Base Language - Feature #2196

implement proper resource deletion of resources created in a procedure ran persistent

10/25/2013 03:12 AM - Constantin Asofiei

Status:	Closed	**Start date:**	10/29/2013
Priority:	Normal	**Due date:**	
Assignee:	Constantin Asofiei	**% Done:**	100%
Category:		**Estimated time:**	24.00 hours
Target version:	Runtime Support for Server Features		
billable:	No	**vendor_id:**	GCD

Description

History

#1 - 10/25/2013 03:13 AM - Constantin Asofiei

From #2120 notes 49 and 50:

Comment by Greg Shah:

> Do we need a new "hook" to handle the class unloading "event" in a RUN PERSISTENT case? For a procedure run as non-persistent, our Finalizable.finished() which is called when the external procedure ends is the equivalent to closing resources when the procedure is unloaded, since these both happen at the same time. But when a procedure is run persistent, then the external procedure will end (and presumeably Finalizable.finished() will be called) long before the procedure is actually unloaded.
>
> I'm worried this has implications for many resources that use Finalizable.

Comment by Constantin Asofiei

> Greg Shah wrote:
>
> > Do we need a new "hook" to handle the class unloading "event" in a RUN PERSISTENT case?
>
> Yes, something like that. In 4GL, the resource might be linked via the INSTANTIATING-PROCEDURE to its "owner": maybe when a persistent procedure is deleted they notify all resources and the ones which have an equal INSTANTIATING-PROCEDURE reference are deleted too? In any case, this is not something query or temp-table specific. I think a separate task is appropriate, to check that all resources (static or dynamic) are deleted properly. And to implement this, I think we will need to register all resources with the current THIS-PROCEDURE instance, at the time of the resource creation; this way, when the procedure gets deleted, the resources can be deleted before the procedure. We might have problems with resources instantiated by the default c'tor, because THIS-PROCEDURE might not be pushed this early (IIRC is pushed when the external procedure block is executed).

#2 - 10/29/2013 03:35 AM - Constantin Asofiei

From #2120 notes 60, 61, 62 - static queries are undeletable, but their handle can be deleted (and a subsequent HANDLE attribute access for the query gives a different handle). Looks like the static resources are kept in a different data structure.

I think this might be a bug in 4GL, as i.e. handle vars referring a static temp-table can't be deleted (a DELETE OBJECT for them is a no-op); same for static BUFFER resources.

#3 - 10/29/2013 08:54 AM - Constantin Asofiei

- Assignee set to Constantin Asofiei

- Start date set to 10/29/2013

- Status changed from New to WIP

#4 - 10/30/2013 08:39 AM - Constantin Asofiei

Notes about implicit deletion:

- from widget cases, only BUTTON, BROWSE and FILL-IN were tested, but the rules should apply to all widgets
- both static and dynamic resources were tested
- resources were created in external procedure, internal procedure, function and exposed via shared vars.
- resources were created in internal procedure, function and exposed via output params
- resource validation was done after
  - external procedure was ran persistent
  - persistent procedure was deleted
  - external procedure was ran non-persistent

The found rules are these:

- resources created via CREATE statement and the PROCEDURE resource do not get deleted if the instantiating procedure is deleted or ran non-persistent.
- the static resources (QUERY, TEMP-TABLE, BUFFER, widgets) are implicitly deleted when the instantiating procedure is deleted, when a non-persistent procedure ends LE: or when the internal procedure/function which created them ends.

#5 - 10/30/2013 09:59 AM - Greg Shah

Does this mean that we just need to defer the processing of TransactionManager.popScope() for external procedures that are run persistent?

Everything else sounds like the behavior that we already have implemented.

#6 - 10/30/2013 11:20 AM - Constantin Asofiei

I missed a rule; the static resources also get deleted when the internal proc/function which created them ends (regardless of how the procedure is ran).

> Everything else sounds like the behavior that we already have implemented.

Yes, only the implicit deletion for static resources is what is missing.

> Does this mean that we just need to defer the processing of TransactionManager.popScope() for external procedures that are run persistent?

Yes, something like this. TM.popScope() should be called (because the external program code does end), but the Block.finalizables should behave differently. The Finalizable.finished for all Block.finalizables should be deferred and executed when the persistent procedure gets deleted.

