

Base Language - Bug #11975

ERROR-STATUS issues due to a wrong silent worker logic

10/12/2026 06:58 AM - Teodor Gorghe

Status:	New	Start date:	
Priority:	Low	Due date:	
Assignee:		% Done:	0%
Category:		Estimated time:	0.00 hour
Target version:		reviewer:	
billable:	No	production:	No
vendor_id:	GCD	env_name:	
case_num:		topics:	
version_reported:			
version_resolved:			
Description			

History

#1 - 10/12/2026 07:11 AM - Teodor Gorghe

Testcase:

```
define output parameter pRes as character no-undo.

define variable cIn as character no-undo.
define variable x as integer no-undo.

function f returns integer (input p as character):
  x = 1 no-error.
  cIn = (if error-status:num-messages = 0 then "-" else string(error-status:get-number(1))).
  return 0.
end function.

procedure p:
  define input parameter p as character no-undo.
  x = 1 no-error.
  cIn = (if error-status:num-messages = 0 then "-" else string(error-status:get-number(1))).
end procedure.

run p (string(100, "99")) no-error.
pRes = "run: inside=" + cIn + " statement="
      + (if error-status:num-messages = 0 then "-" else string(error-status:get-number(1))).
// FWD: run: inside=74 statement=-; function: inside=74 statement=-
// Progress 4GL: run: inside=- statement=74; function: inside=- statement=74

x = f(string(100, "99")) no-error.
pRes = pRes + "; function: inside=" + cIn + " statement="
      + (if error-status:num-messages = 0 then "-" else string(error-status:get-number(1))).
// FWD: run: inside=74 statement=-; function: inside=74 statement=-
// Progress 4GL: run: inside=- statement=74; function: inside=- statement=74
```

To understand this case, notice string(100, "99") for the procedure and function call. That throws the error **74**, but it doesn't abend right away due to NO-ERROR.

So, that gets set up into the function/procedure error pending list (ErrorManager WorkArea pendingErrorStatus), and when it reaches x = 1 no-error, it looks into the pending list, and puts the **74** error raised from the input parameter, into the ERROR-STATUS of this instruction.

x = 1 is a valid instruction because x is an integer, and the expected behavior there is ERROR-STATUS:NUM-MESSAGES=0. FWD returns 1, which is the deferred **74** error.

Since that error is "consumed", the ERROR-STATUS which is expected to be thrown by the RUN instruction, now it is not being set.

I think the solution here is to be a stack of pending error list (eg. nestedSilent, but I think there might be a much simpler solution.

#2 - 10/12/2026 07:22 AM - Constantin Asofiei

Teodor, please see this:

```
procedure proc0.  
  def input param p as char.  
  
    message "here" .  
    message error-status:error error-status:get-message(1). // no  
end.  
  
do on error undo, leave:  
  message string(100, "99").  
  message "this must be shown: not a raised error condition".  
end.  
  
run proc0(string(100, "99")) no-error.  
message error-status:error error-status:get-message(1). // erro 74
```

The problem I think is that an ERROR condition is not being raised.

What this may mean: we have this code in TM.pushScope around line 4796:

```
// the error manager needs a special notification (it can't be a  
// scopeable because it is used on both the client and the server)  
if (topLevel)  
{  
  // remember if silent mode was enabled or not  
  blk.suppressError = wa.em.isSilent();  
  
  // disable silent mode if it was enabled because silent mode on a  
  // RUN statement or for a function call is something that only  
  // suppresses errors during the dispatching itself and NOT during the  
  // execution of the called code  
  if (blk.suppressError)  
  {  
    wa.em.setSilent(false);  
  }  
}
```

Here I think we need to backup (and restore in Tm.popScope) any errors existing before the call.

#3 - 10/12/2026 08:57 AM - Teodor Gorghe

More testcases, with different causes, that one you might already know:

```
routine-level on error undo, throw.

define variable c as character no-undo.
define frame f with size 20 by 2.

c = string(100, "99") no-error.
message error-status:error error-status:get-number(1).
// Progress 4GL: yes 74
// FWD:          no 74

do with frame f:
  c = string(100, "99") no-error.
  message error-status:error error-status:get-number(1).
  // Progress 4GL: yes 74
  // FWD:          no 74
end.

routine-level on error undo, throw.

define variable c as character no-undo.
define frame f with size 20 by 2.

procedure top:
  c = string(100, "99").
  message "top: after".
end procedure.

procedure nested:
  do with frame f:
    c = string(100, "99").
    message "nested: after".
  end.
end procedure.

run top no-error.
message "run top: " error-status:error error-status:get-number(1).
// Progress 4GL: run top:    yes 74
// FWD:          run top:    yes 74

run nested no-error.
message "run nested:" error-status:error error-status:get-number(1).
// Progress 4GL: run nested: yes 74 (the 74 is thrown, "nested: after" is not shown)
// FWD:          the 74 is shown, "nested: after", then run nested: no 0

routine-level on error undo, throw.

define frame f with size 20 by 2.

session:set-numeric-format("1", ".") no-error.
message error-status:error error-status:get-number(1).
// Progress 4GL: yes 8903
// FWD:          yes 8903

do with frame f:
  session:set-numeric-format("1", ".") no-error.
  message error-status:error error-status:get-number(1).
  // Progress 4GL: yes 8903
  // FWD:          no 8903
end.
```

So, most of the issues is related with how do with frame f block is treated during runtime.

In my opinion, the semantics are about the same like a simple progress DO block (which gets converted into a simple java block).

The case of do with frame f, gets converted into a doBlock(...), which creates a new block into the stack. That tricks

BlockManager.mustManageLegacyError to think that this shouldn't throw the error because it isn't a top level block (which routine-level targets to raise the error).

Effect can be seen from the code, all logic from do with frame f which raises warnings, doesn't get thrown as an `ErrorConditionException`, due to this issue.