Database - Feature #1947

Feature # 1669 (Closed): add support for additional database methods

add conversion support for database methods

01/14/2013 03:42 AM - Eric Faulhaber

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Vadim Nebogatov	**% Done:**	100%
Category:		**Estimated time:**	180.00 hours
Target version:	Conversion Support for Server Features		
billable:	No	**version:**	
vendor_id:	GCD		

Description

History

#1 - 01/14/2013 03:56 AM - Eric Faulhaber

- Assignee set to Vadim Nebogatov

- Target version set to Milestone 4

The following are database, handle-based methods that we need to support (in priority order):

- BUFFER-FIELD (KW_BUF_FLD)
- GET-BUFFER-HANDLE (KW_GET_BUFH)
- BUFFER-COPY (KW_BUF_COPY)
- REPOSITION-TO-ROWID (KW_REPOS_2I)
- FIND-FIRST (KW_FIND_1ST)
- QUERY-CLOSE (KW_QRY_CLOS)
- BUFFER-RELEASE (KW_BUF_REL)
- FIND-BY-ROWID (KW_FIND_BR)
- FIND-UNIQUE (KW_FIND_UNI)
- BUFFER-COMPARE (KW_BUF_COMP)
- ~~DISABLE-LOAD-TRIGGERS (KW_DIS_L_TR)~~ [reassigned to #1949]
- FIND-LAST (KW_FIND_LST)
- GET-CURRENT (KW_GET_CUR)
- REPOSITION-FORWARD (KW_REPOS_F)
- REPOSITION-TO-ROW (KW_REPOS_2R)

For purposes of this task, we only need to support the conversion of these methods, and the result must compile, but the runtime does not need to work yet (so we need to add the appropriate runtime stub methods/classes, but the implementation should be deferred).

#2 - 01/15/2013 08:34 PM - Eric Faulhaber

Vadim Nebogatov wrote:

> Please check if I am on the correct way.
>
> I have started implementation for BUFFER-FIELD (KW_BUF_FLD).
>
> 4GL test is created with BUFFER-FIELD like
>
> ```
> ...
> CREATE BUFFER somebuf FOR TABLE "tt".
> …
> somefields[i] = somebuf:BUFFER-FIELD(i).
> ...
> ```
>
> Expected changes for KW_BUF_FLD support are:
>
> methods-attributes.rules
>
> ```
> <rule>ftype == prog.kw_buf_fld
> <action>methodText = "getFieldBuffer"</action>
> <action>castcls = "P2JQuery"</action>
> </rule>
> ```
>
> P2JQuery interface (with empty implementations in AbstractQuery, QueryWrapper, QueryWrapper.DefaultDelegate)
>
> ```
> public BaseDataType getFieldBuffer(NumberType fieldNumber);
> ```
>
> One problem so far is that following not-lines
>
> ```
> somebuf;
> "tt";
> ```
>
> as result of conversion of
>
> ```
> CREATE BUFFER somebuf FOR TABLE "tt".
> ```

Your results with CREATE BUFFER are expected; we do not yet support the dynamic creation of a buffer (see #1654). To get a buffer handle, I would suggest trying something like:

```
DEFINE VARIABLE hbuf as HANDLE.
DEFINE TEMP-TABLE tt ...
...
DEFINE BUFFER somebuf FOR tt.
hbuf = BUFFER somebuf:HANDLE.
...
```

Your approach with P2JQuery and the related query classes is more appropriate for query-related methods (e.g., QUERY-CLOSE). However, buffer-related methods like BUFFER-FIELD should probably be using the RecordBuffer class instead. Please see other cases of conversion to RecordBuffer methods in method_attributes.rules for examples of this.

#3 - 01/17/2013 06:37 PM - Vadim Nebogatov

```
Changes:

method_attributes.rules, line 926, after <rule>ftype == prog.kw_y:

               <rule>ftype == prog.kw_buf_fld
                  <action>methodText = "getFieldBuffer"</action>
                  <action>castcls = "RecordBuffer"</action>
               </rule>

               <rule>ftype == prog.kw_get_bufh
                  <action>methodText = "getBufferHandle"</action>
                  <action>castcls = "RecordBuffer"</action>
               </rule>

               <rule>ftype == prog.kw_buf_copy
                  <action>methodText = "bufferCopy"</action>
                  <action>castcls = "RecordBuffer"</action>
               </rule>

               <rule>ftype == prog.kw_repos_2i
                  <action>methodText = "repositionByID"</action>
                  <action>castcls = "P2JQuery"</action>
               </rule>

               <rule>ftype == prog.kw_find_1st
                  <action>methodText = "findFirst"</action>
                  <action>castcls = "RecordBuffer"</antion>
               </rule>

RecordBuffer.java, line 3335, after method buffer():

   public void bufferCopy(handle bufferHandle)
   {
      //todo
      BufferReference dstRec = (BufferReference)bufferHandle;
      copy(this, dstRec, true);
   }

   public void findFirst(String whereClause,  NumberType iLock)
   {
      // create query and using P2JQuery.first()
   }

   public BaseDataType getFieldBuffer(NumberType fieldNumber)
   {
      return null;//todo
   }

   public handle getBufferHandle(int fieldNumber)
   {
      return null;//todo
   }
```

#4 - 01/17/2013 07:32 PM - Eric Faulhaber

RecordBuffer is something of an unusual class, in that instances of it are never exposed to converted business logic. Thus, converting buffer handle 4GL methods to public instance methods in RecordBuffer won't work. Please follow the example of prog.kw_avail in methods_attributes.rules, which converts the AVAILABLE 4GL attribute to the static method RecordBuffer.isAvailable. Please read the class-level javadoc for RecordBuffer, which attempts to explain the role of this class. I realize the static method approach is clumsy, and as RecordBuffer becomes more cluttered with new static methods to support the many 4GL methods and attributes, we may try to revisit this design, but not in time for Milestone 4.

On the other hand, the approach you have used with KW_REPOS_2I is appropriate, because classes which implement P2JQuery are actually instantiated by converted business logic.

When adding new token type rules to methods_attributes.rules, we have been following the convention of sorting the rules in alphabetical order by token name for ease of maintenance. Please continue this tradition.

#5 - 01/18/2013 03:16 PM - Vadim Nebogatov

```
All handle-based methods return LOGICAL. I think statement based analogues throw errors on execution time when
 FALSE is returned for corresponding handle-based methods. Is it correct?
For example, for handle-based QUERY-CLOSE we could reuse code used for CLOSE QUERY statement. But what to do w
ith returning code?
Method QueryWrapper.close() does not throw any exceptions.

   public static logical queryClose(Object object)
   {
      ((QueryWrapper)object).close();
      return new logical(true);//todo ?
   }

P2JQuery.repositionByID(...) methods also return void
```

#6 - 01/18/2013 06:20 PM - Vadim Nebogatov

```
Will P2J support conversion of methods with variable parameters amount? I saw in P2J conversion reference book
 that  SHARE-LOCK, …, NO-WAIT  are not supported in several statements so far.

For example, FIND-BY-ROWID  method description is:

FIND-BY-ROWID ( rowid
[,
{ SHARE-LOCK | EXCLUSIVE-LOCK
[ , NO-WAIT ]
])

Adding RecordBuffer method

   public static logical findByRowId(Object object, rowid rowId)
   {
```

```
      ...
   }
```

supports conversion of

hBuffer:FIND-BY-ROWID(r).

Also adding RecordBuffer method

```
   public static logical findByRowId(Object object, rowid rowId, NumberType iLock)
   {
      ...
   }
```

supports conversion of

```
DEFINE VARIABLE lock AS INTEGER.
hBuffer:FIND-BY-ROWID(r, lock).
```

But: adding both these methods does not allow conversion of both these 4GL statements.

#7 - 01/18/2013 06:32 PM - Vadim Nebogatov

Latest variant

RecordBuffer.java:

```
   public static handle getBufferHandle(Object queryRef, int bufferSequenceNumber)
   {
      return null;//todo
   }

   public static BaseDataType getBufferField(Object bufRef, int fieldNumber)
   {
      return null;//todo
   }

   public static BaseDataType getBufferField(Object bufRef, String fieldName)
   {
      return null;//todo
   }

   public static logical bufferRelease(Object bufRef)
   {
      release((BufferReference)bufRef);
      return new logical(true);//todo ?
   }

   public static logical bufferCopy(Object bufDstRef, Object bufSrcRef)
   {
      copy((BufferReference)bufSrcRef, (BufferReference)bufDstRef, true);
      return new logical(true);//todo ?
   }
```

```
   public static logical bufferCompare(Object bufDstRef, Object bufSrcRef)
   {
      logical result = new logical();
      compare((BufferReference)bufSrcRef, (BufferReference)bufDstRef, result);
      return result;
   }

   public static void findFirst(Object bufRef, String whereClause)
   {
      new FindQuery((BufferReference) bufRef, "'" + whereClause + "'", null, null).first();
   }

   public static void findLast(Object bufRef, String whereClause)
   {
      new FindQuery((BufferReference) bufRef, "'" + whereClause + "'", null, null).last();   }

   public static void reposition(Object queryRef, rowid rowId)
   {
      ((P2JQuery)queryRef).reposition(new recid(rowId.getValue()));
   }

   public static logical findByRowId(Object bufRef, rowid rowId)
   {
      return null;//todo
   }

   public static logical queryClose(Object queryRef)
   {
      ((P2JQuery)queryRef).close();
      return new logical(true);//todo ?
   }
```

methods_attributes.rules: (will be sorted later as Eric wrote)

```
               <rule>ftype == prog.kw_buf_fld
                  <action>methodText = "RecordBuffer.getBufferField"</action>
                  <action>methodType = java.static_method_call</action>
               </rule>

               <rule>ftype == prog.kw_get_bufh
                  <action>methodText = "RecordBuffer.getBufferHandle"</action>
                  <action>methodType = java.static_method_call</action>
               </rule>

               <rule>ftype == prog.kw_buf_copy
                  <action>methodText = "RecordBuffer.bufferCopy"</action>
                  <action>methodType = java.static_method_call</action>
               </rule>

               <rule>ftype == prog.kw_buf_comp
                  <action>methodText = "RecordBuffer.bufferCompare"</action>
                  <action>methodType = java.static_method_call</action>
               </rule>

               <rule>ftype == prog.kw_repos_2i
                  <action>methodText = "repositionByID"</action>
                  <action>castcls = "P2JQuery"</action>
               </rule>

               <rule>ftype == prog.kw_find_1st
                  <action>methodText = "RecordBuffer.findFirst"</action>
                  <action>methodType = java.static_method_call</antion>
               </rule>

               <rule>ftype == prog.kw_find_lst
                  <action>methodText = "RecordBuffer.findLast"</action>
                  <action>methodType = java.static_method_call</action>
               </rule>

               <rule>ftype == prog.kw_find_br
                  <action>methodText = "RecordBuffer.findByRowId"</action>
                  <action>methodType = java.static_method_call</action>
               </rule>
```

```
                <rule>ftype == prog.kw_qry_clos
                   <action>methodText = "RecordBuffer.queryClose"</action>
                   <action>methodType = java.static_method_call</action>
                </rule>

                <rule>ftype == prog.kw_buf_rel
                   <action>methodText = "RecordBuffer.bufferRelease"</action>
                   <action>methodType = java.static_method_call</action>
                </rule>

                <rule>ftype == prog.kw_find_uni
                   <action>methodText = "unique"</action>
                   <action>castcls = "P2JQuery"</antion>
                </rule>

                <rule>ftype == prog.kw_get_cur             <rule>ftype ==               <rule>ftype == prog.kw_b
uf_fld
                   <action>methodText = "RecordBuffer.getBufferField"</action>
                   <action>methodType = java.static_method_call</action>
                </rule>
prog.kw_buf_fld
                   <action>methodText = "RecordBuffer.getBufferField"</action>
                   <action>methodType = java.static_method_call</action>
                </rule>

                   <action>methodText = "current"</action>
                   <action>castcls = "P2JQuery"</action>
                </rule>

                <rule>ftype == prog.kw_repos_f
                   <action>methodText = "forward"</action>
                   <action>castcls = "P2JQuery"</action>
                </rule>

                <rule>ftype == prog.kw_repos_2r
                   <action>methodText = "RecordBuffer.reposition"</action>
                   <action>methodType = java.static_method_call</action>
                </rule>
```

#8 - 01/19/2013 03:58 PM - Vadim Nebogatov

Updated latest variant above

#9 - 01/20/2013 12:23 AM - Eric Faulhaber

Vadim Nebogatov wrote:

> All handle-based methods return LOGICAL. I think statement based analogues throw errors on execution time when FALSE is returned for corresponding handle-based methods. Is it correct?

In many cases, yes. But not all handle-based methods return LOGICAL. Some return other types, like HANDLE, etc.

> For example, for handle-based QUERY-CLOSE we could reuse code used for CLOSE QUERY statement. But what to do with returning code? Method QueryWrapper.close() does not throw any exceptions.

```
public static logical queryClose(Object object)
{
   ((QueryWrapper)object).close();
   return new logical(true);//todo ?
}
```

In which class would this example method reside?

In this case, I would suggest catching the error and returning FALSE, otherwise TRUE, as in:

```
public QueryWrapper
{
   ...

   public logical queryClose()
   {
      boolean success = true;

      try
      {
         close();
      }
      catch (ErrorConditionException exc)
      {
         success = false;
      }

      return new logical(success);
   }

   ...
}
```

But please remember we are not implementing the runtime methods at this point (though this one is pretty short, so it wouldn't hurt to do it now).

Also note that in this case, we are using an instance method (non-RecordBuffer referent), not a static method.

Actually, in contradiction to my earlier comment about using static methods with RecordBuffer, Greg and I decided Friday to re-implement the way buffers are referenced in the converted business logic, so we **will** actually be implementing methods, attributes, and builtin functions for buffers as instance methods, not static methods. However, I am not yet finished with this change, so please continue as you have been. I apologize that this will require a bit of rework after I'm done (hopefully tomorrow). I will open a separate Redmine issue for this work, explaining the changes.

#10 - 01/20/2013 12:52 AM - Eric Faulhaber

Vadim Nebogatov wrote:

> Will P2J support conversion of methods with variable parameters amount?

For methods which have optional parameters like [SHARE_LOCK | EXCLUSIVE_LOCK , [NO_WAIT]], we probably will need special rules to generate the correct Java parameter, which would be one of the following:

```
LockType.NONE
LockType.SHARE
LockType.SHARE_NOWAIT
LockType.EXCLUSIVE
LockType.EXCLUSIVE_NOWAIT
```

This will probably require checking if the current AST (i.e., the one with token type KW_FIND_BR) has children and, if so, determining what they are, in order to determine the right choice of LockType. You would then create a peer AST node of that type, which would naturally emit as a parameter of the converted method. This would probably best be done in a function, since these lock type options are likely to appear in more than one method. In any event, NumberType is not the correct Java parameter type here.

Please post a snippet of the AST XML file here that shows the structure of the branch below KW_FIND_BR, for a test case which includes a lock type option (including the NO-WAIT option). Also post the appropriate section of the .p.cache file, so I can correlate the AST with the code.

#11 - 01/20/2013 10:27 AM - Greg Shah

To be clear, the problem with 4GL methods that take "parameters" like SHARE-LOCK or NO-WAIT, is that these things are not normally allowed in 4GL expressions. You can't just pass SHARE-LOCK to your own user-defined function or use it in a calculation. So Progress has created some kind of special cases in this instance. It is unusual, but unfortunately you are the lucky one to have to deal with it.

