Kepler - Bug #2240

add support for null values to data passing among ports

11/02/2005 05:37 PM - Matt Jones

Status:	Resolved	**Start date:**	11/02/2005
Priority:	Immediate	**Due date:**	
Assignee:	Christopher Brooks	**% Done:**	0%
Category:	core	**Estimated time:**	0.00 hour
Target version:	1.0.0beta1	**Spent time:**	0.00 hour
Bugzilla-Id:	2240		

Description

Currently ptolemy and kepler do not support passing null values (sometimes called missing values) among ports, even though this is common in analytical systems like R and SAS. The concept of null is not even defined in the token types. This causes a real problem for data sources that are sparsely populated, as well as data streams that result from data integration operations that might produce null values. We need to extend the underlying token representation to include a concept of null values and the actor framework to protect existing actors that might not know how to handle null values. Because nulls cannot currently be represented in Kepler, none of the existing actors support them. An exception is thrown whenever a missing value is detected by the EML data source, and workflow execution ceases.

Bowers and Jones discussed one possible partial solution to this on IRC, which is summarized here.

1) Override the Token base class to support null values by providing two new methods:
Token.null() sets the token's value to null
boolean Token.isNull() returns true if the token has been set to null

2) Override TypedIOPort to add a new method that takes a boolean "dropNull"
a) by default this could be set to "true"
then the existing get() method would be reimplemented to call the new one with a default of "true" because we can assume that no existing actor written can handle null values, since there is no way to pass null values, and so existing calls to get() will invisibly drop null values.

```
b) if an actor can handle null values, then it passes "false"
        which indicates that the actor knows how to deal with nulls and wants to
receive them

so the changes to IOPort (or maybe TypedIOPOrt) would be:
```

IOPOrt.get(channelindex) becomes get(channelIndex, dropNullValues)
and
IOPort.get(channelIndex, int vectorLength) becomes get(channelIndex, vectorLength, dropNullValues)

so, for example, the new implmenetation of get(channelIndex) would simply call get(channelIndex, true), so existing actors would not even notice the change.

Related issues:

Blocked by Kepler - Bug #2171: Add support for missingValueCode in EML parser	**Resolved**	**08/15/2005**

History

#1 - 12/12/2005 06:11 PM - Christopher Brooks

Hi Edward,
[I'm posting this as a comment to the bug report so as to keep a log of the bug, and will forward it on to Edward if necessary. This is all a little experiment to see if we can discuss the fix in the context of bugzilla.]

One of the Kepler bugs blocking the Kepler release is:
http://bugzilla.ecoinformatics.org/show_bug.cgi?id=2240
which is reproduced below.

Basically, Kepler needs to handle missing data because not all
ecological data sets are complete.

The solution to this problem needs to work in PN and SDF.

The Synchronous/Reactive (SR) domain has the notion of absent values.
For example, domains/sr/lib/Absent.java has this class comment:

```
This actor outputs absent values.  That is, it produces no tokens,
 and it calls the sendClear() method of the output port on each
 firing.
```

The sendClear() method is the IOPort.java sendClear() method.
IOPort also has sendClearInside() and broadcastClear().

Do you have any comments?

Below is the text from the bug report:
Below is the text from the bug report:
--start--
Currently ptolemy and kepler do not support passing null values
(sometimes called missing values) among ports, even though this is
common in analytical systems like R and SAS. The concept of null is
not even defined in the token types. This causes a real problem for
data sources that are sparsely populated, as well as data streams that
result from data integration operations that might produce null
values. We need to extend the underlying token representation to
include a concept of null values and the actor framework to protect
existing actors that might not know how to handle null values.
Because nulls cannot currently be represented in Kepler, none of the
existing actors support them. An exception is thrown whenever a
missing value is detected by the EML data source, and workflow
execution ceases.

Bowers and Jones discussed one possible partial solution to this on
IRC, which is summarized here.

