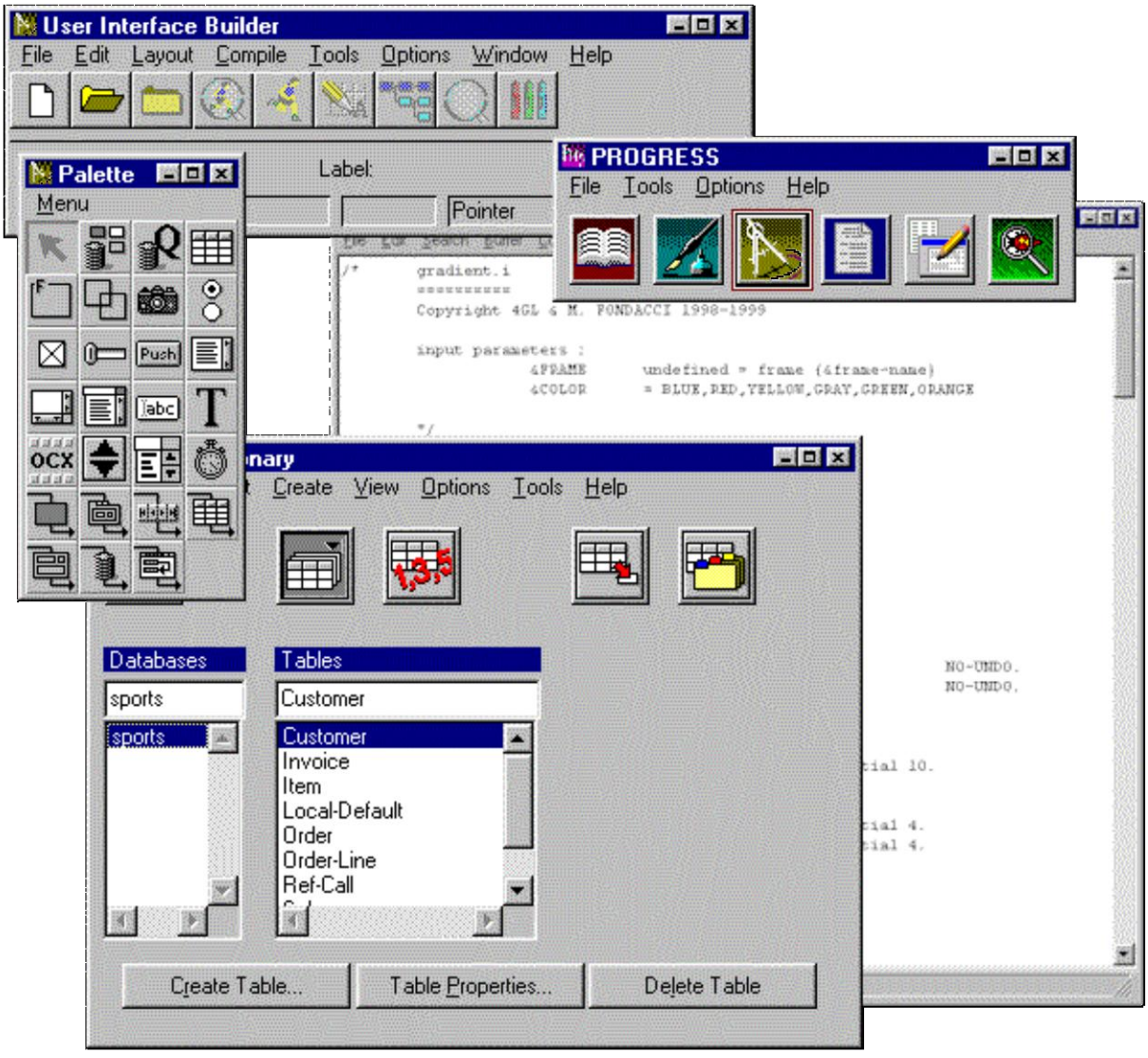
Tips for PROGRESS



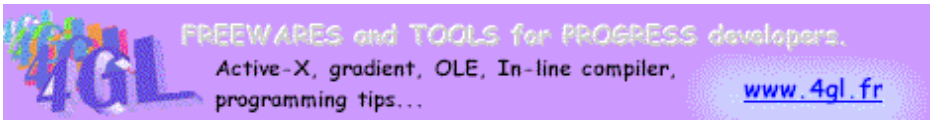
The goal of this publication is to inform you of some tips that you can use in your PROGRESS applications.