The place we currently handle the lock types is in convert/database_access.rules:

```
      <!-- process lock types -->
      <rule>
         (type == prog.kw_no_lock or
          type == prog.kw_sh_lock or
          type == prog.kw_exc_lock) and
         (parent.type == prog.record_phrase or
          parent.type == prog.kw_get)
```

```
         <rule>type == prog.kw_no_lock
            <action>locktype = "LockType.NONE"</action>
         </rule>
         <rule>type == prog.kw_sh_lock
            <action>locktype = "LockType.SHARE"</action>
```

```
         </rule>
         <rule>type == prog.kw_exc_lock
            <action>locktype = "LockType.EXCLUSIVE"</action>
         </rule>

         <!-- add no wait option if needed (only valid for share and
              exclusive locks) -->
         <rule>
            parent.getNumImmediateChildren(prog.kw_no_wait) &gt; 0  and
            type != prog.kw_no_lock
            <action>locktype = sprintf("%s_NO_WAIT", locktype)</action>
         </rule>

      </rule>
```

If it can work, it may be best to handle it there.

#12 - 01/21/2013 01:00 PM - Vadim Nebogatov

Eric Faulhaber wrote:

```
public static logical queryClose(Object object)
   {
      ((QueryWrapper)object).close();
      return new logical(true);//todo ?
   }
In which class would this example method reside?
```

I think all query based methods returning logical should be added to P2JQuery interface:

public logical queryClose();
public logical queryUnique();
public logical queryCurrent();
public logical queryForward(int rows);

Old methods – close(),unique(), current(), forward(...) also should be left in this interface.
Implementation of new methods should be based on old ones: new methods catch ErrorConditionException and return logical

#13 - 01/21/2013 01:04 PM - Eric Faulhaber

Agreed (w/ entry 12).

#14 - 01/21/2013 04:25 PM - Vadim Nebogatov

Latest variant

P2JQuery.java:

```
   public logical queryClose();

   public logical queryUnique();

   public logical queryCurrent();

   public logical queryForward(int rows);

   public logical queryReposition(rowid rowId);

   public logical queryRepositionByID(rowid rowId);
```

methods_attributes.rules: (will be sorted later as Eric wrote)

```
                <rule>ftype == prog.kw_buf_fld
                   <action>methodText = "RecordBuffer.getBufferField"</action>
                   <action>methodType = java.static_method_call</action>
                </rule>

                <rule>ftype == prog.kw_get_bufh
                   <action>methodText = "RecordBuffer.getBufferHandle"</action>
                   <action>methodType = java.static_method_call</action>
                </rule>

                <rule>ftype == prog.kw_buf_copy
                   <action>methodText = "RecordBuffer.bufferCopy"</action>
                   <action>methodType = java.static_method_call</action>
                </rule>

                <rule>ftype == prog.kw_buf_comp
                   <action>methodText = "RecordBuffer.bufferCompare"</action>
                   <action>methodType = java.static_method_call</action>
                </rule>

                <rule>ftype == prog.kw_buf_rel
                   <action>methodText = "RecordBuffer.bufferRelease"</action>
                   <action>methodType = java.static_method_call</action>
                </rule>

                <rule>ftype == prog.kw_find_1st
                   <action>methodText = "RecordBuffer.findFirst"</action>
                   <action>methodType = java.static_method_call</action>
                </rule>

                <rule>ftype == prog.kw_find_lst
                   <action>methodText = "RecordBuffer.findLast"</action>
                   <action>methodType = java.static_method_call</action>
                </rule>

                <rule>ftype == prog.kw_find_br
                   <action>methodText = "RecordBuffer.findByRowId"</action>
                   <action>methodType = java.static_method_call</action>
                </rule>

                <rule>ftype == prog.kw_qry_clos
                   <action>methodText = "queryClose"</action>
                   <action>castcls = "P2JQuery"</action>
                </rule>

                <rule>ftype == prog.kw_find_uni
                   <action>methodText = "queryUnique"</action>
                   <action>castcls = "P2JQuery"</antion>
                </rule>
```

```
            <rule>ftype == prog.kw_get_cur
               <action>methodText = "queryCurrent"</action>
               <action>castcls = "P2JQuery"</action>
            </rule>

            <rule>ftype == prog.kw_repos_f
               <action>methodText = "queryForward"</action>
               <action>castcls = "P2JQuery"</action>
            </rule>

            <rule>ftype == prog.kw_repos_2r
               <action>methodText = "queryReposition"</action>
               <action>castcls = "P2JQuery"</action>
            </rule>

            <rule>ftype == prog.kw_repos_2i
               <action>methodText = "queryRepositionByID"</action>
               <action>castcls = "P2JQuery"</antion>
            </rule>
```

RecordBuffer.java:

```
public static handle getBufferHandle(Object queryRef, int bufferSequenceNumber)
{
   return null;//todo
}

public static BaseDataType getBufferField(Object bufRef, int fieldNumber)
{
   return null;//todo
}

public static BaseDataType getBufferField(Object bufRef, String fieldName)
{
   return null;//todo
}

public static logical bufferRelease(Object bufRef)
{
   release((BufferReference)bufRef);
   return new logical(true);//todo ?
}

public static logical bufferCopy(Object bufDstRef, Object bufSrcRef)
{
   copy((BufferReference)bufSrcRef, (BufferReference)bufDstRef, true);
   return new logical(true);//todo ?
}

public static logical bufferCompare(Object bufDstRef, Object bufSrcRef)
{
   logical result = new logical();
   compare((BufferReference)bufSrcRef, (BufferReference)bufDstRef, result);
   return result;
}

public static void findFirst(Object bufRef, String whereClause)
{
   new FindQuery((BufferReference) bufRef, "'" + whereClause + "'", null, null).first();
}

public static void findLast(Object bufRef, String whereClause)
{
   new FindQuery((BufferReference) bufRef, "'" + whereClause + "'", null, null).last();
}

public static logical findByRowId(Object bufRef, rowid rowId)
{
   return null;//todo
}
```

AbstractQuery.java:

```
   public logical queryClose()
   {
      return queryClose(this);
   }

   public logical queryUnique()
   {
      return queryUnique(this);
   }

   public logical queryCurrent()
   {
      return queryCurrent(this);
   }

   public logical queryForward(int rows)
   {
      return queryForward(this, rows);
   }

   public logical queryReposition(rowid rowId)
   {
      return queryReposition(this, rowId);
   }

   public logical queryRepositionByID(rowid rowId)
   {
      return queryRepositionByID(this, rowId);
   }

   // static methods are for reusing in P2JQuery implementations

   public static logical queryClose(P2JQuery p2JQuery)
   {
      boolean success = true;

      try
      {
         p2JQuery.close();
      }
      catch (ErrorConditionException exc)
      {
         success = false;
      }

      return new logical(success);
   }

   public static logical queryUnique(P2JQuery p2JQuery)
   {
      boolean success = true;

      try
      {
         p2JQuery.unique();
      }
      catch (ErrorConditionException exc)
      {
         success = false;
      }

      return new logical(success);
   }

   public static logical queryCurrent(P2JQuery p2JQuery)
   {
      boolean success = true;

      try
```

```
      {
         p2JQuery.current();
      }
      catch (ErrorConditionException exc)
      {
         success = false;
      }

      return new logical(success);
   }

   public static logical queryForward(P2JQuery p2JQuery, int rows)
   {
      boolean success = true;

      try
      {
         p2JQuery.forward(rows);
      }
      catch (ErrorConditionException exc)
      {
         success = false;
      }

      return new logical(success);
   }

   public static logical queryReposition(P2JQuery p2JQuery, rowid rowId)
   {
      boolean success = true;

      try
      {
         p2JQuery.reposition(new recid(rowId.getValue()));
      }
      catch (ErrorConditionException exc)
      {
         success = false;
      }

      return new logical(success);
   }

   public static logical queryRepositionByID(P2JQuery p2JQuery, rowid rowId)
   {
      boolean success = true;

      try
      {
         p2JQuery.repositionByID(new recid(rowId.getValue()));
      }
      catch (ErrorConditionException exc)
      {
         success = false;
      }

      return new logical(success);
   }
```

QueryWrapper.java and QueryWrapper.DefaultDelegate:

```
   public logical queryClose()
   {
      return AbstractQuery.queryClose(this);
   }

   public logical queryUnique()
   {
      return AbstractQuery.queryUnique(this);
   }

   public logical queryCurrent()
   {
```

```
      return AbstractQuery.queryCurrent(this);
   }

   public logical queryForward(int rows)
   {
      return AbstractQuery.queryForward(this, rows);
   }

   public logical queryReposition(rowid rowId)
   {
      return AbstractQuery.queryReposition(this, rowId);
   }

   public logical queryRepositionByID(rowid rowId)
   {
      return AbstractQuery.queryRepositionByID(this, rowId);
   }
```

#15 - 01/22/2013 08:33 AM - Vadim Nebogatov

4GL supports assigning EXCLUSIVE-LOCK, SHARE-LOCK, NO-LOCK, NO-WAIT to INTEGER as specific INTEGER values:

EXCLUSIVE-LOCK (6207)
SHARE-LOCK (6208)
NO-LOCK (6209)
NO-WAIT (6090)

and passing INTEGER to handle-based methods parameters, for example:

```
DEFINE VARIABLE lock AS INTEGER.
lock = SHARE-LOCK.
DEFINE VARIABLE nowait AS INTEGER.
nowait = NO-WAIT.
hBuffer:FIND-BY-ROWID(rowid, lock, nowait).
```

Not-proper integer values give errors.

For lock:

FIND-BY-ROWID, FIND-FIRST etc., second argument must be NO-LOCK, SHARE-LOCK or EXCLUSIVE-LOCK. (7344)

For nowait:

FIND method argument must be NO-WAIT, or 0 or ?. (7343)

So, handle-based methods could support EXCLUSIVE-LOCK, SHARE-LOCK, NO-LOCK, NO-WAIT as integer parameter with similar validation (throwing ErrorConditionException or return true/false).

Current P2J seems doesn't support assigning these values to INTEGER, not-compilable code is conversion result:

```
lock.assign();
nowait.assign();
```

```
I also think what if use these specific integer values for LockType constants,

private static final int LT_NONE = 6209;

...
```

#16 - 01/22/2013 08:58 AM - Vadim Nebogatov

```
Additionally to above comment. Three methods

   public static logical findByRowId(Object bufRef, rowid rowId)
   {

   }

   public static logical findByRowId(Object bufRef, rowid rowId, integer lock)
   {

   }

   public static logical findByRowId(Object bufRef, rowid rowId, integer lock, integer nowait)
   {

   }
could support conversion of

FIND-BY-ROWID ( rowid
[,
{ SHARE-LOCK | EXCLUSIVE-LOCK
[ , NO-WAIT ]
])

Explicit occurrences of EXCLUSIVE-LOCK, SHARE-LOCK, NO-LOCK, NO-WAIT should be mapped to corresponding INTEGER
 values with the rules like Greg wrote
```

#17 - 01/22/2013 12:00 PM - Greg Shah

Interesting finding! We have seen some other cases where Progress provides some 4GL options as a compiler constant. I suspect this is what they are doing here.

That means we have to convert these sub-expression cases differently from their use as options (e.g. in a statically defined query like a FOR statement).

In the parser (progress.g) you will have to make these changes:

1. Look in the tokens {} array for the place where we define custom tokens for literals. Search for BOOL_TRUE to find this.

In there add:

```
NO_LOCK_LITERAL;
SHARE_LOCK_LITERAL;
EXCLUSIVE_LOCK_LITERAL;
NO_WAIT_LITERAL;
```

2. In the "literal" rule in the parser (progress.g - line 23118 in the version checked into bzr), add code that is something like this (add it just below the "| UNKNOWN_VAL" line:

```
        | nl:KW_NO_LOCK { #nl.setType(NO_LOCK_LITERAL); }
        | sl:KW_SH_LOCK { #sl.setType(SHARE_LOCK_LITERAL); }
        | el:KW_EXC_LOCK { #el.setType(EXCLUSIVE_LOCK_LITERAL); }
        | nw:KW_NO_WAIT { #nw.setType(NO_WAIT_LITERAL); }
```

This code matches the keywords as a literal, but then rewrites the token type to be a unique type that can be unambiguously matched in our TRPL rules.

Also, please document the compiler constant values in the javadoc for that rule the way we have done with others.

In rules/include/report.rules, you have to update the "literal_classify" function to recognize/decode the new *_LITERAL types.

In rules/include/common-progress.rules, you have to update the "literals" function to match the new *_LITERAL types.

In rules/convert/literals.rules, you would add code similar to how we handle BOOL_TRUE/BOOL_FALSE, except you would match on the new *_LITERAL types. On output we probably need to statically import the right constants from the com/goldencode/p2j/persist/LockType class where the constants are implemented as you previously suggested.

In regard to your method signatures for findByRowId(), you need additional overloaded versions where the lock and nowait options can be int instead of integer. Our approach is to create overloaded methods for all possible combinations of data types that can occur in the 4GL. Since the last 2 parms are option and we emit numeric literals as int, this means that we must provide 7 variants:

```
public static logical findByRowId(Object bufRef, rowid rowId)
public static logical findByRowId(Object bufRef, rowid rowId, integer lock)
public static logical findByRowId(Object bufRef, rowid rowId, int lock)
public static logical findByRowId(Object bufRef, rowid rowId, integer lock, integer nowait)
public static logical findByRowId(Object bufRef, rowid rowId, integer lock, int nowait)
public static logical findByRowId(Object bufRef, rowid rowId, int lock, integer nowait)
public static logical findByRowId(Object bufRef, rowid rowId, int lock, int nowait)
```

Also, I am pretty sure Eric will not want you to use Object as the first parameter type. But I will let him comment on his own.

#18 - 01/22/2013 12:14 PM - Eric Faulhaber

Greg Shah wrote:

> Also, I am pretty sure Eric will not want you to use Object as the first parameter type. But I will let him comment on his own.

Yes, please use DataModelObject instead.

Couldn't we replace the following method variants:

```
public static logical findByRowId(DataModelObject bufRef, rowid rowId, int lock)
public static logical findByRowId(DataModelObject bufRef, rowid rowId, int lock, int nowait)
```

with

```
public static logical findByRowId(DataModelObject bufRef, rowid rowId, LockType lockType)
```

?

These two variants are meant to support the case where we have literal integers for lock and nowait, correct? So, shouldn't we have enough information at conversion time to figure out which LockType value to use here?

