Database - Feature #2296

update record buffer validation to prevent database index overflow

05/02/2014 02:56 PM - Eric Faulhaber

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Ovidiu Maxiniuc	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	Cleanup and Stablization for Server Features		
billable:	No	**vendor_id:**	GCD

Description

Related issues:

Related to Database - Feature #2273: address remaining limitations of SQL Ser...	**Closed**

History

#1 - 05/02/2014 04:05 PM - Eric Faulhaber

This task is sort of a follow-on to #2273, but is scoped slightly differently. It is meant to add some generic Progress error-handling that we don't do today, which we will re-use to deal with the SQL Server 900 byte index limitation (and similar limits, if any, for other database implementations).

At certain points in the creation and update of a record in a RecordBuffer instance, we validate the data to be sure it does not violate any mandatory requirements or uniqueness constraints. We have to do this in order to mimic any related Progress errors, and to prevent invalid records from being handed off to Hibernate/JDBC for flushing (which otherwise would cause a SQL-level error and invalidate the current database connection and Hibernate session). As part of this validation, we also purposefully "leak" (i.e., share) uncommitted index updates across sessions to emulate a transaction isolation quirk/defect in Progress.

So, there is an infrastructure already in place in the P2J persistence framework which is aware of changes which would affect any database index associated with a particular record.

We need to add a feature to this infrastructure to validate an update/insert which would otherwise cause an index overflow error in either Progress (to emulate legacy behavior), or in the new backend database (to the extent a limitation on index length exists). Currently, based on research done in #2273, the limitation on index length in Progress appears to by ~188 bytes in certain circumstances (databases created with older versions) and ~1972 bytes in others (databases created with newer versions).

The idea is to add generic validation processing to the DMOValidator class (and associated classes as necessary), which calculates the length of the data in all indexes which are updated by a change to the record currently held in the associated buffer, and which reports a Progress-style error if that length exceeds the allowed length.

Due to the existing checks and the integration with the dirty database manager (i.e., the "leaking" mentioned above), we already have information as to which indexes are affected by a data change. What is missing is:

- an awareness of the maximum index length allowed for the current record buffer (this will be based on defaults in the code, the database dialect-wide, overarching limit (if any), and an optional configuration setting in the directory which specifies a custom limit for the associated database (must be the same or smaller than the dialect's limit));
- an efficient mechanism which knows how to calculate the combined length of the fields in each index affected by a data change;
- a reporting of the error messages and codes reported by Progress when it encounters this type of error.

Note that because this validation may apply stricter length limits than currently are enforced by Progress, this may change application behavior. For instance, in cases where the more lenient, 1972 byte limit is in use in the legacy application, converted applications which use SQL Server will reject records which would generate index lengths where 900 < index length < 1972. This may cause application problems. However, the customer for the current project has agreed that this is a reasonable limitation, given the choice to utilize SQL Server as the backend. If present, these issues will have to be addressed with application- or data-level changes, as this hard limit of a proprietary database implementation is not easily worked around in P2J.

There is an assumption in this approach that the combined lengths of fields in an index actually *can* exceed the maximum index length. In most cases, this will only be true for indexes containing columns of variable length data types (e.g., varchar). Indexes which contain only fixed-length columns whose combined length could never exceed the maximum index length do not need to be validated, of course.

#2 - 05/02/2014 04:28 PM - Eric Faulhaber

Some notes on determining the maximum index limit to apply and whether or not to perform the validation at all:

- We need to add a method to P2JDialect which reports the dialect-specific index length limit, if any. Dialects which have no such limit should report zero.
- If a limit is configured in the directory, it can only be the same or less than the dialect-mandated limit. This value should be read when the database is registered at server startup (or when it is registered as a transient database due to a CONNECT statement). If the configured value is larger than the dialect value, a warning should be written to the server log, which indicates the configuration error and notifies that the smaller of the two values will be used.
- The default behavior should be to disable this validation completely for any dialect which reports a zero limit (i.e., no index length limit). In most cases, an application is unlikely to have behavior dependent upon the legacy, 188/1972 byte limit, and there is no reason to add the overhead of the limit checks if the actual backend no longer has such limits. Thus, the validation should be designed such that it can be bypassed completely in this case. If an application needs the legacy checks, the directory should contain the appropriate configuration item and limit, and it will be honored.

