Base Language - Feature #1600

Feature # 1596 (Closed): complete big decimal implementation

provide a conversion option to emit decimal literals using a string literal instead of double literals

10/19/2012 08:15 AM - Greg Shah

Status:	Closed	**Start date:**	01/30/2014
Priority:	Normal	**Due date:**	02/10/2014
Assignee:	Hynek Cihlar	**% Done:**	100%
Category:		**Estimated time:**	32.00 hours
Target version:	Cleanup and Stablization for Server Features		
billable:	No	**vendor_id:**	GCD

Description

History

#1 - 10/31/2012 01:49 PM - Greg Shah

- Target version set to Milestone 4

#2 - 02/07/2013 10:45 AM - Greg Shah

- Target version changed from Milestone 4 to Milestone 11

#3 - 05/03/2013 10:06 AM - Greg Shah

- Start date changed from 10/19/2012 to 08/09/2013

- Due date set to 08/14/2013

- Assignee set to Ovidiu Maxiniuc

#4 - 01/21/2014 01:33 PM - Greg Shah

- Start date changed from 08/09/2013 to 01/30/2014

- Due date changed from 08/14/2013 to 02/10/2014

- Assignee changed from Ovidiu Maxiniuc to Hynek Cihlar

#5 - 04/28/2014 09:27 AM - Greg Shah

- Subject changed from provide a conversion option to emit decimal literals as string literals instead of double literals AND make all needed overloaded method additions to the runtime to honor this to provide a conversion option to emit decimal literals using a string literal instead of double literals

double literals in Java can encode values in source code which cannot be properly represented by the actual double type at runtime. We actually have encountered some of these in real customer code. BigDecimal can fully represent these values, but it cannot do so when passed a double literal because the data will already have been changed before the BigDecimal constructor gets it. The solution is to put the exact digits in a String and pass this into the decimal constructor.

This code:

```
my-dec-var = 13.1111111111.
```

does this now:

```
myDecVar.assign(13.1111111111);
```

and we need to do this instead:

```
myDecVar.assign(new decimal("13.1111111111"));
```

Initially, I thought about making all needed overloaded method additions in the runtime to honor String parameters and then we would construct the decimal inside the APIs. But that is simply too much work. So, at this time we only need to make sure that decimal literals always emit as "wrapped" (a String parameter to a decimal constructor).

#6 - 05/11/2014 09:34 PM - Hynek Cihlar

- Status changed from New to WIP

- Tracker changed from Feature to Bug

#7 - 05/11/2014 09:45 PM - Hynek Cihlar

- Tracker changed from Bug to Feature

#8 - 05/12/2014 12:46 PM - Hynek Cihlar

- File hc_upd20140512a.zip added

- Tracker changed from Feature to Bug

Please review the attached changes. The related test cases are committed as uast/decimal_literal_precision.p.

#9 - 05/12/2014 12:52 PM - Hynek Cihlar

- Status changed from WIP to Review

#10 - 05/12/2014 02:14 PM - Greg Shah

Code Review 0512a

The code looks fine. My only question is about this:

```
                  <action on="false">
                     big_wrapper = createPeerAst(java.constructor, "decimal", parentid)
                     createPeerAst(java.string, text, big_wrapper)
                  </action>
```

Does our TRPL implementation really allow 2 expressions to be part of a single <action></action>? I thought our implementation would just ignore the second line in this case since the first line is a complete expression (actually it is a primary_expr + ASSIGN operator + expr in the expression rule of com/goldencode/expr/expression.g which is where this gets parsed).

#11 - 05/12/2014 03:33 PM - Hynek Cihlar

- File hc_upd20140512b.zip added

Greg Shah wrote:

> Code Review 0512a
>
> The code looks fine. My only question is about this:
>
> [...]
>
> Does our TRPL implementation really allow 2 expressions to be part of a single <action></action>? I thought our implementation would just ignore the second line in this case since the first line is a complete expression (actually it is a primary_expr + ASSIGN operator + expr in the expression rule of com/goldencode/expr/expression.g which is where this gets parsed).

