Base Language - Feature #6650

improve performance of Scopeable notification in TM.processScopeNotifications

08/01/2022 12:54 PM - Constantin Asofiei

Status:	WIP	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Constantin Asofiei	**% Done:**	90%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**vendor_id:**	GCD
Description			

History

#1 - 08/01/2022 12:58 PM - Constantin Asofiei

I've tried a simple looping block executed (via Java FOR statement) 100k times and the times are with/without executing TM.processScopeNotifications, in milliseconds, for the first 6 runs, starting with the fresh FWD server:

```
283/153
209/93
178/82
186/86
177/88
186/85
```

The test was this:

```
def var i as int.
def var j as int.

repeat i = 1 to 10:
   j = j + 1.
end.
```

Modified like this in the externalProcedure body:

```
   @LegacySignature(type = Type.MAIN, name = "blocktest.p")
   public void execute()
   {
      externalProcedure(Blocktest.this, new Block((Body) () -> 
      {
         long l1 = System.currentTimeMillis();
         for (int k = 0; k < 100000; k++)
         {
            ToClause toClause0 = new ToClause(i, 1, 10);

            repeatTo("loopLabel0", toClause0, new Block((Body) () -> 
            {
               j.assign(plus(j, 1));
            }));
         }

         long l2 = System.currentTimeMillis();
         System.out.println(l2 - l1);
      }));
   }
```

I don't expect the impact to be as great as above (as I completely removed TM.processScopeNotifications from executing, in my test), but I really think FWD wastes a lot of time with notifying all scopeables on each and every block... and for a simple API test in a large app, I see 70k calls for processScopeNotifications.

The main reasoning why FWD may waste lots of time in notifying the Scopeables on each block entry and exit:

- why notify unnamed streams processing if the block isn't requiring it?
- why notify buffer processing if the block is not using buffers (think OO default getter/setters)?
- why notify frame processing if there is no frame usage?
- etc...

#2 - 08/16/2022 09:01 AM - Constantin Asofiei

The complex part of the task is to determine where and when to register a Scopeable with the next (or current) block. There are these Scopeable classes:

```
AccumulatorManager
BufferManager
ChangeBroker
DatabaseTriggerManager
DataSetManager
LogicalTerminal
SharedVariableManager
UnnamedStreams
AbstractParameter$WorkArea
ArrayAssigner$WorkArea
ControlFLowOps$WorkArea
ObjectOps$WorkArea
ProcedureManager$WorkArea
```

For some cases (like UnnamedStreams and SharedVariableManager), the decision is easy - if the context-local WorkArea instance is resolved, then this is a valid usage, and registration must happen.

For other cases, I need some help with any ideas on what should trigger the registration:

- LogicalTerminal - my main problem is with the ENDKEY interactions state; any details are appreciated. The other scoped state I think is related only to frames and menus.
- DatabaseTriggerManager - the problem here is that the context-local instance is referenced by TriggerTracker, which is created for each RecordBuffer instance. So I can't just assume 'any context-local usage must trigger registration'. Instead, I think the registration hook for the scopeable will be in the static methods, called explicitly from the converted code. Ovidiu: please comment, I'm interested if schema triggers are involved (when they are executed) with DatabaseTriggerManager.
- I have not looked at BufferManager and ChangeBroker in-depth yet. My assumption is that any buffer/table parameter, buffer definition, buffer openScope() call, dynamic buffer create, queries?, would need to force registration. Any ideas are welcome.
- DataSetManager - the registration hooks I think should be at dataset definition, parameter, dynamic create. The problem here is that in some cases it needs to be registered in the next top-level block (like dataset definition), and in other cases it needs to be registered in current block (like dynamic dataset create).

The others I think I can solve myself.

#3 - 08/16/2022 02:54 PM - Greg Shah

> For some cases (like UnnamedStreams and SharedVariableManager), the decision is easy - if the context-local WorkArea instance is resolved, then this is a valid usage, and registration must happen.

What are examples of when the context local instance does not resolve for these classes?