#3 - 05/02/2014 04:39 PM - Eric Faulhaber

Since the validation algorithm will be hit quite frequently, it must be as efficient as possible:

- We should figure out the combined lengths of the fixed portions of indexes once and cache these values by index, so we don't have to recalculate them each time we validate.
- When we determine the length of a particular variable-length field, it should be cached short-term (life cycle should be the validation operation itself), so that if that same field appears in multiple indices, we only determine its length once.
- Note that part of the process of calculating the length of varchar data will be knowledge of the number of bytes a character consumes in the encoding used by the associated database. Again, this knowledge should be determined once per database and cached.

#4 - 05/05/2014 05:13 PM - Ovidiu Maxiniuc

- Status changed from New to WIP

Started implementation with initial configuration/validation and research on Progress behavior on this error. Not checked yet the 188 limit of older Progress versions.

Some thought about the validation routine:

- it will only be called on character fields assignments.
- the routine should be lazy, and cache the computed sizes of char fields during the lifetime of DMO. If all fields sizes are known (including fixed ones), when assigning a new value to a field recomputing the new length should be as simple as substracting the length of old value and adding the length of the newly assigned one.
- I believe that the sizes of fixed-length fields can be hardcoded in the respective dialect. I did not do any research but I don't think that this can be obtained dynamically at runtime (in database initialization); if so it would be a very dialect-specific sql query. What I am afraid of, is that even fixed-length fields might be stored using variable number of bytes (Eg. NULL values to take only one byte ?)

#5 - 05/06/2014 08:06 AM - Greg Shah

There is no such thing a a fixed length field in Progress. Any length specification in the database is advisory (i.e. it is only a hint). Without the index limit, you can store up to 32000 bytes in any character field, at any time.

#6 - 05/06/2014 10:43 AM - Ovidiu Maxiniuc

In order to reproduce the exact Progress behavior we need to count the bytes of a index-key, the same way Progress does and then act accordingly. The fields that are not part of an index are left out of this discussion.

I think we are dealing with two different units here:

- the P2JDialect property should reflect the allowed bytes for a index key in respective database (eg. 900 for SQL Server)
- the P4GL target emulated value read from directory (188 or 1970 bytes).

They are rather unrelated and probably should not be compared directly. From my tests of today, it look that Progress has some strange way of storing data in indexes. It seems to me that almost all data is stored in variable length (for example, an integer can be stored on 3 and up to 7 bytes, not a constant 4 bytes as expected). I will investigate the same on SQL Server.

#7 - 05/08/2014 03:55 PM - Ovidiu Maxiniuc

- File om_upd20140508a.zip added

Intermediary update attached. It contains the algorithm for validation that was should be a little changed.

The two index-sizes are rather different:

- The value form registry is the value P2J tries to emulate (display errors about index when the key of a record reach this length on any dialect, including PostgreSQL).
- The value returned by each dialect is the technical limitation and will be used for checking if no database errors will be generated when the record is flushed. A custom message will be printed in this case.

At this moment, H2 and PSQL do not have any known constraints from this point of view. In MSSQL there is the 900 bytes limitation. The good news is that the size of the index-key is easy calculable as the sum of sizes of fields the index is composed.

The big problem is finding the way P4GL stores these index-key within the allowable 1972 or 188 bytes limit. Even the fixed size datatypes are stored on variable byte count. I was able to reverse-engineering the space requirements for values of unknown, logical, character, int, int64 and date datatypes. At this moment I cannot do any estimations for datetime as I found some examples where values form within the same day need 7, 8 or even 12 bytes from the index key. Some particular cases can take as little as 2 bytes. Likewise for decimal, as it a somewhat composed datatype.

#8 - 05/09/2014 04:51 PM - Ovidiu Maxiniuc

- File om_upd20140509a.zip added

I found some more edge-cases that require special handling for emulation of p4gl behavior.
I was forced to rewrite the detection of affected indexes as the vData does not contain the correct lists of uniqueDirtyIndexes and dirtyIndexes when

current record is transient so the computation was not always performed.

#9 - 05/09/2014 05:29 PM - Eric Faulhaber

> I was forced to rewrite the detection of affected indexes as the vData does not contain the correct lists of uniqueDirtyIndexes and dirtyIndexes when current record is transient so the computation was not always performed.

Please explain in more detail the incorrect behavior you have found with vData. Is it a general problem for validation, or is it specific to the specialized index validation you are adding?