1) Override the Token base class to support null values by providing
two new methods:

```
Token.null() sets the token's value to null
   boolean Token.isNull() returns true if the token has been set to null
```

2) Override TypedIOPort to add a new method that takes a boolean
"dropNull"

```
a) by default this could be set to "true" then the existing get()
   method would be reimplemented to call the new one with a default of
   "true" because we can assume that no existing actor written can
   handle null values, since there is no way to pass null values, and
   so existing calls to get() will invisibly drop null values.

b) if an actor can handle null values, then it passes "false"
    which indicates that the actor knows how to deal with nulls and
    wants to receive them
```

so the changes to IOPort (or maybe TypedIOPOrt) would be:

IOPOrt.get(channelindex) becomes get(channelIndex, dropNullValues)
and
IOPort.get(channelIndex, int vectorLength) becomes get(channelIndex,
vectorLength, dropNullValues)

so, for example, the new implmenetation of get(channelIndex) would
simply call get(channelIndex, true), so existing actors would not even
notice the change.
--end--

_Christopher

#2 - 12/13/2005 09:32 AM - Christopher Brooks

I've modified Token to have isMissing() and missing() methods.

If a model drops missing tokens, then it is probably not SDF.
It could be Dynamic Dataflow (DDF) or possibly PN.

Shawn wrote:

> I think it is ok to call these "missing" values, although in database parlance, they are really referred to as NULL values.
>
> I would think that the right functionality would be to literally "drop" missing values (if the flag in get was set as such); which would mean that the loop would treat these tokens with missing values as if they were null (and thus, keep "looping").
>
> It sounds like, however, in SDF, this might mess up the token consumption/production rate (i.e., an actor may receive multiple missing-valued tokens, and not produce anything). This shouldn't be a problem for PN though, right?
>
> One option for SDF would be to literally "propagate" the missing-valued tokens, instead of "dropping" them (as in PN). Then, it seems the consumption/production rates at least wouldn't be violated. Doing this correctly might be a bit tricky, but could probably perform this based on the token production/consumption rates --
>
> If this is a possible work-around, we might want to change the flag from "dropMissingValues" to "ignoreMissingValues" -- with slightly different semantics depending on the particular director used.
>
> Most scientific workflows it seems are really dataflow driven (pipelines), and so DE and SR don't seem like the right fit (based on my limited knowledge of these domains) ... .

Edward wrote:

> Hmm... This now leads me to change my mind... Perhaps this needs to be more carefully thought through. I suspect we will be introducing a back-door mechanism for getting unexpected nondeterminism by this mechanism... Let's not implement it without further discussion...
>
> It is arguable that if you want a well-defined notion of missing tokens, you should be using SR or DE. Both of these have clean semantics for absent tokens, and it's already fully supported...

Steve wrote:

> I don't think this needs to be handled specially. To me this has always seemed like the perfect application of union/variant types. The main difference would be that as a variant type, the receiving actor would have to process each 'missing' token explicitly.
>
> (I think) an SDF model should propagate 'missing' values and generate missing values, just like any other token. I tend to agree that if you want a particular definition of something like missing tokens, then you want to give a fair bit of thought to it. I'm guessing that the semantics of R are to treat it as another data value with anything OP missing = missing.

#3 - 12/13/2005 06:43 PM - Christopher Brooks

Ok, I hacked up the following:

- Token.java now has nil() and isNil() methods
  I went with nil because null is a Java keyword.

The term "missing" is rather appealing since the Token.toString() method usually prints out "present". So, it could be modified to print out "missing". However, I feel that someone is likely to have a parameter named "missing" somewhere. nil seems safer.

```
This is not cast in stone, comments are welcome.
```

```
These methods have tests.
```

- data/expr/Constants.java now defines a "nil" constant which is a Token that has the nil() method:
  ptolemy.data.Token nil = new ptolemy.data.Token();
  nil.nil();
  _table.put("nil", nil);
  Thus, one can now create expressions that have nil in them

  The "nil" constant has a test in data/expr/test/PtParser.tcl

- actor/lib/RemoveNilTokens.java:
  A new actor that reads its input and discards any nil tokens in the fire() method. It might be better to do this in prefire()
  This actor is available in the "More Libraries" -> "Esoteric" section.

  Note that this actor should not be used in SDF.

  ```
  No tests yet.
  ```

- domains/pn/demo/RemoveNilTokens/RemoveNilTokens.xml
  A model that uses RemoveNilTokens.
  I think I'm terminating the PN process poorly. I get 7 outputs instead of 5. I could use some help here.
  Also, I have to explicitly set the type of output of the RemoveNilTokens actor.

The model looks like:

So, now we have a straw man in PN to try out.

