Base Language - Bug #7184

Incorrect conversion of concatenated string literals to a LONGCHAR

03/09/2023 08:42 AM - Vladimir Tsichevski

Status:	Review	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Vladimir Tsichevski	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD		

Description

History

#1 - 03/09/2023 08:43 AM - Vladimir Tsichevski

The code:

```
CLASS unittests.testlongcharconversion:

  METHOD PUBLIC VOID testMethod(INPUT arg AS LONGCHAR):
  END METHOD.

  METHOD PUBLIC VOID test():
     testMethod("asdf" + "qwerty").
  END METHOD.

END CLASS.
```

The call to testMethod is converted as

```
testMethod(new character("asdfqwerty"));
```

which cannot be compiled.

BTW, the call with no concatenation

```
testMethod("asdf").
```

is converted as

```
testMethod(new longchar(new longchar("asdf")));
```

which is IMO suboptimal.

This problem was first mentioned in #7143-51.

#2 - 03/09/2023 08:49 AM - Greg Shah

- Project changed from Conversion Tools to Base Language

#3 - 03/28/2023 07:38 PM - Vladimir Tsichevski

- Status changed from New to WIP

In the course of string literal concatenation the original annotation info is lost.

Here is how a string literal expression passed as a legacy method parameter looks like:

```
                  <ast col="17" id="365072220224" line="8" text="&amp;quot;asdf&amp;quot;" type="STRING">
                    <annotation datatype="java.lang.Boolean" key="is-literal" value="true"/>

                    <annotation datatype="java.lang.Long" key="param_index" value="0"/>
                    <annotation datatype="java.lang.String" key="jtype" value="longchar"/>
                    <annotation datatype="java.lang.Long" key="parmtype" value="702"/>
                    <annotation datatype="java.lang.Boolean" key="wrap_parameter" value="true"/>
                    <annotation datatype="java.lang.String" key="classname" value="longchar"/>
                    <annotation datatype="java.lang.String" key="calltype" value="character"/>
                    <annotation datatype="java.lang.Long" key="support_level" value="16400"/>
                  </ast>
```

And this is how the result of literal concatenation looks like:

```
                  <ast col="0" id="365072220237" line="0" text="&amp;quot;asdfqwerty&amp;quot;" type="STRING">
                    <annotation datatype="java.lang.Boolean" key="is-literal" value="true"/>
                  </ast>
```

#4 - 03/29/2023 10:11 AM - Vladimir Tsichevski

I need some help from TRPL gurus in deciding how to solve this issue. I see the following variants:

1. explicitly copy all **missing** annotations from the first literal node to the result node. This is clumsy, and does not guarantee the problem can arise again in the future if any new annotations are be added, and we forget to update the ruleset for these new annotations.
2. copy **all** annotations from the first literal node to the result node. The problem I see here some annotations may exist which **must not** be copied. I cannot answer this question. The other problem is I do not know how to do this nicely in TRPL.
3. move the whole literal string concatenation operation to some more recent conversion stage **before** all these missing annotations are added. I would prefer this variant.

#5 - 03/29/2023 10:12 AM - Constantin Asofiei

What's the full AST for the method call testMethod("asdf" + "qwerty").?

#6 - 03/29/2023 10:17 AM - Vladimir Tsichevski

- File testlongcharconversion.cls.ast added

Constantin Asofiei wrote:

> What's the full AST for the method call testMethod("asdf" + "qwerty").?

I've attached the full AST as testlongcharconversion.cls.ast.

#7 - 03/29/2023 10:18 AM - Constantin Asofiei

Vladimir Tsichevski wrote:

> Constantin Asofiei wrote:
>
> > What's the full AST for the method call testMethod("asdf" + "qwerty").?
>
> I've attached the full AST as testlongcharconversion.cls.ast.

Please attach also the AST from the F2 phase (after parsing) and also the .cache for the converted program.

#8 - 03/29/2023 10:23 AM - Vladimir Tsichevski

- File testlongcharconversion.cls.cache added

- File testlongcharconversion.cls.F2only.ast added

Constantin Asofiei wrote:

> Please attach also the AST from the F2 phase (after parsing) and also the .cache for the converted program.

Done. The .cache file does not differ from the source.