#19 - 01/22/2013 02:00 PM - Vadim Nebogatov

Eric Faulhaber wrote:

```
These two variants are meant to support the case where we have literal integers for lock and nowait, correct?
```

Yes

```
So, shouldn't we have enough information at conversion time to figure out which LockType value to use here?
```

If P2J will allow assigning EXCLUSIVE-LOCK, SHARE-LOCK, NO-LOCK, NO-WAIT to INTEGER variable, this information will be dynamic and not

known at conversion time.
For example, 4GL understands
loc = NO-LOCK – 2.
as EXCLUSIVE-LOCK

#20 - 01/22/2013 05:43 PM - Vadim Nebogatov

Eric Faulhaber wrote:

```
Yes, please use DataModelObject instead.
```

But handle.get() returns Object and we will receive compilation error:
[javac] /home/vmn/p2j/src/com/goldencode/testcases/testBuffer.java:66: cannot find symbol
[javac] symbol : method findByRowId(java.lang.Object,com.goldencode.p2j.util.rowid)
[javac] location: class com.goldencode.p2j.persist.RecordBuffer
[javac] log.assign(RecordBuffer.findByRowId(hBuffer.get(), r));

#21 - 01/23/2013 05:31 PM - Vadim Nebogatov

Updated variant for new model (with using Buffer interface) without LOCKs and NO-WAIT support so far.

methods_attributes.rules:

```
<rule>ftype == prog.kw_buf_fld
   <action>methodText = "bufferField"</action>
   <action>castcls = "Buffer"</action>
</rule>

<rule>ftype == prog.kw_get_bufh
   <action>methodText = "bufferHandle"</action>
   <action>castcls = "Buffer"</action>
</rule>

<rule>ftype == prog.kw_buf_copy
   <action>methodText = "bufferCopy"</action>
   <action>castcls = "Buffer"</action>
</rule>

<rule>ftype == prog.kw_buf_comp
   <action>methodText = "bufferCompare"</action>
   <action>castcls = "Buffer"</action>
</rule>

<rule>ftype == prog.kw_buf_rel
   <action>methodText = "bufferRelease"</action>
   <action>castcls = "Buffer"</action>
</rule>

<rule>ftype == prog.kw_find_1st
   <action>methodText = "findFirst"</action>
   <action>castcls = "Buffer"</action>
</rule>

<rule>ftype == prog.kw_find_lst
   <action>methodText = "findLast"</action>
```

```
            <action>castcls = "Buffer"</action>
         </rule>

         <rule>ftype == prog.kw_find_br
            <action>methodText = "findByRowId"</action>
            <action>castcls = "Buffer"</action>
         </rule>

         <rule>ftype == prog.kw_qry_clos
            <action>methodText = "queryClose"</action>
            <action>castcls = "P2JQuery"</action>
         </rule>

         <rule>ftype == prog.kw_find_uni
            <action>methodText = "queryUnique"</action>
            <action>castcls = "P2JQuery"</action>
         </rule>

         <rule>ftype == prog.kw_get_cur
            <action>methodText = "queryCurrent"</action>
            <action>castcls = "P2JQuery"</action>
         </rule>

         <rule>ftype == prog.kw_repos_f
            <action>methodText = "queryForward"</action>
            <action>castcls = "P2JQuery"</action>
         </rule>

         <rule>ftype == prog.kw_repos_2r
            <action>methodText = "queryReposition"</action>
            <action>castcls = "P2JQuery"</action>
         </rule>

         <rule>ftype == prog.kw_repos_2i
            <action>methodText = "queryRepositionByID"</action>
            <action>castcls = "P2JQuery"</antion>
         </rule>
```

Buffer:

```
public BaseDataType bufferField(int fieldNumber);

public BaseDataType bufferField(String fieldName);

public handle bufferHandle(int bufferSequenceNumber);

public logical bufferRelease();

public logical bufferCopy(handle bufHandle);

public logical bufferCompare(handle bufHandle);

public void findFirst(String whereClause);

public void findLast(String whereClause);

public logical findByRowId(rowid rowId);

public logical findByRowId(rowid rowId, integer lock);

public logical findByRowId(rowid rowId, integer lock, integer nowait);
```

BufferImpl:

```
public BaseDataType bufferField(int fieldNumber)
{
   return null;//todo
}
```

```
public BaseDataType bufferField(String fieldName)
{
   return null;//todo
}

public handle bufferHandle(int bufferSequenceNumber)
{
   return null;//todo
}

public logical bufferRelease()
{
   release();
   return new logical(true);//todo ?
}

public logical bufferCopy(handle bufHandle)
{
   RecordBuffer.copy((BufferImpl) bufHandle.get(), this, true);
   return new logical(true);//todo ?
}

public logical bufferCompare(handle bufHandle)
{
   logical result = new logical();
   RecordBuffer.compare((BufferImpl) bufHandle.get(), this, result);
   return result;
}

public void findFirst(String whereClause)
{
   new FindQuery(this, "'" + whereClause + "'", null, null).first();
}

public void findLast(String whereClause)
{
   new FindQuery(this, "'" + whereClause + "'", null, null).last();
}

public logical findByRowId(rowid rowId)
{
   return null;//todo
}

public logical findByRowId(rowid rowId, integer lock)
{
   return null;//todo
}

public logical findByRowId(rowid rowId, integer lock, integer nowait)
{
   return null;//todo
}
```

P2JQuery:

```
public logical queryClose();

public logical queryUnique();

public logical queryCurrent();

public logical queryForward(int rows);

public logical queryReposition(rowid rowId);

public logical queryRepositionByID(rowid rowId);
```

AbstractQuery:

```
   public logical queryClose()
   {
      return queryClose(this);
   }

   public logical queryUnique()
   {
      return queryUnique(this);
   }

   public logical queryCurrent()
   {
      return queryCurrent(this);
   }

   public logical queryForward(int rows)
   {
      return queryForward(this, rows);
   }

   public logical queryReposition(rowid rowId)
   {
      return queryReposition(this, rowId);
   }

   public logical queryRepositionByID(rowid rowId)
   {
      return queryRepositionByID(this, rowId);
   }

   // static methods are for reusing in P2JQuery implementations

   public static logical queryClose(P2JQuery p2JQuery)
   {
      boolean success = true;

      try
      {
         p2JQuery.close();
      }
      catch (ErrorConditionException exc)
      {
         success = false;
      }

      return new logical(success);
   }

   public static logical queryUnique(P2JQuery p2JQuery)
   {
      boolean success = true;

      try
      {
         p2JQuery.unique();
      }
      catch (ErrorConditionException exc)
      {
         success = false;
      }

      return new logical(success);
   }

   public static logical queryCurrent(P2JQuery p2JQuery)
   {
      boolean success = true;

      try
      {
         p2JQuery.current();
```

```
      }
      catch (ErrorConditionException exc)
      {
         success = false;
      }

      return new logical(success);
   }

   public static logical queryForward(P2JQuery p2JQuery, int rows)
   {
      boolean success = true;

      try
      {
         p2JQuery.forward(rows);
      }
      catch (ErrorConditionException exc)
      {
         success = false;
      }

      return new logical(success);
   }

   public static logical queryReposition(P2JQuery p2JQuery, rowid rowId)
   {
      boolean success = true;

      try
      {
         p2JQuery.reposition(new recid(rowId.getValue()));
      }
      catch (ErrorConditionException exc)
      {
         success = false;
      }

      return new logical(success);
   }

   public static logical queryRepositionByID(P2JQuery p2JQuery, rowid rowId)
   {
      boolean success = true;

      try
      {
         p2JQuery.repositionByID(new recid(rowId.getValue()));
      }
      catch (ErrorConditionException exc)
      {
         success = false;
      }

      return new logical(success);
   }
```

QueryWrapper and QueryWrapper.DefaultDelegate:

```
   public logical queryClose()
   {
      return AbstractQuery.queryClose(this);
   }

   public logical queryUnique()
   {
      return AbstractQuery.queryUnique(this);
   }

   public logical queryCurrent()
   {
```

```
      return AbstractQuery.queryCurrent(this);
   }

   public logical queryForward(int rows)
   {
      return AbstractQuery.queryForward(this, rows);
   }

   public logical queryReposition(rowid rowId)
   {
      return AbstractQuery.queryReposition(this, rowId);
   }

   public logical queryRepositionByID(rowid rowId)
   {
      return AbstractQuery.queryRepositionByID(this, rowId);
   }
```

#22 - 01/24/2013 04:36 PM - Vadim Nebogatov

I still do not quite good understand why EXCLUSIVE-LOCK child is not added to method signature to be mapped wi th existing Buffer methods.

4GL fragment:

hBuffer:FIND-BY-ROWID(r, EXCLUSIVE-LOCK).

Ast fragment:

```
           <ast col="15" id="12884902119" line="40" text="FIND-BY-ROWID" type="METH_LOGICAL">
              <annotation datatype="java.lang.Long" key="oldtype" value="1271"/>
              <ast col="29" id="12884902120" line="40" text="r" type="VAR_ROWID">
                <annotation datatype="java.lang.Long" key="oldtype" value="2333"/>
                <annotation datatype="java.lang.Long" key="refid" value="12884902070"/>
                <annotation datatype="java.lang.Long" key="peerid" value="34359738505"/>
              </ast>
              <ast col="32" id="12884902121" line="40" text="EXCLUSIVE-LOCK" type="VAR_INT">
                <annotation datatype="java.lang.Long" key="oldtype" value="550"/>
                <annotation datatype="java.lang.Long" key="tempidx" value="204"/>
              </ast>
            </ast>
```

But parameter for EXCLUSIVE-LOCK is not generated:

((Buffer) hBuffer.get()).findByRowId(r);

If I use variable or constant, method is converted properly.

One more question. What if make some child processing in corresponding rule, for example:

```
               <rule>ftype == prog.kw_find_br
                  <action>methodText = "findByRowId"</action>
                  <action>castcls = "Buffer"</action>
                  <variable name="c1" type="com.goldencode.ast.Aast" />
                  <action>c1 = analog.getChildAt(1)</action>
                  <rule> { compare c1!=null and  c1.text with EXCLUSIVE-LOCK and other constants }
```

```
                  <action> {change child type or even replace child} </action>
                </rule>
         { the same for 2nd child }
              </rule>
```

```
Is it correct place for such changes?
```

#23 - 01/24/2013 05:27 PM - Eric Faulhaber

Regarding why the EXCLUSIVE-LOCK parameter does not emit: have you made the parser changes Greg mentions above (in entry 17)? From the AST snippet you provided, it doesn't appear so, unless I've misunderstood Greg's entry.

Regarding changing child nodes: I don't know if methods_attributes.rules is the best place to make these changes. Greg: as the original author of this rule set, can you comment?

Note that when you do make changes to a child node, make sure you are operating on the copy tree, not the this tree. When we run a rule set through the PatternEngine, it walks the source (i.e., this) tree, but it has a copy of the tree in memory on which we make changes (accessed with the copy reference in TRPL).

#24 - 01/24/2013 05:48 PM - Vadim Nebogatov

Yes, I missed some parts of #17 and only added int/integer parameters combinations

#25 - 01/24/2013 05:51 PM - Greg Shah

```
<ast col="32" id="12884902121" line="40" text="EXCLUSIVE-LOCK" type="VAR_INT">
```

This node should not be a VAR_INT type. It is not a variable. It is a kind of literal (compiler constant).

```
One more question. What if make some child processing in corresponding rule, for example:
```

```
              <rule>ftype == prog.kw_find_br
                 <action>methodText = "findByRowId"</action>
                 <action>castcls = "Buffer"</action>
                 <variable name="c1" type="com.goldencode.ast.Aast" />
                 <action>c1 = analog.getChildAt(1)</action>
                 <rule> { compare c1!=null and  c1.text with EXCLUSIVE-LOCK and other constants }
                   <action> {change child type or even replace child} </action>
                 </rule>
         { the same for 2nd child }
              </rule>
```

```
Is it correct place for such changes?
```

No, it isn't.

Please see my note 17 above. You must make parser (progress.g) changes in at least 2 places. The result will be that the text for share-lock which is normally matched as a KW_SHARE_LOCK type will now be SHARE_LOCK_LITERAL. That will emit naturally by making changes in the literals.rules as I mentioned above.

#26 - 01/25/2013 02:18 PM - Vadim Nebogatov

After I went through #17 everything started work.
Examples of converted code:

```
         integer lock = new integer(0);
         integer nowait = new integer(0);
...
            ((Buffer) hBuffer.get()).findByRowId(r, LockType.LT_EXCLUSIVE, LockType.LT_NO_WAIT);
            ((Buffer) hBuffer.get()).findByRowId(r, LockType.LT_SHARE, LockType.LT_NO_WAIT);
            ((Buffer) hBuffer.get()).findByRowId(r, LockType.LT_NONE, LockType.LT_NO_WAIT);
            ((Buffer) hBuffer.get()).findByRowId(r, LockType.LT_EXCLUSIVE);
            ((Buffer) hBuffer.get()).findByRowId(r, LockType.LT_SHARE);
            ((Buffer) hBuffer.get()).findByRowId(r, LockType.LT_NONE);
            ((Buffer) hBuffer.get()).findByRowId(r, LockType.LT_EXCLUSIVE, nowait);
            ((Buffer) hBuffer.get()).findByRowId(r, LockType.LT_SHARE, nowait);
            ((Buffer) hBuffer.get()).findByRowId(r, LockType.LT_NONE, nowait);
            ((Buffer) hBuffer.get()).findByRowId(r, lock, LockType.LT_NO_WAIT);
            ((Buffer) hBuffer.get()).findByRowId(r, LockType.LT_EXCLUSIVE, nowait);
            ((Buffer) hBuffer.get()).findByRowId(r, lock, nowait);
            ((Buffer) hBuffer.get()).findByRowId(r);
            ((Buffer) hBuffer.get()).findByRowId(r, 444, nowait);
```

I made the following additional changes.

LockType:

```
   /** Internal constant for lock type NONE, corresponds to 4GL internal value */
   public static final int LT_NONE = 6209;

   /** Internal constant for lock type SHARE, corresponds to 4GL internal value  */
   public static final int LT_SHARE = 6208;

   /** Internal constant for lock type EXCLUSIVE, corresponds to 4GL internal value  */
   public static final int LT_EXCLUSIVE = 6207;

   /** Internal constant for NO-WAIT, corresponds to 4GL internal value  */
   public static final int LT_NO_WAIT = 6090;
```

literals.rules (in corresponding places)

```
               <rule>
                  (type == prog.no_lock_literal or type == prog.share_lock_literal or
                  type == prog.exclusive_lock_literal or type == prog.no_wait) and
                  ref.type == prog.var_int
                  <action>needsWrap = false</action>
               </rule>

               <rule>
                  (type == prog.no_lock_literal or type == prog.share_lock_literal or
                  type == prog.exclusive_lock_literal or type == prog.no_wait)
                  <action>classname = "integer"</action>
               </rule>
            <!-- NO-LOCK literal -->
            <rule>type == prog.no_lock_literal
               <action>
                  createPeerAst(java.num_literal, "LockType.LT_NONE", parentid)
               </action>
            </rule>

            <!-- SHARE-LOCK literal -->
            <rule>type == prog.share_lock_literal
               <action>
                  createPeerAst(java.num_literal, "LockType.LT_SHARE", parentid)
               </action>
            </rule>

            <!-- EXCLUSIVE-LOCK literal -->
```

```
            <rule>type == prog.exclusive_lock_literal
               <action>
                  createPeerAst(java.num_literal, "LockType.LT_EXCLUSIVE", parentid)
               </action>
            </rule>
```

```
            <!-- NO-WAIT literal -->
            <rule>type == prog.no_wait_literal
               <action>
                  createPeerAst(java.num_literal, "LockType.LT_NO_WAIT", parentid)
               </action>
            </rule>
```

common-progress.rules

```
      <function name="literals">
         <parameter name="ttype" type="java.lang.Integer" />
         ttype == prog.string      or ttype == prog.num_literal  or
         ttype == prog.dec_literal or ttype == prog.bool_true    or
         ttype == prog.bool_false  or ttype == prog.date_literal or
         ttype == prog.unknown_val  or ttype == prog.no_lock_literal or
         ttype == prog.share_lock_literal  or ttype == exclusive_lock_literal or
         ttype == prog.no_wait_literal
      </function>
```

report.rules

```
         <rule>ttype == prog.no_lock_literal
            <action>tname = "NO-LOCK"</action>
         </rule>
```

```
         <rule>ttype == prog.share_lock_literal
            <action>tname = "SHARE-LOCK"</action>
         </rule>
```

```
         <rule>ttype == prog.exclusive_lock_literal
            <action>tname = "EXCLUSIVE-LOCK"</action>
         </rule>
```

```
         <rule>ttype == prog.no_wait_literal
            <action>tname = "NO-WAIT"</action>
         </rule>
```

Buffer – different types combinations

```
   public logical findByRowId(rowid rowId);

   public logical findByRowId(rowid rowId, int lock);

   public logical findByRowId(rowid rowId, integer lock);

   public logical findByRowId(rowid rowId, integer lock, integer nowait);

   public logical findByRowId(rowid rowId, int lock, integer nowait);

   public logical findByRowId(rowid rowId, int lock, int nowait);

   public logical findByRowId(rowid rowId, integer lock, int nowait);
```

progress.g (in corresponding places)

```
…
   NO_LOCK_LITERAL;
   SHARE_LOCK_LITERAL;
   EXCLUSIVE_LOCK_LITERAL;
   NO_WAIT_LITERAL;
…
```

…

```
          | UNKNOWN_VAL
          | nl:KW_NO_LOCK { #nl.setType(NO_LOCK_LITERAL); }
          | sl:KW_SH_LOCK { #sl.setType(SHARE_LOCK_LITERAL); }
          | el:KW_EXC_LOCK { #el.setType(EXCLUSIVE_LOCK_LITERAL); }
          | nw:KW_NO_WAIT { #nw.setType(NO_WAIT_LITERAL); }
          | KW_SAX_COMP
```

…

Greg Shah wrote:

```
On output we probably need to statically import the right constants from the com/goldencode/p2j/persist/LockTy
pe class where the constants are implemented as you previously suggested.
```

I am not sure if it is required, converted code is compilable now.