What have you discovered about how/when Progress does its index length validation? I would expect it to coincide with the other kinds of validation we are doing (i.e., mandatory value checks and unique constraint checks) at key points (like flushing to the database, at the end of assign brackets, and when explicitly requested. If index length validation happens at these same points, have you determined whether it is done before or after the other kinds of validation?

#10 - 05/12/2014 08:43 AM - Ovidiu Maxiniuc

Eric Faulhaber wrote:

> Please explain in more detail the incorrect behavior you have found with vData. Is it a general problem for validation, or is it specific to the specialized index validation you are adding?
>
> What have you discovered about how/when Progress does its index length validation? I would expect it to coincide with the other kinds of validation we are doing (i.e., mandatory value checks and unique constraint checks) at key points (like flushing to the database, at the end of assign brackets, and when explicitly requested. If index length validation happens at these same points, have you determined whether it is done before or after the other kinds of validation?

Apparently this is a problem related to index-key length calculation.

- For the TEMP-TABLES, the indexes are checked when the the record is about to be flushed to database. It does not matter if the affected field is changed in a simple assign statement or in a batch assign. In the case of TEMP-TABLES, triggers cannot be defined in p4gl so timing with ASSIGN/WRITE events cannot be relates but, most likely, the checking is performed after the moment when the WRITE trigger would be called. The index-key length is checked after checking the UNIQUE constraints because when a record with both defects is checked, the only ** tt1 already exists with 123456. (132) error message is displayed (twice). For certain, the index-key validation is NOT performed on simple or batch field assign.

```
FUNCTION f-ballast RETURNS CHARACTER (INPUT len1 AS INTEGER) :
   IF len1 EQ ?
      THEN RETURN ?.

   DEFINE VARIABLE tmp AS CHARACTER INIT "".

   DO WHILE len1 GT 0:
      len1 = len1 - 1.
      tmp = tmp + "a".
   END.

   RETURN tmp.
END FUNCTION.

define temp-table tt1
```

```
    field ballast   as character
    field f-logical as logical
    field f-integer as integer initial 123456
    index idx-integer as unique f-integer asc
    index idx-ballast ballast asc
    index idx-ballast-logical ballast asc f-logical asc
    index idx-ballast-integer ballast asc f-integer asc
    index idx-ballast-logical-integer ballast asc f-logical asc f-integer asc.

define variable pass as logical init No.

CREATE tt1.

/*
CREATE tt1.
DO ON STOP UNDO, LEAVE:
    display "setting ballast" skip.
    pause.
    tt1.ballast = f-ballast(1970 - 3 - 3).

    display "setting f-logical" skip.
    pause.
    tt1.f-logical = yes.

    display "validating" skip.
    pause.
    VALIDATE tt1.

    display "succes" skip.
    pause.
    pass = Yes. /* if validation is OK then the test passed */
END.

IF pass THEN
    DELETE tt1.
*/

CREATE tt1.
DO ON STOP UNDO, LEAVE:
    display "batch assigning" skip.
    pause.
    assign
        tt1.ballast = f-ballast(1970 - 3 - 3)
        tt1.f-logical = yes.

    display "validating" skip.
    pause.
    VALIDATE tt1.

    display "succes" skip.
    pause.
    pass = Yes. /* if validation is OK then the test passed */
END.
```