Beside this, we need to cleanup any handle data generated by the STRING(handle) function calls and ALSO invalidate the resource (i.e. actually call DELETE OBJECT on that resource). This is because the HANDLE attribute might have been saved to another var and the static resource might be leaked to the outer procedure. Any such handle vars are not allowed to use the static resource after its instantiating procedure gets deleted/ends.

Thus, I think is enough to (only for persistent procedures):

1. save the Block.finalizables for the current external procedure at the ProcedureManager, which will process them when the persistent procedure gets deleted.
2. when the created QUERY, TEMP-TABLE, BUFFER or widget resource has DYNAMIC attribute set to false, it gets registered in a set of resources for nearest top-level block.
3. when a top-level block ends, the collected static resources will be:
   - deleted immediately if the top-level block is not the external procedure
   - saved at the ProcedureManager, to be deleted when the persistent procedure gets deleted, if this is the external procedure

#7 - 10/30/2013 11:41 AM - Greg Shah

More questions:

1. In the persistent procedure case, do 4GL transactions (and buffer scopes/record locks) end at the end of the external procedure or when the persistent procedure is deleted?

2. Are there any behaviors of the Finalizable instances that should not be deferred? We may be using Finalizable for a wide range of behaviors, some of which may not be deferred. For example, unnamed stream closing is one that comes to mind. We might also use it for some scoping related features that don't defer.

#8 - 10/30/2013 12:14 PM - Constantin Asofiei

> 1. In the persistent procedure case, do 4GL transactions (and buffer scopes/record locks) end at the end of the external procedure or when the persistent procedure is deleted?

Transactions get committed, but the record locks remain in place.
I've got three programs:

1. program 1: exposes a BUFFER handle to the caller.

```
def shared var h as handle.

open query q for each book.
get first q exclusive-lock.
h = query q:handle.
h = h:get-buffer-handle("book").

book.book-title = "something".

message "transaction:is-open = " string(this-procedure:transaction:is-open).
pause.
```

2. program 2: calls program 1 persistently; the transaction for p2 is ended when the external procedure block ends.

```
def new shared var h as handle.
def var hp as handle.

run p1.p persistent set hp.

message "hp:transaction:is-open = " string(hp:transaction:is-open) "this-procedure:transaction:is-open = "
 string(this-procedure:transaction:is-open).
pause.
delete object hp.
message "record is released now".
```

3. program 3: tries to obtain an exclusive lock on the same record, but does not get it until the persistent procedure is deleted:

```
for each book exclusive-lock:
display book.
end.
```

Thus, record locks do survive, and we can end up having access to buffer referencing an exclusively-locked record outside of a transaction.

2. Are there any behaviors of the Finalizable instances that should not be deferred? We may be using Finalizable for a wide range of behaviors, some of which may not be deferred. For example, unnamed stream closing is one that comes to mind. We might also use it for some scoping related features that don't defer.

Hmm... as you mentioned it, yes, Finalizable is used for some other resource-unrelated code too (Accumulator, UnnamedStreams, appserver, etc). What about this: we change the Finalizable.finished() implementation for the static resources to be a no-op when persistent procedure finishes and call the finished() code when the resource gets deleted.

But Scopeable.scopeFinished() might need some testing too, at least for widgets. Anyway, my testing only touched areas of "is the widget handle valid", I didn't check cases like "what happens when the static widget gets deleted". I think I need some more testing of "what happens with the resource after deletion".

#9 - 10/30/2013 12:35 PM - Constantin Asofiei

Again, something else I've missed. I've been checking handles for static widgets which were only included in a FORM statement; these are reported as non-dynamic and do get deleted after the persistent proc gets deleted/non-persistent proc ends. But, if a static widget is currently being shown at the terminal (via a stream or not), it doesn't get deleted after a persistent proc gets deleted/non-persistent proc ends; and this has nothing to do with the widget's visible state. I think this is because the underlying frame can still be accessed via the widget's FRAME attribute and from this all the other widgets (part of the frame).

#10 - 10/30/2013 12:41 PM - Greg Shah

> Thus, record locks do survive, and we can end up having access to buffer referencing an exclusively-locked record outside of a transaction.

I suspect the buffer is actually referencing a record whose lock has been downgraded to share-lock at the transaction end. This suggests that scoped resources don't exit their scope at the end of the external proc execution when run persistent. At least this is the case for buffers, but I suspect other scoped resources behave the same way.

This makes some sense, since otherwise persistent procedures would not be able to access long-lived resources.

