Base Language - Bug #4602

fixes for OO 4GL and structured error handling

03/30/2020 02:32 PM - Constantin Asofiei

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	40%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD		

Description

Related issues:

Related to Base Language - Feature #4373: finish core OO 4GL support	**New**
Related to Base Language - Bug #4891: Nested method calls ignore 'silent' (no...	**New**
Related to Base Language - Feature #4352: finish progress.lang.apperror and p...	**Rejected**

History

#1 - 03/30/2020 02:32 PM - Constantin Asofiei

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

#2 - 03/30/2020 02:39 PM - Constantin Asofiei

From #4384-121:

> I have FWD fixes for cases of 'nested' calls (for ctors, methods, internal procedures, dynamic functions), which cover RETURN ERROR "foo", RETURN ERROR new AppError, UNDO, THROW ERROR and 'implicit' OE errors (like accessing an attribute in an unknown handle). These work, except the case for RETURN ERROR with nested constructors and default error management (no ROUTINE/BLOCK-LEVEL statements): if this is called from a super-constructor, then 4GL behaves differently - unlike RUN (or other types of calls), the ERROR flag propagates up the stack, until it reaches the NEW operator (and this makes sense, as the object needs to be 'invalidated', because the c'tor failed...); I haven't found a solution in FWD for this, yet.
>
> The other parts which needs to be covered in 4GL tests are:
>
> - direct function calls
> - dynamic new
> - dynamic object method calls
> - external programs
> - ON ERROR UNDO, THROW clauses.
> - error propagation in nested blocks
>
> From these, I think only the last two may need explicit fixing - as it looks like the ERROR can be 'eaten' by a parent block, even if it doesn't have a CATCH block to treat it.

My tests are in testcases/uast/error_handling (must be converted from this folder, with links to ../cfg/ and ../data/). Marian's tests are in xfer testcases project, error_handling folder.

#3 - 04/07/2020 12:11 PM - Constantin Asofiei

Another case to cover: simple statements with NO-ERROR, like ch = h:name NO-ERROR. and input from os-dir("missingdir"). This checks for server-side and client-side raised ERROR conditions.

#4 - 04/26/2020 12:30 PM - Constantin Asofiei

Marian, a weird question: in case of argument validation errors for appserver calls, does the caller's block-level or ROUTINE-LEVEL UNDO, THROW option get 'transferred' to the remote side?

And also, I think I need to check what happens with errors from argument validation for non-appserver RUN (and such) calls.

#5 - 04/27/2020 01:48 AM - Marian Edu

Constantin Asofiei wrote:

> Marian, a weird question: in case of argument validation errors for appserver calls, does the caller's block-level or ROUTINE-LEVEL UNDO, THROW option get 'transferred' to the remote side?
>
> And also, I think I need to check what happens with errors from argument validation for non-appserver RUN (and such) calls.

Constantin, the ROUTINE-LEVEL, BLOCK-LEVEL as well as ON ERROR options do not 'travel' to the other end... it only affects the block on the caller end, if you run something on the appsrv that would lead to an error but that one is shielded by the lack of an ROUTINE-LEVEL, BLOCK-LEVEL on the server side business logic there is no way to influence that from the client side.

Since the remote call is inherently a dynamic call the parameter validation occurs before the remote method is actually executed, in that case an error is thrown back to the client and what happens on the client it depends on error handling on that end.

#6 - 05/12/2020 01:10 PM - Greg Shah

I'm reproducing some discussion from #4349.

#4349-55:

> Greg Shah wrote:
>
> Consider this code:
>
> [...]
>
> LegacyErrorException extends ErrorConditionException which is the equivalent or the ERROR condition in the 4GL. But this code bypasses the ERROR-STATUS processing (recording error/warning state when NO-ERROR is present. The proper processing for ERROR-STATUS is done by the ErrorManager.

Marian wrote:

What we used, and seemed to work fine for our tests, is using undoThrow from BlockManager as in:

```
undoThrow(SysError.newInstance(String.format("Invalid value specified for %s:GetEnum",
                           ObjectOps.getLegacyName(enumClass)), 15246));
```

#4349-56:

> Greg wrote:
>
> Please note that the code above will result in tracked objects (e.g. in the session object chain) being created for the exceptions. If this is necessary, then it will need some attention as well in ErrorManager.

> Marian wrote:
>
> This is correct, the error pop up the stack till is being catch and if is not deleted in that catch block it will linger in the session objects chain - eventually will be garbage collected at some point. It was recommended to delete error objects inside catch block, before they changed the recommendation to not delete anything anymore and let the GC handle everything :)

I'd like to resolve the multiple issues highlighted here.

1. If I understand correctly, any direct usage of new ErrorConditionException(...) will not work properly in code that has structured error handling. But it is not obvious how to leverage ProErro or SysError instances. Without this, I think our catch processing is not going to match the 4GL. The highest priority is in ErrorManager (5 locations) which is used everywhere for proper NO-ERROR support. But I think this affects all of FWD as well. There are 61 other places in FWD that directly construct ErrorConditionException. I think we should create factory methods for construction of the proper errors and then rework the code to use those everywhere.

2. I guess by using SysError.newInstance() we can match the session object chain behavior, but this won't work without modifications.

