# Kepler - Bug #5722

## Check for problems with sanitized RecordToken labels

10/01/2012 06:33 PM - Derik Barseghian

| | | | |
|---|---|---|---|
| **Status:** | Resolved | **Start date:** | 10/01/2012 |
| **Priority:** | Normal | **Due date:** | |
| **Assignee:** | Daniel Crawl | **% Done:** | 90% |
| **Category:** | general | **Estimated time:** | 0.00 hour |
| **Target version:** | 2.5.0 | **Spent time:** | 0.00 hour |
| **Bugzilla-Id:** | 5722 | | |

**Description**

Filing a bug so this doesn't get forgotten - I said I'd try to look into this ptII change, r64639:
---
Sanitize RecordToken or OrderedRecordToken labels, which means that characters like spaces will be converted to underscores. This is necessary so that the string representations of RecordTokens can be reparsed by the expression language. The alternative of modifying the expression language to allow strings as labels causes conflicts in the parser.
---

I worry this change is going to cause problems...anything that expects (e.g. checks equivalence) the data in a label to come out of a RecordToken as it went in will seemingly have a problem. It also sounds like a lossy translation.

### History

#### #1 - 10/01/2012 06:46 PM - Christopher Brooks

The issue here is that if the record field names are not well-formed strings,
then it is not possible to create such a record using a string.

This came up when we tried to use the Test actor with a record that
had a field name.

One alternative would be to change the syntax of the record field so that
spaces and other characters would be permitted. Unfortunately, the
parser started getting conflicts if I use {"field with space" = "foo"}

In either RecordAssembler or RecordDisassembler, my workaround was to check
for a port that matched the field name. If none was found, then to try
substituting spaces for any underscores. This is gross, but worked.

The problem is that arbitrary field names are not going to work as they
did before, the spaces and other characters will be converted to underscores.

#### #2 - 12/05/2012 12:56 PM - Derik Barseghian

So far I haven't been able to come up with a concrete realistic/common usecase where this causes problems. I looked at some of the actors we were concerned about:

The EML 2 Dataset actor actually already converts spaces to underscores for recordToken labels. See "Sedgwick Reserve Aphid Population Growth Rate Experiment" where e.g. "baby greens" becomes "baby_greens".

The DataTurbine actor only uses "timestamps" and "data" for record labels, regardless which of the 2 output type options are chosen, and even on it's newer "specific channel" ports.

The R actor doesn't seem to produce any RecordTokens like I expected. I checked 3 likely candidates:
an R dataframe comes out as:
*dataframe:*/Users/derik/.kepler/cache-2.4/modules/r/Unnamed1_1354739437414/mydataframe-RExpression-1.sav

an R list comes out as:
*object:*/Users/derik/.kepler/cache-2.4/modules/r/Unnamed1_1354739437414/mylist-RExpression-1.sav

an R matrix doesn't come out (ugh).

The Database Query actor produces RecordTokens, but core, cache, and provenance DBs don't have columns with spaces in their names, so I wasn't able to easily cook up an example problem.

I want to keep thinking about this a bit, and discuss w/ others.

If we do keep the r64639 sanitize change, should we consider not making it possible to create unsanitized record labels altogether to be consistent? Right now RecordToken(Map fieldMap) constructor does not sanitize, while RecordToken(String[] labels, Token[] values) does.

#### #3 - 12/05/2012 02:08 PM - ben leinfelder

I would hope that this continues to work.

#### #4 - 12/05/2012 02:28 PM - Derik Barseghian

Labels beginning with digits will also change.
9Volt becomes _9Volt.
9 Volt becomes _9_Volt.

#### #5 - 12/05/2012 02:46 PM - Matt Jones

I don't understand why this is an issue. Records are standard dictionary type data structures, and any string key value containing Unicode characters should be allowed. To do otherwise is stomping on the key space in ways that are just too constraining.

If the expression parser is having trouble with these keys, then it seems to me that is a bug with the expression language. If spaces are used as token delimiters in the expression language, then it must have an escape sequence to indicate that a reserved character such as a space is intended to be used as a literal, not a delimiter. All decent expression languages have this -- couldn't the pt expression language simply use the token escape sequences that are commonly used in regex expressions, such as using a backslash to escape the next character? Unescape it would look like:
my key + 2
versus escaped it looks like:
my\ key + 2

In the second instance, the space is treated as a literal, allowing 'my key' as a key? There are other possibilities from Bash, python, etc. that could be adopted.

Other than spaces, are there any other constraints? Why can't record labels start with a digit? Can they start with punctuation? Are there any other reserved characters?

