The Progress Electronic Magazine

In this issue:

Publisher's Statement:

In this issue, we have an article by Peter Bisset from Queensland, Australia about using dynamic programming techniques to create auditing trails for record changes. One of my pet peeves in many systems, is that they will have a few fields for who created a record, and who modified the record at what date and time – but once the record is changed again – the previous changes are lost! I find it a lot more useful to have a transactional log of changes that have occurred within the system over time. Especially in inventory and financial applications. One of the things Peter doesn't include, is the name of the program that did the change – my little contribution of should be's for the article!

Reach over 700 programmers and companies.

Your ad could be here!

Advertise in the E-Zine for $10.00 per issue!

To your success,
Scott Auge
Founder, Amduus Information Works, Inc.
sauge@amduus.com

Coding Article: Generic Record Change Logging Framework

Written by Peter Bisset pbisset@emergency.qld.gov.au

In our applications, all changes made by a user as part of a procedure that allows record updates are logged in a separate table. We record the before and after values in a character field along with other information that allows identification of who made the change, when, what part of the application, etc.

Prior to the introduction of dynamic queries the way we identified the changes was by comparing an exported text dump of the record before and after the update. This has worked perfectly well for many, many years, but has required one procedure for each dB in which we log the change. It also had the overhead of OS calls to write and read text files. We wanted to remove these things in our conversion to the web.

This article will present an outline of the technique we now employ in the hope that it might assist others with similar needs or prompt further development from the Progress community. It will not explain the usage of the various Progress constructs used – refer to the Progress documentation for this.

Requirements

- You must be running a client that supports dynamic queries (we are on 9.1C and 3.1C). The database must support the raw datatype and be a version accessible by the client.

- Your dB will need to have a table added with the following specification:

  Table name: `mraw`
  Field name: `raw-field`
  Field type: `raw`

  This table is used to hold pre-update copies of records.

- Include the pre-update code in your procedures
- Include the post-update code in your procedures

Figure 1 gives an outline of how to use this mechanism.