> What about this: we change the Finalizable.finished() implementation for the static resources to be a no-op when persistent procedure finishes and call the finished() code when the resource gets deleted.

I think the changes will impact more than just the static resources, since it also affects scoping. It seems to me that we need to differentiate the scoping cases and resource lifetime from the things that actually do still end at the end of the external procedure execution.

I would like to consider an approach that splits the use cases into 2 groups. In the non-persistent proc case, the exit of the external procedure would call both parts. In the persistent proc case, the exit of the external procedure would call only the "part 1" (stuff that does end) and would defer the "part 2" (scoped/long lived resources) to the procedure delete moment. It seems like in the non-persistent case we call the "procedure delete" processing at external proc exit and we defer it in the persistent case.

#11 - 10/31/2013 03:48 AM - Constantin Asofiei

Greg Shah wrote:

> I suspect the buffer is actually referencing a record whose lock has been downgraded to share-lock at the transaction end. This suggests that scoped resources don't exit their scope at the end of the external proc execution when run persistent. At least this is the case for buffers, but I suspect other scoped resources behave the same way.

Eric/Stanislav: is it possible to interrogate what kind of lock it is used for the record loaded in a BUFFER resource?

> I would like to consider an approach that splits the use cases into 2 groups. In the non-persistent proc case, the exit of the external procedure would call both parts. In the persistent proc case, the exit of the external procedure would call only the "part 1" (stuff that does end) and would defer the "part 2" (scoped/long lived resources) to the procedure delete moment. It seems like in the non-persistent case we call the "procedure delete" processing at external proc exit and we defer it in the persistent case.

OK, this is what I'm focusing on now.

#12 - 11/13/2013 10:31 AM - Constantin Asofiei

- File ca_upd20131113c.zip added

It took a while, but I have a good version. In the end, I had these two notifications and INSTANTIATING-PROCEDURE support:

1. Finalizable.deleted() - called when the persistent procedure gets deleted or right after finished(), otherwise
2. Scopeable.scopeDeleted() - called when the external procedure gets deleted or right after scopeFinished(), otherwise

This affects lots of files, but only a few really need them:

1. BufferManager.scopeDeleted - not sure yet about this
2. LogicalTerminal.scopeDeleted - for surviving non-persistent triggers
3. RecordBuffer.deleted - to allow a static buffer to survive outside of its instantiating procedure
4. TemporaryBuffer$Multiplexer.deleted - this will need to make sure the temp-table is cleaned when a persistent proc is deleted
5. GenericFrame.deleted and GenericFrame$Callback.deleted - this are needed to make sure the frame is cleaned when the external procedure is ended, but there is a limitation here (see bellow about how static frames get deleted).

Other notes:

1. I need to finish addressing the static BUFFER and TEMP-TABLE cases, but I need the static TEMP-TABLE implementation and the other BUFFER changes from #1654 first.
2. static frame deletion: a static frame for which its associated INSTANTIATING-PROCEDURE no longer exists will be deleted ONLY when is hidden; more, if at the end of a non-persistent procedure the static frame is not visible, it will be deleted. Need to finish pushing the frame deletion at HIDE.
3. non-persistent UI trigges: they must survive from a persistent procedure until the procedure gets deleted.
4. implicit static resource deletion is performed only if we are explicitly deleting a persistent procedure.
5. fixed the QueryWrapper cleanup: the static query gets cleaned only when is deleted.

#13 - 11/13/2013 01:38 PM - Greg Shah

Code Review 1113c

I like the results very much. Some minor issues:

1. Missing history entry in RecordBuffer and TemporaryBuffer.

2. Some methods are missing javadoc.

What is the plan to complete the open items (other than the dependency on #1654)?

#14 - 11/13/2013 03:09 PM - Constantin Asofiei

Greg Shah wrote:

> 1. Missing history entry in RecordBuffer and TemporaryBuffer.
> 2. Some methods are missing javadoc.

The Record/TemporaryBuffer and BufferManager were left without history entry on purpose (as they are not finished...). About javadoc: I usually did it by comparison, and let it look how its other counterpart (i.e. finished() looked); so in some cases there is javadoc, in other cases there is. Anyway, I'll double check.

> What is the plan to complete the open items (other than the dependency on #1654)?

The only major point is the static frame deletion on HIDE and proper deregistration from client side. Hope to have this done tomorrow.

