada/International

We accept Visa, MasterCard, AMEX (Include Expiration Date) fax credit card orders 970.963.3548 or email to [info@wss.com](mailto:info@wss.com). We also accept company checks drawn on a US Bank. Sorry, no purchase orders. Shipping and Handling included.

**Checks must be in U.S. Dollars payable to White Star Software.** CA residents add 7.75%

### White Star Software

970.963.3545 [info@wss.com](mailto:info@wss.com) [www.wss.com](http://www.wss.com) 970.963.3548 FAX

## Using Queries to do a Find by Justin Martin

I recently had a customer ask me how to do a "find" using a query. If you're still learning about queries, you may not yet have discovered that queries are great for "for each" type situations, but don't (directly) support a "find". The query statement mimics the "for each" in almost all ways. If you're enterprising (meaning that you've actually read the documentation on the for each statement) or you're an experienced Progress user, you know that "for each" supports "for first" and "for last". This means that you can say

```
for first customer where country =
'nederland'.
```

This would be a great way to use a query to support find. However, the query only supports "first" in the second and subsequent record phrases. This is because queries are designed to return record sets, and for first only retrieves one record. So what to do?

Well, of course, you could use the "find" statement. We'll use this as a starting point, so that you'll understand a little about query cursors vs. regular record cursors. A for each or find utilizes the standard Progress record buffer, and there is a cursor pointing to the record in that buffer. (That is, there is a pointer to an index entry for the cursor in that buffer). When you create a query, on the other hand, the record pointed to by the query is in a distinct memory space (the query cache) from the standard record buffer. If you choose to display a record which is pointed to by a browser, for example, you can display to the screen buffer from the query cache, just as you could display data to the screen buffer from a standard record buffer. So technically, you could be pointing to two different records: one in a "find/for each" buffer, and one in the query cache.

Suppose you are looking at all of the customer records, seeing the city and country fields. You have a fill-in which allows you to find a specific city.



Suppose you enter Helsinki. How do you get the browser to point to this record? The code is pretty simple, actually. Suppose the query under the browser is called BrwsCust. Here is the code you'd put under the "find" button:

```
define variable TempRowid as rowid.
find first customer where city begins
fCity:screen-value no-lock no-error.
if available(customer) then do:
    TempRowid = rowid(customer).
    reposition BrwsCust to rowid(temprowid).
end.
else message "No customer with this city"
view-as alert-box.
```

This is all well and good, and many people have taken advantage of this nice fact that query buffers and record buffers don't interfere with each other.

But what happens if you have a second query which you want to use to locate (i.e. find) a single record, and then reposition the first query to that record.

Well, it's not so difficult as it might seem. There are some caveats, but they're no big deal, either. Let's take a look.

First of all, you have a second query in addition to brCust.

```
define variable TempRowid as rowid.
define query TempQuery for customer.
open query TempQuery for each customer where
city begins fCity:screen-value.
get first TempQuery.
```

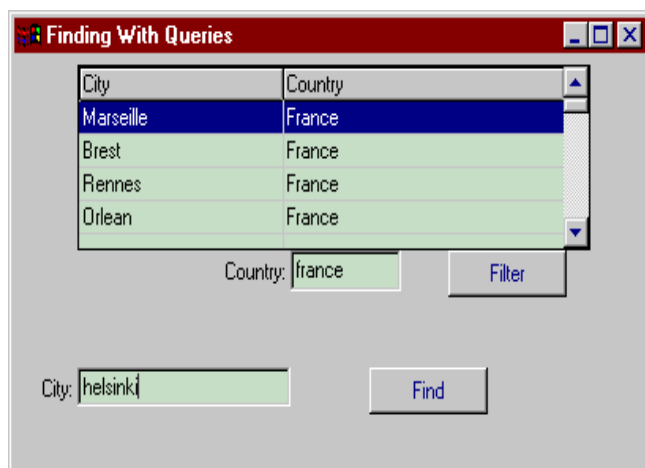
This is the equivalent of performing a find with a query. What is the equivalent of "if not available?". It is simply:

```
if not query-off-end("tempquery") then do:
```