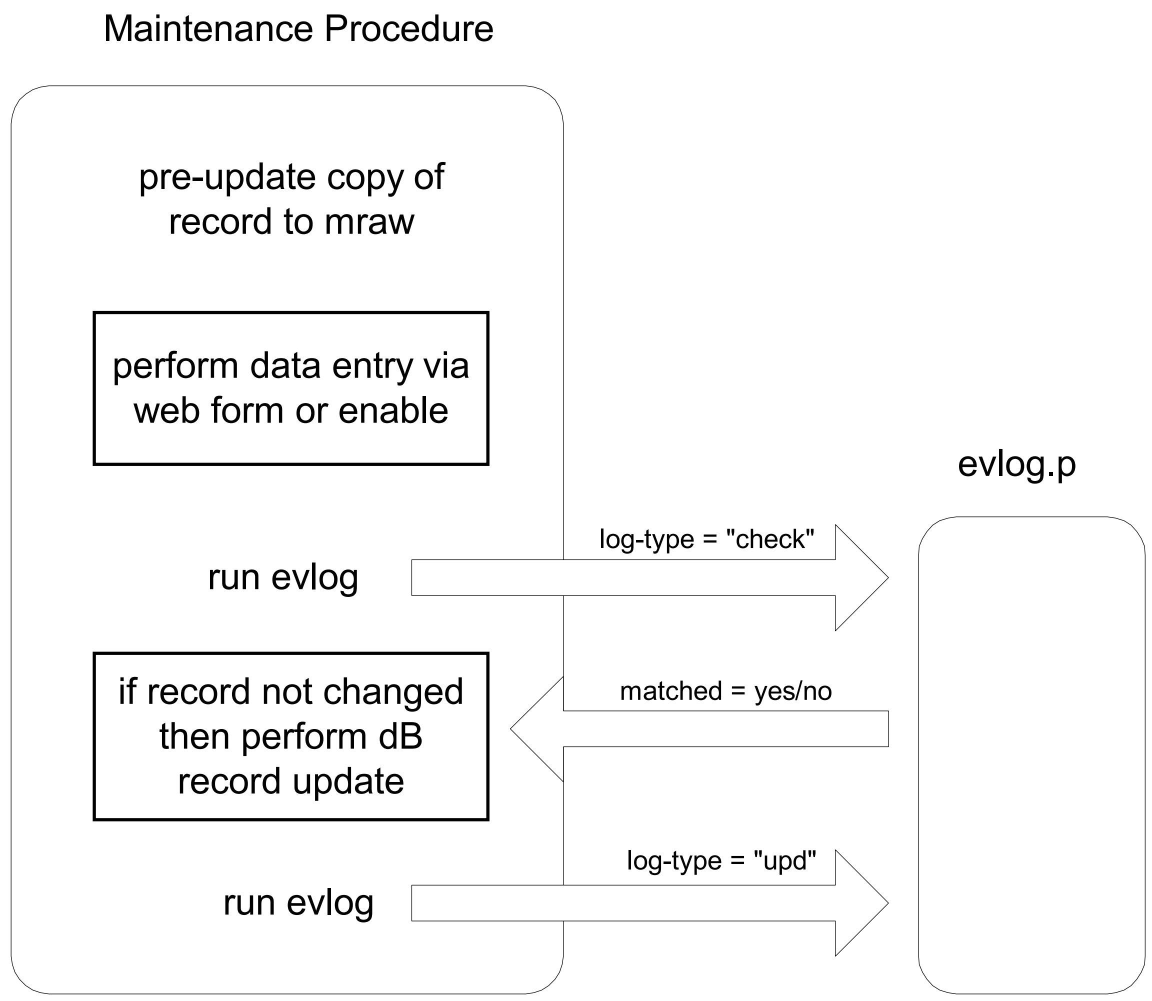

Maintenance Procedure
pre-update copy of record to mraw
perform data entry via web form or enable
run evlog
log-type = "check"
evlog.p
matched = yes/no
if record not changed then perform dB record update
log-type = "upd"
run evlog

Figure 1

Pre-update Process

In your update procedure you need to include the following lines **BEFORE** any changes are made to the record in question:

```
def var mrawid as rowid no-undo.

create mraw.
raw-transfer <table-name> to mraw.raw-data.
assign mrawid = rowid(mraw).
```

This code simply copies the whole `<table-name>` record to the raw field `raw-field` in an `mraw` record. The variable `mrawid` is used to pass the rowid of the `mraw` record to the event logging procedure. Obviously, you need to substitute `<table-name>` for the name of the table being modified.

Transaction Scoping Issue

How you use this code depends on how you scope your transactions. If you have a small transaction scope it should be possible to include the pre-update, update and post-update processes in a single transaction If the transaction scope would be unacceptably large then you need to verify that the record copied to `mraw` has not been changed by another user after the data was displayed on the screen. This is particularly relevant in a web environment. To get around this we call our event logging procedure with an option (`log-type = "check"`) to request confirmation that the `mraw` copy and the dB record are the same. If they are then we go ahead and process the update request, otherwise the user is advised that someone else has altered the record and that they need to start again/refresh/whatever.

By creating the `mraw` record in a separate transaction you have the possibility of it being orphaned should the procedure terminate unexpectedly. You should implement some sort of clean up process to removes these records at a later stage. How you do this depends on your operating environment and whether you can easily identify orphans. For example, a nightly shutdown for a backup is the ideal time to delete all mraw records since there cannot be any updates in progress.

The Gory Details of the Post-update Process

The event logging procedure takes at least four input parameters and should return a value, either as an output parameter or as a return-value indicating the result.

We pass in many more items, such as user name, a code to identify the type of event causing the change, the name of the table in which to record the change and fields to identify the record modified. Flavour to taste.

