Base Language - Bug #2197

fix the error message list cleanup when NO-ERROR mode is enabled and ERROR-STATUS:ERROR is not set

10/25/2013 06:02 AM - Constantin Asofiei

Status:	Closed	**Start date:**	01/13/2014
Priority:	Normal	**Due date:**	01/21/2014
Assignee:	Vadim Gindin	**% Done:**	100%
Category:		**Estimated time:**	32.00 hours
Target version:	Cleanup and Stablization for Server Features		
billable:	No	**case_num:**	
vendor_id:	GCD	**version:**	

Description

Related issues:

Related to Base Language - Bug #2338: fix _MSG function	**Closed**	**07/10/2014**

History

#1 - 10/25/2013 06:04 AM - Constantin Asofiei

From #1612 notes 27 and 28:
Constantin Asofiei comment:

> remember the ErrorManager.recordOrShowError changes related to cases when ERROR flag is not set, but error message is recorded? Looks like they are not enough; I feel like I'm chasing my own tail as I should have tested this before, because the following will fail in P2J:
>
> ```
> def var ch as char init ?.
> session:set-numeric-format(ch, ch) no-error.
> message error-status:num-messages. /* here is 1 */
> session:set-numeric-format(ch, ch) no-error.
> message error-status:num-messages. /* here must be 1 too */
> ```
>
> This fails as the second statement will not clear the error list, and will register a new error on top of the previous one. I think Evgeny's initial fix to clear the error list when silentErrorEnable is called is the one to use, but I can't explain why ErrorManager was initially coded to clear the list when silentErrorDisable is called - maybe there is a case when we don't need to clear the list? This is what bugs me.

Greg Shah comment:

> No, something else is wrong. I know that clearing in the disable is correct (or mostly correct). It is possible to see the old list during the new no-error statement/expression's processing:
>
> ```
> def var ch as char init ?.
> session:set-numeric-format(ch, ch) no-error.
> message error-status:num-messages. /* here is 1 */
> if error-status:num-messages < 1 then message "There should have been an error message logged.". else sess
> ion:set-numeric-format(ch, ch) no-error.  /* if we clear it in the enable, then this would fail */
> message error-status:num-messages. /* here must be 1 too */
> ```

#2 - 01/06/2014 01:09 PM - Greg Shah

- Target version changed from Milestone 7 to Milestone 11

#3 - 01/13/2014 05:04 PM - Greg Shah

- Estimated time set to 32.00

- Due date set to 01/21/2014

- Assignee set to Vadim Gindin

- Start date set to 01/13/2014

Start by writing cases to investigate this behavior, then propose a solution.

#4 - 01/21/2014 04:57 PM - Vadim Gindin

First of all I need to understand the problem. As we can see from the tests Java program does not clear error list, but Progress does clear it.

Q1. How it should work? The Progress remembers the error attributes in ERROR-STATUS until next execution of a statement with NO-ERROR. Why there are possible of more than one error?! - It's for cases where could be more than one error during execution of one statement and only for such cases. Do I correctly understand this moment or there are possible to accumulate errors during execution several statements with NO-ERROR?

Q2. Implementation. There are 2 places where we are clearing an error list and 2 scenarios of that.
2 places:
1) inside the method ErrorManager.ServerDataAccess.setPending(boolean) (line 2186)
2) inside the method ErrorManager.silentErrorDisable() (line 442)
2 scenarios:
1) Between enable and disable. During execution of some of ErrorManager methods adding errors. They are calling ErrorManager.ServerDataAccess.setPending(boolean)
2) During disable.

In the first place and during the first scenario only pending value (and it's previous value) is used to determine: clear or don't clear.

In the second place there is an additional flag WorkArea.dontClean is used to suppress clearing at the last moment. This flag could be set externally through static method ErrorManager.setDontClean() and internally - in only one method recordOrShowError.

This logic is not very simple, but we can understand that WorkArea.dontClean could be set (and designed especially for that) to TRUE in exceptional cases when an error occurs but ERROR-STATUS:ERROR is not set. Flag pending in such cases will be FALSE. And we can understand that in such cases the error list will not be cleared.