#### #6 - 12/05/2012 03:50 PM - Christopher Brooks

The problem here is that we made a poor design decision many years ago about record keys. We decided that record keys need to be identifiers that are compatible with the expression language, which basically means that record keys need to be Java identifiers.

I'd entertain a proposal and implementation that would support arbitrary record keys in the expression language.

I took a quick look at modifying the parser to support spaces and got conflicts.

#### #7 - 12/20/2012 07:00 PM - Derik Barseghian

I believe that by reverting these files as follows:

- 64638 ptolemy/actor/lib/OrderedRecordAssembler.java
- 64638 ptolemy/actor/lib/io/test/auto/ReadCSV2.xml
- 64638 ptolemy/actor/lib/RecordDisassembler.java
- 64632 ptolemy/data/type/RecordType.java
- 64632 ptolemy/data/RecordToken.java
- 64638 ptolemy/data/OrderedRecordToken.java

gets us back to Kepler 2.3 behavior -- e.g.:
1) An Ordered Record Assembler can produce tokens with pre and postfix numbers, spaces, and parens in label:
[1Day of Week1 = {"Mon", "Tue"}, 2Rain Fall (in)2 = {3, 7}]

1b) You can also use the Record Disassembler to disassemble that token without any character changes.

2) Trying to use the same token in an Expression will fail for three reasons: the included prefix numbers, spaces, and parens in the labels.

After reverting the above files, I made a change to PtParser.jjt (attached) that allows prefix number and spaces (I haven't gotten parens to work). I initially tried many things to specifically not change the rule for ID and instead make specific rules for the two types of recordToken, but I wasn't able to do this.

While the change gives us Kepler 2.3 behavior plus better handling in the expression language, it's not ideal -- it doesn't support arbitrary strings, and tests failing in data/expr are now 4 instead of 3. The new failure:

==== $PTII/ptolemy/data/expr/test/PtParser.tcl: PtParser-15.0 Test parsing to end of expression. ==== Contents of test case:

```
set p [java::new ptolemy.data.expr.PtParser]
    catch {$p generateParseTree "1 + 2 foo"} errmsg
```

1. Error message is machine dependent, so we look at the first # two lines

1. This hack is necessary because of problems with crnl under windows
   regsub -all [java::call System getProperty "line.separator"] $errmsg "\n" output
   set lines [split $output "\n"]
   list [lindex $lines 0] [lindex $lines 1] [lindex $lines 2]

==== Result was:
java0x3545 {} {}
---- Result should have been: {ptolemy.kernel.util.IllegalActionException: Error parsing expression "1 + 2 foo"} Because: {Encountered " <ID> "foo "" at line 1, column 7.}
---- PtParser-15.0 FAILED

I don't understand this yet. Typing "1 + 2 foo" without quotes into the Expression actor fails in 2.3 and trunk without my changes, and yet the test passes on trunk without my changes.

#### #11 - 12/20/2012 07:13 PM - Derik Barseghian

I've attached two the workflows I've been using to test. One uses the Test actor, and with the patch and reversion, spaces and prefix numbers work in the label and in the test.

Christopher, do you remember what kind of use of the Test actor you were seeing fail that prompted your r64639 change?

Also let me know if you have ideas on a better more general approach, or problems that my change in the patch to how ID's are handled may cause.

#### #12 - 12/21/2012 05:48 PM - Derik Barseghian

I haven't gotten parenthesis (or better, arbitrary strings) working.

Here's some testing showing behavior w/ my patch. All seems to work except note the inconsistency it introduces with being unable to refer to a label with a prefix number in its name.

[myLabel= {4,5}].myLabel

{4, 5}

[myLabel A= {4,5}].myLabel A

{4, 5}

[myLabel A 5= {4,5}].myLabel A 5

{4, 5}

[5myLabel A 5= {4,5}].5myLabel A 5

Error parsing expression "[5myLabel A 5= {4,5}].5myLabel A 5"
Because:
Encountered " <DOUBLE> ".5 "" at line 1, column 22.

