Base Language - Feature #4347

add runtime support for STOP-AFTER block option

10/15/2019 09:28 AM - Constantin Asofiei

Status:	WIP	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Eduard Soltan	**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**vendor_id:**	GCD

Description

Related issues:

Related to Base Language - Feature #3751: implement support for OO 4GL and st...	**Closed**
Related to Base Language - Feature #4373: finish core OO 4GL support	**New**

History

#1 - 10/15/2019 09:28 AM - Constantin Asofiei

The STOP-AFTER option requires runtime - see the BlockManager.stopAfter stubs.

#2 - 10/15/2019 09:29 AM - Constantin Asofiei

- Related to Feature #3751: implement support for OO 4GL and structured error handling added

#3 - 10/30/2019 09:55 AM - Greg Shah

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

#4 - 05/31/2022 10:14 AM - Constantin Asofiei

We need tests to see what happens if there are nested blocks with STOP-AFTER - as I think the STOP should target the block with the STOP-AFTER clause which expired. And this needs to cleanup any other nested STOP-AFTER stopper threads.

#5 - 05/31/2022 10:19 AM - Greg Shah

I would like to avoid having a thread for each STOP-AFTER block. I think we need to sort the timeouts and process them all on 1 thread.

#6 - 05/31/2022 10:22 AM - Constantin Asofiei

Greg Shah wrote:

> I would like to avoid having a thread for each STOP-AFTER block. I think we need to sort the timeouts and process them all on 1 thread.

OK, makes sense, I think it can be done, we just need to figure out the cleanup (when block terminates, cleanup all nested blocks when targeting an outer block, etc).

#8 - 04/22/2024 03:12 PM - Greg Shah

- Assignee set to Eduard Soltan

#9 - 04/22/2024 03:12 PM - Greg Shah

- Status changed from New to WIP

#10 - 04/24/2024 04:19 AM - Alexandru Lungu

Eduard and I had a chat yesterday on this task an I took some "thinking" time on my own on this matter:

#4347-6 is not a trivial thing from my POV - in fact, it is the hardest part of this task. I think the 4GL statements are atomic, so in 4GL the functionality is something like:

```
do stop after 3:
// if 3 seconds already passed, do stop.
message "a".
// if 3 seconds already passed, do stop.
message "b".
// if 3 seconds already passed, do stop.
// ...
end.
```

To replicate this in Java, we need all **converted** statements to check if there was a time-out or not, so it can raise stop. To do that, we can either:

- Cherry-pick some places in FWD run-time: BlockManager is the obvious one, but there may be others like AbstractQuery.next (converted statement) or BDT.assign (converted statement), etc.
- Use AOP and annotate the methods used exclusively from the conversion side.

No need to use parallel thread. For each session, we can keep a stack of deadlines and the tightest deadline (current-time + after clause). For the hotspots we choose, we need to check if the tightest deadline was passed, so we raise the STOP condition. We can do that in BlockManager for each session context.

Either way, we still need to handle, as Constantin mentioned:

- Nested blocks with STOP AFTER. I don't think this is necessarily hard - if a certain time-out occurred, simply raise STOP, wherever the execution stands (in a top-block or a nested-block).
- Edge cases:
  - Database (schema or session) triggers.
  - UI statements including WAIT-FOR or UI triggers.

Most important from my POV:

- What happens when a long running IO operations happens (including database interaction). Is this the case in #8573?
- If so, we can't cancel a DB query AFAIK (or maybe we can with some specific JDBC settings). How is 4GL dealing with this? Even if 4GL is waiting for the query to end, in FWD we don't have cursor walking, but special queries to prefetch data (like ProgressiveResults), which are not that granular.

My end-game here: if the urgency of #4347 is due to #8573 and #8573 is timing out due to a long running query, maybe we should optimize that first, right?
If we need to handle this **now**, I suggest doing some detective work for the edge cases and IO operations. After, we can deliver a more relaxed implementation where only BlockManager is honoring the STOP AFTER.

#11 - 04/24/2024 04:30 AM - Constantin Asofiei

Alexandru, the STOP AFTER needs to use a similar mechanism we have for CTRL-C support (which actually raises a STOP condition in ChUI). This will interrupt the target Conversation thread once the timeout exceeded and it will perform a rollback/rollup until the targeted DO STOP AFTER block is reached, via a STOP condition.

If you run this test in ChuI:

```
do on quit undo, leave:
   do on stop undo, leave:
     do stop-after 1:
       pause 3.
     end.
     message "no stop".
  end.
  message "no quit".
end.

message "done".
```

you will see that a STOP condition is being raised for both CTRL-C and the STOP-AFTER timeout.

#12 - 04/24/2024 08:34 AM - Greg Shah

To replicate this in Java, we need all **converted** statements to check if there was a time-out or not, so it can raise stop. To do that, we can either:

- Cherry-pick some places in FWD run-time: BlockManager is the obvious one, but there may be others like AbstractQuery.next (converted statement) or BDT.assign (converted statement), etc.
- Use AOP and annotate the methods used exclusively from the conversion side.

I disagree here. We don't want to do this. It will make a mess of the code and will also add new processing that is a performance drag.

No need to use parallel thread. For each session, we can keep a stack of deadlines and the tightest deadline (current-time + after clause).

Actually, a timer thread is exactly what is needed. And as Constantin notes, we have the interruption mechanism already available to generate the STOP. This will work for all cases (base lang, UI, database, processing on the client or the server). See Control.interrupt().

This timer thread can:

- Be a single thread for all sessions. In this case, we must save some context local state to identify the session to be interrupted along with the scheduled deadline. OR
- Be a single thread per session that handles all timers for that context. In this case, the thread itself would have the context needed to know which session is being interrupted.

  - Database (schema or session) triggers.
  - UI statements including WAIT-FOR or UI triggers.

Most important from my POV:

- What happens when a long running IO operations happens (including database interaction). Is this the case in #8573?
- If so, we can't cancel a DB query AFAIK (or maybe we can with some specific JDBC settings). How is 4GL dealing with this? Even if 4GL is waiting for the query to end, in FWD we don't have cursor walking, but special queries to prefetch data (like ProgressiveResults), which are not that granular.

The Control.interrupt() works for all of these cases.