#9 - 03/29/2023 10:27 AM - Constantin Asofiei

Somewhere in TRPL (I think annotations) we transform this AST:

```
<ast col="24" id="365072220212" line="7" text="+" type="PLUS">
  <annotation datatype="java.lang.Long" key="param_index" value="0"/>
  <annotation datatype="java.lang.String" key="jtype" value="longchar"/>
  <annotation datatype="java.lang.Long" key="parmtype" value="702"/>
  <annotation datatype="java.lang.Boolean" key="wrap_parameter" value="true"/>
  <annotation datatype="java.lang.String" key="classname" value="longchar"/>
  <annotation datatype="java.lang.String" key="calltype" value="character"/>
  <annotation datatype="java.lang.Long" key="support_level" value="16400"/>
  <ast col="17" id="365072220215" line="7" text="&amp;quot;asdf&amp;quot;" type="STRING">
    <annotation datatype="java.lang.Boolean" key="is-literal" value="true"/>
    <annotation datatype="java.lang.Long" key="support_level" value="16400"/>
  </ast>
  <ast col="26" id="365072220216" line="7" text="&amp;quot;qwerty&amp;quot;" type="STRING">
    <annotation datatype="java.lang.Boolean" key="is-literal" value="true"/>
    <annotation datatype="java.lang.Long" key="support_level" value="16400"/>
  </ast>
</ast>
```

to this AST:

```
<ast col="0" id="365072220237" line="0" text="&amp;quot;asdfqwerty&amp;quot;" type="STRING">
  <annotation datatype="java.lang.Boolean" key="is-literal" value="true"/>
  <annotation datatype="java.lang.Long" key="peerid" value="575525617842"/>
</ast>
```

Run conversion using -Drules.tracing=true JVM argument, and this will add some byrule- annotations which point to the TRPL file and line where this annotation has been set. This will help you find where TRPL concatenates the two literals.

Now, regarding what annotations need to be copied - look at what the longachar arg has at the method call:

```
<ast col="17" id="365072220224" line="8" text="&amp;quot;asdf&amp;quot;" type="STRING">
  <annotation datatype="java.lang.Boolean" key="is-literal" value="true"/>
  <annotation datatype="java.lang.Long" key="param_index" value="0"/>
  <annotation datatype="java.lang.String" key="jtype" value="longchar"/>
  <annotation datatype="java.lang.Long" key="parmtype" value="702"/>
  <annotation datatype="java.lang.Boolean" key="wrap_parameter" value="true"/>
  <annotation datatype="java.lang.String" key="classname" value="longchar"/>
  <annotation datatype="java.lang.String" key="calltype" value="character"/>
  <annotation datatype="java.lang.Long" key="support_level" value="16400"/>
</ast>
```

which already exist at the PLUS AST (and the 'get lost' during concatenation). So you need to copy these annotations:

```
  <annotation datatype="java.lang.Long" key="param_index" value="0"/>
  <annotation datatype="java.lang.String" key="jtype" value="longchar"/>
  <annotation datatype="java.lang.Long" key="parmtype" value="702"/>
  <annotation datatype="java.lang.Boolean" key="wrap_parameter" value="true"/>
  <annotation datatype="java.lang.String" key="classname" value="longchar"/>
  <annotation datatype="java.lang.String" key="calltype" value="character"/>
```

from the PLUS literal to the final, concatenated STRING literal.

#10 - 03/29/2023 10:33 AM - Vladimir Tsichevski

Constantin Asofiei wrote:

> Somewhere in TRPL (I think annotations) we transform this AST:
> [...]
> to this AST:
> [...]
>
> Run conversion using -Drules.tracing=true JVM argument, and this will add some byrule- annotations which point to the TRPL file and line where this annotation has been set. This will help you find where TRPL concatenates the two literals.
>
> Now, regarding what annotations need to be copied - look at what the longachar arg has at the method call:
> [...]
> which already exist at the PLUS AST (and the 'get lost' during concatenation). So you need to copy these annotations:
> [...]
> from the PLUS literal to the final, concatenated STRING literal.

Exactly. I knew all this already. The concatenation happens in the constant_expressions.rules (which is overcomplicated already).

The question is which of 3 ways of **fixing** this problem to choose? :-(

#11 - 03/29/2023 10:41 AM - Constantin Asofiei

Vladimir Tsichevski wrote:

> The question is which of 3 ways of **fixing** this problem to choose? :-(

First, you need to run the full conversion with -Drules.tracing=true and also add a printfln("%s", copy.dumpTree(true)) in constant_expression.rules on line 157. This will show you all annotations added to the PLUS node at the time the concatenation is done.