#27 - 01/26/2013 05:50 PM - Vadim Nebogatov

Two questions.

1) Methods

FIND-FIRST (predicate-expression[, lockmode[, wait-mode]])
FIND-LAST (predicate-expression[, lockmode[, wait-mode]])
FIND-UNIQUE (predicate-expression[, lockmode[, wait-mode]])
GET-CURRENT (NO-LOCK | SHARE-LOCK [, NO-WAIT] | EXCLUSIVE-LOCK [, NO-WAIT])

support locks and no-wait.

Will we add these parameters to interface?

For their implementation it was supposed reusing FindQuery().first(), FindQuery().last(), FindQuery().first(), p2JQuery.unique(), p2JQuery.current() having no parameters.

2) REPOSITION-TO-ROWID ({rowid1 [, rowid2| rowid-array] ...})

Now it is supported only for one parameter. Should we support array parameters too?

#28 - 01/28/2013 10:11 AM - Eric Faulhaber

Vadim Nebogatov wrote:

> Two questions.
>
> 1) Methods
>
> FIND-FIRST (predicate-expression[, lockmode[, wait-mode]])
> FIND-LAST (predicate-expression[, lockmode[, wait-mode]])
> FIND-UNIQUE (predicate-expression[, lockmode[, wait-mode]])
> GET-CURRENT (NO-LOCK | SHARE-LOCK [, NO-WAIT] | EXCLUSIVE-LOCK [, NO-WAIT])
>
> support locks and no-wait.
>
> Will we add these parameters to interface?
>
> For their implementation it was supposed reusing FindQuery().first(), FindQuery().last(), FindQuery().first(), p2JQuery.unique(), p2JQuery.current() having no parameters.

For the FIND variants, do you mean for the conversion to look something like:

```
new FindQuery(<usual args (other than LockType)>).{first|last|unique}(lockmode [, waitmode]);
```

? If so, I agree.

...and for GET-CURRENT:

```
query.current([LockType.[NONE|SHARE|SHARE_NO_WAIT|EXCLUSIVE|EXCLUSIVE_NO_WAIT]);
```

? In what case would P2JQuery.unique() come into play for one of these?

> 2) REPOSITION-TO-ROWID ({rowid1 [, rowid2| rowid-array] ...})
>
> Now it is supported only for one parameter. Should we support array parameters too?

The existing conversion and runtime signature:

```
P2JQuery.repositionByID(rowid, rowid...)
```

should support multiple, individual, rowid parameters already (i.e., the first case of rowid1 [, rowid2 [, ...]], but you are right, it will not support an array. It was originally written to support the REPOSITION to ROWID language statement, which does not have such an option. Yes, please add support for the array option.

#29 - 01/28/2013 03:04 PM - Vadim Nebogatov

Eric Faulhaber wrote:

```
In what case would P2JQuery.unique() come into play for one of these?
```

I confused with P2JQuery.unique(). FIND-UNIQUE is buffer-handle based method, corrected

#30 - 01/31/2013 03:07 PM - Vadim Nebogatov

- File vmn_upd20130231_test_cases.zip added

- File vmn_upd20130231.zip added

Two updates are attached:

```
1) vmn_upd20130231.zip - support database methods
2) vmn_upd20130231_test_cases.zip - associated test cases
```

#31 - 01/31/2013 05:11 PM - Vadim Nebogatov

- File isue1668.ods added

Attached table with information about supported attributes

#32 - 01/31/2013 11:58 PM - Eric Faulhaber

Please merge all of your code in these updates (primarily the code update) with the latest code in bzr and re-submit as soon as possible. It is too difficult for me to review because your files are based on an outdated version, especially methods_attributes.rules. I can't easily tell which diffs are due to your intentional changes and which are due to the base file being out of date.

When naming update archives, please use the naming convention vmn_updYYYYMMDD<letter>.zip, where <letter> is a for the first update of a given day, b is the second, c the third, and so on...

#33 - 02/05/2013 01:18 PM - Vadim Nebogatov

On 02/01/2013 02:49 PM, Vadim wrote:

```
Hello Eric,

It seems "castcls =" does not work after I updated sources from Bazaar.

Caused by: com.goldencode.expr.ExpressionException: Expression error castcls = "Buffer"]
    at com.goldencode.expr.Expression.getCompiledInstance(Expression.java:627)
    at com.goldencode.expr.Expression.execute(Expression.java:325)
    at com.goldencode.p2j.pattern.Rule.apply(Rule.java:401)

Do you know, what could be the reason?

Or should I use "hwrap =" instead? It works for interfaces, not for FieldReference, and converts to another co
```

```
de.

Regards,
Vadim
```

On 01.02.2013 23:53, Greg Shah wrote:

```
Vadim,

Yes, hwrap is now the replacement for castcls.  You should specify the name of the interface that contains the
 method you will be calling.  The code is generated differently, but it should generally work for you without
changes.

In regard to FieldReference, that may be the implementation object instance that is stored inside the handle,
but the interface is what needs to be specified for hwrap.

Greg
```

On 02/04/2013 01:48 PM, Vadim wrote:

```
Hello Greg,
I still cannot understand
hwrap = "Window" (Window is not an interface),
hwrap = "Procedure" - there are no such interface in code.
All my tests passed earlier, both getter and setter based ones. Now, when I used hwrap = "FieldReference", get
ter based tests passed, but setter tests do not pass.
For example, in case
hField:COLUMN-LABEL= ch.
setter
public void changeColumnLabel(character columnLabel) in FieldReference class
is not found.
Also the line which was converted properly earlier
hBuffer:BUFFER-VALUE(4) = hBuffer:BUFFER-VALUE(6).
is converted now to wrong code
4;
handle.unwrapFieldReference(hBuffer).changeValue(6, handle.unwrapFieldReference(hBuffer).value());
Thanks,
Vadim
```

On 05.02.2013 12:26, Constantin Asofiei wrote:

```
Vadim,

There is a pending update which solves some issues related to attributes/methods accessed via handles. The fix
 is at task #1920 (last one attached to comments) - you can use it until I release it officially (the fix is c
urrently under regression testing).
```

On 05.02.2013 19:18, Eric Faulhaber wrote:

```
Vadim,

Did Constantin's response and interim update get you past this problem?

Thanks,
Eric
```

On 02/05/2013 12:44 PM, Vadim wrote:

```
Eric,
After merge with Constantin's update.
Problem with converting
hBuffer:BUFFER-VALUE(4) = hBuffer:BUFFER-VALUE(6).
is fixed.
Problem with converting
hField:COLUMN-LABEL= ch.
still is not fixed.
What about problem with converting
ch = hField:STRING-VALUE(1).
Result is
[javac] Compiling 3 source files to /home/vmn/p2j/build/classes
[javac] /home/vmn/p2j/src/com/goldencode/testcases/Vadim22.java:75: cannot find symbol
[javac] symbol : method assign(int,com.goldencode.p2j.util.character)
[javac] location: class com.goldencode.p2j.util.character
[javac] ch.assign(1, handle.unwrapFieldReference(hField).stringValue());
[javac] ^
[javac] 1 error
Should I add corresponding assign(int, ...) method to character class?
Thanks,
Vadim
```

On 05.02.2013 21:51, Greg Shah wrote:

```
Vadim,

The problem seems to be this:

ch.assign(1, handle.unwrapFieldReference(hField).stringValue());

I think it probably needs to be this:

ch.assign(handle.unwrapFieldReference(hField).stringValue(1));

Greg
```

#34 - 02/06/2013 12:02 PM - Vadim Nebogatov

- File vmn_upd20130205a.zip added

- File vmn_upd20130205a_test_cases.zip added

Two updates are attached:

1) vmn_upd20130205a.zip - support database methods and attributes
2) vmn_upd20130205a_test_cases.zip - associated test cases

#35 - 02/06/2013 03:00 PM - Eric Faulhaber

- % Done changed from 0 to 50

Code review 20130205a:

- It seems that you must have changed handle.java, but it was not included in the update.
- In the archive, the file uast/progress.g should be located at src/com/goldencode/p2j/uast/progress.g, so that it is applied properly when unzipped.
- All "hwrap" types in methods_attributes.rules must be interfaces, not concrete implementation classes. For example, an hwrap of "FieldReference" refers to a concrete class. It should refer to an interface that defines the methods which represent the converted 4gl methods/attributes. Currently, there is no such interface, so it must be created. The backing, concrete class (FieldReference) must implement that interface.
- You have added a Buffer[Impl].rowId method which returns integer, for the ROWID buffer handle attribute. Please note there is already a Buffer[Impl].rowID method (originally added to support the converted ROWID builtin function) which returns the correct type. Please use that instead of adding the new method.
- Although I realize we have some naming inconsistencies across files, please follow the camel-casing convention already in use in a file when adding new methods. For example, Buffer already has a rowID method. So, when adding methods to support FIND-BY-ROWID, please name the associated method(s) findByRowID rather than findByRowId.
- Why is there a method Buffer[Impl].characterLocked? The 4gl documentation indicates the LOCKED attribute returns a logical, not character. Did you find the documentation was wrong (this is not unusual)? If not, please convert the LOCKED attribute to the existing Buffer.locked method instead.
- Please remove BufferImpl.queryOfEnd unless it has a purpose.
- There shouldn't be any public methods in BufferImpl that are not also in Buffer, and vice versa; e.g., error and changeError are in BufferImpl, but not Buffer.
- Some of the javadoc in BufferImpl is incorrect (possible cut-and-paste errors?); e.g., changeError.
- You have introduced a subtle bug in LockType.java by changing the internal values of the LT_* constants: compareTo relies on the order of these internal values; it is now broken.
- The QueryWrapper.DefaultDelegate inner class should not do any real work. It is simply meant to provide a safe, no-op implementation when a wrapper has no real query yet (i.e., between DEFINE QUERY and OPEN QUERY). There is no need to call static methods in AbstractQuery from any of this class' methods. Also, a formatting issue: please add the new QueryWrapper methods before the inner class, rather than after.
- The various forms of the value and changeValue methods in FieldReference.java should deal with Object, rather than DataModelObject. DataModelObject represents an entire record (all fields), whereas Object represents the value of the individual field.
- I'm not sure the new static methods in AbstractQuery.java are necessary, particularly since they should not be called from QueryWrapper.DefaultDelegate. What is their purpose? I guess I did not fully understand this from the discussions above, but now that I look at the update as a whole, they do not seem necessary. If they need to remain for some reason, please follow the coding standards with regard to method ordering within the class file (i.e., static methods should be defined above instance methods).
- The version of rules/convert/literals.rules you submitted is out of date with the latest version in bzr. Please re-merge.

#36 - 02/07/2013 01:26 PM - Vadim Nebogatov

```
It seems that you must have changed handle.java, but it was not included in the update.
```

Now included

```
In the archive, the file uast/progress.g should be located at src/com/goldencode/p2j/uast/progress.g, so that
it is applied properly when unzipped.
```

Corrected

```
All "hwrap" types in methods_attributes.rules must be interfaces, not concretesome  implementation classes. Fo
r example, an hwrap of "FieldReference" refers to a concrete class. It should refer to an interface that defin
es the methods which represent the converted 4gl methods/attributes. Currently, there is no such interface, so
 it must be created. The backing, concrete class (FieldReference) must implement that interface.
```

FieldInterface is created

```
You have added a Buffer[Impl].rowId method which returns integer, for the ROWID buffer handle attribute. Pleas
e note there is already a Buffer[Impl].rowID method literals.rules (originally added to support the converted
ROWID builtin function) which returns the correct type. Please use that instead of adding the new method.
```

Concerning documentation and some errors in types and so on. From scratch I supposed that P2J does not cover version 11 of 4GL, but only version 9.11. So, I checked only reference in version 9 literals.rules and googled if some method description was not found. Only after Constantin's table it was clear for me that version 11 is supposed to be supported. So, I really found sometimes wrong documentation (maybe out-to-date), for example, did not find method ROWID, but found method NUM-ROWID returning integer and supposed by mistake that it was KW_ROWID.
Now I use documentation from version 11 (OpenEdge).

```
Although I realize we have some naming inconsistencies across files, please follow the camel-casing convention
 already in use in a file when adding new methods. For example, Buffer already has a rowID method. So, when ad
ding methods to support FIND-BY-ROWID, please name the associated method(s) findByRowID rather than findByRowI
d.
```

Corrected

```
Why is there a method Buffer[Impl].characterLocked? The 4gl documentation indicates the LOCKED attribute retur
ns a logical, not character. Did you find the documentation was wrong (this is not unusual)? If not, please co
nvert the LOCKED attribute to the existing Buffer.locked method instead.
```

The same confusing with documentation like I described above

```
Please remove BufferImpl.queryOfEnd unless it has a purpose.
```

Removed

```
There shouldn't be any public methods in BufferImpl that are not also in Buffer, and vice versa; e.g., error a
nd changeError are in BufferImpl, but not Buffer.  literals.rules
```

Corrected

```
Some of the javadoc in BufferImpl is incorrect (possible cut-and-paste errors?); e.g., changeError.
```

Corrected

```
You have introduced a subtle bug in LockType.java by changing the internal values of the LT_* constants: compa
reTo relies on the order of these internal values; it is now broken.
```

Fixed

```
The QueryWrapper.DefaultDelegate inner class should not do any real work. It is simply meant to provide a safe
, no-op implementation when a wrapper has no real query yet (i.e., between DEFINE QUERY and OPEN QUERY). There
 is no need to call static methods in literals.rules AbstractQuery from any of this class' methods. Also, a fo
rmatting issue: please add the new QueryWrapper methods before the inner class, rather than after.
```

Corrected

```
The various forms of the value and changeValue methods in FieldReference.java should deal with Object, rather
than DataModelObject. DataModelObject represents an entire record (all fields), whereas Object represents the
value of the individual field.
```

Corrected

```
I'm not sure the new static methods in AbstractQuery.java are necessary, particularly since they should not be
 called from QueryWrapper.DefaultDelegate. What is their purpose? I guess I did not fully understand this from
 the discussions above, but now that I look at the update as a whole, they do not seem necessary. If they need
 to remain for some reason, please follow the coding standards with regard to method ordering within the class
 file (i.e., static methods should be defined above instance methods).
```

They are wrapping methods, do not throwing ErrorConditionException, but returning logical instead. Needed only for reusing in classes extending P2JQuery – AbstractQuery and QueryQrapper.

```
The version of rules/convert/literals.rules you submitted is out of date with the latest version in bzr. Pleas
e re-merge.
```

Merged

#37 - 02/07/2013 01:28 PM - Vadim Nebogatov

- File vmn_upd20130207a_test_cases.zip added

- File vmn_upd20130207a.zip added

Two updates are attached:

1) vmn_upd20130207a.zip - support database methods and attributes
2) vmn_upd20130207a_test_cases.zip - associated test cases

#38 - 02/07/2013 06:06 PM - Eric Faulhaber

Code review 20130207a:

This version of the update still had a few issues, but it was close enough that I just made some edits myself so we can get this feature set into regression testing:

- I moved the static unwrap* methods you added to group them with the other public static methods in handle.java. I removed unwrapP2JQuery because it was redundant with the unwrapQuery method that was already there. I renamed unwrapFieldInterface to unwrapBufferField to make it better match the original 4gl concept of a buffer field. I also changed the implementation of that method, which was still using FieldReference.class instead of FieldInterface.class.
- I made the associated changes in methods_attributes.rules (set hwrap to "Query" where it was "P2JQuery" and to "BufferField" where it was "FieldInterface").
- In Buffer[Impl].java, I changed the return type of the bufferField methods to handle, from BaseDataType. I missed this in my earlier review.
- FieldInterface.java: minor changes to match coding standards.
- QueryWrapper.java: I implemented the new DefaultDelegate methods to return safe values, rather than null (generally false for logicals, and unknown value for other types). Although we weren't targeting the runtime changes for this release, there were no TODOs here, and it seemed just as easy to change the return values as it would have been to add the TODO markers.
- The updates to progress.g produced several nondeterminism warnings during parser generation. On Greg's advice, I removed the 4 new literals from the reserved_variables rule, which eliminated the ambiguity.