OK, but Constantin's test shows that this logic is wrong: session:set-numeric-format(ch, ch) do not set the flag ERROR-STATUS:ERROR and the Progress clears its error list and outputs
1
1

Why it was needed to process ERROR-STATUS:ERROR=no case separately here? May be there was a corresponding task?

Q3. Greg, I didn't understand your example. Why it would fail if we will move clearing to enable-method?

> Why there are possible of more than one error?! - It's for cases where could be more than one error during execution of one statement and only for such cases.

No, this is not the problem. Progress does have an internal list of errors for each statement/expression. You can use ERROR-STATUS:NUM-MESSAGES to see how many have occurred (and some methods to obtain each one). But this has nothing to do with the problem in this task. The only reason Constantin's testcase above uses this feature is because it allows detection of the P2J bug. The multiple errors behavior is not really the problem.

I also should note that NO-ERROR can be used on statements that can cause nested logic to execute. For example, using RUN program.p NO-ERROR can cause quite a complex situation, since there will be nested blocks/multiple statements in that program that can also use NO-ERROR. Likewise, as part of an expression like my-var = something + my-function(). the call to the user-defined function can have nested blocks/multiple statements inside it.

For this reason, there is careful tracking and stacking of the ERROR-STATUS state. This is one of the reasons the logic is so complex.

> Q2. Implementation
> ...

Constantin, I think you are the better person to answer this one.

> Q3. Greg, I didn't understand your example. Why it would fail if we will move clearing to enable-method?

I think my testcase is not quite right. Perhaps this will explain it better:

```
def var ch as char init ?.
def var num as int.

session:set-numeric-format(ch, ch) no-error.
message error-status:num-messages. /* outputs "1" */

num = error-status:num-messages no-error.  /* num will be 1 here even though we are in a statement that is NO-ERROR */

if num < 0 then message "There should have been an error message logged.".
else session:set-numeric-format(ch, ch) no-error.  /* if we clear it in the enable, then this would fail */

message num error-status:num-messages. /* outputs "1 1" */
```

The key point here is that if the clearing is done on the enable, then the num = error-status:num-messages no-error. would yield 0, right? That would be wrong. So the ERROR-STATUS state must be visible even INSIDE the next NO-ERROR statement. I think this is why we previously implemented the "deferred" clearing in the disable.

#6 - 06/06/2014 12:50 PM - Constantin Asofiei

Vadim, regarding Q2 in note 4: as Greg mentioned, we can't clean the list too early and the right place to me looks like is when silentErrorDisable is called.

Also, we are missing some things; when a NO-ERROR clause is involved, there are two states:

1. the prior state, as before the NO-ERROR statement is invoked.
2. the current state, affected by the current NO-ERROR statement.

Take a look at this case, which does something similar as Greg's case, but with a twist:

```
procedure foo.
   message error-status:error error-status:num-messages error-status:get-message(1).
   return error.
end.

def var h as handle.
def var ch as char.
ch = h:name no-error.

run foo no-error.
message error-status:error error-status:num-messages error-status:get-message(1).
```

Inside foo, the state of ERROR-STATUS is preserved and available. After the RUN completes (as an ERROR is returned), it will discard the previous data and replace it with the new one. There might be an even better case to demonstrate this, but I hope you got the idea. What this means for us:

1. ErrorManager needs to keep track of two separate error-lists: recordOrShow/Throw will use the "inactive" error-list, which is made "active" only when silentErrorDisabled is called. When silentErrorEnabled is called, the "inactive" error-list is cleaned (actually, this can be done right after silentErrorDisabled replaces the "active" with the "inactive" error-list). This will allow us to get rid of the dontClean and possible other flags and make the implementation cleaner.
2. see #1632 - I think the _MSG implementation can't use the same error-list as the NO-ERROR clause... _MSG looks like it tracks actual raised errors, while NO-ERROR needs to be aware only of unraised errors. We need to fix this.

#7 - 06/12/2014 05:11 AM - Vadim Gindin

It seems we need not only additional errors list. We also need additional flags like error, pending and some others. I stuck..

Current implementation for me does not seems clear. Why don't we use a Stack structure to maintain procedure or function calls? I see from your examples, that all information (error list and flags) are stacked and after procedure call ended it should be recovered to a previous state. Sequence of calls can be nested in several levels. The sequenced "state" here is not only error list but the flags also. All this state should be stacked. Am I wrong?