#12 - 03/29/2023 10:55 AM - Vladimir Tsichevski

Constantin Asofiei wrote:

> Vladimir Tsichevski wrote:
>
> > The question is which of 3 ways of **fixing** this problem to choose? :-(

> First, you need to run the full conversion with -Drules.tracing=true and also add a printfln("%s", copy.dumpTree(true)) in constant_expression.rules on line 157. This will show you all annotations added to the PLUS node at the time the concatenation is done.

Done. Seems, we need to transfer the annotations from the PLUS node, not from the first argument of the PLUS operator:

```
+ [PLUS]:365072220212 @7:24
      (param_index=0, jtype=longchar, parmtype=702, wrap_parameter=true, classname=longchar, calltype=characte
r, support_level=16400, support_level-ByRule=gaps/gap_analysis_marking.xml:435)
   "asdf" [STRING]:365072220215 @7:17
         (is-literal=true, support_level=16400, support_level-ByRule=gaps/gap_analysis_marking.xml:435)
   "qwerty" [STRING]:365072220216 @7:26
         (is-literal=true, support_level=16400, support_level-ByRule=gaps/gap_analysis_marking.xml:435)
```

#13 - 03/29/2023 10:56 AM - Constantin Asofiei

Vladimir Tsichevski wrote:

> Seems, we need to transfer the annotations from the PLUS node, not from the first argument of the PLUS operator:

Yes, this was my assumption, 'option 4', transfer all annotations from the PLUS node to the new STRING literal node. In constant_expression.rules, you need to change this code:

```
        <rule>litType != null
            <action>ref = copy.parent</action>
            <action>refIdx = copy.indexPos</action>

            <action>copy.remove()</action>

            <action>ref = createProgressAst(litType, literal.toString(), ref, refIdx)</action>
            <action>ref.putAnnotation("is-literal", true)</action>
         </rule>
```

to do this:

- save the copy reference in some ref2 var, before doing copy.remove().
- copy all annotations from ref2 to the ref AST

#14 - 03/29/2023 10:58 AM - Vladimir Tsichevski

Constantin Asofiei wrote:

- copy all annotations from ref2 to the ref AST

Is there a nice way to do this in TRPL?

#15 - 03/29/2023 11:00 AM - Constantin Asofiei

Vladimir Tsichevski wrote:

> Constantin Asofiei wrote:
>
> - copy all annotations from ref2 to the ref AST
>
> Is there a nice way to do this in TRPL?

Something like this:

```
<rule>iter = ref2.annotationKeys()</rule>
<while>iter.hasNext()
   <action>akey = #(java.lang.String) iter.next()</rule>
   <action>ref.putAnnotationObject(akey, ref2.getAnnotation(akey))</rule>
</while>
```

#16 - 03/29/2023 11:30 AM - Vladimir Tsichevski

- Status changed from WIP to Review

- File constant_expressions.rules added

- % Done changed from 0 to 100

Constantin Asofiei wrote:

> Vladimir Tsichevski wrote:
>
> > Constantin Asofiei wrote:
> >
> > - copy all annotations from ref2 to the ref AST
> >
> > Is there a nice way to do this in TRPL?
>
> Something like this:
> [...]

Great! Now it works as charm!

The fixed constant_expressions.rules is attached. Please, review.

#17 - 03/29/2023 11:34 AM - Constantin Asofiei

Why did you remove the descent-rules portion? There can be nested PLUS ASTs, with complex expression with or without STRING literal. The changes need to be only in this code:

```
        <rule>litType != null
            <action>ref = copy.parent</action>
            <action>refIdx = copy.indexPos</action>

            <action>copy.remove()</action>

            <action>ref = createProgressAst(litType, literal.toString(), ref, refIdx)</action>
            <action>ref.putAnnotation("is-literal", true)</action>
         </rule>
```

#18 - 03/29/2023 11:51 AM - Vladimir Tsichevski

Constantin Asofiei wrote:

> There can be nested PLUS ASTs, with complex expression with or without STRING literal.

Would you provide an example, please.

#19 - 03/29/2023 11:53 AM - Constantin Asofiei

Vladimir Tsichevski wrote:

> Constantin Asofiei wrote:
>
> > There can be nested PLUS ASTs, with complex expression with or without STRING literal.
>
> Would you provide an example, please.