This document will (try to…) summarize some of advanced features (not documented or unknown) that you can't find in PROGRESS examples or demos.

You can submit to this publication any tip that you find useful for the PROGRESS community at support@4gl.fr

Any help or information can be obtained at info@4gl.fr

www.4gl.fr



Tip#1 : locking outside of a transaction

(18-jan-2000)
submitted by 4GL

One of the more amazing tips that this one.

You have heard and read many times in PROGRESS manuals that your PROGRESS development system is a *single-transaction-at-a-time* system…
And you must deal with that.

However, sometimes it would be nice to get another solution…

For a development, we were confronted to a problem:

> *In a V8 smart-application, we had to lock some records to enter in our update process. That would signal to other users that these records were locked and allows the program to direct the process on another folder. In case of not locked record, we would acquire a lock on this record(s).*
> ***We did not want to start a transaction because of side effects** on our application (and of the logic of some Smart-objects : ie no-transaction panel).*

The first idea that we had at this time was to create a lock file in the database and then find the good locking record each time we want a record… and update it… Many objections, the first one was: what does it occur if the user deads? (we know that WINDOWS users never hangs, but our customer was not sure of that…). Is the record released? How to automate the delete of records in the lock file?

The better solution was to leave PROGRESS lock this record for us.
… then the second idea was to use PROGRESS logic. You remember that:

```
DO TRANSACTION :
    FIND customer WHERE … … EXCLUSIVE-LOCK.
    END.
```

… leaves a share-lock on the record beyond the scope of the transaction. That was a good idea, but not sufficient. In fact, the next time we had to read another *customer* record, the lock was released (without our agreement). This lock was not persistent… not persistent?

Why do no set the FIND EXCLUSIVE as **PERSISTENT ?**

We have encapsulated a FIND in a persistent procedure which locks the desired record(s)

…and this works fine!

How to do ?

A persistent procedure like :

FIND customer WHERE … … exclusive-lock.

called by :

Run <MyPersistentLock>.p **PERSISTENT.**

locks the customer record while no transaction is started in the calling program.
How to unlock this record? Simply **DELETE** the persistent PROCEDURE. In case of a program crash, all locks are smoothly released…

This sample screen shot shows the output of a PROMON utility. Two users are active, each of them is locking 2 records (customer and order).

Record Locking Table:

Usr Name	Chain #	Rec-id	Lock Flags
3 Marcel	REC 20	2510	SHR
2 Paul	REC 30	2437	SHR
3 Marcel	REC 68	1894	SHR
2 Paul	REC 70	1896	SHR

RETURN - repeat, U - continue uninterrupted, Q - quit:

*And at the **same** time, the PROMON transaction control says that there is no active transaction.*

Transaction Control:

Usr Name	Trans	Login	Time	R-comm?	Limbo?	Crd?	Coord	Crd-task

RETURN - repeat, U - continue uninterrupted, Q - quit:

And if you test the **TRANSACTION** status in your program, you must agree that there is no active transaction. However any other process has to wait to read this record: he is locked.

There are many benefits of using this method…

The demo for Tip#1 :

You'll find in the Tips kit two programs to easily demonstrate this behavior : **lock.p** (the persistent procedure) and **locktest.w** (the main locking program).
Create a sports database and run a server on it (proserve sports). Then run two sessions on it and try with locktest.w the different situations of record locks. An editor window will show you the current status of each task (TRANSACTION or NOT).
You can use freely this procedures for your own usage and can modify lock.p to meet your needs. Take a look to the source code.