Questions:
1) Would something like this meet the needs of the Kepler group?
2) Should RemoveNilTokens do something in prefire()?
I'm not sure if I can get access to the Token and call Token.isNil() in prefire().
3) Is "nil" an ok name?
4) How do I get PN to terminate properly after I see 5 non-nil tokens?

#4 - 02/07/2006 11:13 AM - Christopher Brooks

Dan wrote:

I would like to add some thoughts to this issue of null/nil tokens in Kepler/Ptolemy.

Consider first some comments from the R system about what it calls 'not available' or 'missing value' elements

from R documentation ------

"In some cases the components of a vector may not be completely known. When an element or value is \223not available\224 or a \223missing value\224 in the statistical sense, a place within a vector may be

reserved for it by assigning it the special value NA. In general any operation on an NA becomes an NA. The motivation for this rule is simply that if the specification of an operation is incomplete, the result cannot be known and hence is not available. The function is.na(x) gives a logical vector of the same size as x with value TRUE if and only if the corresponding element in x is NA.
> z <- c(1:3,NA); ind <- is.na(z)

Notice that the logical expression x NA is quite different from is.na(x) since NA is not really a value but a marker for a quantity that is not available. Thus x NA is a vector of the same length as x all of whose values are NA as the logical expression itself is incomplete and hence undecidable.

Note that there is a second kind of \223missing\224 values which are produced by numerical computation, the so-called Not a Number, NaN, values. Examples are

```
> 0/0
 > Inf - Inf
which both give NaN since the result In summary, is.na(xx) is TRUE
is.nan(xx) is only TRUE for NaNs."
```

Note that R typically works with vectors (arrays) of values. NAs are often kept as part of these vectors so that sizes do not change. The NAs can be explicitly removed from a vector when required with a statement like

```
> y <- x[!is.na(x)]
```

which creates a new vector y which will contain the non-missing values of x, in the same order.

So I suggest that in Kepler/Ptolemy we need to consider nil/null values in arrays as well as the stand-alone nil/null token. For example, if we are working with a data table that has some NA values in some columns we may be working with individual cell, table rows, or table columns. If Kepler automatically drops null/nil values, this may cause problems relating rows or columns. And columns are usually the same structural type, so we may well be working with a TokenArray to handle the whole column. All tokens in a token array need to be of the same type, so how do we handle nilTokens in an array of, say, DoubleTokens?

So it seems to me that we really do not want a NilToken; we really want tokens of any type that have a nil/null property [Christopher has already added this to the Token class, so this should apply to any of the classes derived from Token.] This is the approach that R has taken to handle NA value - simple set one double/float, integer, or character value to represent a NA value. {See my email of 12/22/2005). One can then create an actor to do things like strip NAs from an array. And existing actors that do not consider nils will continue to work without change.

So a column in a table with type 'double' that has one missing value would have one DoubleToken with a property of nil/null. But we would also like any mathematical operation applied to the token to give a result that is also nil/null. This would imply that the various operator methods of Tokens (e.g. _add or _multiply) should check the nil property and respond appropriately. And then there is the problem of an actor that just gets the value of a Token and does not check the nil property. This might be handled by setting the value to some predefined value that can be recognized (e.g. setting a float/double nil value to NaN or some very large number; NaN is very convenient since primitive mathematical operations on NaN return NaN.)

Note that some of the Java GIS actors that are already in Kepler must handle missing data values in spatial rasters. In that case, I simply defined any double value over a very large threshold to represent a 'missing value'. This is a custom case of the nil/null problem and it works with the current version of Kepler because the rasters are passed by reference to filenames, rather than through tokens directly containing data.

So much for my 2 cents ;)

Dan

#5 - 02/07/2006 02:23 PM - Christopher Brooks

Jing will modify the EML source.
Dan will modify the R actor.
I'll will look at the XYPlot actor and possibly other actors such as
SequencePlotter.
I'll create Array2Array and remove nils, changing the length.
SequenceToArray, preserves nils in the array.
ArrayToSequence, preserves nils in the output.

What happens when we hook up an actor that produces nils and we call getValue(),
what happens. Right now do we throw an exception?