> LogicalTerminal - my main problem is with the ENDKEY interactions state; any details are appreciated. The other scoped state I think is related only to frames and menus.

Hmm. I was hoping to never think about this again. ;)

Please recall that the 4GL has an event called END-ERROR. When this event is raised, it will become either an END-KEY condition or an ERROR condition.

The interactions counter has its primary purpose to determine which condition should be raised when an END-ERROR event occurs. We make this decision in ThinClient.processCondition(). Based on this we either throw an EndConditionException or an ErrorConditionException. Since this determination is made on the client in response to low level key processing, we use ServerState to synchronize the interactions counter to the client side.

We have some related processing of ErrorConditionException on the server side which needs to be differentiated by the same counter tracked in the TM. This differentiation of error processing must be aware of whether the ErrorConditionException was caused by an END-ERROR event. You can see this in a catch block in BlockManager.editingBlock() and in BlockManager.processCondition(). Toe manage this we must update the TM interactions counter at the same time as we manage the ServerState counter.

All the rest of the processing of the interactions counter is simply to manage its value for those usage locations in TC and BM.

In regard to the Scopeable processing, the LT.scopeStart() must save off the current interactions counter onto the blocks deque and then resets the ServerState counter to 0 for the new block. The stack is used because we must sometimes "rollup" this value when we exit the block. When we are in rollup mode, we accumulate the interactions of nested blocks as we exit back into the enclosing blocks. When not in rollup mode, we just simply restore the original interactions counter. These behaviors are dependent upon whether the block has the END-KEY property and whether the TM.isEnclosingRollup() is true.

#4 - 08/16/2022 02:55 PM - Greg Shah

Ovidiu and Eric: Please review the persistence related Scopeables in #6650-2 and see if you can offer some guidance.

#5 - 08/16/2022 03:25 PM - Ovidiu Maxiniuc

Constantin Asofiei wrote:

> - DatabaseTriggerManager - the problem here is that the context-local instance is referenced by TriggerTracker, which is created for each RecordBuffer instance. So I can't just assume 'any context-local usage must trigger registration'. Instead, I think the registration hook for the scopeable will be in the static methods, called explicitly from the converted code. Ovidiu: please comment, I'm interested if schema triggers are involved (when they are executed) with DatabaseTriggerManager.

The DatabaseTriggerManager requires to be notified by scopable events, not because of schema triggers but because of

- the session triggers added during a scope - they are alive only in that scope and must be dropped when the scope end;
- the disable trigger statements which also have effect only in current scope - when the scope ends, any disabled schema trigger becomes back enabled.

OTOH, I think that setting up the TriggerTracker for each RecordBuffer instance is an overkill. I think it is logic that the TriggerTracker to be in 1:1 relation to the table instead - but I need to run some tests to confirm this. This is probably not covered by this task: even if the number of these object will decrease, I do not expect to do by much because usually only the default buffer is used in 90% of ABL code.

> - I have not looked at BufferManager and ChangeBroker in-depth yet. My assumption is that any buffer/table parameter, buffer definition, buffer openScope() call, dynamic buffer create, queries?, would need to force registration. Any ideas are welcome.

Can the procedure (for which they are parameters) notify them when the scope ends? Is that a better solution from performance POV?

> - DataSetManager - the registration hooks I think should be at dataset definition, parameter, dynamic create. The problem here is that in some cases it needs to be registered in the next top-level block (like dataset definition), and in other cases it needs to be registered in current block (like dynamic dataset create).

Is there another solution to keep track of the objects of this kind in relation to the scopes? Can we register it/them with the (external) procedure and call notifications from there, if they exist?

#7 - 08/29/2022 02:53 PM - Constantin Asofiei

I can not find a good way to limit BufferManager/ChangeBroker. I have changes which sync them (so they are always notified together), but they will always be registered when BufferManager.isImportantBlockTransition() is true.

Unfortunately, this includes almost all blocks. A big problem here is that BufferManager is responsible for tx management, too - we should extract this in a separate class for scope processing, as I can't find how the dictionaries kept in BufferManager are being used by beginTx, TxWrapper, and others. And I think my failing attempts to 'slow down' the BufferManager registration are all because it needs isImportantBlockTransition() for all the tx management.