- For permanent tables, the index-checking is performed BEFORE the ASSIGN trigger was executed (the 4GL programmer does not have the chance the fix a value that is about to trigger the 11353 STOP condition). In fact the check is performed very early, in both assign cases (simple and batch) none of the UNIQUEness is displayed.

```
/*
 * NOTE:
 *    Make sure only the (idx-ballast and) idx-ballast-integer indexes are active!
 */

FUNCTION f-ballast RETURNS CHARACTER (INPUT len1 AS INTEGER) :
   IF len1 EQ ?
      THEN RETURN ?.

   DEFINE VARIABLE tmp AS CHARACTER INIT "".

   DO WHILE len1 GT 0:
      len1 = len1 - 1.
      tmp = tmp + "a".
   END.
```

```
   RETURN tmp.
END FUNCTION.

/* return 0 if result as expected, 1 on other outcome */
FUNCTION t-integer RETURNS INTEGER (
      INPUT test AS INTEGER,     /* testing integer values          */
      INPUT n AS INTEGER,        /* expected bytes taken by index-key */
      INPUT fail AS LOGICAL):    /* true if test is expected to fail  */

   DEFINE VARIABLE pass AS LOGICAL INITIAL No.

   /* cleanup */
   for each pt1:
      delete pt1.
   end.

   CREATE pt1.
   /* automatic flush, we need two records with same f-integer in order to
      break the UNIQUEss constraint of uidx-integer index*/
   pt1.f-integer = 2147483647. /* also need 9 bytes */

   CREATE pt1.

   pt1.f-integer = 2 + 147483647. /* also need 9 bytes */

/*
   VALIDATE pt1.
*/

   on assign of pt1.ballast do:
        display "assigning pt1.ballast" length(pt1.ballast) with frame f1 no-labels.
        /* pt1.ballast =  "eeeee". */
   end.

   DO ON STOP UNDO, LEAVE:

      display "batch assigning..." with frame f1 no-labels.
      pause.

      assign
         pt1.ballast = f-ballast((1971 - n) - 1) /* fill the index area all except n bytes */
         pt1.f-integer = test
         .

      display "assigning  ballast" skip with frame f1 no-labels.
      pause.
      pt1.ballast = f-ballast((1971 - n) - 1).

      display "assigning f-integer" skip with frame f1 no-labels.
      pause.
      pt1.f-integer = test.

      display "assigned, validating..." skip with frame f1 no-labels.
      pause.
      VALIDATE pt1.

      pass = Yes. /* if validation is OK then the test passed */
   END.

   DELETE pt1.

   IF pass EQ fail THEN
   DO:
        DISPLAY "Failed to store value [" test "] on" n "bytes" with frame f1 no-labels.
        RETURN 1.
   END.

   RETURN 0. /* return 0 if pass or fail as expected*/
END FUNCTION.

DEFINE VARIABLE errs AS INTEGER INITIAL 0.
DISPLAY "Testing integer datatype" with frame f1 no-labels.
errs = t-integer(2147483647, 8, Yes).
DISPLAY errs "errors." SKIP  with frame f1 no-labels.
```

#11 - 05/12/2014 12:41 PM - Ovidiu Maxiniuc

Why are not vData.dirtyUnique and vData.dirtyNonUnique correctly initialized, from the point of view of index checking?
When executing in the above example:
pt1.ballast = f-ballast((1971 - n) - 1).
The converted code is:
pt1.setBallast(fBallast(new integer(minus(minus(1971, n), 1))));
At execution time, the detectChange() will eventually call DMOValidator.validate() and then the DMOValidator.checkIndexMaxSize() is call with vData.dirtyUnique empty and vData.dirtyNonUnique containing only the idx_ballast because it's the only index with all field components changed (record is transient so fullMatch == true and no other field was yet touched). In the case of temp-tables, we got worse, as both lists are empty because in TempValidationHelper.listDirtyIndexes(), for non-unique indexes, the empty collection is returned.

As consequence, the members of vData only contains index information that is needed by other validation (not null and unique) - the list of indexes that are dirty from these points of view. To correctly check if the record does not pass the index-key limits I need the list of all affected indexes, meaning all indexes that the modified field (ballast in our case) is part of, so all the declared indexes have to be checked.

#12 - 05/19/2014 05:13 PM - Ovidiu Maxiniuc

During last days I encountered inconsistent readings about the errors I get and whether I get some errors:
I did some research in more detail, in parallel with normal (permanent tables - PT1) and temp tables (TT1). As you will see below there are lots of differences between the two kind of tables for same code executed.

Both tables have:

- a f-integer is defined with init value of 123456. It needs 5 bytes in the index key to be stored.
- a ballast character field that will be filled with 1967 characters so that the key will overflow, provoking the STOP condition.
- an index ballast-integer with fields ballast and f-integer for testing the index key length.
- an unique index uinteger with f-integer for testing the relation to unique index validation.

Before running the code fragments, a test record is created and f-integer = 123457. The fragments are executed on a new freshly CREATE d record .

Case 1: Independent assignments

Subcase 1.1 If no other fields were altered, VALIDATE / CREATE / RELEASE will report invalid index key size for both TT / PT:

```
pt1.ballast = f-ballast(1967).
validate pt1.
```

PT	The total length of the fields in an index exceeds max key size. Index ballast-integer of table p2j_test.pt1 (129) (11353)
TT	The total length of the fields in an index exceeds max key size. Index idx-ballast-integer of table (TEMP-TABLE)tt1 (129) (11353) The total length of the fields in an index exceeds max key size. Index idx-ballast-integer of table (TEMP-TABLE)tt1 (129) (11353)

Note the message is printed twice in the case of temp-tables.