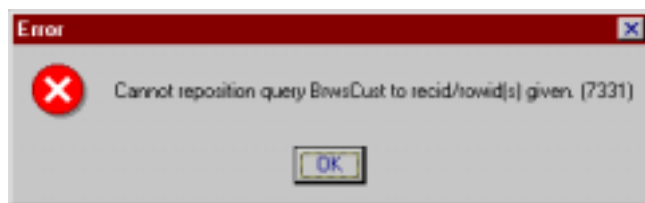
So now we have the ability to redirect the first query to the record retrieved by the second. (Again, this is possible because the two queries have different query caches, and thus can point to different records at the same time).

```
TempRowid = rowid(customer).
reposition BrwsCust to rowid(temprowid).
```

This is all well and good, but what do you do in the following situation: you have filtered the browser for the customers in France, but are trying to locate a customer in Helsinki (which, when last we checked, was not a suburb of Paris).

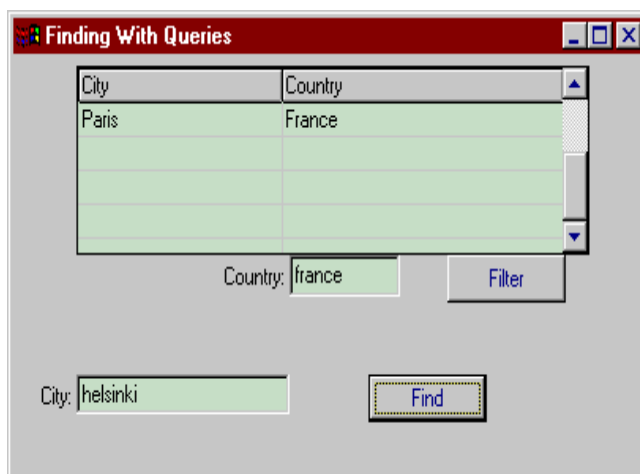


Now what? Well, for one thing, if you don't do anything, what will happen is as follows:



So, you add the "no-error" to the reposition statement.

```
reposition BrwsCust to rowid(temprowid) no-
error.
```



The result isn't bad, but then again, it isn't right, either. But you know, that since you're using Progress, there's a simple solution. And, of course, it involves only 1 more line of code.

```
if error-status:error then
    message "Record not in original set"
    view-as alert-box.
```

So it's just that simple: use a second query, perform a find; reposition the first to the rowid located by the second. If the first is not re-locatable, use an error message. Here is the code all in one piece.

```
define variable TempRowid as rowid.
define query TempQuery for customer.
open query TempQuery for each customer where
city begins fCity:screen-value.
get first TempQuery.
if not query-off-end("tempquery") then do:
    TempRowid = rowid(customer).
    reposition BrwsCust to rowid(temprowid) no-
error.
    if error-status:error then
        message "Record not in original set"
        view-as alert-box.
end.
```

Shouldn't everything be this easy?

## Editors and Productivity ED For Windows by Neville Franks

Stop and think about how many hours you spend each day using an editor and then give some thought to where your time goes while you're editing. ED for Windows is designed to extend your capabilities and let you move beyond the basics. This article will cover some of the features you should be looking for in an editor and help you to get the most out of your day to day editing activities.

### Reducing Coding Errors

A good programmer's editor can go a long way in helping to reduce coding errors, and thus minimize the time you waste in compile-edit-compile cycles.

Effective color syntax highlighting lies at the core of reducing errors. People who have used the Progress editor have enjoyed the recent addition of color coded syntax. ED4 goes beyond these capabilities. For example, you can designate a special color for tables names. ED is extremely flexible in allowing you to designate your own categories of syntax and their colors. It also makes code easy to read, by letting you focus on specific areas within your code, while excluding others. For example comments can appear to melt into the the background when you're not interested in reading them and vice-versa.