Yes, you are correct. I am attaching the correct version.

#12 - 05/12/2014 03:53 PM - Greg Shah

Code Review 0512b

Looks good. Get it regression tested. Both conversion and runtime testing are necessary. The conversion testing will be tricky because of so many diffs. Try to sort it out as best you can.

#13 - 05/13/2014 12:07 PM - Hynek Cihlar

- Tracker changed from Bug to Feature

#14 - 05/16/2014 08:47 AM - Hynek Cihlar

- Tracker changed from Feature to Bug

I am wondering, whether we should emit the wrapper even for (effectively) whole numbers? Ex. instead of .assign(new decimal("2.0")) emit .assign(2.0). This would make the code leaner.

#15 - 05/16/2014 09:04 AM - Hynek Cihlar

- Tracker changed from Bug to Feature

#16 - 05/16/2014 09:37 AM - Greg Shah

We very much like to avoid wrapping when unnecessary. Your idea about whole numbers is a good one. It does have to following issues:

1. The Java double can only faithfully encode integers up to 17 digits in size. The 4GL decimal can handle up to 50 digits (or 40+ if there is a fractional part to the number). So we would have to re-introduce some safety code in the conversion implementation. This is not the biggest deal, especially because such very large integers are less common.

2. The bigger concern is that it is easy for future readers/editors of the code to modify that double later, perhaps changing it from a whole number to one with a fraction, without realizing the potential for loss of precision. In other words, it requires the person seeing the code later to have more knowledge in order to safely edit the code.

We need to weigh these issues versus the improvement in the code.

#17 - 05/16/2014 10:26 AM - Hynek Cihlar

Greg Shah wrote:

> We very much like to avoid wrapping when unnecessary. Your idea about whole numbers is a good one. It does have to following issues:
>
> 1. The Java double can only faithfully encode integers up to 17 digits in size. The 4GL decimal can handle up to 50 digits (or 40+ if there is a fractional part to the number). So we would have to re-introduce some safety code in the conversion implementation. This is not the biggest deal, especially because such very large integers are less common.
>
> 2. The bigger concern is that it is easy for future readers/editors of the code to modify that double later, perhaps changing it from a whole number to one with a fraction, without realizing the potential for loss of precision. In other words, it requires the person seeing the code later to have more knowledge in order to safely edit the code.
>
> We need to weigh these issues versus the improvement in the code.

- If the literal is an integer (people usually omit the decimal point), it will be emitted without the wrapper resulting in the similar situation - the editor may not realize the potential loss of precision when changing to a decimal literal.
- The pattern of potentially loosing precision is no different from the plain Java API, so the editor should be able to recognize the error.

The reason I brought this up is that I am seeing many places in the converted code, where the decimal literal is effectively an integer and comparably only a few occurrences of real decimals.

#18 - 05/16/2014 11:45 AM - Greg Shah

Good points. OK, go ahead with your approach. Please do put in the code to detect the 17 digit issue at conversion time. If we are over that limit, then use the string approach. Otherwise, the double literal is fine for whole numbers.

#19 - 05/18/2014 06:32 PM - Hynek Cihlar

- File hc_upd20140518a.zip added

Attached is the implementation with the special handling of mathematical integers. Now the decimal literal is emitted without the decimal wrapper when the decimal value is effectively a mathematical integer and the value fits in a Java double without loosing precision. I didn't use the previous implementation of counting the number of digits when determining whether precision would be lost but rather converting the value to double and comparing it back to the original literal value. Also the implementation is now more robust in terms of localization.

The changes are being regression tested.

#20 - 05/19/2014 09:31 AM - Greg Shah

Code Review 0518a

The changes look good. I really like your approach of using the Java double and BigDecimal implementations to test for loss of precision.

One thought: instead of always emitting the literal as a string, we could just use the string approach when a loss of precision case is found.

The upside is that this would greatly limit the number of places that would be affected (it should be a rare condition after all).

The downside is that future maintainers of the code will not have any warning when they change the literal and it loses precision.

What are your thoughts?