#39 - 02/08/2013 07:13 AM - Vadim Nebogatov

What is Query? I saw hwrap to it. Is it still not implemented new interface which will be extended by P2Query?

#40 - 02/08/2013 08:07 AM - Eric Faulhaber

Vadim Nebogatov wrote:

> What is Query? I saw hwrap to it. Is it still not implemented new interface which will be extended by P2Query?

It is just the suffix to the handle.unwrapQuery method name. The hwrap value doesn't necessarily need to match the interface name, so handle.unwrapQuery returns P2JQuery, which is what you needed in many cases.

You could have an hwrap of Foo, but then have handle.unwrapFoo return an instance of Bar:

```
public static Bar unwrapFoo(handle h)
{
```

```
    return unwrapImpl(h, Bar.class);
}
```

#41 - 02/08/2013 08:43 AM - Greg Shah

All true. I would still prefer Query over P2JQuery.

#42 - 02/08/2013 10:57 AM - Eric Faulhaber

- File vmn_upd20130207b.zip added

#43 - 02/10/2013 12:01 PM - Vadim Nebogatov

Implemented, tested and commented attributes:
BUFFER-NAME (KW_BUF_NAME),
INDEX-INFORMATION (KW_IDX_INFO)
LITERAL-QUESTION (KW_LIT_QSTN)
VALIDATE-MESSAGE (KW_VAL_MSG).
MANDATORY (KW_MAND)
VALIDATE-EXPRESSION (KW_VAL_EXPR)
CURRENT-ITERATION (KW_CUR_ITER)
FORWARD-ONLY (KW_FWD_ONLY)
RECID (KW_RECID)
PREPARE_STRING (KW_PREP_STR)
(commment moved to #1668)

#44 - 02/11/2013 07:07 PM - Eric Faulhaber

- File vmn_upd20130211a.zip added

This version of your update is merged with the latest code revision in bzr (10166), and with overlapping changes in a separate update from Costin, which adds support for 4gl socket features. I merged in the socket update because we are regression testing this update together with that one and there were overlapping changes in several common files. Note that I only merged the common files, but the changes are all additive, so they should not cause a problem with your update.

CORRECTION: you will need Costin's update (cs_upd20130211c.zip, attached to #1639) to compile. Note that Costin's update must be applied **before** vmn_upd20130211a.zip, since cs_upd20130211c.zip contains older versions of handle.java, literals.rules, methods_attributes.rules, and common-progress.rules.

#45 - 02/11/2013 11:04 PM - Eric Faulhaber

- File vmn_upd20130211a.zip added

Greg,

The changes we made to literal processing (specifically, the "literals" function in common-progress.rules) caused a regression in implicit_where_clause.rules. The NO-LOCK option to a FOR EACH statement was being interpreted as a literal to be used in an implicit where clause generated from the primary index.

The rule in implicit_where_clause.rules which is now incorrect because of the new implementation of the "literals" function is:

```
ref != null and evalLib("literals", ref.type)
```

where ref is the second child of a record phrase of a FOR EACH.

I fixed this by changing the rule to

```
ref != null and
evalLib("literals", ref.type) and
!evalLib("lock_type_literals", ref.type)
```

and adding a new "lock_type_literals" function to common-progress.rules, which tests for the 4 new lock-related literal token types.

Does this seem like a safe and reasonable fix?

I have refreshed vmn_20130211a.zip to include the updated common-progress.rules file and the fixed implicit_where_clause.rules file.

#46 - 02/11/2013 11:05 PM - Eric Faulhaber

- File deleted (vmn_upd20130211a.zip)

#47 - 02/12/2013 02:12 AM - Eric Faulhaber

- File deleted (vmn_upd20130211a.zip)

#48 - 02/12/2013 02:14 AM - Eric Faulhaber

- File vmn_upd20130211a.zip added

Yet another version of this update, with a fixed version of methods_attributes.rules and literals.rules. The previous versions broke conversion in regression testing.

#49 - 02/12/2013 08:25 AM - Greg Shah

The change in implicit_where_clause.rules is correct and as you say, it is very necessary.

#50 - 02/14/2013 02:36 PM - Vadim Nebogatov

This update was regression tested, is committed to bzr revision 10168 & 10169.

#51 - 02/14/2013 02:37 PM - Vadim Nebogatov

- Status changed from New to WIP

- % Done changed from 50 to 100

#52 - 02/14/2013 03:27 PM - Eric Faulhaber

- Status changed from WIP to Closed

#53 - 02/15/2013 07:56 AM - Greg Shah

There is a bug related to the parser changes for making the lock types into literals:

```
FIND FIRST whatever WHERE  NO-LOCK.
```

There are many possible variants, but the idea is that in the 4GL an empty WHERE clause is valid. BUT now that we treat the lock types (and NO-WAIT) as literals, the parser naturally matches on this as a valid expression.

I think the solution is to put a semantic predicate into the where_clause rule in progress.g:

```
where_clause
   :
      KW_WHERE^
      (
         options { generateAmbigWarnings = false; }
         :
         {
           LA(1) != KW_SH_LOCK && LA(1) != KW_EXC_LOCK &&
           LA(1) != KW_NO_LOCK && LA(1) != KW_NO_WAIT
         }?
         expr
      )?
   ;
```

#54 - 02/15/2013 10:55 AM - Vadim Nebogatov

Greg Shah wrote:

> There is a bug related to the parser changes for making the lock types into literals:
>
> [...]
>
> There are many possible variants, but the idea is that in the 4GL an empty WHERE clause is valid. BUT now that we treat the lock types (and NO-WAIT) as literals, the parser naturally matches on this as a valid expression.

> I think the solution is to put a semantic predicate into the where_clause rule in progress.g:
>
> [...]

Do you mean runtime? In conversion time

FIND FIRST btest WHERE NO-LOCK.

is converted to

new FindQuery(btest, (String) null, null, "btest.id asc").first();

#55 - 02/15/2013 11:01 AM - Eric Faulhaber

Vadim Nebogatov wrote:

> Do you mean runtime? In conversion time
>
> FIND FIRST btest WHERE NO-LOCK.
>
> is converted to
>
> new FindQuery(btest, (String) null, null, "btest.id asc").first();

But it should be converted to:

```
new FindQuery(btest, (String) null, null, "btest.id asc", LockType.NONE).first();
```

The default lock type in Progress if none is specified is SHARE-LOCK (not NO-LOCK), which we honor in P2J. So, this conversion (i.e., without the specific lock type of LockType.NONE) is incorrect.

#56 - 02/15/2013 11:02 AM - Greg Shah

This is a conversion time problem only (since it is in the parser, which has no runtime implications).

#57 - 02/15/2013 11:25 AM - Vadim Nebogatov

Greg Shah wrote:

> This is a conversion time problem only (since it is in the parser, which has no runtime implications).

After changes you wrote, it is generated correctly:

new FindQuery(btest, (String) null, null, "btest.id asc", LockType.NONE).first();

Will I include these changes in progress.g to next update (with corresponding comment and reference to # in issue)?

#58 - 02/15/2013 11:27 AM - Greg Shah

Yes, please do.

#59 - 02/27/2013 12:04 PM - Eric Faulhaber

- Assignee changed from Vadim Nebogatov to Ovidiu Maxiniuc

- Status changed from Closed to WIP

Needed to re-open this for some conversion corrections to query methods GET-NEXT, GET-FIRST, etc. Please see note 61 in #1985 for details.

#60 - 02/27/2013 12:37 PM - Ovidiu Maxiniuc

- File om_upd20130227b.zip added

- Assignee changed from Ovidiu Maxiniuc to Vadim Nebogatov

This update fixes generation of LockType literals into converted java source.

#61 - 02/27/2013 01:14 PM - Eric Faulhaber

- Priority changed from Normal to High

Vadim, please fix the query handle GET-* return type problem, as described in #1985, note 61 (repeated below for convenience). **This is higher priority than any runtime work.**

From #1985:

Eric Faulhaber wrote:

> Constantin Asofiei wrote:
>
> > ...
> > the query-related methods like get-next/get-prev/etc, don't have the proper return type set in P2JQuery - there will be compile-time problems, as the server project uses this returned value.

Yes, instead of mapping GET-NEXT to public void next(...), we need to add wrapper methods, to be implemented in AbstractQuery, like:

```
public logical getNext(LockType lockType)
{
   boolean success = true;

   try
   {
      next(lockType);
   }
   catch (ConditionException exc)
   {
      success = false;
   }

   return (new logical(success));
}
```

This should just be a matter of changing methods_attributes.rules to map KW_GET_NEXT (for example) to new methods in P2JQuery:

```
public logical getNext();
public logical getNext(LockType lockType);
```

Then implement public logical AbstractQuery.getNext(LockType lockType) as noted above.

#62 - 02/27/2013 01:25 PM - Eric Faulhaber

Correction: I don't think we'll need P2JQuery.getNext() (only P2JQuery.getNext(LockType lockType)); seems the 4gl GET-NEXT method requires a lock type parameter.

#63 - 02/27/2013 01:29 PM - Eric Faulhaber

Ignore my last post; I mis-read the documentation. NO-LOCK is the default if not specified. So, it seems we will need a very simple implementation of AbstractQuery.getNext() as well:

```
public logical getNext()
{
   return getNext(LockType.NONE);
}
```

#64 - 02/27/2013 02:17 PM - Greg Shah

The om_upd20130227b.zip is close, but I have made a couple of changes to the code. The post-rules section is moved to the end of the file, which is where we keep it (by convention) since it runs after the stuff in front of it. The actual literal emission portion was moved inside the while loop where the parentid var is being processed. Placing it outside of that loop as dangerous and may not work in all cases. I'm not entirely sure that the loop is still needed, but for now it is more correct to put the code inside the loop. I have uploaded my changes. They are being conversion tested now.

#65 - 02/27/2013 02:18 PM - Greg Shah

- File om_upd20130227c.zip added

#66 - 02/27/2013 03:53 PM - Greg Shah

om_upd20130227c.zip has passed conversion testing and is checked into bzr as revision 10223.

#67 - 02/28/2013 03:40 AM - Vadim Nebogatov

The question: BUFFER-COMPARE/ BUFFER-COPY support also should be corrected. May I do this in the same update?

BUFFER-COPY (source-buffer-handle [, except-list [, pairs-list [, no-lobs]]])
public logical bufferCopy(handle bufHandle);
public logical bufferCopy(Buffer buffer);
public logical bufferCopy(handle bufHandle, String except, String pairs);
public logical bufferCopy(handle bufHandle, character except, character pairs, logical noLobs);
Should be added
public logical bufferCopy(handle bufHandle, character except);
and methods with all replacements character to String
Method
public logical bufferCopy(Buffer buffer);
possible should be removed at all.

BUFFER-COMPARE (source-buffer-handle [, mode-exp [,except-list , [pairs-list [, no-lobs]]]])
Now only one method:
public logical bufferCompare(handle srcBufHandle);
Should be even more methods and parameters than for BUFFER-COPY

Vadim,

Please always put questions of a technical nature in the associated Redmine task, and move your question below and my answer there. Your status report can be short and just summarize what you've done that day, and highlight any items blocking you (but always with the details in Redmine).

Regarding your question about the other Buffer methods, go ahead and add those you suggest, but do not remove any. I'm not sure I understand what you mean by "and methods with all replacements character to String". Leave the methods which accept String there. They are needed for string literals. In fact, I guess you will need to add methods with mixed signatures (e.g., bufferCopy(handle, String, character) and bufferCopy(handle, character, String)).

If you can get all these changes done and submitted before start of business here tomorrow, then yes, please include them in the same update. Otherwise, submit the GET-* update first, since we know those cases are in the customer's code.

Thanks,
Eric

#68 - 02/28/2013 03:47 AM - Vadim Nebogatov

The same question concerning getNext parameters. Do we really need method with signature using P2J type LockType?

public logical getNext(LockType lockType)
Conversion actually needed methods with signatures

public logical getNext();
public logical getNext(int lock);
public logical getNext(integer lock);
public logical getNext(int lock, int nowait);
public logical getNext(int lock, integer nowait);
public logical getNext(integer lock, int nowait);
public logical getNext(integer lock, integer nowait);

#69 - 02/28/2013 03:50 AM - Ovidiu Maxiniuc

- File om_upd20130227b.zip added

- Priority changed from High to Normal

I think

```
public logical bufferCopy(Buffer buffer);
```

should remain, here is an example form 4GL reference I already tested:

```
bh:BUFFER-COPY(BUFFER Customer:HANDLE, ?, "cust-sales-rep,SalesRep").
```

Probably BUFFER-COMPARE has a similar valid syntax... ?

#70 - 02/28/2013 03:51 AM - Ovidiu Maxiniuc

- Priority changed from Normal to High

I don't understand... stupid chrome or redmine ?

#71 - 02/28/2013 03:52 AM - Ovidiu Maxiniuc

- File deleted (om_upd20130227b.zip)

#72 - 02/28/2013 06:00 AM - Vadim Nebogatov

Eric Faulhaber wrote:

> Ignore my last post; I mis-read the documentation. NO-LOCK is the default if not specified. So, it seems we will need a very simple implementation of AbstractQuery.getNext() as well:
> [...]

According doc, lock is mandatory, no-wait is optional:

GET-NEXT (NO-LOCK| SHARE-LOCK [, NO-WAIT] | EXCLUSIVE-LOCK [, NO-WAIT])

So, getNext() seems not needed too.

#73 - 02/28/2013 06:11 AM - Constantin Asofiei

> According doc, lock is mandatory, no-wait is optional:
>
> GET-NEXT (NO-LOCK| SHARE-LOCK [, NO-WAIT] | EXCLUSIVE-LOCK [, NO-WAIT])
>
> So, getNext() seems not needed too.

Vadim, over time, we've learned not to put all our trust in the official docs. Is best to test in the 4GL environment, before making a decision. For this case, 4GL compiles this code:

```
def var h as handle.
h:get-next().
```

So, the lock type is optional.

#74 - 02/28/2013 06:38 AM - Vadim Nebogatov

OK, good. What do you think about #68 (getNext(LockType lockType))?

#75 - 02/28/2013 07:34 AM - Constantin Asofiei

Vadim Nebogatov wrote:

> OK, good. What do you think about #68 (getNext(LockType lockType))?

Actually, NO-LOCK and similar convert to NO_LOCK_LITERAL for which Ovidiu added conversion support to convert to LockType.* constants. So, no work needed for these. As get-next() accepts integer too, you need to add getNext(integer) and getNext(long) and getNext(int64) versions also (as integer and int64 can be used), and the runtime will determine the type of lock based on the value.

#76 - 02/28/2013 07:35 AM - Constantin Asofiei

LE: when I say no work needed, I refer to the lock types. You will still need getNext(LockType) version to be added.

#77 - 02/28/2013 08:03 AM - Vadim Nebogatov

Constantin Asofiei wrote:

> Vadim Nebogatov wrote:
>
> > OK, good. What do you think about #68 (getNext(LockType lockType))?
>
> Actually, NO-LOCK and similar convert to NO_LOCK_LITERAL for which Ovidiu added conversion support to convert to LockType.* constants. So, no work needed for these. As get-next() accepts integer too, you need to add getNext(integer) and getNext(long) and getNext(int64) versions also (as integer and int64 can be used), and the runtime will determine the type of lock based on the value.