For Progress, there is a "dictionary" of approximately 2,000 keywords broken up into six categories, each category with its own color specification is supplied with ED. Each of ED's 33+ languages has its own dictionary, which can hold an unlimited number of keywords in 14 individual categories. The dictionaries come in simple Ascii text files which are quick and easy to change to suit your specific needs.

### Help Me Write this Code

At any point in time writing code usually involves working with a small set of well defined keywords, which are repeated over and over. This set of keywords will change in part as you move from one area of the code to another.

Like Progress, ED's code automation and completion capabilities can assist by finishing keywords for you, and writing complete code fragments ready for you to fill in the blanks. Simply type the first few characters and press Space or Escape and then let ED do the rest. Perhaps you are annoyed by certain hard-wired features of Progress' auto-completion. You have an alias for "for each", but if

you type in "open query myquery <cr> for each" you will get Progress' auto-completion whether or not you want it. ED allows you to "force" auto-completion with the ESC key, so you don't get it where you don't want it. This is also true of do...end and many other places where Progress automatically inserts completions into your syntax when you don't necessarily want them.

ED comes with around 150 Progress aliases ready to use, and you can easily create your own or change the ones we supply. It also will auto-complete variable names for you if you ask it to! No more three-letter variable names because programmers don't like to type!

Macros are another arm of ED's code automation capabilities which enable you to record, save and replay complex series of commands, greatly reducing the effort required to perform highly repetitive tasks.

### Finding Stuff

Programmers tend to spend a lot of time trying to find stuff, so your editor needs to have very good searching capabilities. ED V4's new search engine includes full Regular Expression support, including searching for an expression which may cross multiple lines and minimal and maximal repeat operators, plus the ability to let you choose from Posix, Microsoft and ED Regular Expression dialects.

Regular expressions let you find text using complex pattern matching. For example: "screen(ing|ed)" will find screened or screening. The expression syntax is familiar to most people who have done Unix scripting.

Often you're not interested in finding matches inside strings or comments, so ED's Filters let you include and exclude text from a list of 33 syntax elements. Other options include Match case and Match whole word only. Effective use of the various find capabilities helps you focus only on the matches that matter and not get weighed down by extraneous clutter. You can also search on white space elements. For example, the pattern can be searched for with a space at the end, so that incidences of and in the middle of a word (candy) are ignored.

ED has a strong design philosophy of letting you do what you want when you want. For this reason it's Find and Replace dialog is modeless and also compact. Modeless means you can edit text or do anything else in ED without having to keep opening and closing the dialog. You can even set Bookmarks on every match and visit them when you please.

```

54      1/06/98  GFS Removed _REFRESHABLE
55      1/97    SLK Added _vbx2ocx temp table
56
57      -----*/
58      /* _U - Universal Widget
59         Contains the attributes that are commonly found in most widgets */
60
61      /* Max tables in query definition */
62      #Glob MaxTbl 20
63      /* Max tables in query definition */
64      #Glob MaxFld 127
65      /* Max User defined lists */
66      #Glob MaxUserLists 6
67
68      DEFINE (1) SHARED TEMP-TABLE _U
69      FIELD _ALIGN          AS CHAR      INITIAL "L" /* "L"eft "C"olon "R"ight */
70      FIELD _HANDLE        AS WIDGET    LABEL "Handle"
71      FIELD _CON-HANDLE    AS COMPONENT-HANDLE LABEL "Con Handle"
72      FIELD _NAME          AS CHAR      LABEL "Name"          FORMAT "X(32)"
73      FIELD _BUFFER        AS CHAR      LABEL "Buffer"          INITIAL ?
74      FIELD _CONTEXT-HELP-ID AS INTEGER LABEL "Context Help ID" INITIAL ?
75      FIELD _DBNAME        AS CHAR      LABEL "DB Name"        INITIAL ?
76      FIELD _DEFINED-BY    AS CHAR      LABEL "DB,User,Tool"    INITIAL "Tool"

```

ED's Find in Files progresses from searching within the current file to searching files on disk. You can search entire sub-directory trees for specific files, search just a single directory or search all of the files in one or all ED Projects. The resulting matches are listed in one of ED's Find in Files Output windows. Selecting a match from the list jumps to it in the relevant source file. Plus, you can do search and replace across multiple files, so if you want to change a variable name, you can do so without worrying about missing it in another program. ED makes extensive use of multi-threading so functions like Find in Files work away quickly and quietly in the background so that you can keep on working.