Anything like "a" + ch + "B" + "c" + ch2 or even integer expressions, etc, or even "a" + "b" + func0("c" + "d" + "e") + "f" + "g"). The code was written that way for a reason, please keep the changes pinpointed to fix this specific issue.

#20 - 03/29/2023 12:01 PM - Vladimir Tsichevski

Constantin Asofiei wrote:

> Vladimir Tsichevski wrote:
>
> > Constantin Asofiei wrote:
> >
> > > There can be nested PLUS ASTs, with complex expression with or without STRING literal.
> >
> > Would you provide an example, please.
>
> Anything like "a" + ch + "B" + "c" + ch2 or even integer expressions, etc, or even "a" + "b" + func0("c" + "d" + "e") + "f" + "g"). The code was written that way for a reason, please keep the changes pinpointed to fix this specific issue.

All examples of yours are compiled nicely.

```
CLASS unittests.testlongcharconversion:

  METHOD PUBLIC longchar testMethod(INPUT arg AS LONGCHAR):
      return arg.
  END METHOD.

  METHOD PUBLIC VOID testMethod(INPUT arg AS DECIMAL):
  END METHOD.

  METHOD PUBLIC VOID test():
     testMethod("asdf" + "qwerty").
     testMethod("asdf").
     // Now with all quote combinations
     testMethod("a'sdf" + 'qwe"rty').
     testMethod('as"df' + "qwe'rty").
     testMethod('as"df' + 'qwe"rty').
     // NOW three arguments
     testMethod('as"df' + 'qwe"rty' + "third arg").

     // nested mixed expression
     testMethod(4 + 5).
     testMethod('asdf' + STRING('qwerty') + 'zxcv').
     DEFINE VARIABLE ch AS CHARACTER NO-UNDO.
     DEFINE VARIABLE ch2 AS CHARACTER NO-UNDO.
     testMethod("a" + ch + "B" + "c" + ch2).
     testMethod("a" + "b" + testMethod("c" + "d" + "e") + "f" + "g").

END METHOD.
```

Converts to

```
   @LegacySignature(type = Type.METHOD, name = "test")
   public void test()
   {
      character ch = TypeFactory.character();
      character ch2 = TypeFactory.character();

      internalProcedure(Testlongcharconversion.class, this, "test", new Block((Body) () ->
      {
         testMethod(new longchar(new longchar("asdfqwerty")));
         testMethod(new longchar(new longchar("asdf")));

         //  Now with all quote combinations
         testMethod(new longchar(new longchar("a'sdfqwe\"rty")));
         testMethod(new longchar(new longchar("as\"dfqwe'rty")));
         testMethod(new longchar(new longchar("as\"dfqwe\"rty")));
         testMethod(new longchar(new longchar("as\"dfqwe\"rtythird arg")));
         testMethod(new decimal(plus(4, 5)));
         testMethod(new longchar(new longchar(concat("asdf", valueOf("qwerty"), "zxcv"))));
         testMethod(new longchar(new longchar(concat("a", ch, "B", "c", ch2))));
         testMethod(new longchar(concat("ab", testMethod(new longchar(new longchar("cde"))), "f", "g")));
      }));
   }
```

The only problem I see if the new longchar(new longchar.. construct. I mentioned this already in the first task comment. But this problem is not related to the original problem and the ruleset.

#21 - 03/29/2023 12:41 PM - Constantin Asofiei

OK, lets back-track a little. Sorry I was so abrupt in the initial note, I was expecting a pin-point fix, not a full refactor. Some notes of my reasoning:

- when refactoring a file, please explain the reasoning before asking for the review (or just a heads up).
- after looking again at the file, I think there is no reason to add addDictionaryString("constant", "exprType", ecw.expressionType(this, false)). I don't recall why I did it in the first place, and your refactoring makes sense.
- the literal can be dropped completely, as ref.text is always STRING...
- we will need to regression test the conversion of the original app where this was done, plus some other customer apps. Although the refactoring makes sense, as this is not 'just a pinpoint fix', we can't take a chance without regression testing.

Also, can you track down where the new longchar(new longchar(gets emitted?

And another question: is this bug only for OO method calls, or for function calls with longchar parameters and 'concatenated string' argument, too?