Other usages of the tip :

- in case of multiple updates, you can easily lock multiple records at the same time and *reserve* them for future use. Simply run as many time as needed the lock.p program. Each persistent program has its own context. But don't forget to delete all of them at the end of program!
- use it if you want all the benefits of no-transaction update panels without the message "*This record has been changed by another user…"*

Tip#2 : get the current working directory

(18-jan-2000)
submitted by 4GL

It's sometimes useful to know what's the current directory your application is working on. The main situation where you need that if when working with external programs such as API, WORD…

The **SEARCH** function returns a *relative-to-propath* path and the other process can't access this file. The file name returned by search could be ".\documents\mydoc.doc". When you send that to Word, this file is unknown if the working directory is not the same as the PROGRESS directory.

You can use…

to place the current directory in your **Current-Dir** variable.

```
Def var pDir            as MEMPTR.
Def var Current-Dir     as CHAR NO-UNDO.

Set-Size(pDir) = 256.              /* get a 256 bytes memory area */

Run getCurrentDirectoryA(255, pDir).

Current-Dir = get-String(pdir, 1).

Set-Size(pDir) = 0.                /* release the memory area */
```

```
PROCEDURE GetCurrentDirectoryA EXTERNAL "KERNEL32" :
  DEFINE INPUT  PARAMETER lpSize    AS LONG.
  DEFINE INPUT  PARAMETER lpBuffer  AS MEMPTR.
END PROCEDURE.
```

24-feb-2000 :

A **new and better suggestion** submitted by PROGRESS users (Carl Verbiest , Michael Rüsweg-Gilbert…) :

```
file-info:file-name = ".".
Dir = file-info:full-pathname.
```

…provides the same result within the 4GL language… method that you can use with every file-name to get his absolute path.

Forget the first one !

Tip#3 : Font manipulation – characters justification

Do you remember the time we just needed to count spaces for correct alignement of field, text…
With the proportional WINDOWS fonts, all is gone away and texts are always displayed at the left side of the widget… It would be sometimes useful to right justify or center a text into a given widget (titles, info…). We propose in this tip a "right-justify and a center justification" procedure logic. Use it as a baseline to develop some other tool or understand how it works.

If you already have dowloaded ***adjustlabel.i***, you have seen that we have a such method to set the field label at the right place. Let try to generalize that.

What's the problem ?

- With proportional fonts, each character has its own width, different from the other : "I" is smaller than "W". This is not the case with a fixed font where `"I"` and `"W"` have a same length,
- If we want to justify a text, we need to know how much *fill* characters we can insert to obtain the correct width,
- Finally, of course, we need to know the widget width where we want to justify our text.

PROGRESS point of view:

- The width of the text is given by:
  FONT-TABLE:GET-TEXT-WIDTH-PIXELS(<Text-Value>, <Font number>).
  Gives the number of pixels of a given text for a font.
- The width of a widget is:
  <WIDGET>:WIDTH-PIXELS
- The third element to know is the width of the fill character:
  FONT-TABLE:GET-TEXT-WIDTH-PIXELS(<fill-char>, <Font number>).
- The widget font may be well known or default to **? font**. In this case the parent font applies. But if the containing frame has a default font too, we must use the PROGRESS default font.
- If the widget is an enabled fill-in, we must keep track of the **borders**. In the example below, we assume that the fill-in is **view-as TEXT**, that's the most common situation.
- The size of widget can be insufficient for deal with the literal value, so no justifying can occur.