### Checking For Progress Syntax Errors

The ability to jump directly to the location of syntax errors in your code is another way an editor can help you work smarter and faster. This is easily accomplished by enabling Progress to control ED and pass it error information.

Tim Townsend has written a Progress Check Syntax/

Compile add-in which lets you work through and fix your errors directly in ED. The advantage over the Progress syntax checker is that each of the errors are listed in a separate portion of the ED window rather than as messages which disappear once you dismiss them. Visit Tim's site at [www.tttechno.com/edtop.htm](http://www.tttechno.com/edtop.htm) and check out what Tim has to offer.

### What's the Difference?

How often have you changed some code only to discover something has now broken or isn't behaving quite as it used to? This is where Version Control Systems and a good Difference Analysis program help save the day. (You are all using a VCS aren't you?)

Difference Analysis (DA) highlights the changes between different versions of a file so you can quickly identify potential problem areas. For example, boxes are drawn around changed text and lines connect the changes in the two files. As you scroll around everything stays in step between the two files and navigating the differences is a breeze. DA in ED is a powerful tool not offered by the

Progress editor.

### Code Navigation and Browsing

Finding your way around large complex projects with hundreds of files can be a daunting task, especially if you are new to a project or it has been a while since you've worked on it and that intimate familiarity you once had has long since dissipated.

Tools like Find in Files which we mentioned earlier and Code Browsers are able to ease this burden for you. Code browsing capabilities vary considerably from one editing environment to the next, with many lower end editors providing no browsing support whatsoever.

Code browsers parse your source code and build up lists of important code elements, such as functions and procedures. These are typically stored in a database for fast access. The editor will then provide a number of ways of using the information stored in the database to enable you to jump instantly to any item and also to visualize its contents, usually in a tree format.

The Source Database (SDB) in ED V4 builds a comprehensive resource of every Progress function, procedure and various ON statements. The SDB is built and maintained using background threads and is updated in real time as you edit your code, so that it is always up to date. And unlike compilers and some other editors you don't need a syntactically correct program before you can use ED's code browsing capabilities.

With ED's SDB you can Right click on any function/procedure call etc. and instantly jump to its definition. Or you can search the database for a specific item, and use regular expressions to broaden the search if you're not exactly sure what you are looking for. The ClassView tree in the Workspace window displays the entire database and lets you jump to any item. ED's Navigate Toolbar has a drop-down list of all functions/procedures etc. in the current source file and displays the name of the function you are currently positioned within. You can select from the list to jump to any function. The same toolbar also includes a list of all include files in the current file providing quick access to any one.

Some of ED's other navigation aids include; Goto Next/Previous function/procedure, Open include file at the cursor, and Statement and Bracket matching.

Statement matching helps you find mismatched blocks of code. For example, if you right click or use Goto|Locate Object on a DO: statement ED will jump to its matching

END statement and vice-versa. This is another of ED's many capabilities that our Progress customers find indispensable.

By delivering a comprehensive and carefully thought through suite of code navigation capabilities, built around a fast real time database, ED can help unravel the complex nature of your code and let you concentrate on the task at hand without getting bogged down in the minutiae.

### Projects

The Projects feature lets you arrange related sets of files together into harmonious groups. As soon as you open a Project all of its files are instantly accessible via a Project Tree. Files can be organized into virtual folders within the project tree, making them easier to manage and locate. With ED you can have multiple projects open at the same time.

This is ideal for working with application and library files for example, or even including other applications so you have quick access to their code.

