Base Language - Bug #5651

conditions raised from a finally block

09/03/2021 11:54 AM - Constantin Asofiei

Status:	WIP	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	20%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD	**version:**	

Description

Related issues:

Related to Base Language - Feature #4373: finish core OO 4GL support	**New**
Related to Base Language - Bug #5743: finally block: finalizable processing	**New**

History

#3 - 09/03/2021 12:00 PM - Constantin Asofiei

TransactionManager.processFinallyBlock is not protected at all for conditions raised from within the finally block. Any condition will abend the processing of:

```
iterateWorker(WorkArea)
popScope(WorkArea)
processRetry(WorkArea, BlockDefinition)
```

and leaves the context in a corrupted state.

I saw this in a scenario where there an ERROR condition 12327, Cannot delete a BY-REFERENCE PARAMETER dataset or table in the called procedure is raised from within the finally block, which 'abends' the popScope processing (as a side note, I need to investigate this - if it should be raised or not - but regardless this does not affect this finally abend problem in FWD).

My first instinct would be to execute the Block.fini call from a BlockManager.doBlock (so all block processing is in place).

I need to write some testcases and see if some conditions are 'consumed' and others are honored from the finally block.

#4 - 09/03/2021 01:58 PM - Constantin Asofiei

- Status changed from New to WIP

Looks like the following are consumed from a finally block:

```
undo, leave
undo, retry
undo, next
next
retry
```

and the following are allowed from a finally block:

```
undo, return
undo, return error
undo, throw
leave
return
return error
quit
stop
ERROR
```

My first problem is that I don't see how to distinguish between undo, leave and leave, when is caught via LeaveUnwindException. I think maybe these statements are no-op if they are executed in the finally, and they target the owner block.

Now, the second problem is that it seems for some conditions the finally block is not executed (like quit, stop). I need to expand this, and I think depends if the owner block has an ON clause or not.

Greg, a change I think is most likely needed is to move everything after the processFinallyBlock in a finally, so that is executed regardless if there is a condition thrown or not.

#5 - 09/03/2021 02:32 PM - Greg Shah

> My first problem is that I don't see how to distinguish between undo, leave and leave, when is caught via LeaveUnwindException. I think maybe these statements are no-op if they are executed in the finally, and they target the owner block.

We could create UndoLeaveUnwindException which is a subclass of LeaveUnwindException. Normal cases would continue to work, but this special case could be detected.

> I think depends if the owner block has an ON clause or not.

For DO blocks, only DO TRANSACTION and DO ON ERROR support FINALLY. I don't think DO ON ENDKEY will work, for example.

> Greg, a change I think is most likely needed is to move everything after the processFinallyBlock in a finally, so that is executed regardless if there is a condition thrown or not.

Agreed. Please put a new try/finally in like this from processRetry():

```
...
      try
      {
         // my excellent comments :)
         processFinallyBlock(wa, blk);
      }

      finally
      {
         wa.handleDeferredError();

         // check the interrupted status and throw the stop condition if needed
         wa.honorStopCondition();
      }
```

I don't want to move the following logic into an existing finally block. The reason: this would change the behavior in the case where something is raised inside that following logic. Things like handleDeferredError() are where they are for a reason.

#6 - 09/03/2021 02:32 PM - Greg Shah

Make sure to put your testcases in testcases/uast/finally/ and update the readme there.

#7 - 09/04/2021 05:49 AM - Constantin Asofiei

Greg Shah wrote:

> Agreed. Please put a new try/finally in like this from processRetry():
>
> [...]
>
> I don't want to move the following logic into an existing finally block. The reason: this would change the behavior in the case where something is raised inside that following logic. Things like handleDeferredError() are where they are for a reason.

Yes, that was my intent for moving the code.

OTOH, just this change stabilizes FWD and the original abend in the customer application is solved.

I suggest leaving this on the 'back burner' while I work on some more high priority tasks.

#8 - 09/06/2021 07:27 AM - Greg Shah

I think you can go ahead with the change.

#9 - 09/06/2021 09:55 AM - Constantin Asofiei

I just found that another customer application is using RETURN "<val>" from the finally block.

I'll right a report to find which kind of tokens are used from a finally block, and fix those at this time.

#10 - 09/06/2021 03:49 PM - Greg Shah

I think we need to handle all the cases now. We can't know what minor edit will cause new cases to be used tomorrow which are not there today. I don't want to have to debug this later to find out that something is missing.

#11 - 09/15/2021 02:43 PM - Constantin Asofiei

Greg, there is an issue which is kind of tricky. For looping blocks, the finally can not be processed at the popScope() - this is because for example at the finally you can have a NEXT and at the block you can have a LEAVE - the NEXT in the FINALLY will override the LEAVE:

```
def var k as int.
def var i as int.

i = 0.
k = 0.
repeat i = 1 to 100  :
  k = k + 1.
  leave.
  finally:
     k = k + 1.
     next .
  end.
end.
message k i.
```

If I let the finally block to be executed in popScope (after the LEAVE), then there is no way for the loop to resume processing (and for FINALLY to override conditions/unwind from the main block).