Constantin, you noted two couples of lists: "inactive" and "active" for "not silent" and "silent" modes. Did I correctly understand? Another pair is raised and unraised lists for _MSG function. I understood that "unraised" list should contain errors that do not set "error" flag. Did you mean separate cases or "unraised" is "inactive" and "raised" is "active" lists? This confused me.

About dontClean flag. As far as I understood it used only for situation when one expression is raising several errors. During processing of the error list this flag is set to contain and record all of them not only the first. It seems like some separate case. I'm not sure if it possible to use errorPending flag here.

There are a lot of flags. I'm not sure, that I won't brake this code. Could you help me?

#8 - 06/12/2014 06:12 AM - Constantin Asofiei

Vadim Gindin wrote:

> Current implementation for me does not seems clear. Why don't we use a Stack structure to maintain procedure or function calls? I see from your examples, that all information (error list and flags) are stacked and after procedure call ended it should be recovered to a previous state. Sequence of calls can be nested in several levels. The sequenced "state" here is not only error list but the flags also. All this state should be stacked. Am I wrong?

I think you are onto something, and we should stack all the data; but you need to create some tests to definitely prove this.

> Constantin, you noted two couples of lists: "inactive" and "active" for "not silent" and "silent" modes. Did I correctly understand?

Yes, that was the idea, but your stack idea might be better.

> Another pair is raised and unraised lists for _MSG function.

I think _MSG works only with raised error conditions, not silent errors. Anyway, leave this for later, after the NO-ERROR stuff.

> About dontClean flag. As far as I understood it used only for situation when one expression is raising several errors. During processing of the error list this flag is set to contain and record all of them not only the first. It seems like some separate case. I'm not sure if it possible to use errorPending flag here.

The idea of the dontClean flag was this: in some cases, an error message is recorded, but the ERROR-STATUS:ERROR flag is not set. The dontClean flag prevents cleaning up the recorded errors, when silentErrorDisable is called - previously, this was cleaning the error list, if the ERROR flag was not being set by the recorded error, which was wrong. If the stack pays out, the dontClean flag will need to be removed.

> There are a lot of flags. I'm not sure, that I won't brake this code.

I think you got this backwards: the code is already broken as it doesn't work as in 4GL - go with your instinct on this one, create some testcases to prove that we need to stack this info.

#9 - 06/14/2014 06:44 AM - Vadim Gindin

I made some tests and committed them into /errlist subfolder. There are also your tests there. Most of my tests contains two internal subprocedures foo and bar. Main procedure calls foo, and foo calls bar procedure. I variated error expressions in these procedures with different options to find out stacking behavior. Here are my findings with references to tests.

1. The main conclusion is there are no stacking behavior with error lists. I.e. the error list is common for all procedures. NO-ERROR option applied to RUN command is applied to this statement only and not to the statements of called procedure. In other words behaviors of 2 different options NO-ERROR in 2 different statements will not intersects. It means that silent field can be applied only to current executed statement and there is not need to stack this attribute. See Constantin's test.

2. When some statement of procedure raises an error, this breaks execution of this procedure without error for calling RUN statement. Execution flow is returned to parent procedures and continues with the statement, the next after RUN. When some procedure executes the statement return error., this raises the error for RUN statement and breaks it's procedure execution. Herewith no error is added to error list. Only the error flag is set to true. See *noerr.p*, *return_err.p*.

3. It seems, that NO-ERROR and usual errors are contained in separate lists. See tests *noerr_diff_err.p*. The error from set-numeric-format survived till the end of main procedure and despite of the next error, raised after bar call in foo procedure. The last message outputs set-numeric-format's error. It also shows the next problem. silentErrorDisable is placed right after set-numberic-format but raised error is surviving. It means that we it is wrong to clean the error list in silentErrorDisable method.

4. The errors that are not set the error flag can be contained in usual or NO-ERROR error lists. It seems that such errors do not differ from other except the error flag.

Questions.