At this time, automated testing of an app shows 5-10% improvement of overall time. But BufferManager scope notifications still consumes ~5%. I'll post the changes for review (at the task) tomorrow, and I'll do some more testing, especially for the UI (LogicalTerminal) part.

My plan is for BufferManager tx scope notification to be reworked in another class, but after I get current changes released.

#8 - 09/01/2022 09:37 AM - Constantin Asofiei

- % Done changed from 0 to 90

- File scopeable_6650_patch_20220901a.patch added

- Status changed from New to WIP

- Assignee set to Constantin Asofiei

The changes are attach. Greg/Eric: please review and let me know if you see something dangerous.

The patch is on top of 3821c/14216.

#9 - 09/01/2022 09:37 AM - Constantin Asofiei

You can apply the patch using:

```
patch -p1 < scopeable_6650_patch_20220901a.patch
```

#10 - 09/01/2022 11:48 AM - Constantin Asofiei

Eric: if you can give me some hints at extracting BufferManager's tx scope support, I can give it a try. To me, it looks like it would be enough to extract the beginTx and endTx in a separate scopeable, which is executed for isImportantBlockTransition().

The other scopeable state can be explicitly handled when buffers are used/defined in certain cases (I have the changes for this, but they are not included in the patch).

Something I forgot to mention - there are some methods in BufferManager which are never called, and I marked them with Deprecated. I would like to remove these.

#11 - 09/02/2022 06:22 AM - Eric Faulhaber

Constantin Asofiei wrote:

> Eric: if you can give me some hints at extracting BufferManager's tx scope support, I can give it a try. To me, it looks like it would be enough to extract the beginTx and endTx in a separate scopeable, which is executed for isImportantBlockTransition().

This seems right. As you refactor, take note that a database transaction is not started immediately at beginTx, unless the corresponding database is "active" (i.e., has at least one open buffer). If it is not active, the transaction start is deferred until a buffer is opened during that application-level transaction.

At least that's how it is supposed to work. I was looking at this code just now, and the reference counting for the open buffers which determines whether a database is active doesn't look right. It increments when a buffer scope is opened and when a buffer is initialized, and it decrements for every open buffer when a scope is deleted or finished. Plus it decrements when a dynamic buffer is deregistered. I'm not sure this is balanced.

> The other scopeable state can be explicitly handled when buffers are used/defined in certain cases (I have the changes for this, but they are not included in the patch).

This has changed significantly since I first wrote it, and I don't fully understand all the procedure features that are now integrated with it. It would be great if we could do something more efficient than all the collection copying work that is going on today.

> Something I forgot to mention - there are some methods in BufferManager which are never called, and I marked them with Deprecated. I would like to remove these.

I did not see any methods marked @Deprecated in 3821c. There is a comment in BufferManager.resolvePendingBufferClasses that it may be deprecated, but that is all. Do you mean in 6129a?

#12 - 09/02/2022 06:27 AM - Greg Shah

Code Review scopeable_6650_patch_20220901a.patch

Overall, I think the changes are quite good.

My concern is mostly that the changes cause a more tight coupling between various parts of the system, sometimes in circular ways where the older approach was a bit more abstract and mostly had one-way dependencies.

1. Tighter coupling:

- The Scopable interface now encodes knowledge of all implementers of the interface. I think the ScopeId enum should be external to this class, to make this a little cleaner.
- StandardServer.invoke() now encodes logic that was once internal to the LogicalTerminal. Perhaps we should move the registration of the "cleaner" Finalizable back into a helper method in LT so that it is a little more encapsulated.

2. Circular dependencies. The TM now has hard links "back" to LogicalTerminal, BufferManager and AccumulatorManager. These circular dependencies make the code more tightly coupled and make the TM less clean.

3. Registration in a "deregistration location". In LT.deregisterFrame() and in LT.applyChanges() for the frame dereg processing we have a call to TM.registerBlockScopeable(). At a minimum, this makes the code more confusing. Perhaps we need a comment in those locations to explain why this is needed and why it does not cause any unbalanced processing.