For this reason I need to move the blockSetup inside the inner 'do ... while' loop for i.e. BlockManager.coreLoop:

```
         loop:
         while (true)
         {
            boolean runBlockSetup = true;

            do
            {
               try
               {
                  if (runBlockSetup)
                  {
                     wa.tm.blockSetup();
                     runBlockSetup = false;
                  }
                  ...
                  if (!needsIterate(wa.tm, to, expr))
                  {
                     break loop;
                  }

                  processBody(wa, block, on);
               }
               ... // all the other catch blocks
               catch (Throwable thr)
               {
                  wa.tm.abnormalEnd(thr);
               }
            }
            while (wa.tm.needsRetry());

            if (wa.tm.isBreakPending())
            {
               break loop;
            }
         }
```

needsIterate had to be moved **after** blockSetup, so that it will be executed after the FINALLY block (and not before it), for an iteration.

My problem is I don't see how safe is that the tx-related logic is executed twice at the end of the loop, once for the blockSetup (for the last FINALLY call) and once in popScope.

As a side note, I need to duplicate my tests with other block types (currently I used only REPEAT, and had to refactor coreLoop a lot...).

#12 - 09/15/2021 04:33 PM - Greg Shah

Ughhh. Supporting NEXT in a finally which overrides a prior LEAVE is an awful "feature" and a terrible idea. What could they have been thinking when they implemented that? Or perhaps they didn't think about it at all and it is just one more poorly conceived implementation quirk.

I'm still thinking about this, but I don't think it is right to use blockSetup() for the popScope() execution of finally. It will break the transaction processing at a minimum. The other changes to make it work also are quite dangerous. For example, needsIterate() cannot be executed inside the needsRetry() inner loop. The retry is independently processed and needsIterate() must never re-execute (there can be side effects from the WHILE expression). Even if we avoid the re-execution, I think there are problems with the ordering. And these changes also seem likely to break the control flow for how we process exceptions.

There must be a better way. I'll think about this some more.

#13 - 09/15/2021 04:43 PM - Constantin Asofiei

And something I didn't expect: same suite ran in ChUI and GUI (OE 11.6.3) gives different results in ~5% of cases. They all seem to be part of some cases where a raised ERROR condition is involved.

#14 - 09/15/2021 05:30 PM - Greg Shah

I am willing to postpone the implementation of NEXT in a FINALLY. We can put code in to detect this at conversion and render a WARNING.

Are there other cases that would drive a radical rework of our block processing? If not, let's create a separate task and work it later.

#15 - 09/17/2021 03:27 PM - Constantin Asofiei

Greg, to answer your question, yes, the unwind can be overridden in the FINALLY (via LEAVE/NEXT/RETRY or even another condition). The problem is only the owner block (of the finally) decides to unwind, and the FINALLY block overrides this and decides to continue the loop (as in this case FINALLY will be executed from popScope).

Leaving the 'unwind override' from the FINALLY aside, the solution was to catch any unwind/condition from the FINALLY, deffer it and throw it in processBody - so it will be treat it in the main code for this block.

But, the FINALLY block needs to be executed before the loop expression (which can be whatever, a WHILE <expr> or var to expr()). So I need the coreLoop to look like this:

```
         boolean firstRun = true;
         boolean secondRun = false;

         loop:
         while (true)
         {
            if (firstRun)
            {
               if (!needsIterate(wa.tm, to, expr))
               {
                  break loop;
               }

               wa.tm.blockSetup();
               firstRun = false;
            }

            if (secondRun)
            {
               wa.tm.blockSetup();

               if (wa.deferredFinallyThrow == null && !needsIterate(wa.tm, to, expr))
               {
                  break loop;
               }
            }
            secondRun = true;
```

BTW, the iteration expression has the same behavior as the FINALLY block - no TX is seen at that level.

But I think you are right, this change still seems dangerous, as the blockSetup will be executed one additional time (for iteration mode) than the current way. I'll comment this firstrun/secondRun code and leave it behind, to at least have some 'breadcrumbs' for when we get back to it.

I've tested with 'before and after' and there are 59 failures which should be related to 'unwind override' and additional 67 failures if I comment out the firstRun/secondRun code (this is from a 819 test suite).

#16 - 09/27/2021 09:33 AM - Constantin Asofiei

3821c/13001 adds the try/finally protection for TM.processFinallyBlock.

#17 - 09/28/2021 04:04 PM - Greg Shah

Code Review Task Branch 3821c Revision 13001

No objections.

#18 - 09/28/2021 04:05 PM - Greg Shah

Are there any known customer issues that are caused by the remaining issues? If not, we will defer those items.

#19 - 09/28/2021 04:07 PM - Greg Shah

- Related to Feature #4373: finish core OO 4GL support added

#20 - 09/29/2021 06:22 AM - Constantin Asofiei

Greg Shah wrote:

> Are there any known customer issues that are caused by the remaining issues? If not, we will defer those items.

Yes, there are QUIT, STOP, RETURN and some other cases where an ERROR condition can be thrown. I need to stabilize these changes and finish the FINALLY for the top-level block.

#22 - 11/11/2021 11:22 AM - Greg Shah

- Related to Bug #5743: finally block: finalizable processing added

#23 - 11/15/2022 09:51 AM - Greg Shah

- % Done changed from 0 to 20