1. What should I test more?
2. There are the warning attribute in ErrorManager.WorkArea. What does it mean? In the docs I only found WARNING attribute of COMPILE statement.. Is it relates to this attribute or means no to throw an Exception or it means just the errors, that are not set the error flag?
3. There are several attributes like soapFault, error and so on. I suspect that we will need to have 2 copies of them for both lists: usual and NO-ERROR. But I don't know exactly behavior of them except the error and silent. What do you think?

#10 - 06/17/2014 06:05 AM - Constantin Asofiei

Vadim Gindin wrote:

> 1. The main conclusion is there are no stacking behavior with error lists. I.e. the error list is common for all procedures. NO-ERROR option applied to RUN command is applied to this statement only and not to the statements of called procedure. In other words behaviors of 2 different options NO-ERROR in 2 different statements will not intersects. It means that silent field can be applied only to current executed statement and there is not need to stack this attribute. See Constantin's test.

See the nested_noerror.p, for a more complex example, using functions - this shows (as you found) that the ERROR-STATUS does not restore the errors from a nested NO-ERROR clause (from another top-level block). Note that the silent field (if set) is backed up and restored each time a top-level block is ran (i.e. function/procedure/trigger), by TransactionManager. From this test, I think we can conclude that the "inactive" error list I've

mentioned gets cleared when silentErrorDisabled is called - to allow the "parent" NO-ERROR statement to record any data to it.

> 2. When some statement of procedure raises an error, this breaks execution of this procedure without error for calling RUN statement. Execution flow is returned to parent procedures and continues with the statement, the next after RUN. When some procedure executes the statement return error., this raises the error for RUN statement and breaks it's procedure execution. Herewith no error is added to error list. Only the error flag is set to true. See *noerr.p*, *return_err.p*.

Yes, this should work properly in P2J.

> 3. It seems, that NO-ERROR and usual errors are contained in separate lists. See tests *noerr_diff_err.p*. The error from set-numeric-format survived till the end of main procedure and despite of the next error, raised after bar call in foo procedure. The last message outputs set-numeric-format's error. It also shows the next problem. silentErrorDisable is placed right after set-numberic-format but raised error is surviving. It means that we it is wrong to clean the error list in silentErrorDisable method.

Please hard-code the expected behaviour within the test (see the nested_noerror.p test I've added, the checkError function). I'm not sure I understand your notes: when NO-ERROR clause is in effect, an ERROR condition is not raised; instead, the ERROR-STATUS reports details about the error which was suppressed (i.e. not raised) by the NO-ERROR clause.

> 4. The errors that are not set the error flag can be contained in usual or NO-ERROR error lists. It seems that such errors do not differ from other except the error flag.

Again, the ERROR-STATUS:ERROR flag is set only when NO-ERROR clause is in use; when an error is actually raised (or displayed on screen, if only a message is shown), then ERROR-STATUS is not affected. Wwhat do you mean by "usual error list"?

> 1. What should I test more?

Adjust your tests to include the expected behaviour and only show data to screen if data is not the expected one.

> 2. There are the warning attribute in ErrorManager.WorkArea. What does it mean? In the docs I only found WARNING attribute of COMPILE statement.. Is it relates to this attribute or means no to throw an Exception or it means just the errors, that are not set the error flag?

This warning field is only used internally by P2J: when a code normally would raise an ERROR condition (and the NO-ERROR clause is not present), if the code is used within a UI statement, then an ERROR condition is not raised, but the message is recorded.

> 3. There are several attributes like soapFault, error and so on. I suspect that we will need to have 2 copies of them for both lists: usual and NO-ERROR. But I don't know exactly behavior of them except the error and silent. What do you think?

Yes, I think a new container (named ErrorStatus) needs to be added (as a ErrorManager inner class), with the following fields:

1. error, to map the ERROR-STATUS:ERROR attribute. Replacement for WorkArea.error and WorkArea.pendingError.
2. soapFault, to map the ERROR-STATUS:ERROR-OBJECT-DETAIL attribute. Replacement for WorkArea.lastSoapFault and WorkArea.pendingSoapFault
3. errList, with the details about the recorded errors. Replacement for WorkArea.errList.