#22 - 03/29/2023 02:19 PM - Vladimir Tsichevski

Constantin Asofiei wrote:

> OK, lets back-track a little. Sorry I was so abrupt in the initial note, I was expecting a pin-point fix, not a full refactor. Some notes of my reasoning:
>
> - when refactoring a file, please explain the reasoning before asking for the review (or just a heads up).

Correct. Mea culpa.

> - after looking again at the file, I think there is no reason to add addDictionaryString("constant", "exprType", ecw.expressionType(this, false)). I don't recall why I did it in the first place, and your refactoring makes sense.

The good thing is that the original author (you) is available and can help :-)

> - the literal can be dropped completely, as ref.text is always STRING...

Done. Thanks.

> - we will need to regression test the conversion of the original app where this was done, plus some other customer apps. Although the refactoring makes sense, as this is not 'just a pinpoint fix', we can't take a chance without regression testing.

OK. How can we do this technically?

> Also, can you track down where the new longchar(new longchar(gets emitted?

In progress. Right now I see this problem is legacy method calls specific, it does not exist for a function calls.

> And another question: is this bug only for OO method calls, or for function calls with longchar parameters and 'concatenated string' argument, too?

This bug is only for OO method calls.

#23 - 03/29/2023 02:28 PM - Constantin Asofiei

Vladimir Tsichevski wrote:

- we will need to regression test the conversion of the original app where this was done, plus some other customer apps. Although the refactoring makes sense, as this is not 'just a pinpoint fix', we can't take a chance without regression testing.

OK. How can we do this technically?

Please create a task in Regression Testing project and we will discuss there.

And another question: is this bug only for OO method calls, or for function calls with longchar parameters and 'concatenated string' argument, too?

This bug is only for OO method calls.

Thanks, I think this is because for function calls we emit an EXPRESSION (or PARAMETER? I don't recall exactly) parent node instead of the argument being directly attached to the FUNC_ call AST. But, there are cases when this doesn't happen... I think when a function call is as an argument to another function call.

#24 - 03/29/2023 03:17 PM - Vladimir Tsichevski

If I remove the "wrap_parameter" annotation from the literal, both OO method and function calls convert correctly with no double-wrapping.
I suppose, this annotation causes the wrapping, and must be removed when the wrapping is done. Probably, it is added twice somehow.

#25 - 03/29/2023 04:59 PM - Vladimir Tsichevski

First time the literal is wrapped as a parameter in expression.rules:

```
      <!-- wrap parameter expressions, if needed -->
```

```
      <rule>isNote("wrap_parameter") and getNoteBoolean("wrap_parameter")
         <action>classname = getNoteString("classname")</action>
         <action>createPeerAst(java.constructor, classname, closestPeerId)</action>
      </rule>
```

Second time the literal is wrapped in literals.rules:

```
            <rule>classname != "date"        and
                  classname != "datetime"    and
                  classname != "datetimetz"  and
                  !classname.equals("BaseDataType") and
                  type != prog.unknown_val
               <!-- insert an intermediate constructor node -->
               <action>
                  lastid = createJavaAst(java.constructor, classname, closestPeerId)
               </action>

               <!-- 'temporary' peerid -->
               <action>lastid != 0 and putNote('peerid', lastid)</action>
            </rule>
```

I think, only non-literal the parameters should be wrapped, so the first block of code should be look like:

```
      <rule>isNote("wrap_parameter")
            and getNoteBoolean("wrap_parameter")
            and type != prog.string
            ^^^^^^^^^^^^^^^^^^^^^^^
```

This cures the problem in question at least. May be some generalization on all literal types is required/possible?

#26 - 03/30/2023 11:20 AM - Constantin Asofiei

I think the solution would be in literals.rules on line 371:

```
         <rule>
            !isNote("force_no_wrap") and
```

to add a !(isNote("wrap_parameter") and getNoteBoolean("wrap_parameter")) condition beside !isNote("force_no_wrap") - the point is, if this literal is wrapped for the parameter, it should not be wrapped again.

Please create 7184a with these two changes; it will need to be regression-tested for conversion, but is a low priority at this time.

#27 - 03/30/2023 11:34 AM - Vladimir Tsichevski

Constantin Asofiei wrote:

> I think the solution would be in literals.rules on line 371:
> [...]
> to add a !(isNote("wrap_parameter") and getNoteBoolean("wrap_parameter")) condition beside !isNote("force_no_wrap") - the point is, if this literal is wrapped for the parameter, it should not be wrapped again.