4. We will need to ensure that the LT usage in TM is not used in any call path that can be reached by code running in the "ultimate" server-side resources mode (where there is no client). For example, the usage of LT.isInteractive() is not safe if there is no LT. It is currently protected by wa.lt != null but this kind of protection will be required everywhere the LT is used from the TM. It is more fragile (and less clear) than if these dependencies did not exist.

#13 - 09/02/2022 06:39 AM - Constantin Asofiei

Eric Faulhaber wrote:

> > Something I forgot to mention - there are some methods in BufferManager which are never called, and I marked them with Deprecated. I would like to remove these.
>
> I did not see any methods marked @Deprecated in 3821c. There is a comment in BufferManager.resolvePendingBufferClasses that it may be deprecated, but that is all. Do you mean in 6129a?

I mean in the patch I posted, in BufferManager there are new methods marked Deprecated, like isTransactionAt.

#14 - 09/02/2022 07:37 AM - Constantin Asofiei

Greg Shah wrote:

> Code Review scopeable_6650_patch_20220901a.patch

Overall, I think the changes are quite good.

My concern is mostly that the changes cause a more tight coupling between various parts of the system, sometimes in circular ways where the older approach was a bit more abstract and mostly had one-way dependencies.

1. Tighter coupling:

- The Scopable interface now encodes knowledge of all implementers of the interface. I think the ScopeId enum should be external to this class, to make this a little cleaner.
- StandardServer.invoke() now encodes logic that was once internal to the LogicalTerminal. Perhaps we should move the registration of the "cleaner" Finalizable back into a helper method in LT so that it is a little more encapsulated.

Fixed.

2. Circular dependencies. The TM now has hard links "back" to LogicalTerminal, BufferManager and AccumulatorManager. These circular dependencies make the code more tightly coupled and make the TM less clean.

I'm thinking how to solve this.

3. Registration in a "deregistration location". In LT.deregisterFrame() and in LT.applyChanges() for the frame dereg processing we have a call to TM.registerBlockScopeable(). At a minimum, this makes the code more confusing. Perhaps we need a comment in those locations to explain why this is needed and why it does not cause any unbalanced processing.

I'll improve documentation. The location does not matter if is a "deregistration" - it matters that it affects the frames.

4. We will need to ensure that the LT usage in TM is not used in any call path that can be reached by code running in the "ultimate" server-side resources mode (where there is no client). For example, the usage of LT.isInteractive() is not safe if there is no LT. It is currently protected by wa.lt != null but this kind of protection will be required everywhere the LT is used from the TM. It is more fragile (and less clear) than if these dependencies did not exist.

AFAIK even if the FWD client process no long exists, LT will still exist, but TC will be on the server-side. If we remove LT, this would mean frames are no longer being used at all - which we can't guarantee. But I'll see how I can improve this.

#15 - 09/02/2022 10:47 AM - Greg Shah

> LT will still exist, but TC will be on the server-side. If we remove LT, this would mean frames are no longer being used at all - which we can't guarantee. But I'll see how I can improve this.

If we are getting rid of the client process, then my assumption has been there would be no UI at all (no TC and no LT). On the other hand, there are definitely batch mode/appserver use cases for redirected terminal, but making the UI work on the server side is a bigger change. I was not planning to take that on right now.

#16 - 09/05/2022 04:22 AM - Greg Shah

I think you can go ahead and check in your first pass changes from #6650-14. Or did you plan to check it in with the rest of the changes?

#17 - 09/06/2022 08:53 AM - Constantin Asofiei

There is an inherent feature of ScopedDictionary, where 'global' scope is used: this is not something created by the ScopedDictionary instance itself, but instead the user of this dictionary must be careful to add the global scope as the first one in the stack... this was a problem when refactoring BufferManager, I need to finish checking all other scopeables for 'global scope' usage.