Projects are the cornerstone of some of ED's more powerful productivity tools such as the Source Database and Find in Files. They also enable you to specify include file search paths and API Help files for fast access to include and Help files.

Project files are portable, meaning they can be moved to a different drive or directory, along with their source files, and still work correctly. By grouping files into Projects you mimic your real world view of your source files and gain optimal access to them.

Sometimes, we settle for what we have, even though it may not do everything we want. ED and its extensions allows you to work as a stand-alone editor (including database connections), and to substitute ED for the App Builder section editor as well. It provides all of the functionality of the Progress editor and then some, and numerous other features not at all native to the Progress work environment. It is an excellent alternative for developing Progress code quickly and efficiently.

## Neville Franks

Neville is the author of ED for Windows. He can be contacted at [support@getsoft.com](mailto:support@getsoft.com) For more information on ED V4 and a trial copy, check [www.getsoft.com](http://www.getsoft.com)

## Dynamic Temp-tables 101 by John Campbell

As the development paradigm evolves to support new ways of exchanging information and doing business, the Progress language has evolved in some subtle but phenomenally powerful ways, too. It will not be long before completely dynamic programming will be possible. What does this look like, and what does it mean for the dozens, if not hundreds of applications which already exist?

There have been recent articles in Progressions about dynamic queries and dynamic browsers. These are some of the early, useful tools to add to existing programs. Dynamic queries allow a report writer to construct complicated selection statements "on the fly", giving substantially more flexibility and power (read improved performance) to reports which are otherwise unwieldy and often slow. What we will see in coming months and years are programs which perform dynamic data selection, use dynamic memory storage structures, and create dynamic interfaces. The repository-based Dynamics (originally ICF) from Progress, and other products such as DWP from NetSetup, incorporate a great deal of dynamic programming. The value is reduction of client side code, and the ability to store screen definitions, and even a certain amount of business logic, in database repositories; to be extracted at runtime and used to "build" the application on the fly. In addition to addressing topics of ongoing interest to non-dynamic programmers, this and future issues of Progressions will explore the nature and mechanics of dynamic programming.

### Dynamic Temp-tables

Who hasn't used temp-tables to solve some complicated data storage or reporting problem? Static temp-tables are an ideal solution in the host-based or client-server world, where there is but one system holding the temporary data; static temp-table definitions and data were passed as shared structures or as parameters. This worked wonderfully, and served many different situations very well. Today, though, there are different kinds of demands on software. Code that people write today often has to work on character, client-server, and the web. So how is it possible to do this?

Many of the newer deployment models require the app server. Static temp-tables still work just fine in this environment: it is possible to pass static parameters to programs on different "sides" of the app server. While it is possible to use legacy code structures, there are some downsides to this.

Imagine that you are using the AppServer. This is becoming more and more common as environments diversify. So you have two different environments: the database server, and the app server. With luck, these machines are under one roof. But this is not always the case, so you may have code deployed in two different physical environments. This means that if you make a change to the definitions in one environment, you will have to remember to make the exact same changes in the other app server environment. Not too big a headache; but still, a layer of tracking you might not want to have to do.

Now, imagine that users are deploying your code to their webclient system. This means that you don't necessarily know who has your code. So the web client is good at keeping track of whether the code is different between the source machine and the webclient machine, but this is an annoyance to the end user to be continually downloading code.

Imagine further that you don't necessarily know what kind of data structures are coming your way. I once had a client who consolidated data from several hundred different reporting sources. For whatever reason, the reporting format was not uniform between these sources. In addition, the nature of the data being reported changed from one month to the next. Since the reports "dribbled in" throughout the month, at any given time, we had an uncountable number of different report formats which were all current. If we coded each format we would have done nothing but write data-reading routines all the time. This leads to insanity in very short order. Our solution was to take an electronic form that contained the format, read it into a database table, and generated code from that database table which could then read the incoming data.