#15 - 11/14/2013 10:24 AM - Constantin Asofiei

My changes in static frame deletion somehow ended up digging some other problems:

1. the pause-before-end behaviour: if the program has no frame displayed on the TERMINAL and no text in message area, then the pause at the end is not done; the program just exits. To test this, start a non-UI program from the command line, not the editor:

```
def var i as int.
/* no pause is seen when running this. */
```

2. VISIBLE attribute for frames redirected to unnamed stream targeting something other than the terminal: somehow, in this case, the frame's VISIBLE attribute remains set to no. Can't explain their reason for doing this, because if the frame is redirected to a named stream or to the terminal (regardless if output to terminal. is used), the frame's VISIBLE attribute gets set to yes.
   How does this affect this task: as the frame is not visible, it does not survive the external procedure. The fix for this is not that simple, because in

> P2J the VISIBLE attribute is an intrinsic part of the UI code when working with a frame... for now, I think is best to leave it as is, and let these kind of redirected frames to survive the external procedure (and thus be deleted at the appropriate times).

#16 - 11/14/2013 10:33 AM - Greg Shah

If you can fix the pause-before-end issue quickly, include it with this task.

I am OK for you to open a new Bug for the VISIBLE issue.

#17 - 11/14/2013 11:50 AM - Constantin Asofiei

Greg Shah wrote:

> If you can fix the pause-before-end issue quickly, include it with this task.

I think I have a fix for this, hope the regression testing will not contradict me <g>.

> I am OK for you to open a new Bug for the VISIBLE issue.

Actually, I think I've found the problem. For frames, the server-side state of the VISIBLE flag (at FrameConfig) is not sync'ed with the client-side state of this flag. It seems to work if I use GenericFrame.isVisible instead (as this interrogates the client-side).
The only limitation I see now is case 1 from bellow:

```
output to "something.txt".
display "foo" with frame f1.
if frame f1:visible then message "case 1: frame must not be visible!".
output close.
if frame f1:visible then message "case 2: frame must not be visible!".
```

For case 2, the frame's visibility is moved back to false in P2J, when the unnamed output stream is closed.

#18 - 11/14/2013 12:11 PM - Greg Shah

Perhaps we should just sync the flag. I assume that would fix the case 1 issue? It would also solve some number of other unspecified app problems where we might read the server state only.

#19 - 11/14/2013 12:55 PM - Constantin Asofiei

OK, I'll try this too, as it will save lots of round-trips to the client. At this time, it works using a FrameConfig.isVisible implementation which calls frame.isVisible() (frame is the associated GenericFrame instance), but is expensive.

Something else I've missed this during testing: static resources created at i.e. internal procedure must be initialized in P2J when i.e. the internal procedure executes, as this is how the static resources get initialized when defined in an internal proc; each call of an internal procedure will create a new static resource. The converted code would look like:

```
public Foo
{
   QueryWrapper query1 = null; /* this is defined at the internal proc, not external proc; so no reason to ins
tantiate it here, but as queries are expected to be instance fields, we should leave them here. */

   public void proc0()
   {
      internalProcedure(new Block()
      {
         public void init()
         {
            query1 = new QueryWrapper();
         }
      }
   }
```

For frames, it will require to be register immediately (we can place send a signal to GenericFrame before block.init() is called, to now what kind of finalizable registration to expect).
Gladly, the registration with the external procedure or internal procedure was written with this in mind, so only a couple of conversion changes are required.

#20 - 11/15/2013 01:21 PM - Constantin Asofiei

- File ca_upd20131115b.zip added

This update contains the following:

1. more changes related to DYNAMIC attribute (to mark the resource as dynamic or not when is created)
2. some misc fixes to query scopes and frame names (when scoped to a function)
3. a buffer scoping problem
4. static resources scoped to triggers/internal procedures/functions must be created when that trig/proc/func is invoked; the only exception is the implicit buffer for a table. Thus, the code will now look like:

```
   public class StaticResScope
```

```
{
   StaticResScopeProc0F2 proc0F2Frame = null;
   StaticResScopeProc0F1 proc0F1Frame = null;
   Book.Buf bbBuf2 = null;
   TempRecord1.Buf ttBuf2 = null;
   QueryWrapper query0 = null;
   QueryWrapper query1 = null;

   public void proc0()
   {
      internalProcedure(new Block()
      {
         {
            // they are created when the Block is instantiated
            proc0F2Frame = GenericFrame.createFrame(StaticResScopeProc0F2.class, "proc0-f2");
            proc0F1Frame = GenericFrame.createFrame(StaticResScopeProc0F1.class, "proc0-f1");
            bbBuf2 = RecordBuffer.define(Book.Buf.class, "p2j_test", "bbBuf2");
            ttBuf2 = TemporaryBuffer.define(tt1, "ttBuf2");
            query0 = new QueryWrapper("q0", false);
            query1 = new QueryWrapper("q1", false);
         }
         public void body()
         {
...
```