Also:

1. WorkArea.dontClean flag needs to be removed
2. there will be two ErrorStatus instance fields:

- WorkArea.errorStatus, with the current values reported by the ERROR-STATUS handle. (i.e. with the details about the "active" error list)
- WorkArea.pendingErrorStatus, used when NO-ERROR clause is in effect. (i.e. with the details about the "inactive" error list)

As I mentioned above, when silentErrorDisable is called I think WorkArea.errorStatus will be set to WorkArea.pendingErrorStatus and WorkArea.pendingErrorStatus will be re-initialized.

#11 - 06/18/2014 04:40 AM - Vadim Gindin

I've updated tests as you asked. In cases when actual error is raised there are no possibility to hardcode it to the procedure text, but for NO-ERROR cases it is possible and was done.

About point 3 in my previous note. I wanted to say the following thing. First of all the error is happen (with NO-ERROR) bar procedure and was survived till the end of the main procedure. After that the second error was happen in foo procedure after bar procedure call and it does not override the first error. This example just showed that there are 2 lists should be in implementation: for raised and not raised (NO-ERROR) errors.

#12 - 06/18/2014 08:46 AM - Constantin Asofiei

Vadim Gindin wrote:

> I've updated tests as you asked. In cases when actual error is raised there are no possibility to hardcode it to the procedure text

Yes, the ERROR-STATUS handle does not record info about the raised errors, but the _MSG function allows access to some details - but this is a different stuff.

Some notes about the error messages: how do you run the program? Because the error message ** bar /home/vig/err/55noerr_diff_err.p: Unable to evaluate field for assignment. (143) suggests that you run it like pro -p /home/vig/err/..., which is not the best way; instead, run the program directly, via pro -p noerr_diff_err.p, and the error message will be independent of the full path: ** bar noerr_diff_err.p: Unable to evaluate field for assignment. (143).

> About point 3 in my previous note. I wanted to say the following thing. First of all the error is happen (with NO-ERROR) bar procedure and was survived till the end of the main procedure. After that the second error was happen in foo procedure after bar procedure call and it does not override the first error. This example just showed that there are 2 lists should be in implementation: for raised and not raised (NO-ERROR) errors.

Take a look at the noerr_diff_err.p - the comments are confusing (I think they might be from a previous test). What I think you are referring is that the raised errors can't be added to the same list as the silenced errors (by the NO-ERROR clause): for this reason, we will need to separate the raised errors from the silenced errors. But leave that for later - first focus on the NO-ERROR case.

#13 - 06/25/2014 07:09 AM - Vadim Gindin

- File vig_upd20140625a.zip added

Take a look at first NO-ERROR implementation please. I want to understand if I am on a right way. Also note at the places of clearList() calls. I cant

set them to silentErrorDisable. Take a look at the test *noerr_diff_err.p*. The error that happened in the bar procedure survived till the end of main procedure, i.e. it survived through several silenErrorDisable() calls.

Separate question about separate error list. I thought, that implementation will have 2 separate list as it have in this update. One list is for NO-ERROR case, another - for raised errors (does it needed?). In current update errorList is used for NO-ERROR case and pendingErrorList is not used at all. It seems I don't fully understand your idea about them. Please explain in detail what you meant in previous notes. If there is no need of error list for raised error, than why we need pendingErrorList?

#14 - 06/26/2014 06:21 AM - Constantin Asofiei

Vadim Gindin wrote:

> Take a look at first NO-ERROR implementation please. I want to understand if I am on a right way. Also note at the places of clearList() calls. I cant set them to silentErrorDisable.

silentErrorDisable doesn't need to clear list, it just needs to switch the errorStatus with the pendingErrorStatus instance and re-initialized pendingErrorStatus. Why do you want to clear it?

> Take a look at the test *noerr_diff_err.p*. The error that happened in the bar procedure survived till the end of main procedure, i.e. it survived through several silenErrorDisable() calls.

No. You have only a NO-ERROR clause in that test. Thus, ERROR-STATUS will always report what happened the last time some code with NO-ERROR was invoked. Are you sure you are working with the same file as the one on bzr?

> Separate question about separate error list. I thought, that implementation will have 2 separate list as it have in this update. One list is for NO-ERROR case, another - for raised errors (does it needed?).