In each of these described scenarios, static temp tables work, but there are downsides to them. Dynamic temp-tables are an ideal solution. The last situation, that of unknown data formats, may sound familiar to you if you are using EDI or XML. This particular scenario will be the subject of a future article. In the meantime, I'd like to expand on the earlier article by Bruce Gruenbaum and Mary Szekely on temp-table handles, show how to use dynamic temp-tables, and how to pass information from one session to another (with the app server in the picture).

Let's start with static temp-tables and review how they work so that the structure of dynamic Temp-tables is not so esoteric sounding. When you work with a static temp-table you begin with

```
define temp-table MyTable
```

```
field FieldA as char,    etc.
index Index1 as primary Field FieldA.
```

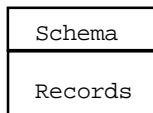
Here you define a structure which represents the "schema" of the temp-table.

Next, you create records in the table.

```
create MyTable.
assign FieldA = "value1".
```

And so on. Here, you are creating records and populating specific fields within the temp table. This is very similar to what we do with database records.

Here you see a schema with individual fields defined. This schema structure cannot hold records. It simply holds the definitional information for the records. So, as we create records, we come to have (visually, not literally) a scenario that looks like this:



This represents the schema, at top, with records created in a separate 'space' from the schema itself. Now, let's imagine creating the temp table dynamically. First of all, this requires having handles to point to these structures. Why? If you have a defined structure, you have no other way to refer to it (traditionally, that is) other than by the names you have given the structures (MyTable, FieldA, etc.) And, of course, you can't manipulate the structure dynamically. Let's look at the dynamic method of building temp-tables. It takes more code—the first time, but once you have created a model, and the code to build a temp-table structure, you don't have to do (most of) it over for each subsequent table. We'll show two models for creating a temp-table from a database table. We'll use the schema as a model, but if you had your own repository of tables and fields, the code would not be significantly different.

The first method is extremely simple, but can be put to good advantage with a little imagination. Here is the code to create a dynamic temp-table identical to the customer table.

```
define variable hTempTable as handle.
define variable FileName as char init "cus-
tomer".
create temp-table hTempTable.
hTempTable:create-like(FileName).
```

At this point, we have created the dynamic "schema" , similar to the one pictured above. Just as we can't look

at the schema of a temp-table, we can't really look directly at the schema of a dynamic temp-table.

Next, let's look at the structure of this temp-table, so you get an idea of the methods and attributes you can use. Before we can use the temp-table, we have to signal that we are done adding fields to it. (Think of a static temp-table: you can define it like a DB table, and then specify additional fields.) So we need one more statement before we go on:

```
hTempTable:temp-table-prepare("ttcustomer").
```

We can make this more abstract by saying:

```
hTempTable:temp-table-prepare("tt" +
filename).
```

At this point, you would think that you could loop through all the fields and inspect them. But remember, you can't do that to a static temp-table. Here is the process for a dynamic one: you "define" a record buffer, and inspect this record buffer. This buffer is referred to as the default buffer for the temp-table, and its handle is the default-buffer-handle. Here goes:

```
define variable hTTBuffer as handle.
hTTBuffer = hTempTable:default-buffer-handle.
```

Now we have our record buffer, so we are going to look at each field. Since this is completely dynamic, we have to define a handle as a pointer to the fields.

```
define variable cntnr      as int      no-undo.
define variable hTTField  as handle no-undo.
```

We have an attribute which tells us the number of fields in the buffer, and one which tells us the name of the field. Here's the rest.

```
do cntnr = 1 to hTTBuffer:num-fields with
down:
  hTTField = hTTBuffer:buffer-field(cntnr).
  display hTTField:name format "x(20)" with
use-text.
end.
```

Now we have it. Let's create and populate a table by looking at all the fields. You don't need to do this when you have a table definition (like customer), but remember that you may want to draw this information from a repository.

```
find first _file where _file-name = "customer"
no-lock.
find first _field no-lock of _file where
_field-name = "name".
```

```
hTempTable:add-like-field(_field-name,_file-
name + "." + _field-name).
```

This syntax may seem to be a little confusing at first: the temp-table field needs a name (the first argument), and the source ("add-like") field must include the table name as well.