- **Parameters**

```
/* string to indicate type of procedure call - update, check, etc. */
```

```
def input parameter log-type     as char   no-undo.

/* The name of the table that is being logged */
def input parameter tname     as char   no-undo.

/* The row id of the record after the changes have been logged */
def input parameter record-id as rowid no-undo.

/* The row id of the mraw record that contains a snap shot of the record
 * taken when the update was started i.e. before any of the changes were
 * committed
 */
def input parameter mraw-id   as rowid no-undo.

/* The return value of the compare */
def  output parameter matched as log   no-undo.
```

- **Internal Variables**

```
/* A handle to mraw.raw-data so that we can retrieve the data */
def var mraw-field as handle no-undo.

/* A handle to the mraw table */
def var mraw-handle as widget-handle no-undo.

/* Handle to the updated record */
def var new-buffer as widget-handle no-undo.

/* Dynamic query to find updated record */
def var new-query  as widget-handle no-undo.

/* Temp table to hold old record */
def var ttable     as handle        no-undo.

/* Hold the name of the first field */
def var fname as char no-undo.

/* A counter */
def var i as i no-undo.

/*
 * Variables used to step through the fields in the two tables so
that
 * they can be compared.
 */
def var fh-new     as widget-handle extent 300 no-undo.
def var fh-old     as widget-handle extent 300 no-undo.

def var vlabel as char no-undo.
def var vold   as char no-undo.
def var vnew   as char no-undo.
def var arrayi as int  no-undo.
```

- **Steps**

The first step is to create a dynamic query to the table passed as a parameter in `tname` and to create the temp table `ttable` like the table `tname`.

```
/* Create a dynamic buffer to the table */
create buffer new-buffer for table tname.

/* Create and open a dynamic query into the table */
create query new-query.

/* Tell the dynamic query which table to use */
new-query:set-buffers(new-buffer).

/* Create the temp table to hold the old record */
create temp-table ttable.

/* Make the temp table have the same fields as the record we are
checking */
ttable:create-like(new-buffer).
ttable:temp-table-prepare("ttable").

/* Create a buffer to the temp table */
tt-handle = ttable:default-buffer-handle.
```

There is a bug in all current versions of Progress that prevents `raw-transfer` working on a temp table that has no index defined. We have such tables in our applications used as system parameter tables with one record. If you don't have this situation then you can skip the next section. If you have the problem you will receive the following run time errors when using `raw-transfer` (the field number and table name will vary depending on your definitions):

```
SYSTEM ERROR: Cannot read field 3 from record, not enough fields.
(450)
SYSTEM ERROR: Failed to extract field 3 from ttable record (table
2) with recid 129. (3191)
SYSTEM ERROR: bffld: nxtfld: scan past last field. (16)
```

As with more than a few Progress error messages, the message doesn't help you easily identify the reason. The only hint is the table name.

If you do have this situation, then you will need to do one of the following: 1) add an index to the affected tables; 2) use the workaround as described; 3) wait for Progress to issue a service pack.

We are on 9.1C, so no fix will be forthcoming, and 9.1D is not guaranteed to be fixed since there is a relatively simple workaround and it is not viewed as a high priority bug.

The workaround is to simply create an index on a field in the temp table. While it is easy enough to find a field using the facilities in the language for manipulating dynamic queries and make an index on it, you need to ensure that the field you find is not an array - they cannot be indexed. The simplest solution is to make a new index on the first available field. It doesn't matter if one already exists – it is simpler to just create one.

```
/*
 * To get around the 3191 bug - you must have an index on a temp-
table
 * to use raw transfer -we have to do the following
 */

assign fname = "".

/* Get the name of the first field that is not an array */
do i = 1 to tt-handle:num-fields:

    mraw-field = tt-handle:buffer-field(1).
    if mraw-field:extent > 0
    then next.

    fname = mraw-field:name.
    leave.
end.

/* If all the fields in the table are arrays then we cannot audit
log this record */

if fname = ""
then do:
    assign matched = true.
    message "All fields in the table:" tname "are arrays. Audit
logging not possible!".
    delete mraw.
    mraw-handle:buffer-release().
    return.
end.

/* Clean up */
ttable:clear().

/* Start again this time adding the index */
/* Create the temp table to hold the old record */
create temp-table ttable.
```

```
/* Make the temp table have the same fields as the record we are
checking */
ttable:create-like(new-buffer).

/* Add the index to the field we found – this needs to be unique
*/
ttable:add-new-index("XXXX1", false, false).
ttable:add-index-field("XXXX1", fname).

/* Reprepare the temp-table for use */
ttable:temp-table-prepare("ttable").

/* Create a buffer to the temp table */
tt-handle = ttable:default-buffer-handle.

/*
 * End of 3191 bug changes
 */
```