All the discussion was about the NO-ERROR case, which really needs to separate ErrorStatus instances (thus two error lists). For the raised errors, we will need a distinct list, where we will need to record them.

> In current update errorList is used for NO-ERROR case and pendingErrorList is not used at all. It seems I don't fully understand your idea about them. Please explain in detail what you meant in previous notes. If there is no need of error list for raised error, than why we need pendingErrorList?

Think about it this way: when a code with NO-ERROR is executed, the code has accessed to two different states of the ERROR-STATUS handle (again, I'm not talking about real raised ERROR conditions, I'm talking about ERROR conditions which were/need to be ignored, as the NO-ERROR is in effect):

- the 4GL application code has access only to the state as before this code was executed. This means that any errors recorded into ERROR-STATUS are seen by the NO-ERROR 4GL code (i.e. foo proc in a RUN foo NO-ERROR will see the ERROR-STATUS handle as before the RUN foo NO-ERROR was executed).
- another special state, to which we are referring as pending, and this state is used by the P2J runtime. If an ERROR condition is raised while the NO-ERROR clause is in effect, the P2J runtime will write it to the pending state; also, this pending state needs to be checked by the P2J runtime, when it needs to know if there are pending errors.
- when the NO-ERROR is statement is finished (i.e. silentErrorDisable is called), P2J needs to replace the current state with the pending state and clear the pending state. This ensures that subsequent code has access to the proper ERROR-STATUS state (the state from the last executed NO-ERROR statement).

#15 - 06/30/2014 03:05 PM - Vadim Gindin

- File err20140630a.log added

About *noerr_diff_err.p* It seems I made wrong conclusion. There are only one silentErrorDisable right after setCurrentWindow(h) call.

I'm trying to debug my current implementation and I faced with a problem during this process. See attached *err20140630a.log*. I converted *noerr_diff_err.p* and during the last call of checkError function in the bar function implementation the error is happen (line 118). The real place where the error is happen is the method BlockManager.checkJavaCall(..) line (6988). Here is the code:

```
            handle thisProc = ProcedureManager.thisProcedure();
            referent = thisProc.get();

            // we need to determine the legacy name of this call, so we can push it to the call
            // stack. fortunately, unlike the external program names, the internal entry names are
            // reported as they are defined, not called
            iename = ProcedureManager.resolveClosestMethod(referent, function, externalProgram);
```

referent variable becoming null and in the following method call NPE error is raised. How is that possible?

#16 - 07/01/2014 05:49 AM - Constantin Asofiei

Vadim Gindin wrote:

> I'm trying to debug my current implementation and I faced with a problem during this process. See attached *err20140630a.log*.

The noerr_diff_err.p test does not abend with the latest P2J rev. There must be something in your changes.

#17 - 07/01/2014 05:44 PM - Vadim Gindin

- File vig_upd20140701a.zip added

You're right. Thank you I managed this error and prepared the next update, that contains NO-ERROR implementation with your last remarks.

Two questions
1) Take a look at isPending, setPending, isPendingError methods. Did I correctly interpreted them after I removed pending flag from WorkArea?
2) I'm not sure of one moment about where to use work field and where to use da field form ErrorManager in my changed methods.

Please take a look.

#18 - 07/02/2014 03:38 AM - Constantin Asofiei

Review for 0701a.zip:

- please move ErrorManager.isPendingError back to line 1338
- the ErrorManager.setError APIs need to look like this:

```
  /**
    * Sets the error variable.  This is a no-op, as the ERROR-STATUS:ERROR attribute
    * is read-only.
    *
    * @param    value
    *           The new state of the variable.
    */
   @LegacyAttribute(name = "ERROR", setter = true, ignore = true)
   public static void setError(boolean value)
   {
      // no-op
   }
```

  They are added only because of the Errorable interface - ERROR-STATUS:ERROR is read-only.
- ErrorManager.clearList, RemoteErrorData.clearList, ServerDataAccess.setError, ServerDataAccess.clearList, RemoteErrorData.setError APIs need to be removed - they are no longer needed.
- otherwise, the logic looks good. I've checked your update with a testcase like this:

```
def var h as handle.

h = this-procedure.
h:add-super-procedure(h, search-target) no-error.
message error-status:num-messages error-status:get-message(1).
h:add-super-procedure(h, search-self) no-error.
message error-status:num-messages error-status:get-message(1).
h:add-super-procedure(h, search-target) no-error.
message error-status:num-messages error-status:get-message(1).
h:add-super-procedure(h, search-self) no-error.
message error-status:num-messages error-status:get-message(1).
```

  and the error list is cleared properly.