{5myLabel A 5= {4,5}}.length()

1

foo = {5myLabel A 5= {4,5}, 6myLab 6={6,7}}

{5myLabel A 5 = {4, 5}, 6myLab 6 = {6, 7}}

bar = {6myLab 6={6,7}}

{6myLab 6 = {6, 7}}

merge(foo,bar)

{5myLabel A 5 = {4, 5}, 6myLab 6 = {6, 7}}

intersect(foo,bar)

{6myLab 6 = {6, 7}}

foo + foo

{5myLabel A 5 = {8, 10}, 6myLab 6 = {12, 14}}

foo * foo

{5myLabel A 5 = {16, 25}, 6myLab 6 = {36, 49}}

foo / foo

{5myLabel A 5 = {1, 1}, 6myLab 6 = {1, 1}}

foo = foo + foo

{5myLabel A 5 = {8, 10}, 6myLab 6 = {12, 14}}

#### #13 - 01/23/2013 03:38 PM - Marten Lohstroh

If certain strings are unacceptable as labels because the parser cannot deal with them, then the only right behavior for records is to reject such labels (as was established by changeset 64633). The problem is that actors like RecordDisassembler and RecordAssembler use a number of complex type constraints that tie port names to labels in RecordTypes. Bluntly renaming labels results in undesired behavior and unexpected typing problems. Because the the sanitation mapping is not invertible, the original port names can no longer function as proper identifiers. E.g., what kind of record should a RecordAssembler with two inputs "a b" and "a_b" produce? And how can RecordDisassembler ensure that its input record has two distinct fields that correspond two its two outputs "c d" and "c_d"? Errors like these would be trapped by RecordType as it requires labels to be unique, but the resulting errors are not very friendly, and the solution as a whole is not very elegant.

If we want to allow record labels that are equally expressive as port names, the expression language needs to be adapted. This is a task that requires some careful thought. I don't have time to look into this right now, but I will look do so by the end of February and come up with a proposal / candidate implementation.

#### #14 - 03/07/2013 02:33 PM - Daniel Crawl

After discussing with Ilkay, we decided this wasn't necessary for 2.4, so retargeting.

#### #15 - 03/27/2013 02:31 PM - Redmine Admin

Original Bugzilla ID was 5722

#### #16 - 04/24/2013 12:55 PM - Marten Lohstroh

*- % Done changed from 0 to 100*

*- Assignee changed from Christopher Brooks to Marten Lohstroh*

Arbitrary string are allowed as record labels as of rev. 66121; the expression language is adapted to make this possible. The change is backward compatible because strings that qualify as valid Java identifiers are still allowed to be used without quoting them e.g., {a=1, b=2} is still accepted, but now {"a"=1, "b "=2} is accepted too. The only thing which should be mentioned that does not extend to the use of arbitrary strings as record labels, is the dot notation to retrieve fields e.g., {a=1,b=2}.a works, but {"a"=1, "b "=2}."b " does not work. This is not a problem because the dot notation is only a shorthand for the get() method, so {"a"=1, "b "=2}.get("b ") works fine. Quotes within quotes must be escaped e.g., {a=1, "b "=2, "_c\"!"=3}.get("_c\"!") yields the value 3.

One additional problem that has been solved is the following: because arbitrary strings include the use of periods, and periods are not allowed in port names (because it makes their identifier ambiguous), therefore RecordAssembler and RecordDisassembler now use the display name of ports (which has no formatting restrictions) to construct their types. Note that the display name is used instead of name only if it is actually defined.

#### #17 - 08/12/2015 03:56 PM - Daniel Crawl

Derik's test workflow expressionOrderedRecordTokenTest.kar is still failing.

#### #18 - 08/14/2015 04:07 PM - Marten Lohstroh

*- % Done changed from 100 to 90*

*- Assignee changed from Marten Lohstroh to Daniel Crawl*

Daniel Crawl wrote:

> Derik's test workflow expressionOrderedRecordTokenTest.kar is still failing.

I reviewed the test case. It contains the following expression:

```
[1Day of Week1 = {"Mon", "Tue"}, 2Rain Fall2 = {3, 7}]
```

Arbitrary strings as labels are allowed, but unless they are valid Java identifiers (and these are not), they must be surrounded by quotation marks.
In Ptolemy II (svn trunk), the following expression parses just fine:

```
["1Day of Week1" = {"Mon", "Tue"}, "2Rain Fall2" = {3, 7}]
```

In Kepler 2.4, however, it fails.

#### #19 - 08/14/2015 04:18 PM - Daniel Crawl

*- Status changed from New to Resolved*

Marten, thanks for clarifying that the labels need quotes. It's fine if it works in the trunk but not 2.4.

### Files

| | | | |
|---|---|---|---|
| recordLabelSpaces.kar | 6.39 KB | 12/05/2012 | ben leinfelder |
| PtParser.jjt.patch | 689 Bytes | 12/21/2012 | Derik Barseghian |
| recordAssemblerTrunkTest.kar | 7.26 KB | 12/21/2012 | Derik Barseghian |
| expressionOrderedRecordTokenTest.kar | 7.41 KB | 12/21/2012 | Derik Barseghian |