One point to note is that you must do `create` the temp-table twice because you can only do a `prepare` once the schema changes have finished. However, you can only look at the schema once it has been prepared. So, the first one gives us a schema we can look at to find a valid field. You must then delete it and recreate it so we can add the index, then do the `prepare`.

Now we have to retrieve the modified record using the rowid passed in with the parameter `record-id`.

```
/* Tell the query about the for each we need to use.  NB this
must be simple */
new-query:query-prepare("for each " +  new-buffer:name + " no-
lock").

/* Open the query */
new-query:query-open.

/* Reposition the query to the required row and retrieve the
resulting row */
new-buffer:find-by-rowid(record-id).
```

Now retrieve the pre-update copy stored in the `mraw` table using the row id passed in with the parameter `mraw-id` and place it into the raw variable `mraw-field`.

```
find first mraw
      where rowid(mraw) = mraw-id
      exclusive-lock no-error no-wait.
```

```
/* Get a handle to the table */
mraw-handle = buffer mraw:handle.

/* Retrieve the raw data */
mraw-field = mraw-handle:buffer-field(1).

/* Put the raw data into the tempory table table */
tt-handle:raw-transfer(false, mraw-field) no-error.
```

See if any fields were changed using the `buffer-compare` method. This shows how we check if the record has been changed after it was copied to `mraw`. If a value of "`check`" is passed in the parameter `log-type` or if this is an update and there weren't any changes then clean up and return. The logical variable `matched` is an output parameter.

```
matched = tt-handle:buffer-compare(new-buffer).

if log-type = "check" /* We just need to be sure that nothing has
changed */
or (matched and log-type = "upd") /* No changes to log */
then do:
       mraw-handle:buffer-release().
       new-buffer:buffer-release().
       new-query:query-close().
       delete object new-query.
       delete mraw.
       return.
end.
```

Now step through each field and compare it to the pre-update version.

The two array variables `fh-new` and `fh-old` are used to step through each field in a record and so need to be as large as the largest table in your database. 300 well and truly covers the largest table in our applications. We have not investigated defining this at run time. If a field is an array you have to iterate through each element.

```
/* Compare each of the fields to see what has changed */
do i = 1 to new-buffer:num-fields:
      fh-new[i] = new-buffer:buffer-field(i).
      fh-old[i] = tt-handle:buffer-field(i).

      if fh-new[i]:extent > 0 /* an array */
      then do arrayi = 1 to fh-new[i]:extent:
            /* If nothing has changed then skip it */
            if fh-new[i]:buffer-value(arrayi) = fh-old[i]:buffer-
      value(arrayi)
            then next.
            /* The label of the field */
            vlabel = trim(fh-new[i]:label) + ": ".

            /* The old value */
            vold = trim(fh-old[i]:buffer-value(arrayi)).

            /* The new value */
            vnew =  trim(fh-new[i]:buffer-value(arrayi)).

            /*
             * Create the audit log here - we record the values
        in vold,
             * vnew and vlabel so that the audit log shows before
        and
             * after values.
             * Use your own code here, depending on needs and
        design,
             */

        end.
        else do: /* not an array */

            /* If nothing has changed then skip it */
            if fh-new[i]:buffer-value = fh-old[i]:buffer-value
            then next.

            /* The label of the field */
            vlabel = trim(fh-new[i]:label) + ": ".

            /* The old value */
            vold = trim(fh-old[i]:buffer-value).

            /* The new value */
            vnew =  trim(fh-new[i]:buffer-value).

            /*
             * Create the audit log here - we record the values
        in vold,
             * vnew and vlabel so that the audit log shows before
        and
             * after values.
             * Use the same code as above.
             */
        end.

    end.
```

Finally, delete all objects before the end of the procedure is reached. Apart from deleting the `mraw` record, the rest is optional since Progress will remove the data structures once the procedure closes.

```
/* Clean up */
delete mraw.
mraw-handle:buffer-release().
new-buffer:buffer-release().
new-query:query-close().
delete object new-query.
```

Conclusion

We have found this process works extremely well with little, if any, measurable performance impact.