Another way to create a temp-table like a DB table would be as follows. Granted, this is way too much work for when you have the "create-like-table" method, but it may give you ideas for other situations.

```
for each _field of _file no-lock:
  hTempTable:add-like-field(_field-name,
  _file-name + "." + _field-name).
end.
```

Once you're done with the creation of all the fields, you need to add indices. The simplest way given what we have just accomplished, is to use the add-like-index syntax.

```
hTempTable:add-like-index(_field-name,
_field-name,_file-name).
```

Here are the methods for adding a custom index (i.e. one which is not in the DB metaschema).

```
hTempTable:add-new-
index("CountryCity",true,true).
hTempTable:add-index-
field("CountryCity","Country").
```

In order to add an index like one in the database, you could do something like the following:

```
for each _index of _file no-lock:
  hTempTable:add-like-index(_index-
name,_index-name,_file-name).
end.
```

OK, now you have built a temp-table. Before you can add data to it, you need to do two things. One is to signify that you have completed your work with the schema. (This is similar to the "commit schema" option in the dictionary.) Here is the syntax for this operation.

```
hTempTable:temp-table-prepare("TT" +
_file-name).
```

## TOOLS FOR PROGRESS



Visual Printing  
and Enhanced Reports

- ✓ uses Windows printer drivers
- ✓ data processing with 4GL
- ✓ incl. tool for graphic design
- ✓ no runtime licence cost
- ✓ supports bmp/jpg images
- ✓ embedding rtf-texts (font,...)
- ✓ generates pdf-files (email)
- ✓ supports WebSpeed /-Client



CASE-Extension for the  
Data Dictionary

- ✓ view Progress-DB structures
- ✓ create/update DBs directly
- ✓ reengineer Progress-DBs
- ✓ read/write Progress df-files
- ✓ compare/maintain versions
- ✓ supports areas
- ✓ autogenerates references
- ✓ print resizable ER-Diagrams



Project management,  
compiling, analyzing

- ✓ compiles project file lists
- ✓ includes compiler server
- ✓ also compiles in char-mode
- ✓ uses different Progress vers.
- ✓ compiles for different OS
- ✓ contains xref-analyze frontend
- ✓ shows db structure & content
- ✓ keeps track of project errors

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



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

The argument to this method gives the temp-table a name. Don't be fooled: you can't say "for each ttcustomer".

The next task is to create a (record) buffer to put data into. In other words, in the Progress DB, you don't put data directly into the schema. You create a record buffer, and then put data into that buffer, and put that buffer into the DB. This is the familiar code to do this with a static buffer.

```
do transaction:
  create customer.
  update name city country.
end.
```

To do this dynamically, you have to begin by creating a buffer for the temp-table. Not surprisingly, it involves a handle.

```
define variable hTTBuffer as handle no-undo.
hTTBuffer = hTempTable:default-buffer-handle.
```

Next, in order to place a value into the temp-table, we will define a variable, update the variable, create a blank temp table record (buffer), and put it into the temp table field. Of course, we need a handle for the field as well.

```
define variable hTTField as handle no-undo.
define variable vName like customer.name.

hTTField = hTTBuffer:buffer-field(
  _field-name). /* name */

update vName with frame a side-label three-d.

hTTBuffer:buffer-create.
hTTField:buffer-value = vName.
```

Here we are. We have a temp-table that we have fashioned in one way or another, and we need to get this information back into the database. We won't go into passing the table-handle parameter back and forth; hopefully you've read and understood Bruce Gruenbaum's article on this subject. We'll assume that the temp-table buffer handle which you have is now in a session which is connected to the database. What we want to be able to do at this point is to create a record buffer and put data into it (or find an existing record buffer and do the same).