Ok, I understood concerning getNext(LockType lockType).
But I did not understand you properly about all other methods. Are all methods below required? (From tests I see that - YES)

```
public logical getNext();
public logical getNext(int lock);
public logical getNext(integer lock);
public logical getNext(int lock, int nowait);
public logical getNext(int lock, integer nowait);
public logical getNext(integer lock, int nowait);
public logical getNext(integer lock, integer nowait);
```

And you suggest to add also ALL combinations with int64 and long? For lock only or for nowait too? In any case the list of methods will be increased very much.

#78 - 02/28/2013 08:14 AM - Constantin Asofiei

The syntax for this method would be (example for GET-NEXT):

```
public logical getNext( [ LockType | long | int64 | integer ] [, { LockType | long | integer | int64 } ] );
```

where both parameters are optional, and nowait appears only if first is set. We can reduce this by using NumberType instead of integer and int64, but I don't want to do this, because we will expose something not accepted by 4GL. This I think will yeald 21 versions (1 no-params, 4 with only first param, 16 with both params), and I don't think we can reduce this.

Greg/Eric: can you check if are cases where get-next and alike get a parameter which is not the name of a lock, i.e. an int var/field/constant ? If not, then we can get away with adding only the LockType versions.

#79 - 02/28/2013 08:26 AM - Vadim Nebogatov

LockType aggregates lock and wait statuses, what is the sense of using it with other parameters, especially with one more LockType?

#80 - 02/28/2013 08:33 AM - Constantin Asofiei

Vadim Nebogatov wrote:

> LockType aggregates lock and wait statuses, what is the sense of using it with other parameters, especially with one more LockType?

Because the parser should emit the second param to NO_WAIT_LITERAL (and convert it to LockType.NO_WAIT) if NO-WAIT is used. But currently I think there is a problem in the parser, because it doesn't convert it to NO_WAIT_LITERAL.

A final note, in 4GL following is valid:

```
def var i as int.
def var i64 as int.
def var h as handle.

h:get-next().

h:get-next(NO-LOCK).
h:get-next(1).
h:get-next(i).
h:get-next(i64).

h:get-next(NO-LOCK, 1).
h:get-next(NO-LOCK, i).
h:get-next(NO-LOCK, i64).
h:get-next(1, 1).
h:get-next(1, i).
h:get-next(1, i64).
h:get-next(i, 1).
h:get-next(i, i).
h:get-next(i, i64).
h:get-next(i64, 1).
h:get-next(i64, i).
h:get-next(i64, i64).
h:get-next(NO-LOCK, NO-WAIT).
h:get-next(1, NO-WAIT).
h:get-next(i, NO-WAIT).
h:get-next(i64, NO-WAIT).
```

#81 - 02/28/2013 08:39 AM - Vadim Nebogatov

Will we cast long to int before using LockType.fromInt(int type)?

#82 - 02/28/2013 08:40 AM - Constantin Asofiei

If you set the parameter to java native long, then all native int values will be accepted, so no casting is needed. Idea is, beside LockType, each parameter will have "long"to accept native int and long values, and "int64" and "integer" for wrapper types.

#83 - 02/28/2013 08:49 AM - Vadim Nebogatov

If we want to reuse current(LockType lockType) we will need to create LockType from other types including long

#84 - 02/28/2013 08:51 AM - Constantin Asofiei

Vadim Nebogatov wrote:

> If we want to reuse current(LockType lockType) we will need to create LockType from other types including long

This is the runtime's resposibility, not conversion's. For now, don't worry about this, just stub all the new APIs.

#85 - 02/28/2013 10:05 AM - Eric Faulhaber

Constantin Asofiei wrote:

> where both parameters are optional, and nowait appears only if first is set. We can reduce this by using NumberType instead of integer and int64, but I don't want to do this, because we will expose something not accepted by 4GL.

I don't understand this. Why can't we use NumberType to reduce the explosion of APIs? We already use this in the numeric setter methods of generated DMOs.

#86 - 02/28/2013 10:07 AM - Constantin Asofiei

Eric Faulhaber wrote:

> I don't understand this. Why can't we use NumberType to reduce the explosion of APIs? We already use this in the numeric setter methods of generated DMOs.

The idea was to not let values which are not integer or int64 to be passed to the parameters (i.e. to let the 4GL code fail at compile time if this is encountered)... but if you think this is a non-issue, then we can use NumberType instead of integer and int64.

#87 - 02/28/2013 10:19 AM - Eric Faulhaber

Constantin Asofiei wrote:

> Vadim Nebogatov wrote:
>
> > If we want to reuse current(LockType lockType) we will need to create LockType from other types including long
>
> This is the runtime's resposibility, not conversion's. For now, don't worry about this, just stub all the new APIs.

From my perspective, if we have the literals at conversion time, we have enough information to generate the correct LockType constant. So, why do we need all the following combinations?

```
public logical getNext(int lock, int nowait);
public logical getNext(long lock, int nowait);
public logical getNext(int lock, long nowait);
public logical getNext(long lock, long nowait);
```

It seems to me all these would be covered by:

```
public logical getNext(LockType lockType);
```

Am I missing something?

#88 - 02/28/2013 10:22 AM - Constantin Asofiei

> Am I missing something?

yes, you can pass vars/expressions to these parameters, not just literals:

```
def var h as handle.
def var i as int.
h:get-next(i + 1). /* lock-type is not a literal */
```

I didn't check runtime, just compile. And that's why I've asked at note 78 to check if there are non-literal cases for GET-NEXT (and others) in the project. If there are none, then is enough to add support for the LockType cases.

#89 - 02/28/2013 10:36 AM - Eric Faulhaber

Constantin Asofiei wrote:

> Am I missing something?

> yes, you can pass vars/expressions to these parameters, not just literals:
>
> ```
> def var h as handle.
> def var i as int.
> h:get-next(i + 1). /* lock-type is not a literal */
> ```

OK, but wouldn't the i + 1 expression be converted to plus(i, 1) (which returns an integer wrapper instance), such that this would be handled by the following?

```
public logical getNext(integer lock)
```

I'm not proposing we remove that one, just the variants with combinations of **only** Java primitive types. I think we need to keep the ones with a mixture of NumberType and int (the Java primitive), to handle the cases where one parameter is a (4gl) variable or expression and the other is a literal.

> I didn't check runtime, just compile. And that's why I've asked at note 78 to check if there are non-literal cases for GET-NEXT (and others) in the project. If there are none, then is enough to add support for the LockType cases.

I'm not sure yet, but if we can cover all combinations with a reasonable number of APIs, I'd like to do that now.

#90 - 02/28/2013 10:41 AM - Constantin Asofiei

> OK, but wouldn't the i + 1 expression be converted to plus(i, 1) (which returns an integer wrapper instance), such that this would be handled by the following?

Correct, it will be covered by that. My example was to show that it can use expressions.

> I'm not proposing we remove that one, just the variants with combinations of **only** Java primitive types. I think we need to keep the ones with a mixture of NumberType and int (the Java primitive), to handle the cases where one parameter is a (4gl) variable or expression and the other is a literal.

Actually, it would be the java primitive long instead of int, to accept int64 literals. If you look at note 78 again, you will see that my proposal was [LockType | long | int64 | integer] for each parameter (java int literals was covered by long, as long accepts int literals). If we reduce it to [LockType | long | NumberType], we end up with 13 combinations. If we no longer emit LockType.NO_LOCK for a NO-LOCK literal, then we can use [long | NumberType] and reduce it to ~~6~~ 7 cases. These are our options.

#91 - 02/28/2013 10:45 AM - Vadim Nebogatov

Variant with 100+ methods (integer, int64, long, LockType) is ready

GET-NEXT (KW_GET_NEXT)
GET-PREV (KW_GET_PREV)
GET-FIRST (KW_GET_FIRST)
GET-LAST (KW_GET_LAST)
GET-CURRENT (KW_GET_CUR)

As I understand I should hold it...

#92 - 02/28/2013 11:08 AM - Eric Faulhaber

Yes, please hold this update for the moment.

I think we can cover all cases with:

```
getNext()
getNext(LockType lockType)
getNext(NumberType lock, NumberType nowait)
getNext(NumberType lock, long nowait)
```

```
getNext(long lock, NumberType nowait)
```

The no-arg variant obviously gets used if there are no parameters to the GET-NEXT call.

The LockType variant only gets used if we have a constant/literal for lock and (optionally) a constant/literal for nowait. This is resolved at conversion time into a single LockType argument, and I think is actually the common case.

Anything other than these cases (such as the use of expressions, variables, fields, etc. as one of the parameters) drives us to use one of the other 4 variants.

Is there any case which is not covered here?

As a separate issue, I'm not actually sure why we need long instead of int, because the only valid literals are well-known compiler constants. Does Progress compile cases which use literals that are not one of the well-known constants?

#93 - 02/28/2013 11:14 AM - Constantin Asofiei

Eric Faulhaber wrote:

> Yes, please hold this update for the moment.
>
> I think we can cover all cases with:
>
> [...]

You forgot these:

```
getNext(NumberType lock)
getNext(long lock)
```

> The LockType variant only gets used if we have a constant/literal for lock and (optionally) a constant/literal for nowait. This is resolved at conversion time into a single LockType argument, and I think is actually the common case.

This does not work, with current bzr rev:

```
def var h as handle.
h:get-next(no-lock, no-wait).
```

converts to:

```
handle.unwrapQuery(h).next(LockType.NONE, LockType.SHARE_NO_WAIT);
```

> As a separate issue, I'm not actually sure why we need long instead of int, because the only valid literals are well-known compiler constants. Does Progress compile cases which use literals that are not one of the well-known constants?

Yes, this is valid:

```
h:get-next(20, 20).
h:get-next(999999999999999999, 999999999999999999).
```

#94 - 02/28/2013 11:34 AM - Eric Faulhaber

Constantin Asofiei wrote:

> You forgot these:
>
> ```
> getNext(NumberType lock)
> getNext(long lock)
> ```

The first one, yes, I missed that.

But, do we need the second one? What does it actually mean in Progress when we get a literal that is not one of the well-known constants? My guess is that it is silently ignored and we get some default behavior (NO-LOCK? SHARE-LOCK?). It must be deterministic. We can detect this at conversion time. We need some tests to determine what the behavior is in Progress for nonsense literals. Or does Progress raise a runtime error in this case? If so, then yes, we need the extra API, and we'll have to mimic the runtime error.

> The LockType variant only gets used if we have a constant/literal for lock and (optionally) a constant/literal for nowait. This is resolved at conversion time into a single LockType argument, and I think is actually the common case.

> This does not work, with current bzr rev:
>
> ```
> def var h as handle.
> h:get-next(no-lock, no-wait).
> ```
>
> converts to:
>
> ```
> handle.unwrapQuery(h).next(LockType.NONE, LockType.SHARE_NO_WAIT);
> ```

OK, but doesn't that just mean we're converting it wrong today? IMO, this should convert to:

```
handle.unwrapQuery(h).next(LockType.NONE);
```

...and, for example,

```
def var h as handle.
h:get-next(share-lock, no-wait).
```

should convert to:

```
handle.unwrapQuery(h).next(LockType.SHARE_NO_WAIT);
```

> As a separate issue, I'm not actually sure why we need long instead of int, because the only valid literals are well-known compiler constants. Does Progress compile cases which use literals that are not one of the well-known constants?

Yes, this is valid:

```
h:get-next(20, 20).
h:get-next(999999999999999999, 999999999999999999).
```

Again, if the literals don't match the well-known compiler constants, Progress must default to some lock behavior deterministically or raise a runtime error. We should do the same, but we need to know what that behavior is.

LE: For both of the examples above of what I think the conversion should be, I meant to write getNext, not next.

#95 - 02/28/2013 11:36 AM - Constantin Asofiei

> Again, if the literals don't match the well-known compiler constants, Progress must default to some lock behavior deterministically or raise a runtime error. We should do the same, but we need to know what that behavior is.

Ok, now I understand what you mean. Then someone needs to test the runtime and see how 4GL acts when not using the well-known compiler constants.

#96 - 02/28/2013 11:59 AM - Eric Faulhaber

Constantin Asofiei wrote:

> > Again, if the literals don't match the well-known compiler constants, Progress must default to some lock behavior deterministically or raise a runtime error. We should do the same, but we need to know what that behavior is.
>
> Ok, now I understand what you mean. Then someone needs to test the runtime and see how 4GL acts when not using the well-known compiler constants.

Vadim, please test this to determine what the behavior is. First, determine if Progress produces a runtime error for literals which are not well-known. If so, we have our answer.

If not, then I think you will need to have two user sessions try to access the same record simultaneously. For this, you will need to start a database server in Progress for the p2j_test database (see the dbadm.pdf - Database Management: Database Administration documentation on how to start a server process), then connect to it in shared mode (use mpro instead of pro) from both user sessions.

The first session should start a transaction, exclusively lock a record, and pause (to hold the lock), while the other tries to get a lock on that same record. So the first case looks something like this:

```
do transaction:
  find first book exclusive-lock.
  pause.
end.
```

In the second session, first try to obtain the lock using get-first with a known lock type (to confirm you are properly locking the record). Then, use other forms of get-first with non-standard literals to see what happens.

#97 - 02/28/2013 01:13 PM - Vadim Nebogatov

I have tested in 4GL the following

```
DO WHILE QUERY QueryTtTest:GET-NEXT(N):
END.
```

and substituted different values for N. It works without errors only if N is one from constants below
public static final int LT_NONE = 6209;
public static final int LT_SHARE = 6208;
public static final int LT_EXCLUSIVE = 6207;
Else error is shown : The first argument to QUERY GET methods must be NO-LOCK, SHARE-LOCK or EXCLUSIVE-LOCK (7314)

#98 - 02/28/2013 02:05 PM - Eric Faulhaber

OK, then we must emulate this error handling at runtime, which means we need the following APIs:

```
getNext()
getNext(LockType lockType)
getNext(long lock)
getNext(long lock, long nowait)
getNext(NumberType lock)
getNext(NumberType lock, NumberType nowait)
getNext(NumberType lock, long nowait)
getNext(long lock, NumberType nowait)
```

I still want to convert all cases of known lock and nowait constants to the getNext(LockType) API.

In AbstractQuery, all the other APIs will call this one at runtime. All the variants which accept one or both numeric values should call a private worker method to do the error handling and get the right LockType value, since all getXXXX methods will have to do the same), then call getNext(LockType), which will then call next(LockType) in a try-catch and return the proper logical value, as discussed in note 61.

BTW, please test what happens when the second parameter is not the proper literal for NO-WAIT. This may produce a different error message and number. We will have to emulate that as well.

In annotations/database_general.rules, please add a rule to test the current node for the no-lock, share-lock, and exclusive-lock literals with a parent node of the appropriate attribute type (KW_GET_NEXT, etc.). Determine whether a next sibling exists. If not, putNote("javaname", "LockType.{NONE|SHARE|EXCLUSIVE}"). If there is a next sibling AND it is the no-wait literal, putNode("javaname", "LockType.{NONE|SHARE_NO_WAIT|EXCLUSIVE_NO_WAIT}). Also, call setHidden(true) on that next sibling no-wait literal node. This will hide it, so we don't try to emit it during core conversion. If the next sibling exists, but it is NOT the no-wait literal, we have to assume it is an expression, variable, field, etc. that may or may not evaluate to the no-wait value (6090) at runtime. In this case, putNote("javaname", "LockType.LT_{NONE|SHARE|EXCLUSIVE}") and DO NOT hide the next sibling.

In convert/literals.rules, rewrite the rules regarding lock type literals (added in the last update) as follows. When you find a lock literal, check getNoteBoolean("javaname"). Use the value of the "javaname" annotation to create a java.reference node with that text. The second parameter (if any) should emit naturally.