for a case like:

```
procedure proc0.
   open query q0 for each tt1.
   define query q1 for tt1.
   form "ch" with frame f1.
   define buffer bb for book.
   define buffer tt for tt1.
   define frame f2.
end.
```

TODOs:

- clean empty default c'tor blocks in triggers
- improve the VISIBLE attribute, to use the server-side flag and not interrogate the P2J client.

This is going through regression testing now.

#21 - 11/15/2013 01:35 PM - Constantin Asofiei

- File resource_deletion_tests20131115.zip added

Added tests related to implicit resource deletion and INSTANTIATING-PROCEDURE attribute.

#22 - 11/16/2013 02:02 PM - Constantin Asofiei

My buffer scoping changes exposed some incorrect usage of the DictionaryWorker. Currently, the DictionaryWorker uses case-insensitive keys in a dictionary. This is OK for 4GL names and other cases not related to java names, but fails when it maps java names. Thus, something like this will fail:

```
def temp-table ttvoid field f1 as int.
def temp-table tt-void field f2 as int.

ttVoid.f1 = 10.
tt-void.f2 = 10.
```

as it will convert the ttVoid and @tt-Void buffer scopes to the same name. The problem is in annotations/buffer_name_conflicts.rules, which uses javanames as the keys, in a case-insensitive manner.

On a side note, there are some shared frame problems which at first made me think they are related to the FrameWidget changes which bypass the frame's config and use GenericFrame for various VISIBLE attribute.

#23 - 11/16/2013 02:33 PM - Constantin Asofiei

- File ca_upd20131116e.zip added

- % Done changed from 0 to 70

Attached update fixes the buffer_name_conflicts.rules problem and some other regressions found during testing. The FrameWidget changes (related to VISIBLE and HIDDEN) are temporary so the shared frame code will pass; I will focus on the VISIBLE attribute tomorrow.

#24 - 11/18/2013 10:54 AM - Greg Shah

Code Review 1116e

I am OK with the changes. I do have some questions:

1. The addition of a BLOCK node as a prior sibling to CS_INSTANCE_VARS in the templates, seems a bit strange. It isn't something I would have expected to work, since we previously assumed that the CS_* nodes would be the first children.

2. Is it always true that the browse cells/columns are never dynamic? If so, put a comment in the related code to explain this.

#25 - 11/20/2013 11:27 AM - Constantin Asofiei

Greg Shah wrote:

1. The addition of a BLOCK node as a prior sibling to CS_INSTANCE_VARS in the templates, seems a bit strange. It isn't something I would have expected to work, since we previously assumed that the CS_* nodes would be the first children.

Yes, this surprisingly works. The idea at the time was that it was easier to just add a block and let it emit (as it did all the work I needed). But I think it will be more readable if a constructor is added to a CS_CONSTRUCTORS section (so the fields get initiated by an existing default c'tor for the class, not the implicit one). I don't expect this to take too much time.

2. Is it always true that the browse cells/columns are never dynamic? If so, put a comment in the related code to explain this.

Actually, I meant to put a note "need to test if the browse cells/columns are dynamic or not"; I will add this note, but more testing can be postponed until the dynamic BROWSE widget will be implemented (as BROWSE widget is a beast of its own, and I don't think is necessary to start digging into how it works internally for this task).

#26 - 11/21/2013 09:12 AM - Constantin Asofiei

Greg, is there a specific reason the Reader/Writer threads don't have a security context set? I ask this because some state incoming from the client (as in "this frame is no longer visible") which is processed by the Reader thread on server-side, may lead to deleting the frame. But, the Reader thread doesn't have a security context, thus my question.

#27 - 11/21/2013 09:14 AM - Constantin Asofiei

LE: the queue may be multiplexed, thus the reason... I think we need to do something similar to how Dispatcher.processInbound works, to push/pop the security context only for the duration of the state synchronization.