Another part was that BufferManager.openBuffers and RecordBuffer.activeScopeDepth are somehow dependent on the 'real app stack size'. There is this code in RecordBuffer.validate (which gets executed following the same logic as current 3821c, i.e. beginTx will call txHelper.registerCommitAt for each isImportantBlockTransition.

```
         int currentScope = bufferManager.getOpenBufferScopes();
         boolean exitingActiveScope = currentScope == activeScopeDepth && (!global || !worldScope);
```

I had a failure in this code, as 'currentScope' and 'activeScopeDepth' were equal and exitingActiveScope was true, thus a flush was forced (when it should not have been). The app stack at the moment of this flush had a few other blocks on it, so previously this worked fine (as 'getOpenBufferScopes' was following the app stack). I've fixed this by forcing BufferManager to push scopes also for a block where a buffer is registered as dirty, or a buffer is initialized. This solved the problem.

Overall, the full patch seems to have a 10~13% improvement over the results with 3821c/14112 ran on Jul. 28.

#18 - 09/06/2022 02:25 PM - Constantin Asofiei

- File scopeable_6650_patch_20220906a.patch added

The latest patch is attached, built on top of 3821c/14225. I need to finish testing with other apps, there is an issue with static class variables I need to fix.

In the mean time, Eric, please review, I've extracted TxWrapper from BufferManager in a standalone class, and this has scopeable support for anything needed related to transaction management.

Greg: I haven't yet decoupled TransactionManager from the direct references to BufferManager/LogicalTerminal, I'll do this as a low priority... I have an idea how to do it, but I don't want to delay this patch anymore.

#19 - 09/07/2022 01:25 PM - Greg Shah

> Greg: I haven't yet decoupled TransactionManager from the direct references to BufferManager/LogicalTerminal, I'll do this as a low priority... I have an idea how to do it, but I don't want to delay this patch anymore.

Absolutely. I did not expect this to block check in.

#20 - 09/07/2022 10:38 PM - Eric Faulhaber

Code Review scopeable_6650_patch_20220901a.patch:

I did not find a revised patch with the #6650-14 changes after Greg's code review, so I reviewed the original, primarily the persistence-related changes. Everything seems ok. I'm not as familiar with the data set or database trigger manager implementations, but the changes to these looked pretty straightforward.

I think the block at TransactionManager:4393 needs to be protected with a wa.bm != null check, in case of non-server-side use.

#21 - 09/08/2022 10:13 AM - Eric Faulhaber

Eric Faulhaber wrote:

> Code Review scopeable_6650_patch_20220901a.patch:
>
> I did not find a revised patch with the #6650-14 changes after Greg's code review, [...]

Ugh, this is why I shouldn't do code review late at night. I don't know how I didn't see scopeable_6650_patch_20220906a.patch. I need to review this...

#22 - 09/09/2022 02:11 AM - Constantin Asofiei

- File ca_upd_20220909a_3821c_14233.zip added

Eric, I'm attaching the changes directly, please apply them over 3821c/14233 and review the p2j.persist part. Thanks.

#23 - 09/10/2022 04:24 AM - Eric Faulhaber

I've reviewed ca_upd_20220909a_3821c_14233.zip update (the persistence portions, mostly).

I think overall this looks like a good update. I like how much less work is being done in BufferManager scope transitions.

In TM.registerScopeable, you removed the safety code to prevent ConcurrentModificationException. Are you sure no registration can ever occur while processing Scopeable events?

In TM.pushScope, dereferencing wa.bm at line 4412 still needs to be protected by a null check for client-side use.

#24 - 09/23/2022 10:12 AM - Constantin Asofiei

- File p2j_oo_object_local_vars.txt added

- File ca_upd_20220923a_3821c_14253.zip added

The latest update is attached. All projects look OK.

To fix the p2j.oo code, I've analyzed it using a crude spoon Java source processor (https://spoon.gforge.inria.fr/); all TypeFactory usage related to extent or object types are fixed. But, there are lots of other local object variables defined within the BlockManager API, and their reference directly assigned to some expression (like a method call or CAST). For now, the ObjectOps scopeable support is automatically registered whenever some Java method in a *BaseObject* sub-type is being invoked (see the BlockManager.topLevelBlock and functionBlock changes).