  Two questions
  1) Take a look at isPending, setPending, isPendingError methods. Did I correctly interpreted them after I removed pending flag from WorkArea?

Yes, they look OK.

  2) I'm not sure of one moment about where to use work field and where to use da field form ErrorManager in my changed methods.

da needs to be used in APIs which can be called from the P2J client side, too. The usage looks OK.

Next step: clean the code (remove the TODOs, add javadoc, etc) and get it regression tested.

#19 - 07/04/2014 03:32 PM - Vadim Gindin

- File vig_upd20140702a.zip added

Here is the current update, that passed regression tests. Only *gso_ctrlc_3way_tests* tests are failed (6/6). I'm not sure it is valuable.

#20 - 07/05/2014 11:53 AM - Vadim Gindin

I ran the *msg_test.p* and *msg_test2.p* several times and I found that there are no stack scope at all (stack scope which Greg mentioned in the note 7 of the task #1632. It seems the errors numbers stack is never gets cleared and there is no need to scope it. At least I can't find the case when it gets cleared. In the tests mentioned above there are several blocks such as several "do" blocks and calls of external and internal procedures with its own do blocks with own scopes. After execution all of them the error numbers stack contains all of the happened errors. Am I wrong?

#21 - 07/07/2014 02:43 AM - Constantin Asofiei

Vadim Gindin wrote:

> Here is the current update, that passed regression tests. Only *gso_ctrlc_3way_tests* tests are failed (6/6). I'm not sure it is valuable.

About the update:

- ErrorManager.setError(logical value) needs to have the same LegacyAttribute annotation as its boolean version.
- the history entry for RemoteErrorData should be Removed clearList and setError methods. methods (there is no ServerDataAccess class in RemoteErrorData).

#22 - 07/07/2014 03:41 AM - Constantin Asofiei

Vadim Gindin wrote:

> I ran the *msg_test.p* and *msg_test2.p* several times and I found that there are no stack scope at all (stack scope which Greg mentioned in the note 7 of the task #1632. It seems the errors numbers stack is never gets cleared and there is no need to scope it. At least I can't find the case when it gets cleared. In the tests mentioned above there are several blocks such as several "do" blocks and calls of external and internal procedures with its own do blocks with own scopes. After execution all of them the error numbers stack contains all of the happened errors. Am I wrong?

I've added notes to #1632, lets continue the discussion there.

#23 - 07/08/2014 03:04 AM - Vadim Gindin

- File vig_upd20140707a.zip added

Corrected update for note 8.

#24 - 07/08/2014 05:35 AM - Constantin Asofiei

Vadim Gindin wrote:

> Corrected update for note 8.

If all your tests work OK, go ahead and release it.

#25 - 07/09/2014 03:46 PM - Vadim Gindin

Committed to bzr. Rev #10565

#26 - 07/10/2014 10:43 AM - Greg Shah

Vadim: please open a new task to resolve the remaining issue (_MSG problems). Make sure that Constantin and myself are both added as watchers. In that task, document exactly what you have found, what you have done so far and what remains to be done. Post any testcases and code as needed to save off your current status. Then we are going to close this task.

#27 - 07/11/2014 07:49 AM - Greg Shah

- % Done changed from 0 to 100

- Status changed from New to Closed

#28 - 11/16/2016 12:07 PM - Greg Shah

- Target version changed from Milestone 11 to Cleanup and Stablization for Server Features

Files

vig_upd20140625a.zip	59.6 KB	06/25/2014	Vadim Gindin
err20140630a.log	28.4 KB	06/30/2014	Vadim Gindin
vig_upd20140701a.zip	16.3 KB	07/01/2014	Vadim Gindin
vig_upd20140702a.zip	16.2 KB	07/04/2014	Vadim Gindin
vig_upd20140707a.zip	16.2 KB	07/08/2014	Vadim Gindin