Subcase 1.2. If other fields part of index was previously altered error is:

```
pt1.ballast = f-ballast(1967).
pt1.f-integer = 123457.
```

PT	The total length of the fields in an index exceeds max key size. Index ballast-integer of table p2j_test.pt1 (129) (11353)
TT	tt1 already exists with 123457. (132) tt1 already exists with 123457. (132) tt1 already exists with 123457. (132) tt1 already exists with 123457. (132)

For both PT and TT, the messages are generated by the second line. Note the message for TT is displayed 4 times.

Case 2: Batch assignments
In these cases no VALIDATE is necessary.

Subcase 2.1: unique constraint violated before index-key size

```
assign
   pt1.f-integer = 123457
   pt1.ballast = f-ballast(1967).
```

PT	pt1 already exists with f-integer 123457. (132) Cannot execute user-defined function, method or property accessor 'f-ballast' in an ASSIGN statement after an indexing error. (13917) pt1 already exists with f-integer 123457. (132)
TT	tt1 already exists with 123457. (132) Cannot execute user-defined function, method or property accessor 'f-ballast' in an ASSIGN statement after an indexing error. (13917) tt1 already exists with 123457. (132) tt1 already exists with 123457. (132) tt1 already exists with 123457. (132) tt1 already exists with 123457. (132)

This particular testcase gives me something new about the insides of how Progress executes batch assignments. I need to dig deeper here. If, for example, the ballast will be assigned a shorter value everything would be ok. However, because the validation occurred after the first assignment from batch, the subsequent assignments are not executed, so the record is invalidated even if all assignments from batch would bring the record to a valid internal state.

Subcase 2.2: unique constraint violated after index-key size

```
assign
   pt1.ballast = f-ballast(1967)
   pt1.f-integer = 123457.
```

PT	The total length of the fields in an index exceeds max key size. Index ballast-integer of table p2j_test.pt1(129) (11353)
TT	tt1 already exists with 123457. (132) tt1 already exists with 123457. (132) tt1 already exists with 123457. (132) tt1 already exists with 123457. (132)

Subcase 2.3: What if the index-key size condition does not occur when setting the ballast ? (0 as integer need only 2 bytes in the key instead of 5 for 123457) The condition is caused by the second assignment in batch.

```
tt1.f-integer = 0.
assign
   pt1.ballast = f-ballast(1967)
   pt1.f-integer = 123457.
```

PT	The total length of the fields in an index exceeds max key size. Index ballast-integer of table p2j_test.pt1 (129) (11353)
TT	tt1 already exists with 123457. (132) The total length of the fields in an index exceeds max key size. Index idx-ballast-integer of table (TEMP-TABLE)tt1 (129) (11353) The total length of the fields in an index exceeds max key size. Index

	idx-ballast-integer of table (TEMP-TABLE)tt1 (129) (11353) The total length of the fields in an index exceeds max key size. Index idx-ballast-integer of table (TEMP-TABLE)tt1 (129) (11353)

#13 - 05/23/2014 02:50 PM - Ovidiu Maxiniuc

- File om_upd20140523a.zip added

Uploaded om_upd20140523a.zip for review. It is merged with latest (10536) bzr rev.
The big issue here is that I could not find the exact pattern that Progress uses for storing the index components in the index key. The algorithm for integer/int64/date are close but I still could accidentally find values that do not fit the pattern.
The other archive contains my main testcases along with a .df export for testing permanent table.
As workaround for the unique null issue of SQL server, the ID is added in the dialect for unique indexes, too.

#14 - 05/23/2014 02:58 PM - Ovidiu Maxiniuc

- File om_upd20140523b.zip added

#15 - 05/23/2014 03:05 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> As workaround for the unique null issue of SQL server, the ID is added in the dialect for unique indexes, too.

I thought we had agreed this was not a viable approach, since it makes duplicate rows unique.

#16 - 05/23/2014 03:15 PM - Ovidiu Maxiniuc

Eric Faulhaber wrote:

> Ovidiu Maxiniuc wrote:
>
> > As workaround for the unique null issue of SQL server, the ID is added in the dialect for unique indexes, too.
>
> I thought we had agreed this was not a viable approach, since it makes duplicate rows unique.

This is true, I only added it (and documented this in the code) as a quick fix until we find a correct solution. Otherwise some of the complex tests will stop early. I will investigate this issue further with the #2186 and #2142.