I think, there is something wrong with the idea in general: there are two clashing methods:

1. convert (wrap) values to the expected type is cases the usage requires it (to use values as parameters, for example). This sounds reasonable.
2. convert all literals regardless of the usage.

I think, only the first method should remain, and followed thoroughly.

Other solution could be: create a procedure which **assures** the value is of the desired type in Java. If the value type matches the expected, then do nothing, otherwise add conversion wrapper.

> Please create 7184a with these two changes; it will need to be regression-tested for conversion, but is a low priority at this time.

By the "two changes" you mean the fix to the original problem and and the fix to double-wrapping problem?

#28 - 03/30/2023 11:39 AM - Constantin Asofiei

Vladimir Tsichevski wrote:

> 1. convert (wrap) values to the expected type is cases the usage requires it (to use values as parameters, for example). This sounds reasonable.
> 2. convert all literals regardless of the usage.
>
> I think, only the first method should remain, and followed thoroughly.

These are two separate 'wrapping' and each must remain:

- "wrap_parameter" in expressions.rules ensures that the argument is wrapped in a type for the expected call (for i.e. direct OO method calls or function calls). In this case, the argument type **must match** the parameter type of the call target.
- the "wrap" annotation is used (among others) for other Java calls not associated with the first one; think builtin functions like ASC which get converted to their own FWD API call.

So both must remain; but currently there is an overlap (which you found). OTOH, we should find who sets the wrap annotation for the literal - did you convert with -Drules.tracing=true?

> By the "two changes" you mean the fix to the original problem and and the fix to double-wrapping problem?

Both fixes.

#30 - 04/03/2023 09:12 AM - Vladimir Tsichevski

The repo 7184a was created for the fix, and the regression test task #7245.

#31 - 04/03/2023 09:31 AM - Vladimir Tsichevski

Constantin Asofiei wrote:

> OTOH, we should find who sets the wrap annotation for the literal - did you convert with -Drules.tracing=true?

There is no "wrap" annotation in the literal, but there is a "wrap_parameter" annotation, and there is no rule tracing annotation for the "wrap_parameter" annotation :-(

#32 - 04/03/2023 09:34 AM - Constantin Asofiei

Vladimir Tsichevski wrote:

> Constantin Asofiei wrote:
>
> > OTOH, we should find who sets the wrap annotation for the literal - did you convert with -Drules.tracing=true?
>
> There is no "wrap" annotation in the literal, but there is a "wrap_parameter" annotation, and there is no rule tracing annotation for the "wrap_parameter" annotation :-(

It means they are added by the parser. I assume you see other byrule- annotations in the AST, correct?

#33 - 04/03/2023 10:34 AM - Vladimir Tsichevski

Constantin Asofiei wrote:

> It means they are added by the parser. I assume you see other byrule- annotations in the AST, correct?

Correct. Other ByRule- annotations are in place.

#34 - 04/03/2023 04:22 PM - Vladimir Tsichevski

Constantin Asofiei wrote:

> I think the solution would be in literals.rules on line 371:
> to add a !(isNote("wrap_parameter") and getNoteBoolean("wrap_parameter")) condition beside !isNote("force_no_wrap") - the point is, if this literal is wrapped for the parameter, it should not be wrapped again.

This wraps the string literal in annotation values too, which is not correct:

```
   @Test(expected = new character("OEUnit.Assertion.AssertionFailedError"))
                    ^^^^^^^^^^^^^^^
   public void areEqualCharacterMustFail()
   {
      internalProcedure(TestOeunitAssert.class, this, "areEqualCharacterMustFail", new Block((Body) () ->
      {
         Assert.areEqual(characterLeft, characterRight);
      }));
   }
```

#35 - 04/10/2023 09:47 AM - Vladimir Tsichevski

- Assignee set to Vladimir Tsichevski

#36 - 09/18/2023 03:11 PM - Vladimir Tsichevski

Constantin, can we close this?

Files

testlongcharconversion.cls.ast	23.4 KB	03/29/2023	Vladimir Tsichevski
testlongcharconversion.cls.F2only.ast	15 KB	03/29/2023	Vladimir Tsichevski
testlongcharconversion.cls.cache	227 Bytes	03/29/2023	Vladimir Tsichevski
constant_expressions.rules	6.63 KB	03/29/2023	Vladimir Tsichevski