Let's suppose that we have populated every single field in the ttcustomer temp-table. This makes life easier for understanding the process. We are going to have to create a handle for the customer table, and then copy the data from the temp-table to the customer table. As with

everything, there is more than one way to do this. The first way is very easy, but there is a gotcha. Here is the base code:

```
create buffer hTable for table _file-name.
/* customer */
do transaction:
  hTable:buffer-create.
  hTable:buffer-copy(hTTBuffer).
end.
```

Since the structure of the temp-table matches that of the DB table, you get all the fields across. If the two tables don't match, only those fields which do correspond are copied. However, this isn't the gotcha. The trick is, since we are creating a dynamic buffer, the trigger for the record creation in the table is not fired. So, you have to fire it yourself. The code is simple enough:

```
hField = hTable:buffer-field("custnum").
hField:buffer-value = next-
value(nextcustnum).
```

Now, if you want to go through the fields one at a time (again, an exercise to understand more about the process) here is what you would do:

```
do cntr = 1 to hTTBuffer:num-fields.
  assign
  hTTField = hTTBuffer:buffer-field(cntr)
  hField = hTable:buffer-field(hTTField:name)
  hField:buffer-value = hTTField:buffer-value.
end.
```

Actually, very straightforward: you loop through all the fields in the temp table, point a handle to the particular field, then use the name of that field to identify the corresponding field in the customer table, and put the value from the temp table into the database table. Don't forget to execute the "trigger" code after you do this to populate custnum!

So there you have it: soup to nuts. You start with a raw temp table, get data from the screen, and put it into the database. No surprise: if you want to put data from the database into the temp-table, you would take the code above and reverse the temp-table and buffer handle names. That is:

```
hTTBuffer:buffer-create.
hTTBuffer:buffer-copy(hTable).
```

Dynamic temp-tables are an amazing addition to the Progress repertoire. In future issues, you will discover all kinds of nifty ways to put them to use!

# Provide your programmers the essential PROGRESS<sup>®</sup> training nutrition they deserve!

For additional nutrition, try PGA Consulting

Now offering JARGON supplements to keep our clients thin!

**Extreme PROGRESS™**

Care and Feeding of SMARTOBJECTS 2™

Introduction to PROGRESS Programming™

Beyond V6™ (Jump-Start to V8 Character)

**PGA COURSES**  
The Training of Gurus

See our website for nutrition information  
[www.pgasmarts.com](http://www.pgasmarts.com)

Non-Programmer's PROGRESS™

for each customer      wait-for close of this-procedure  
prompt-for item-num  
enable all      display invoice-total  
name:screen-value      update order      output to printer

The graphic features a central orange box with the text 'PGA COURSES The Training of Gurus' and an illustration of a man with glasses at a computer. To the left is a blue pyramid with four levels of course titles. To the right is a blue bowl with a white interior containing various programming-related terms. A starburst and a pill-shaped callout provide additional information.

## Feeding Programmers for 14 Years

Paul Guggenheim & Associates, Inc. is an authorized Jargon Software reseller



**Paul Guggenheim & Associates, Inc.**

1788 Second St., Suite 201 Highland Park, IL 60035

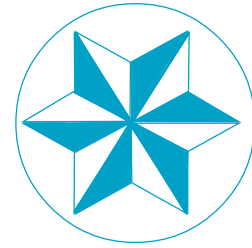
Phone: (847) 926-9800 Fax: (847) 926-9805

Web: [www.pgasmarts.com](http://www.pgasmarts.com)



## Table of Contents

<b>Universal Audit Trails</b>	
Paul Guggenheim .....	2
<b>Hacking the Internet with Progress</b>	
Ethan Tower .....	4
<b>DB Design Rules</b>	
Jop Kluis .....	6
<b>Session Super-Procedures</b>	
John Johnson .....	9
<b>Using Queries for a Find</b>	
Justin Martin .....	14
<b>ED For Windows</b>	
Neville Franks .....	16
<b>Dynamic Temp-tables 101</b>	
John Campbell .....	19



## White Star Software

Every man is a damned fool for  
at least five minutes every day.  
Wisdom consists in not  
exceeding the limit.

Elbert Hubbard