#17 - 05/26/2014 02:35 PM - Ovidiu Maxiniuc

The initial regression test failed. There were no changes in generated ddl and no TRPL rules files affected for the conversion.
CTRL+C: failed to connect to remote db (known issue) and some other tests I could not identify the cause
Main: only tc set had about 10 fails. The majority were on same driver thread and and some being 'dependency chain'. Looking into the process list I observed a hanging client. I had to -9 it, it could't be terminate by simple kill.
I have restarted the runtime regression test.

#18 - 05/27/2014 09:06 AM - Ovidiu Maxiniuc

The second execution took extremely long to finish.
There were a couple less failures in CTRL+C part.
The main part finished with errors in gc testset. The tc, that previously was blocked on driver 4, now only had 2 fails: the tc_job_002 and tc_job_clock_002. I consider it has passed.

The 3rd time I run only the ctrl+c part. Yet, irrelevant results.

#19 - 05/27/2014 12:55 PM - Eric Faulhaber

Code review 20140523a:

Update looks good. The findings you documented in the code comments regarding the Progress widths of certain fields are quite bizarre. Nice work tracking these down.

Are any of the Ctrl+C failures you are encountering in your regression testing attributable to new stop conditions in this update?

#20 - 05/27/2014 01:48 PM - Ovidiu Maxiniuc

Eric Faulhaber wrote:

> Code review 20140523a:
>
> Update looks good. The findings you documented in the code comments regarding the Progress widths of certain fields are quite bizarre. Nice work tracking these down.

Yes, I was expecting some constant values for fixed-size data. The pattern I found have yet exceptions. Unfortunately I was unable to find patterns for all datatypes. The pattern limits are proved by the update attached in note 14 above.

> Are any of the Ctrl+C failures you are encountering in your regression testing attributable to new stop conditions in this update?

At first thought, they should't. The STOP conditions only occur on assign and write-events. Constantin informed me that there are no such events in the CTRL+C testing. So this should not be the cause. Because the long running tests (unusual 180+ minutes) he also suggested to run the test plan directly, so removing the overhead of an extra jvm used by ant. Apparently he is right, after the run, the gso_ctrlc_tests ended flawlessly, only 4 fails in the gso_ctrlc_3way_tests: a database connection and 3 (probably) subsequent semaphore timeouts.

What do you think of the change in RecordBuffer.java:10134, namely restricting the flush() only when the commitNow is to be executed ? This may also be related to #2222.

#21 - 05/27/2014 02:33 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> What do you think of the change in RecordBuffer.java:10134, namely restricting the flush() only when the commitNow is to be executed ? This may also be related to #2222.

The reasoning behind the change appears sound; if it has passed regression testing, I think we can check it in. Did tc_job_clock_002 pass in at least one of the rounds of testing? While this fails frequently due to timing, it should pass eventually (unlike tc_job_002, which always fails at step 40).

#22 - 05/28/2014 09:59 AM - Ovidiu Maxiniuc

Here is the summary of all the 8 tests:
20140523_172349 (CTR+C): massive fails
20140523_195225 (main): all test sets pass but tc_tests. tc_job_clock_002 passes.
20140526_093148 (CTRL+C): massive fails
20140526_115359 (main): some fails in gc_tests, tc_tests has only two fails: tc_job_002 and tc_job_clock_002
20140527_042409 (CTRL+C): massive fails
20140527_080513 (CTRL+C): massive fails
20140527_121558 (CTRL+C): gso_ctrlc_tests passes, 6 fails in gso_ctrlc_3way_tests
20140527_134000 (CTRL+C): gso_ctrlc_tests fails but gso_ctrlc_3way_tests passes

In conclusion, the update should be clear so I will do the check in.

#23 - 05/28/2014 12:49 PM - Ovidiu Maxiniuc

The update was committed to bzr in rev 10537 and distributed by mail.

#24 - 05/30/2014 04:58 PM - Eric Faulhaber

- % Done changed from 0 to 100

- Status changed from WIP to Closed

#25 - 11/16/2016 12:07 PM - Greg Shah

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

Files

om_upd20140508a.zip	174 KB	05/08/2014	Ovidiu Maxiniuc
om_upd20140509a.zip	175 KB	05/09/2014	Ovidiu Maxiniuc
om_upd20140523a.zip	160 KB	05/23/2014	Ovidiu Maxiniuc
om_upd20140523b.zip	16.3 KB	05/23/2014	Ovidiu Maxiniuc