I hope that I have reproduced everything correctly. Any errors or omissions, probably in the area of missing variable definitions, are mine and are due solely to preparing the document for publication. The important code is all shown and it should not be difficult to implement this in your application. If you have any questions or comments then feel free to email me.

The hard work in figuring all this out (not for the faint of heart) was done by our contract programmers:

Kylie Leonard - Adaptive Enterprises (kleonard@adaptive-enterprises.com.au)
Wendy Keating - r-code design (wkeating@ozemail.com.au)

About the author: Peter Bisset is a systems analyst/administrator/programmer with the Department of Emergency Services in Queensland, Australia. He has been using Progress for more years than he cares to remember (14 at last count) and still thinks it is a pleasure to work with.

Publishing Information:

Scott Auge publishes this document. I can be reached at sauge@amduus.com.

Amduus Information Works, Inc. assists in the publication of this document by providing an internet connection and web site for redistribution:

Amduus Information Works, Inc.
1818 Briarwood
Flint, MI 48507
http://www.amduus.com

Other Progress Publications Available:

This document focuses on the programming of Progress applications. If you wish to read more business oriented articles about Progress, be sure to see the Profile's magazine put out by Progress software:

http://www.progress.com/profiles/

There are other documents/links available at www.peg.com.

Products Available From Amduus:

Amduus Information Works, Inc. is a Progress reseller and ASPen partner. We primarily develop UNIX/Linux based applications for the web. We also perform integration of Progress applications through such languages and tools as MQ Series, C, and C++.

Amduus provides support for the following applications: Blue Diamond, Denkh, Denkh HTML Reporter, Red Arrow Portal (CMS), Survey Express and other software.

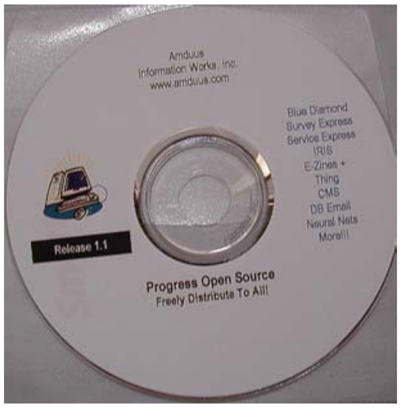
Article Submission Information:

Please submit your article in Microsoft Word format or as text. Please include a little bit about yourself for the About the Author paragraph.

Looking for technical articles, *marketing Progress* articles, articles about books relevant to programming/software industry, white papers, etc.

Order Form for Progress Open Source CD-ROM

COUPON 001A



This is an offer for the CD-ROM at lower than list savings!
This is a great way to support the E-Zine too!

Mail this form to:
Amduus Information Works, Inc.
1818 Briarwood
Flint, MI 48507

Please send ______ copies of the Open Source CD-ROM at $25.00 per disk to:

Name ______________________________

Company ______________________________

Address ______________________________

City ______________________________

State ______________ Country ______________

Zip ______________________________

Please make your checks/money orders out to: Amduus Information Works, Inc. Cash works too!
This offer only valid in the United States of America.

The CD-ROM includes (all source code included):

- Blue Diamond/IRIS – Webspeed alternatives
- Survey Express – easily create text templates of surveys and then have the program generate the web pages automatically
- Service Express – Web based Help Desk.
- The Progress E-Zines, books on learning to program in Webspeed (PDF/Word/HTML)
- Denkh HTML Reporter – web based report writer
- CMS – a web content management system
- DB Email – Use pop3 to download emails into a Progress database
- Neural Networks – experiments in spam recognition and text message classification
- Denkh – create PDF file reports for Webspeed/UNIX CHUI!
- More!