#28 - 11/21/2013 01:53 PM - Greg Shah

Generally speaking, to increase security, it is a good practice to limit the use of credentials (e.g. security context) whenever it is not really needed. The reader/writer threads are good examples of threads that really don't have much need for context.

You are right that the queue can be multiplexed, but it should not happen in client sessions, which are all direct sessions today.

I think the reason that client-to-server state sync is working today is that each user's queue has a specific LogicalTerminal instance registered as the state synchronizer. LogicalTerminal.applyChanges() will be called synchronously from the reader thread (which has no context), but it is called on the proper instance and thus the state changes are applied properly even though there is no security context. So long as no security-sensitive methods are invoked, this should work.

What is there in the visible flag case that requires the context?

#29 - 11/21/2013 11:42 PM - Constantin Asofiei

> What is there in the visible flag case that requires the context?

The problem is that a frame (for which its instantiating-procedure is no longer available) needs to be deleted exactly when its visibility changes from true to false; this means that, when the state synchronizer sets the frame to not visible/hidden, I need to check: is the instantiating procedure still valid? If not, go ahead and delete the frame. One problem is when checking the instantiating procedure: I need the ProcedureManager's context-local data to determine if the procedure is still valid. The other is when actually the frame is deleted: this will need the context-local LogicalTerminal instance (obtained via LogicalTerminal.locate()) to destroy the frame and clean up after it. Even if I can solve the ProcedureManager dependency, the LogicalTerminal part required at frame deletion can't be removed easily.

#30 - 11/22/2013 09:41 AM - Greg Shah

> this will need the context-local LogicalTerminal instance (obtained via LogicalTerminal.locate()) to destroy the frame and clean up after it. Even if I can solve the ProcedureManager dependency, the LogicalTerminal part required at frame deletion can't be removed easily.

Isn't the this reference in LogicalTerminal.applyChanges() the same as the instance returned by LogicalTerminal.locate()?

#31 - 11/22/2013 09:42 AM - Constantin Asofiei

Greg Shah wrote:

> Isn't the this reference in LogicalTerminal.applyChanges() the same as the instance returned by LogicalTerminal.locate()?

locate() calls instance.get(); thus, without proper security context set, it will not work.

#32 - 11/22/2013 09:49 AM - Greg Shah

My point is that the applyChanges() method is already running in the proper instance because that instance is what is registered as the StateSynchronizer instance. If only instance methods are being called from there, then no calls to locate() would be needed.

#33 - 11/22/2013 10:11 AM - Constantin Asofiei

Yes, I agree with you, but that is not enough. To make all call-paths reachable from the frame deletion code independent of context-local data, I will need to refactor some of the LogicalTerminal APIs; at a minimum, the LogicalTerminal instance should be saved for each GenericFrame (so that there will be a i.e. LogicalTerminal.destroyFrame instance method).

Plus, I will need to change some INSTANTIATING-PROCEDURE code, to reverse the way it works: instead of interrogating ProcedureManager with a "is this procedure handle still valid?" request, I need to register ALL created resources with their instantiating procedure and notify them when that gets deleted.

#34 - 11/22/2013 10:13 AM - Greg Shah

OK, then let's get the context set properly instead of reworking all of this code.

What are the implications for setting context permanently for the reader/writer threads on DIRECT sessions only?

#35 - 11/22/2013 10:25 AM - Constantin Asofiei

Greg Shah wrote:

> What are the implications for setting context permanently for the reader/writer threads on DIRECT sessions only?

My concern is: what happens if somehow the read message has a different security context than the one set at the thread? The Reader thread is theoretically exposed to the outside world, and we should not immediately trust the read data; at least, check the message's key against the thread's security context.

#36 - 11/22/2013 11:06 AM - Greg Shah

We can partially mitigate this (for the body of the message) by only setting the context during the state sync processing. But this portion would still be exposed.

Another approach is to move the state sync processing to the waiting thread (in the 4GL case, this will be the conversation thread, but other use cases for the protocol could be different). In most cases, we know we have a waiting thread that is about to be unblocked. Before we return on that thread, we could potentially run the state sync there. That thread already has context. The only tricky part is for asynch messages. Do we even support state sync during async messages? If so, we might have to exclude that case...

#37 - 11/22/2013 12:12 PM - Constantin Asofiei

Greg Shah wrote:

> We can partially mitigate this (for the body of the message) by only setting the context during the state sync processing. But this portion would still be exposed.