3. Our session object chain is pinned in memory todat but in the 4GL the chaining is equivalent to using a WeakReference in Java. The 4GL chained references can be garbage collected. I do wonder what that means for the object chain (does the 4GL automatically clean up the chain when something is GC'd or can you get an invalid object reference back). I'm guessing invalid references are possible.

Constantin: please post your thoughts.

I'd like to resolve this ASAP since it will manifest in many bad ways at runtime. Also, I'd like to get a clean and common approach retrofitted everywhere so that in can be consistent in all future work.

#7 - 05/12/2020 05:48 PM - Constantin Asofiei

Greg Shah wrote:

> 1. If I understand correctly, any direct usage of new ErrorConditionException(...) will not work properly in code that has structured error handling.

Yes, looks like these will need to be refactored.

> But it is not obvious how to leverage ProErro or SysError instances.

I don't think the OE runtime ever throws ProError; I can't find any docs about this. The docs state that only SysError is thrown (and neither can be sub-classed or instantiated by application code).

> Without this, I think our catch processing is not going to match the 4GL. The highest priority is in ErrorManager (5 locations) which is used everywhere for proper NO-ERROR support. But I think this affects all of FWD as well. There are 61 other places in FWD that directly construct ErrorConditionException. I think we should create factory methods for construction of the proper errors and then rework the code to use those everywhere.

LegacyErrorException was meant to only be used via undoThrow or undoThrowTopLevel APIs. And the usage in LegacyEnum.getEnum doesn't have a compatible top-level block. This will pose problems, as the current runtime requires all LegacyErrorException usage to be from 4GL-compatible blocks, even if you force a throw new LegacyErrorException.

Note that when implementing the p2j.oo classes, we may need to mark them as ROUTINE-LEVEL UNDO, THROW or BLOCK-LEVEL UNDO, THROW (Marian already did this for some on which he is working). Otherwise, all the management for the OO-style ERROR conditions (like SysError) will not be able to be processed correctly by parent blocks.

> 2. I guess by using SysError.newInstance() we can match the session object chain behavior, but this won't work without modifications.

I'm not sure I understand what is needed here - this creates a 4GL-compatible ObjectResource. We already do reference tracking and destroy objects once they are no longer referenced, as long as the instance is held in a ObjectVar. And automatically delete any instance which was never assigned in the first place (when the top-level block which created it finishes).

> 3. Our session object chain is pinned in memory todat but in the 4GL the chaining is equivalent to using a WeakReference in Java. The 4GL chained references can be garbage collected. I do wonder what that means for the object chain (does the 4GL automatically clean up the chain when something is GC'd or can you get an invalid object reference back). I'm guessing invalid references are possible.

We need tests for this. Something like 'create a new instance and save it in var A, check the chain, set var A to unknown, check the chain'. IIRC 4GL cleans after the reference pretty quickly.

But regardless, even if the chain would have invalid references, the application code can't do anything with them. At most it can do a valid-object test before accessing that instance.

#8 - 05/12/2020 06:01 PM - Greg Shah

I want to avoid using undoThrow() everywhere in our runtime. This was designed to be used from within a BlockManager block context, not from the runtime. I prefer that we continue using EM..recordOrThrowError() etc... and hide the setup of things inside there.

#9 - 05/12/2020 06:30 PM - Constantin Asofiei

Greg Shah wrote:

> I want to avoid using undoThrow() everywhere in our runtime. This was designed to be used from within a BlockManager block context, not from the runtime. I prefer that we continue using EM..recordOrThrowError() etc... and hide the setup of things inside there.

I think at least BlockManager.mustManageLegacyError should be checked before throwing a LegacyErrorException. But ErrorManager.recordOrThrowError is aware of this. So if you add another dedicated API, use mustManageLegacyError.

My concerns at this time are:

- pl.Contains - this is from PL/Java and must remain as throw new ErrorConditionException, or some different approach, as we can't check mustManageLegacyError in this 'headless' context.
- persist.TemporaryBuffer usage I think is OK to remain as is (same as DmoAsmWorker usage)
- p2j.ui usage I think can be switched safely
- p2j.util can be switched, except:
  - Agent which should throw new ErrorConditionException, and the AppServerHelper should decide what to do with this
  - SharedVariableManager.errorHelper has a comment there
  - other notes in CFO - I think at least some need to be changed to recordOrThrowError; currently the usage at least for parameter validation seems wrong, as if NO-ERROR clause is set, the error message is not recorded.

#10 - 06/20/2020 11:54 AM - Greg Shah

- % Done changed from 0 to 40

Task branch 4231b has been merged to trunk as revision 11347.

#11 - 09/15/2020 08:56 AM - Greg Shah

- Related to Bug #4891: Nested method calls ignore 'silent' (no-error) option. added

#12 - 06/02/2022 03:45 PM - Greg Shah

This work from #4352 needs to be worked here:

> apperror vs syserror - these weren't tested comprehensively, the support is pretty good, but when combined with ROUTINE-LEVEL or BLOCK-LEVEL statements, we may have missing issues (as I recall, we need to convert ERROR condition to SysError class). AppError is used at least by RETURN ERROR statements.
>
> We need to test and fix any deviations.

This will complete both ON THROW and UNDO THROW.

#13 - 06/02/2022 03:45 PM - Greg Shah

- Related to Feature #4352: finish progress.lang.apperror and progress.lang.syserror added