Example : To demonstrate the use of this procedure, try **justify.w** in the working directory.

```
PROCEDURE Justify:
DEF INPUT PARAMETER hWidget              as HANDLE no-undo.
DEF INPUT PARAMETER TextToJustify        as CHAR   no-undo.
DEF INPUT PARAMETER HowTo                as CHAR   no-undo.

Def var     hParent         as Handle    no-UNDO.
Def var     fontNumber      as Int       no-UNDO  initial ?.
Def var     textWidth       as int       no-UNDO.
Def var     fillWidth       as int       no-UNDO.

/*     Get the widget font number */

If hWidget:FONT = ? then do :
        hParent = hWidget:PARENT.
        Do while(valid-Handle(hParent)
                 and (NOT can-query(hParent, "FONT")
                      OR hParent:FONT = ?)) :
                 hParent = hParent:PARENT.
                 end.

        If valid-handle(hParent)
        and hParent:FONT <> ? then       /* The parent has a font value */
                    fontNumber = hParent:FONT.
        end.
else
        fontNumber = hWidget:FONT.

/*     Assume that fill char is space */
&SCOP  Sep     " "

/*     Get width of text and fill char */
if fontNumber = ? then
    ASSIGN
            textWidth  = FONT-TABLE:GET-TEXT-WIDTH-PIXELS(TextToJustify)
            fillWidth  = FONT-TABLE:GET-TEXT-WIDTH-PIXELS({&Sep}).
else
            ASSIGN textWidth  = FONT-TABLE:GET-TEXT-WIDTH-PIXELS(TextToJustify, fontNumber).
                   fillWidth  = FONT-TABLE:GET-TEXT-WIDTH-PIXELS({&Sep},        fontNumber).

/* If HowTo = "C" then a centered text will be displayed,
   otherwise a right justify occurs.
*/
HWidget:SCREEN-VALUE =
                        Fill({&Sep},
                             MAX(
                                 INT(TRUNCATE( (hWidget:WIDTH-PIXELS - textWidth) /
fillWidth
                                                       / (If HowTo = "C" then 2 else 1)
                                   , 0  )
                                     )
                                 , 0)
                            )
                                          + TextToJustify.

END PROCEDURE.
```

Tip#4 : Loading of selection-list, combo-boxes or radio-sets (V8 and V9 with field-pairs).

One of the most current tasks we have to do is the loading of selection-lists, combo-boxes or radio-sets. The use of ADD-LAST or ADD-FIRST method is expensive in terms of CPU charge, so we use the **LIST-ITEMS** attribute. You will find below some useful includes to manipulate combo, selection or radio-sets.

```
/*          load.i

        Load a combo or a selection list
        We use a TAB character to delimite the fields.
        Private-Data is used to store the rowid value of the record.
*/
                DO :            /* load.i can be embedded */

&SCOP       I             {&SEQUENCE} /* So we can use this include more than once */

Def VAR     L-{&I}        AS CHAR             NO-UNDO.
Def VAR     P-{&I}        AS CHAR             NO-UNDO.

&IF "{&OBJECT}" = "" &THEN            /* Widget and Object are allowed */
    &SCOP OBJECT    {&WIDGET}
    &ENDIF

&IF DEFINED( UNDEFINED ) > 0 &THEN    /* Undefined value */
                    L-{&I} = "{&UNDEFINED}".
                    &ENDIF

FOR EACH {&FILE} NO-LOCK
                            WHERE {&WHERE}
                            {&BREAK} :
        ASSIGN               L-{&I} =   L-{&I}  +  chr(9) + {&FIELD}.
                             P-{&I} =   P-{&I}  + "," + String(  Rowid( {&File} ) ).
        END.

&IF DEFINED( UNDEFINED ) = 0        &THEN
        ASSIGN      {&OBJECT}:DELIMITER                     = chr(9)
                            {&OBJECT}:LIST-ITEMS            = SUBSTRING(L-{&I}, 2)
                            {&OBJECT}:PRIVATE-DATA          = SUBSTRING(P-{&I}, 2).
&ELSE
        ASSIGN      {&OBJECT}:DELIMITER                     = chr(9)
                            {&OBJECT}:LIST-ITEMS            = L-{&I}
                            {&OBJECT}:PRIVATE-DATA          = P-{&I}.
&ENDIF

  /* first entry by default */

{&OBJECT}:SCREEN-VALUE = ENTRY(1, {&OBJECT}:LIST-ITEMS, {&OBJECT}:DELIMITER).

                            END.
```