Please get the conversion part of this done ASAP (including the P2JQuery changes and the stubs in AbstractQuery) and submit it first. Work on the runtime portion next and submit that as a later update.

#99 - 02/28/2013 02:22 PM - Eric Faulhaber

One correction to my last posting:

In the case where the first argument is NOT one of the lock literals, AND we have a no-wait literal for the second argument, the no-wait literal will not emit naturally; it will be ignored. In this case, we want to emit LockType.LT_NO_WAIT for the no-wait literal.

So, we'll need an extra rule in convert/literals.rules that tests for the no-wait literal and creates a java.reference node with the text LockType.LT_NO_WAIT. This is safe without any additional protection logic in the case where the first argument is a lock type literal, because we will have hidden the node during annotations, so it will not be visited during core conversion.

#100 - 02/28/2013 03:11 PM - Vadim Nebogatov

I tested in 4GL the following

```
DO WHILE QUERY QueryTtTest:GET-NEXT(NO-LOCK, N):
END.
```

And substituted different values for N. It works without errors if N is

public static final int LT_NO_WAIT = 6090;

Else error is shown : QUERY GET second argument must be NO-WAIT or 0. (7360)
It works with 0 without errors too; - as far as I understood

```
GET-NEXT(NO-LOCK, 0)
```

is equivalent to

```
GET-NEXT(NO-LOCK)
```

The same for SHARE-LOCK.

#101 - 02/28/2013 04:02 PM - Eric Faulhaber

Vadim Nebogatov wrote:

> It works with 0 without errors too; - as far as I understood
>
> ```
> GET-NEXT(NO-LOCK, 0)
> ```
>
> is equivalent to
>
> ```
> GET-NEXT(NO-LOCK, 0)
> ```
>
> The same for SHARE-LOCK.

In that case, you also will want to modify the database_general.rules logic described above to hide the next sibling node if it is a prog.num_literal with text "0", and set the "javaname" annotation in the lock literal node to "LockType.{NONE|SHARE|EXCLUSIVE}".

#102 - 02/28/2013 04:41 PM - Constantin Asofiei

I'm posting these here too, from #2068:

More errors related to database methods:

```
    [javac] .../server/app/Idryoau0.java:862: error: no suitable method found for findFirst()
    [javac]             handle.unwrapBuffer(ihTttableBuf).findFirst();
```

```
    [javac] .../server/app/Iodtcab0.java:464: error: method addBuffer in interface P2JQuery cannot be applied
to given types;
    [javac]             handle.unwrapQuery(ohQryInputTt).addBuffer(ihInputTtbuf);
    [javac]   required: Buffer
    [javac]   found: handle
```

Both are cases of missing APIs

LE: GES removed customer-specific directory names from this entry.

#103 - 03/01/2013 03:46 AM - Vadim Nebogatov

- File vmn_upd20130201a_test_cases.zip added

- File vmn_upd20130301a.zip added

Attached update vmn_upd20130301a.zip with tests vmn_upd20130301a_test_cases.zip containing
1) the conversion and runtime stubs for
GET-NEXT (KW_GET_NEXT)
GET-PREV (KW_GET_PREV)
GET-FIRST (KW_GET_1ST)
GET-LAST (KW_GET_LAST)
GET-CURRENT (KW_GET_CUR)
2) kw_get_1st – duplicate removed from methods_attributes.rules

#104 - 03/01/2013 07:03 AM - Ovidiu Maxiniuc

- File om_upd20130301a.zip added

This update removes the unneeded handle: token from the chaining like:

```
BUFFER book:HANDLE:BUFFER-COPY(h).
viCount = QUERY q:HANDLE:NUM-BUFFERS.
```

The issue is more documented in #2068, notes 14-16.

#105 - 03/01/2013 07:16 AM - Constantin Asofiei

The code looks good, I'm putting this through conversion regression testing.

#106 - 03/01/2013 08:49 AM - Constantin Asofiei

It passed conversion regression testing, please go ahead and checkin/ distribute.

#107 - 03/01/2013 08:50 AM - Vadim Nebogatov

Constantin Asofiei wrote:

> It passed conversion regression testing, please go ahead and checkin/ distribute.

Do you mean #103 or #104 or both?

#108 - 03/01/2013 08:57 AM - Eric Faulhaber

I think he means 104. I'm reviewing 103 now. It is not ready; I'll provide detail shortly.

#109 - 03/01/2013 08:58 AM - Constantin Asofiei

Sorry I have not specified, it was Ovidiu's update.

#110 - 03/01/2013 09:05 AM - Vadim Nebogatov

- File vmn_upd20130201a_test_cases.zip added

- File vmn_upd20130301b.zip added

Attached update vmn_upd20130301b.zip with tests vmn_upd20130301b_test_cases.zip
containing fixes for # 1947/ #102:

1) corrected for:

FIND-UNIQUE (KW_GET_UNI),
FIND-FIRST (KW_FIND_1ST),
FIND-LAST (KW_FIND_LST)

2) Added addBuffer(handle bufHandle) (KW_ADD_BUF)

#111 - 03/01/2013 09:09 AM - Vadim Nebogatov

- File deleted (vmn_upd20130201a_test_cases.zip)

#112 - 03/01/2013 09:09 AM - Vadim Nebogatov

- File vmn_upd20130201b_test_cases.zip added

#113 - 03/01/2013 09:40 AM - Eric Faulhaber

Code review 20130301a:

- The update is incomplete. I provided detailed instructions on the needed changes to rules/annotations/database_general.rules and rules/convert/literals.rules for lock type conversion. These changes were not included with the update. If something about my proposed changes is not clear, please ask.
- Please rename the getPrev methods to getPrevious. Although "getPrev" matches the Progress naming convention, please note that we already have broken with that convention with the previous() and previous(LockType) methods. Let's maintain consistency.
- Method variant getPrevious(long, NumberType) is missing from the hierarchy.
- The queryCurrent instance method was removed from the query hierarchy, but the static method of the same name is still in AbstractQuery.
- Finally, just an aesthetic note: please do not change the javadoc format when adding or changing methods. For example, the existing methods in this file (indeed, across the project) indent the text after param, return, throws tags. Long lines are wrapped on that indent. The code you have added does not indent the javadoc on the return tag the same way. It's not a problem from a functional point of view, but it makes the code look inconsistent. We are delivering the source code to customers, so we want to maintain a uniform standard.

#114 - 03/01/2013 10:55 AM - Vadim Nebogatov

Changes in *.rules files. OK, will correct. My impression was that it will be needed for runtime.

Static queryCurrent – maybe it should be removed at all so far? I added it in time where I tried to implement runtime.

javadoc format – I use IntelliJ Idea, unfortunately it's current settings do not show me any differences. So, I supposed that everything is ok – usually I add/remove often on line base. How is better check problems you wrote? Maybe I should generate javadocs each time and check them?

#115 - 03/01/2013 11:22 AM - Eric Faulhaber

Vadim Nebogatov wrote:

> Changes in *.rules files. OK, will correct. My impression was that it will be needed for runtime.

No, this is required to make the lock type and no-wait parameters convert correctly so the code will compile.

> Static queryCurrent – maybe it should be removed at all so far? I added it in time where I tried to implement runtime.

I just saw that you removed other forms of queryCurrent, and it is not referenced from anywhere, so I assumed this was obsolete.

> javadoc format – I use IntelliJ Idea, unfortunately it's current settings do not show me any differences. So, I supposed that everything is ok – usually I add/remove often on line base. How is better check problems you wrote? Maybe I should generate javadocs each time and check them?

I'm referring to the format of the javadoc in the *source code*, not the generated javadoc. I did not build with your update and generate javadoc, but I assume that will work OK. I'm just saying that the other methods in the runtime source code look like this:

```
/**
 * Method description...
 *
 * @param   arg1
 *          Description of arg1.
 *
 * @return  This is a long description of the return value of this
 *          method. It is aligned with the indented description of the
 *          param tags. Additional text...
 */
```

...and yours looks like this:

```
/**
 * Method description...
 *
 * @param   arg1
 *          Description of arg1.
 *
 * @return This is a long description of the return value of this
 * method. It is not aligned with the indented description of the
 * param tags. Additional text...
 */
```

For consistency, I would like you to indent and wrap the text in the source code like the first example, since that is the way the rest of the code in the project looks. When each developer does things differently, it makes the code look messy over time. Again, not a critical issue, but maintaining the consistency makes the source code cleaner.

Do you expect you can deliver this corrected update today?

#116 - 03/01/2013 11:42 AM - Vadim Nebogatov

Yes, I hope

#117 - 03/01/2013 12:15 PM - Eric Faulhaber

Vadim Nebogatov wrote:

> Yes, I hope

If not, please post what you have at the end of the day so we have the option of finishing it ourselves without starting over.

One other note I forgot in the code review: please leave out back up files from your updates. For example, the last one contained src/com/goldencode/p2j/persist/QueryWrapper.java~, which looks like a backup your editor created.

#118 - 03/01/2013 04:05 PM - Vadim Nebogatov

- File vmn_upd20130301c.zip added

- File vmn_upd20130201c_test_cases.zip added

Attached update vmn_upd20130301c.zip with tests vmn_upd20130301c_test_cases.zip is replacement of update vmn_upd20130301a.zip.
I fixed all comments including changes in annotations/database_general.rules, but had no time to make changes in convert/literals.rules

#119 - 03/02/2013 05:23 AM - Vadim Nebogatov

Eric Faulhaber wrote:

> In convert/literals.rules, rewrite the rules regarding lock type literals (added in the last update) as follows. When you find a lock literal, check getNoteBoolean("javaname")...

Did you mean getNote("javaname")?

#120 - 03/02/2013 09:34 AM - Eric Faulhaber

Vadim Nebogatov wrote:

> Eric Faulhaber wrote:
>
> > In convert/literals.rules, rewrite the rules regarding lock type literals (added in the last update) as follows. When you find a lock literal, check getNoteBoolean("javaname")...
>
> Did you mean getNote("javaname")?

Oh, typo. Actually I meant getNoteString("javaname").

#121 - 03/02/2013 04:52 PM - Greg Shah

- File ges_upd20130302a.zip added

As found in #1985 (and documented in note 3 of #1655), the DELETE-RESULT-LIST-ENTRY method should have been added previously.

The attached implements the conversion change WITHOUT the runtime stubs.

Here is what the method should look like in the runtime (in P2JQuery and whatever implements it):

```
/**
 * This will delete the current row in the query's results.  This implements the
 * 4GL DELETE-RESULT-LIST-ENTRY() method.
 *
 * @return  true on success.
 */
public logical deleteResultListEntry();
```

In the 4GL this can also be executed against a BROWSE, but I suspect it is just redirected to the browse's internal query.

This change (methods_attributes.rules only) is being conversion tested now.

#122 - 03/02/2013 06:11 PM - Greg Shah

The ges_upd20130302a.zip passed conversion testing and checked in as bzr revision 10232.

#123 - 03/03/2013 03:28 PM - Vadim Nebogatov

- File vmn_upd20130303a_test_cases.zip added

- File vmn_upd20130303a.zip added

Attached update vmn_upd20130303a.zip with tests vmn_upd20130303a_test_cases.zip is next replacement of update vmn_upd20130301a.zip.
I have added changes in convert/literals.rules

#124 - 03/03/2013 09:48 PM - Eric Faulhaber

Code review 20130303a:

- As I noted earlier, please don't include backup files (e.g., QueryWrapper.java~) in your updates.
- rules/convert/methods_attributes.rules was provided with your last update, but it is missing from this update. Please merge your changes with the latest version of this file (which has changed in bzr, see notes 121-122 above) and re-submit.

- rules/annotations/database_general.rules is out of date. Please merge with the latest version in bzr and resubmit.
- All the getPrev methods in the hierarchy were changed to getPrevious except getPrev(NumberType).
- Your test case misses testing the majority of your conversion changes. Currently, it only tests the scenarios of (1) no arguments; and (2) numeric literals which do not match the lock type and no-wait compiler constants. While such tests are necessary, you are not testing the common cases of using the lock type literals, with and without the no-wait literal. It also does not test the less common cases of using variables or expressions as parameters. Because of this, you cannot have tested the logic you added to database_general.rules and literals.rules to convert the literal arguments, which is by far the more complex aspect of this update. Please update your test case to add these scenarios.
- The name of your test case should be something more meaningful to the context of the features you are testing than "Vadim22.p". It should be in a directory named uast, so it can be checked into the testcases project.

Please correct these issues and re-submit this update as your top priority.

#125 - 03/04/2013 03:04 AM - Vadim Nebogatov

I did not notice that Ubuntu Nautilus does not show backup files by default...

#126 - 03/04/2013 04:05 AM - Vadim Nebogatov

I have analyzed one more time. Solution you wrote is useful if both parameters are literals. But many combined cases is possible when one or both of parameters are valid numbers like

```
GET-NEXT(SHARE-LOCK, 6090).                    //  NO-WAIT
GET-NEXT(SHARE-LOCK, 0).                         //  GET-NEXT(SHARE-LOCK).
GET-NEXT(6207, NO-WAIT).                           //  6207 = EXCLUSIVE-LOCK
...
```

Moreover, these values could be expressions calculated in runtime:

```
GET-NEXT(SHARE-LOCK, 6090/2 + 6090/2).                    //  NO-WAIT
```

I suggest do not replace parameters to LockType constants on conversion time, but
create static function in LockType:

```
   public static LockType fromLockNowait(long lock, long nowait)
   {
      if (lock == LT_NONE)
      {
         if (nowait == 0)
         {
            return NONE;
         }
         else if (nowait == LT_NO_WAIT)
         {
            throw new IllegalArgumentException("Invalid combination: lock type NONE is not allowed with NO-WAI
T");
         }
         else
         {
            throw new IllegalArgumentException("Invalid combination: lock type NONE is not allowed with nowait
 =  " + nowait);
         }
      }
      else if (lock == LT_SHARE)
      {
         if (nowait == 0)
         {
            return SHARE;
         }
```

```
         else if (nowait == LT_NO_WAIT)
         {
            return SHARE_NO_WAIT;
         }
         else
         {
            throw new IllegalArgumentException("Invalid combination: lock type SHARE is not allowed with nowai
t =  " + nowait);
         }
      }
      else if (lock == LT_EXCLUSIVE)
      {
         if (nowait == 0)
         {
            return EXCLUSIVE;
         }
         else if (nowait == LT_NO_WAIT)
         {
            return EXCLUSIVE_NO_WAIT;
         }
         else
         {
            throw new IllegalArgumentException("Invalid combination: lock type EXCLUSIVE is not allowed with n
owait =  " + nowait);
         }
      }
      else
      {
         throw new IllegalArgumentException("Invalid lock:  " + lock);
      }
   }
```

and use it in runtime implementations of all getNext() methods like

```
   public logical getNext(long lock)
   {
      return next(this, LockType.fromLockNowait(lock, 0));
   }
   public logical getNext(NumberType lock, long nowait)
   {
      return next(this, LockType.fromLockNowait(lock.longValue(), nowait));
   }
...

where next() method in AbstractQuery contains all logic and will be reused both in AbstractQuery and QueryWrap
per

   protected static logical next(P2JQuery p2JQuery, LockType lockType)
   {
      boolean success = true;
      try
      {
         p2JQuery.next(lockType);
      }
      catch (ErrorConditionException exc)
      {
         success = false;
      }
      return new logical(success);
   }
```

#127 - 03/04/2013 09:44 AM - Eric Faulhaber

Vadim Nebogatov wrote:

> I have analyzed one more time. Solution you wrote is useful if both parameters are literals. But many combined cases is possible when one or both of parameters are valid numbers like
> [...]
> Moreover, these values could be expressions calculated in runtime:
>
> [...]

Yes, I know this; the solution I wrote intentionally only deals with the common case where both parameters are lock literals. The fact that it is possible to use other parameters was the basis of the long discussion above and the reason we added all the API variants that accept long and NumberType. But all of the APIs that accept long and/or NumberType are just there to deal with the possibility that a developer has not used the well-known constants. This is not the common case.