All these object local vars need to be reviewed and refactored to the standard FWD conversion rules for variables. The report is attached. Once this is fixed, the BlockManager.topLevelBlock and functionBlock changes to automatically register *BaseObject* calls need to be removed.

#25 - 09/27/2022 12:07 AM - Eric Faulhaber

The persistence-related changes in ca_upd_20220923a_3821c_14253.zip look good to me.

#26 - 09/27/2022 03:02 PM - Constantin Asofiei

- File ca_upd_20220927a_3821c_14253.zip added

Another update, includes some fixes after testing #6672.

#27 - 09/27/2022 04:30 PM - Greg Shah

Code Review ca_upd_20220927a_3821c_14253.zip

I have no objections. Has this been tested with one of the large GUI applications?

#28 - 09/30/2022 05:29 AM - Greg Shah

Greg Shah wrote:

> Code Review ca_upd_20220927a_3821c_14253.zip
>
> I have no objections. Has this been tested with one of the large GUI applications?

To what degree has this been tested with the large GUI applications? I'd like get this merged into 3821c (and either rebase 6129a or merge it into 6129a).

That assumes you agree it is ready.

#29 - 10/02/2022 11:22 AM - Constantin Asofiei

There was a problem in an app related to dataset-handle parameters - BufferManager support is required for blocks defining these. This looks safe to be merged.

#30 - 10/02/2022 11:23 AM - Constantin Asofiei

And the update.

#31 - 10/02/2022 12:18 PM - Constantin Asofiei

- File ca_upd_20221002b_3821c_14261.zip added

Constantin Asofiei wrote:

> And the update.

The latest update, previously had redmine errors.

#32 - 10/03/2022 03:49 PM - Greg Shah

Please merge ca_upd_20221002b_3821c_14261.zip into 3821c.

#33 - 10/05/2022 05:01 AM - Constantin Asofiei

ca_upd_20221002b_3821c_14261.zip was committed to 3821c/14269

#34 - 01/10/2023 07:38 AM - Constantin Asofiei

6129b/14350 improves:

- OUTPUT parameter processing: track if an output parameter is associated with a table field, so batch processing is done only in these cases, for output parameter assignment.
- BufferManager.scopeStart: keep a reverse mapping of bound buffers, per each external program, to avoid iterating the entire set, when a scope starts.

#35 - 01/20/2023 09:20 AM - Constantin Asofiei

After what was done in #6650-34, scope processing is still an expensive area, although the possibilities to improve it further may be difficult. Completion remains set at 90%.

#36 - 12/21/2023 06:14 AM - Constantin Asofiei

7026f rev 14902 includes:

- Refactored the WidgetPool scopeable support to register for processing as needed.
- ObjectOps scopeable support is registered only when: a var is defined, a new instance is being created, or a function/method returns 'object'.
- Refactored the this/source/target-procedure stacks, to remain only one for each case.

#37 - 12/21/2023 10:34 AM - Constantin Asofiei

7026f rev 14903 fixes some regressions in WidgetPool support.

#38 - 12/21/2023 02:39 PM - Constantin Asofiei

Branch 7026f was merged into trunk revision 14897 and archived.

Files

scopeable_6650_patch_20220901a.patch	115 KB	09/01/2022	Constantin Asofiei
scopeable_6650_patch_20220906a.patch	171 KB	09/06/2022	Constantin Asofiei
ca_upd_20220909a_3821c_14233.zip	1010 KB	09/09/2022	Constantin Asofiei
ca_upd_20220923a_3821c_14253.zip	1.16 MB	09/23/2022	Constantin Asofiei
p2j_oo_object_local_vars.txt	31.9 KB	09/23/2022	Constantin Asofiei
ca_upd_20220927a_3821c_14253.zip	1.17 MB	09/27/2022	Constantin Asofiei
ca_upd_20221002b_3821c_14261.zip	1.17 MB	10/02/2022	Constantin Asofiei