Actually, this is my current approach: the state sync processing is bracketed in push/pop of the message's security context. I hope to clear the frame related errors after the testing I'm running now (the errors were related to shared frames).

> Another approach is to move the state sync processing to the waiting thread.

I'll look into this and see if it can be easily done.

#38 - 12/09/2013 09:39 AM - Constantin Asofiei

- File ca_upd20131209d.zip added

With this version all hand-written tests pass. Needs to be regression tested.

#39 - 12/09/2013 06:14 PM - Greg Shah

Code Review 1209d

Eric: please review the changes in the persist package.

I am generally fine with the changes. Of course, I am a bit nervous about the scoping and visibility change, but they look OK. I have these questions:

1. Why do we now call applyChanges() from both Conversation.block() and Queue.transact()?

2. Why is it safe to eliminate the scopeValid and needsPageElementCleanup from GenericFrame?

#40 - 12/10/2013 09:13 AM - Constantin Asofiei

- File ca_upd20131210b.zip added

Greg Shah wrote:

> 1. Why do we now call applyChanges() from both Conversation.block() and Queue.transact()?

From my initial analysis, when the Conversation thread is started, the received message is sent directly to the Conversation. But, I think I've missed a case: if a received message is not processed by the Conversation thread and is sent to the Dispatcher (as CTRL-C is sent), the state-sync changes will not be applied. Thus, I think the following locations should process the state sync changes:

1. move the state sync call from Queue.transact() to the end of Queue.transactImpl - this will cover all call paths.
2. move the state sync call from Conversation.block() to Dispatcher.processInbound - this will cover both sync and async calls.

> 2. Why is it safe to eliminate the scopeValid and needsPageElementCleanup from GenericFrame?

- The GenericFrame.scopeValid field was used only in GenericFrame.valid, to determine if a frame is valid or not; but, in resource terms, a frame must be valid unless is deleted... that's why I've removed it and changed GenericFrame.valid to check the underlying frame widget.
- The GenericFrame.needsPageElementCleanup was removed because page header/footer state survives the original frame scope; a static frame created in external procedure and exposed to the caller can be used by the caller as a page header/footer for subsequent stream output.

Attached update contains changes related to the applyChanges calls. I'll re-run regression to make sure it doesn't brake some other frame behaviour.

#41 - 12/10/2013 09:58 AM - Greg Shah

Code Review 1210b

I'm OK with the changes. You can check in and distribute when they pass testing.

#42 - 12/10/2013 10:19 AM - Eric Faulhaber

1209d/10b:

Persistence changes look OK.

#43 - 12/12/2013 02:39 AM - Constantin Asofiei

- % Done changed from 70 to 100

1210b.zip was committed to bzr revision 10419.

#44 - 12/12/2013 06:44 AM - Greg Shah

- Status changed from WIP to Closed

#45 - 07/28/2014 04:43 AM - Constantin Asofiei

- File ca_upd20140728a.zip added

Attached update fixes a case when a frame (for which its instantiating procedure is no longer alive) is implicitly hidden on the client-side (by another frame). The Frame.onFromVisibleToHidden API was changed to notify the server-side that the frame was hidden. Server-side will check if it can be deleted - if so, it will delete and destroy the frame resource.

#46 - 07/28/2014 10:40 AM - Greg Shah

Code Review 0728a

The change looks good. I am actually very, very surprised we got away without this for so long.

#47 - 07/28/2014 11:02 AM - Constantin Asofiei

0728a.zip passed testing and was committed to bzr rev 10582.

#48 - 11/16/2016 11:42 AM - Greg Shah

- Target version changed from Milestone 7 to Runtime Support for Server Features

Files

ca_upd20131113c.zip	714 KB	11/13/2013	Constantin Asofiei
ca_upd20131115b.zip	940 KB	11/15/2013	Constantin Asofiei
resource_deletion_tests20131115.zip	8.8 KB	11/15/2013	Constantin Asofiei
ca_upd20131116e.zip	938 KB	11/16/2013	Constantin Asofiei
ca_upd20131209d.zip	1.01 MB	12/09/2013	Constantin Asofiei
ca_upd20131210b.zip	1.02 MB	12/10/2013	Constantin Asofiei
ca_upd20140728a.zip	98.4 KB	07/28/2014	Constantin Asofiei