> I suggest do not replace parameters to LockType constants on conversion time,

No. If you will carefully re-read note 98 above, you will see that I stated that I want to deal with the common case of combinations of the well-known {NO|SHARE|EXCLUSIVE}-LOCK and NO-WAIT literals into our LockType values **at conversion time**. This is why I provided detailed instructions on how to change the database_general and literals rule sets.

> but create static function in LockType:
>
> [...]
> and use it in runtime implementations of all getNext() methods like
>
> [...]

Yes, this is what I have been saying, **except that I still want to convert the common case of the known lock constants and nowait constant to use the LockType variant of the getXXXX methods**. Again, please re-read note 98.

If you will just fix the issues noted in the last code review, that is all I want right now. Do not include LockType.fromLockNowait; it does not properly emulate the Progress error handling that you tested in notes 97 and 100 above.

#128 - 03/04/2013 01:19 PM - Eric Faulhaber

Vadim Nebogatove wrote (in email):

> I tried to make tests today with literal constants. Possible I implemented changes in rules not quite correct like you wanted, because result of conversion looks like getNext(null, null). getNodeString returns null for some reason.

It sounds like the "javaname" annotation is not being written to the lock type literal node. Look at the .ast file for your test case. Search for the lock type literal node which is associated with the GET-NEXT call that is converting improperly. Does it have a "javaname" node? If not, make sure the conditions you are testing for in database_general.rules when you write that annotation actually exist in the AST. Is there a lock type literal node with a parent of the appropriate type? Do the next sibling conditions match what you are looking for in the rules?

BTW, I noticed one problem in the database_general changes, but I don't think it's the cause of this. When you call setHidden(true), call it like this:

```
copy.parent.nextSibling.setHidden(true)
```

instead of like this:

```
parent.nextSibling.setHidden(true)
```

There are two ASTs in memory when TRPL executes the rule-sets: a source AST and a copy AST. The source AST is meant for reading and testing conditions against. It is the one implicitly in scope when you specify a rule like

```
type == prog.kw_no_lock_literal
```

or

```
parent.type == prog.kw_get_next
```

The copy AST is the one that should be updated with annotations. Convenience methods like putNote automatically operate on the copy AST. However when you are explicitly modifying a particular node, you have to specify copy as the referent, as in:

```
copy.parent.nextSibling.setHidden(true)
```

The TRPL engine walks the source tree, executing a rule-set against that walk. At the end of the walk, the source tree is discarded and the copy tree is saved, to become the source tree for the next rule-set walk. Thus, if you make any change to the source tree during a walk, it will be discarded and lost.

#129 - 03/05/2013 07:40 AM - Vadim Nebogatov

- File vmn_upd20130305a.zip added

- File vmn_upd20130305a_test_cases.zip added

Attached update

vmn_upd20130305a.zip / vmn_upd20130305a_test_cases.zip

is next replacement of update vmn_upd20130301a.zip/vmn_upd20130303a.zip.
Also I included here update vmn_upd20130301b.zip #110 (corrected for: FIND-UNIQUE,FIND-FIRST (KW_FIND_1ST), FIND-LAST (KW_FIND_LST), and added addBuffer(handle bufHandle) (KW_ADD_BUF)

Some comments:

there were some problems with hide nodes: conversion result in this case looks like

```
query0.getNext(LockType.LT_NONE, );
```

so, method still expects second parameter. Maybe it happens because of intermediate EXPRESSION nodes between GET-NEXT and LOCK nodes and also it was impossible to hide also EXPRESSION node because of some errors.
So, I did not hide, but simply removed expression node in such cases. It seems everything works.

Tests with variables and expressions.

I have found out that 4GL does not allow using expressions instead of lock/nowait parameters, only numeric values and variables are allowed.
For example, statement

```
QUERY hQuery:GET-CURRENT(1, 4).
```

is compilable but will give exception in runtime, as I wrote above. But statements

```
QUERY hQuery:GET-CURRENT(1+2, 4).
QUERY hQuery:GET-CURRENT(1, 4-1).
QUERY hQuery:GET-CURRENT(1, (4-1)).
```

are even not compilable.

#130 - 03/05/2013 07:43 AM - Constantin Asofiei

Vadim,

are even not compilable.

the correct syntax for you tests is this:

```
QUERY hQuery:GET-CURRENT(1 + 2, 4).
QUERY hQuery:GET-CURRENT(1, 4 - 1).
QUERY hQuery:GET-CURRENT(1, (4 - 1)).
```

Note the space char between the operand and the operator.

#131 - 03/05/2013 09:31 AM - Vadim Nebogatov

Yes, it is true. Tests passed in this case too with the results like

query1.getNext(LockType.LT_EXCLUSIVE, plus(7, 1));
query1.getNext(plus(7, 1), LockType.LT_NO_WAIT);
query1.getNext(plus(int1, 1), LockType.LT_NO_WAIT);

#132 - 03/05/2013 09:40 AM - Vadim Nebogatov

- File deleted (vmn_upd20130305a_test_cases.zip)

#133 - 03/05/2013 09:41 AM - Vadim Nebogatov

- File vmn_upd20130305a_test_cases.zip added

Updated test cases with expressions

#134 - 03/05/2013 03:38 PM - Eric Faulhaber

- File vmn_upd20130305b.zip added

- File vmn_upd20130305c.zip added

Code review 20130305a:

The changes all look good, but rules/convert/literals.rules was out of date with bzr (conflicts with rev. 10237, checked in yesterday morning) and rules/annotations/database_general.rules was still out of date with bzr (conflicts with rev. 10229, checked in 4 days ago). I merged these up to the latest revision.

The test case was good and thorough. I renamed it get-with-locks.p to be a little more specific as to its content, and I moved it under a "uast" directory, instead of "test", so it can easily be added to the testcases project.

Although the database_general changes were correct, the test case results made it clear to me that there were additional combinations of parameters which could be converted to LockType constants deterministically. I enhanced the rules to deal with the well-known numeric literals (the compiler constants for NO-LOCK, SHARE-LOCK, EXCLUSIVE-LOCK, and NO-WAIT).

The modified code and test case updates are attached. The former currently is undergoing conversion regression testing.

#135 - 03/05/2013 03:44 PM - Vadim Nebogatov

Eric, sorry, but I updated code today from bzr minimum 5 times during the day with bzr update command. Last time immediately before update. It is some strange for me...

#136 - 03/05/2013 03:49 PM - Vadim Nebogatov

I made fresh check out today morning and merged manually from updates with adding only my changes. Maybe I missed something on this stage. But I updated after that several times and there were no conflicts in any case

#137 - 03/05/2013 03:55 PM - Vadim Nebogatov

I tried to update code from bzr now (I did not make any changes since the time I sent update).
Please see result - no conflicts

```
vmn@vmnComp:~/p2j$ bzr update
M  rules/convert/control_flow.rules
M* rules/include/common-progress.rules
M  src/com/goldencode/p2j/convert/NameMappingWorker.java
M* src/com/goldencode/p2j/schema/SchemaDictionary.java
M  src/com/goldencode/p2j/util/SourceNameMapper.java
All changes applied successfully.
Updated to revision 10243 of branch /home/vmn/projects/repo/p2j_repo/p2j
```

#138 - 03/05/2013 05:30 PM - Eric Faulhaber

Update 20130305b hit a snag in conversion regression testing. It did not cause a regression per se, because the change was intentional, but I did not realize there were instances of GET-FIRST and GET-NEXT method use in Majic. The conversion of these has of course changed from first() to getFirst() and from next() to getNext(), respectively. At the moment, this causes a problem, since there is no runtime implementation of the getFirst() and getNext() methods in AbstractQuery and QueryWrapper, so this change will break Majic.

Vadim, please implement the following methods in AbstractQuery and QueryWrapper as your top priority:

```
getFirst()
getNext()
getLast()
getPrevious()
getCurrent()
```

For now, we only care about these no-argument variants.

Don't add any new static methods to AbstractQuery, just implement the instance methods directly, as described in note 61 above (but without the LockType parameter). The QueryWrapper versions should just be one line each, returning the result of a call to the delegate query. For example:

```
public logical getNext()
{
```

```
    return getDelegate().getNext();
}
```

Please post an update with these changes as soon as they are ready. If the code includes any other runtime changes in process -- *for recently added methods only, not untested changes to existing methods* -- that's OK, those other new methods won't be called by Majic. Such partially implemented methods don't need to work, but the update must compile.

#139 - 03/05/2013 11:46 PM - Eric Faulhaber

Vadim, please note that Ovidiu soon will be committing to bzr some changes to files which will overlap with your update, so a merge will be necessary.

#140 - 03/06/2013 08:49 AM - Vadim Nebogatov

- File vmn_upd20130306a_test_cases.zip added

- File vmn_upd20130306a.zip added

Attached update
vmn_upd20130306a.zip / vmn_upd20130306a_test_cases.zip (start/final revisions 10245/10253)
The following methods are implemented in AbstractQuery and QueryWrapper :
getFirst()
getNext()
getLast()
getPrevious()
getCurrent()

#141 - 03/06/2013 08:54 AM - Constantin Asofiei

Vadim, literals.rules is not merged with 10245 (evl_upd20130305a.zip). I'll let Eric review the rest of the update.

#142 - 03/06/2013 09:21 AM - Vadim Nebogatov

Update #134 from Eric was attached at 2013-03-05 11:38PM, so I supposed that it was merged with revision 10245 that was committed at 2013-03-05 16:37:01 -0500

As Greg wrote, I updated from bzr up to revision 10245, replaced files from update, removed my history entries, executed bzr update (up to latest revision) and created my history entries again

#143 - 03/06/2013 09:25 AM - Vadim Nebogatov

As I see, update from Eric was not merged with revision you pointed. Maybe I am wrong with time calculation?

#144 - 03/06/2013 09:35 AM - Vadim Nebogatov

I will merge everything manually and resubmit

#145 - 03/06/2013 10:29 AM - Eric Faulhaber

Code review 20130306a:

Other than the literal.rules merge issue noted above (which I do think was my mistake), everything here looks good.

#146 - 03/06/2013 10:58 AM - Vadim Nebogatov

- File vmn_upd20130306b.zip added

I have attached one more variant of the same update, but merged manually

#147 - 03/06/2013 11:21 AM - Eric Faulhaber

- File vmn_upd20130306b.zip added

I made a minor change to the database_general.rules header. Please check in and distribute the attached version of your update.

LE: BTW, this version passed conversion regression testing.

#148 - 03/06/2013 11:21 AM - Eric Faulhaber

- File deleted (vmn_upd20130306b.zip)

#149 - 03/06/2013 01:37 PM - Eric Faulhaber

- Status changed from WIP to Closed

- Priority changed from High to Normal

Vadim committed this update to bzr as revision number 10256.

#150 - 03/11/2013 11:05 AM - Constantin Asofiei

- File ca_upd20130311d.zip added

Added missing API bufferHandle(integer). Committed to bzr revision ~~10276~~ 10277.

#151 - 03/22/2013 05:23 AM - Constantin Asofiei

See note 202 from #2068, which documents a server failure. Note that the REPOSITION-TO-ROWID can accept mixed extent and non-extent rowid values, as in:

```
def var h as handle.
def var rids as rowid extent 10.
def var rr as rowid.
h:REPOSITION-TO-ROWID(rr).
h:REPOSITION-TO-ROWID(rids).
h:REPOSITION-TO-ROWID(rr, rids).
h:REPOSITION-TO-ROWID(rids, rr).
h:REPOSITION-TO-ROWID(rr, rids, rr).
/* etc */
```

#152 - 03/22/2013 06:06 AM - Constantin Asofiei

I'm working on note 151.

#153 - 03/22/2013 06:49 AM - Constantin Asofiei

- File ca_upd20130322c.zip added

Solved note 151 by always expanding array vars passed to the REPOSITION-TO-ROWID method. This avoids adding an queryRepositionByID(Object... ids) API. Note that with this approach, if functions returning extent are passed, it will still not work, but I don't think this is encountered in the server project.

I'm putting this through conversion testing.

#154 - 03/22/2013 07:42 AM - Constantin Asofiei

- File ca_upd20130322e.zip added

Adds missing queryRows(integer) API.

#155 - 03/22/2013 09:30 AM - Constantin Asofiei

Both 0322c.zip and 0322e.zip have passed conversion regression testing.

#156 - 03/22/2013 10:40 AM - Eric Faulhaber

Both updates look good, please check them in.

#157 - 03/22/2013 10:46 AM - Constantin Asofiei

CA 0322c.zip was committed to bzr revision 10315 and CA 0322e.zip to 10316.

#158 - 04/18/2013 11:09 AM - Eric Faulhaber

- Estimated time set to 180.00

#159 - 11/16/2016 11:06 AM - Greg Shah

- Target version changed from Milestone 4 to Conversion Support for Server Features

Files

vmn_upd20130231.zip	370 KB	01/31/2013	Vadim Nebogatov
vmn_upd20130231_test_cases.zip	3.1 KB	01/31/2013	Vadim Nebogatov
isue1668.ods	23.4 KB	01/31/2013	Vadim Nebogatov
vmn_upd20130205a.zip	377 KB	02/06/2013	Vadim Nebogatov
vmn_upd20130205a_test_cases.zip	1.82 KB	02/06/2013	Vadim Nebogatov
vmn_upd20130207a.zip	386 KB	02/07/2013	Vadim Nebogatov
vmn_upd20130207a_test_cases.zip	1.82 KB	02/07/2013	Vadim Nebogatov
vmn_upd20130207b.zip	384 KB	02/08/2013	Eric Faulhaber
vmn_upd20130211a.zip	390 KB	02/12/2013	Eric Faulhaber
om_upd20130227b.zip	6.49 KB	02/27/2013	Ovidiu Maxiniuc
om_upd20130227c.zip	6.2 KB	02/27/2013	Greg Shah
vmn_upd20130301a.zip	72 KB	03/01/2013	Vadim Nebogatov
vmn_upd20130201a_test_cases.zip	1.88 KB	03/01/2013	Vadim Nebogatov
om_upd20130301a.zip	5.77 KB	03/01/2013	Ovidiu Maxiniuc
vmn_upd20130301b.zip	63.2 KB	03/01/2013	Vadim Nebogatov
vmn_upd20130201b_test_cases.zip	1.18 KB	03/01/2013	Vadim Nebogatov

vmn_upd20130301c.zip	61.5 KB	03/01/2013	Vadim Nebogatov
vmn_upd20130201c_test_cases.zip	1.43 KB	03/01/2013	Vadim Nebogatov
ges_upd20130302a.zip	16.5 KB	03/02/2013	Greg Shah
vmn_upd20130303a.zip	67.8 KB	03/03/2013	Vadim Nebogatov
vmn_upd20130303a_test_cases.zip	1.43 KB	03/03/2013	Vadim Nebogatov
vmn_upd20130305a.zip	78.1 KB	03/05/2013	Vadim Nebogatov
vmn_upd20130305a_test_cases.zip	39.9 KB	03/05/2013	Vadim Nebogatov
vmn_upd20130305b.zip	77.7 KB	03/05/2013	Eric Faulhaber
vmn_upd20130305c.zip	1.14 KB	03/05/2013	Eric Faulhaber
vmn_upd20130306a.zip	79.3 KB	03/06/2013	Vadim Nebogatov
vmn_upd20130306a_test_cases.zip	1.27 KB	03/06/2013	Vadim Nebogatov
vmn_upd20130306b.zip	79.3 KB	03/06/2013	Eric Faulhaber
ca_upd20130311d.zip	36.6 KB	03/11/2013	Constantin Asofiei
ca_upd20130322c.zip	5.1 KB	03/22/2013	Constantin Asofiei
ca_upd20130322e.zip	38.2 KB	03/22/2013	Constantin Asofiei