#21 - 05/19/2014 11:46 AM - Hynek Cihlar

Greg Shah wrote:

> Code Review 0518a
>
> The changes look good. I really like your approach of using the Java double and BigDecimal implementations to test for loss of precision.
>
> One thought: instead of always emitting the literal as a string, we could just use the string approach when a loss of precision case is found.

Yes, this is sort of what is happening with with 0518a (if I am following you right), let me illustrate.

```
i = 123456789.5555555555.
i = 123456789.0.
i = 1234567891234567891.0.
i = 123456789.50.
```

converts to

```
i.assign(new decimal("123456789.5555555555"));
i.assign(123456789.0);
i.assign(new decimal("1234567891234567891.0"));
i.assign(new decimal("123456789.50"));
```

Or are you saying that we should not care about the literal being effectively an integer (and do not check for this) and only do the loss of precision check emitting the decimal wrapper (new decimal("0.1")) only when we would lose precision?

> The upside is that this would greatly limit the number of places that would be affected (it should be a rare condition after all).

Yes, this would make the code better maintainable and hopefully prevent other kinds of errors resulting from the cluttered code.

> The downside is that future maintainers of the code will not have any warning when they change the literal and it loses precision.

Certainly not good. On the other hand, this is no different from plain Java and no new model to learn.

#22 - 05/19/2014 02:05 PM - Greg Shah

> Or are you saying that we should not care about the literal being effectively an integer (and do not check for this) and only do the loss of precision check emitting the decimal wrapper (new decimal("0.1")) only when we would lose precision?

Yes, this is what I mean. This brings safety to the cases (at conversion time) where there is an issue, but doesn't over-wrap the cases where it is not needed. Let's go forward with this approach.

#23 - 05/19/2014 02:10 PM - Hynek Cihlar

Greg Shah wrote:

> > Or are you saying that we should not care about the literal being effectively an integer (and do not check for this) and only do the loss of precision check emitting the decimal wrapper (new decimal("0.1")) only when we would lose precision?
>
> Yes, this is what I mean. This brings safety to the cases (at conversion time) where there is an issue, but doesn't over-wrap the cases where it is not needed. Let's go forward with this approach.

Very well, I'll update the logic.

#24 - 05/19/2014 02:29 PM - Hynek Cihlar

- File hc_upd20140519a.zip added

Attached is the change set with the removed check for decimal literals being effectively mathematical integers. Now the logic is clean and simple - does it fit into Java double? (1) No? Wrap it using decimal(String) constructor. (2) Yes? Emit the literal as it is as Java double literal.

#25 - 05/19/2014 03:18 PM - Greg Shah

Code Review 0519a

It looks good. Sorry that I didn't think about this approach before now. It was your good idea for the precision loss testing that brought it to mind.

Go ahead with conversion regression testing. Only do runtime testing if there are changes in the generated MAJIC code.

#26 - 05/19/2014 04:38 PM - Hynek Cihlar

Greg Shah wrote:

> Code Review 0519a
>
> It looks good. Sorry that I didn't think about this approach before now. It was your good idea for the precision loss testing that brought it to mind.

No problem at all.

> Go ahead with conversion regression testing. Only do runtime testing if there are changes in the generated MAJIC code.

There are differences in the converted code, all of them are straightforward and expected. Runtime regression test is in-progress.

#27 - 05/20/2014 05:07 PM - Hynek Cihlar

Regression tests passed. Code changes committed to bzr revision 10535.

#28 - 05/20/2014 05:08 PM - Hynek Cihlar

- % Done changed from 0 to 100

#29 - 05/21/2014 08:41 AM - Greg Shah

- Status changed from Review to Closed

#30 - 11/16/2016 12:07 PM - Greg Shah

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

Files

hc_upd20140512a.zip	8.49 KB	05/12/2014	Hynek Cihlar
hc_upd20140512b.zip	38.7 KB	05/12/2014	Hynek Cihlar
hc_upd20140518a.zip	63.5 KB	05/18/2014	Hynek Cihlar
hc_upd20140519a.zip	32.1 KB	05/19/2014	Hynek Cihlar