Typical usage :

```
        { load.i        &FILE       = customer
                        &OBJECT     = mySelectionList
                        &WHERE      = "cust-num > 100"
                        &BREAK      = "BY customer.Name"
                        &FIELD      = customer.Name
                        &UNDEFINED  = "I don't know."     /* If necessary */
                        }
```

The problem now is to display the correct value (in local-display-fields for example) or to set the reference value (in local-assign).

To set the value, we must read the related record. Use the following:

```
/*  reference.i

    Find the corresponding record for a SELECTION or COMBO widget
*/

&SCOP   I   {&SEQUENCE}

DEF VAR R-{&I}      as ROWID NO-UNDO.

&IF "{&TABLE}" = "" &Then

    &SCOP   Table   {&FILE}
    &ENDIF

&IF "{&OBJECT}" = "" &Then
    &SCOP   OBJECT   {&WIDGET}
    &ENDIF

  Assign Frame {&FRAME-NAME} {&OBJECT}.

  Find {&Table} where ROWID( {&Table} ) =
              TO-ROWID(
                          ENTRY(
                                LOOKUP({&OBJECT}, {&OBJECT}:LIST-ITEMS, {&OBJECT}:DELIMITER),
                                {&OBJECT}:PRIVATE-DATA
                                )
                      ) NO-LOCK NO-ERROR.
```

This procedure reads the related record (assumed that this combo/selection was previously loaded by a load.i and that there is no duplicate value in the list)

```
{ reference.i     &FILE       = state
                  &OBJECT     = myState
                  }
If available state then
        Customer.state = State.State.
... ...
```

The display of field can be done by:

```
/*  display.i

    Find the related record and display the correct value in the list
*/

&If "{&TABLE}" = "" &THEN
    &SCOP TABLE {&FILE}
    &ENDIF

  Find {&Table} where {&WHERE} NO-LOCK NO-ERROR.

  If available {&Table} then do WITH FRAME {&Frame-Name} :
           {&OBJECT}:SCREEN-VALUE =
                       {&Object}:ENTRY( MAX(LOOKUP( String(ROWID( {&TABLE} )),
                       {&OBJECT}:PRIVATE-DATA ), 1) ).
           END.
```

Example :

```
{ reference.i    &TABLE      = state
                 &WHERE      = "of customer"
                 &OBJECT     = myState
                 }
```

What happens with the field pairs?

The field pairs attribute allows your program to display and set values directly from the widget.
The **display.i** and **reference.i** are not needed in this case.
For loading a field pair widget, use loadpair.i :

```
/*          loadpair.i

                Load combo or selection-list (V9+)
*/

                        DO :

&SCOP        I                {&SEQUENCE}
Def VAR      L-{&I}        AS CHAR             NO-UNDO.

&IF "{&OBJECT}" = "" &THEN
    &SCOP OBJECT     {&WIDGET}
    &ENDIF


For each {&FILE} NO-LOCK
                 WHERE {&WHERE}
                 {&BREAK} :
        L-{&I} = L-{&I} + chr(9) + {&FIELD} + chr(9) + {&KEY}.
        END.

&IF DEFINED( UNDEFINED ) > 0 &THEN
           L-{&I} = "{&UNDEFINED}" + chr(9) + " " + L-{&I}.          /* Empty value */
&else
           L-{&I} = substring( L-{&I}, 2 ).
&endif


ASSIGN  {&object}:DELIMITER       = chr(9)
        {&object}:LIST-ITEM-PAIRS = L-{&I}        NO-ERROR.

                        END.
```

In this case, the key value is returned by an ASSIGN and a DISPLAY statement displays the correct value without anything else…

Load of radio-set :

```
/*          loadradio.i

                Load values in a radio-set
*/

                    DO :

&SCOP       I               {&SEQUENCE}
Def VAR     L-{&I}       AS CHAR            NO-UNDO.


&IF "{&OBJECT}" = "" &THEN
    &SCOP OBJECT    {&WIDGET}
    &ENDIF

{&OBJECT}:auto-resize = true.               /* Don't forget to resize */

For each {&FILE} NO-LOCK
                 WHERE {&WHERE}
                 {&BREAK} :
        L-{&I} = L-{&I} + chr(9) + {&Field} + chr(9) + {&KEY}.
        END.

&IF DEFINED( UNDEFINED ) > 0 &THEN
           L-{&I} = "< Unknow value >" + chr(9) + L-{&I}.
&else
           L-{&I} = substring( L-{&I}, 2 ).
&endif

ASSIGN  {&object}:DELIMITER     = chr(9)
        {&object}:RADIO-BUTTONS = L-{&I}        NO-ERROR.

                    END.
```

The nature of radio-set allows doing a direct ASSIGN like ASSIGN RADIO-SET-1. So the **{&KEY}** gives the corresponding value and **{&FIELD}** the displayed value.

Typical usage :

```
DO WITH FRAME {&FRAME-NAME} :  /* if necessary */

    { loadradio.i     &FILE   = customer
                      &OBJECT       = myRadioSet
                &WHERE        = "cust-num < 100"
                      &BREAK        = "by customer.name"
                      &KEY          = string(Customer.Cust-Num)
                      &FIELD= customer.Name
                      }

    END.
```

Tip#5 : Scanning from PROGRESS.

The use of a scanner can be easily implemented in a PROGRESS application. The first thing to know is that the standard protocol to do that is TWAIN. TWAIN is a industry standard, powerful but complex. A more easy to use interface is EZTWAIN (Easy TWAIN), a freeware developed by Hugh (Spike) McLarty.
You can download **EZTWAIN 1.x** freeware at www.dosadi.com/eztwain1.htm
You will find in the tips directory a sample program **EZTwain.w** a program to scan, view and save a document.
Sample program: the EZTWAIN logic.

```
Run GetParent( {&WINDOW-NAME}:hwnd, Output h_Window).

set-size(Z)                       = length(FileName) + 1.  /* Store the filename */
put-string(Z,1)                   = FileName.
put-byte(Z, 1 + length(FileName)) = 0.

B-Image:LOAD-IMAGE( ? ).

Run    TWAIN_OpenSourceManager( h_Window, output h).

Run    TWAIN_SetHideUI( HideUi ).                 /* Scanner dialog-Box */

Run    TWAIN_AcquireNative(
                           h_Window,
                           TWAIN_ANYTYPE,         /* Color scale */
                           output DIB
                         ).

If DIB = 0 then do :
             Message "SCANNING ERROR." view-as alert-box ERROR.
             RETURN.
             end.

Run    TWAIN_WriteNativeToFilename(       /* Store into file */
                           DIB,
                           Z,
                           output R
                           ).

If R <> 0 then
       Message
            "Return code from 'writeNativeToFile = " R view-as alert-box WARNING.

set-size(Z) = 0.                          /* Free PROGRESS memory */

Run    TWAIN_FreeNative( DIB ).           /* And free DIB memory */
```

Tip#6 : Using RAS connections from PROGRESS.

(21-apr-2000)
submitted by 4GL

If you want to use remote connections from within your application, you will be confronted to the use of **RAS WINDOWS API**.

How to initialize a connection, list active or available connections, close and terminate an opened one ?

You'll find a demo program called dRas.w that you can use as a template for your own usage.

NOTE : you can see in the dial procedure that you must provide user name and password for connecting. It's the way to inhibit a connection out of your application.
In this case, don't specify any user name or password in the standard connection icon and the user will not be able to connect without using your application…

INDEX