Also, we need to handle IntToken.intValue() for nil tokens.
Also and perhaps the constructors that take a string need to properly handle
"nil"

```
% set d [java::new ptolemy.data.DoubleToken "nil"]
ptolemy.kernel.util.IllegalActionException: For input string: "nil"
% set d [java::new ptolemy.data.DoubleToken "1.0"]
java0x6
%
```

Perhaps we need some sort of constructor that constructs a nil token.
It makes no sense to set a Double to a value and then set it nil.
Should Double(boolean isNil) be able to create a nil token?

#6 - 02/09/2006 10:31 AM - Christopher Brooks

[I'm going to try to keep most of the email in the null value bug so
that I don't lose track while I'm on vacation 2/15-2/28]

Hi Dan,
Yep, Token.toString() returns "nil"

```
--start--
cxh@carson 46% $PTII/bin/ptjacl
% set token [java::new ptolemy.data.Token]
java0x1
% $token nil
% $token toString
nil
%
--end--
```

One big problem is that Tokens are supposed to be immutable, so we need
a constructor that will set the Token to nil. We need constructors for
all the Tokens that will set them to nil. I'm a little unclear on the type
of a nil Token. In many ways it is of type "General", where it can be
used anywhere.

I need to set up a way to create nil tokens of a particular type using
the expression actor. I'm thinking that a function call nil() that takes
a type would be the way to go. Thus nil("double") would create a nil
DoubleToken. I've not thought about this much though.

Once we have that, we can create ArraysTokens that have nil elements.

The notion of logical arrays sounds like it could be implemented without
much effort.

_Christopher

Dan writes:

> Hi Christopher,
> I don't know if you guys discussed this or not, but in looking at
> the RExpression actor I noted that I use the the Token 'toString()'
> method. Thus, if 'at' is an arrayToken containing the 3 integers 1,2,3,
> then at.toString() is used to create the string "{1,2,3}".
> So the question is, if one of the tokens in an array is 'nil', then
> what is the result of the 'toString()' method?
> The R system just returns the string 'NA'. If Ptolemy returned 'NA'
> or maybe 'nil' that would make it easy to convert the RExpression actor.

In any case, it seems to me that it would be useful to have Kepler/PT return some string value for 'toString' when a token has the nil property rather than throw an exception.

On a slightly different topic, we talked about an actor that would take an array with nils and produce a smaller array without nils. A somewhat more general approach used in R that might be very useful is the idea of of a 'logical array'. We can certainly have an array of boolean values, but R uses a logical array to select parts of another array of the same dimensions. In other words, one could use an actor that has two array inputs, one a logical array and the other any other kind of array with the same dimension. The resulting output is a smaller array where only those values corresponding to true values in the logical array are retained. Thus, one would remove nils by creating a logical array from some arrays where nil values are 'false' and then combining that logical array with the original.

Dan

#7 - 02/28/2006 11:33 AM - Christopher Brooks

Just a status report on this:
I think I have something that works.
I've not done all the tokens. We are to have a review today.
After the review I'll consider updating the other tokens.
The SequencePlotter and NonStrictTest actors both handles nil tokens.

#8 - 03/01/2006 07:09 AM - Christopher Brooks

We did a review of the nil token work yesterday.
Two major issues:
1) Edward suggested:
Why not make a static public member for nil token instead of nil token constructor (the copy constructor)? Advantage: zero overhead in memory because don't need boolean to identify whether is boolean.
2) We probably don't want to support nil tokens in DoubleMatrixTokens.
Edward suggested:
Should we just not support nil tokens in MatrixToken? Because of efficiency reasons. MatrixToken is supposed to support Java base types efficiently. If you want an array of tokens where some of them are nil, use ArrayToken. We don't want to mess with matrix manipulation algorithms to deal with nil.

It would be nice to address these issues. I'll have a little time today, but more time tomorrow. Also, I won't be able to make the telcon at 11:30, I have prior commitments.

Below is the contents of the review.

Identified defects

- Token
  1. isNil() comment: Should link to protected method be in public method comment? [eal]
  2. _nil(): Should be renamed. Seems like make nil. [eal]
  3. toString() comment: There is no nil() method. [eal]
  4. _newNilToken(Type): Shifting a type error to runtime? See issue about static public member. [eal] * DoubleToken
  1. DoubleToken(String): What is the difference between a DoubleToken("nil") and a StringToken("nil")? What if we actually want a token that has the string "nil"? Maybe should not parse the string "nil" in any token class? Push this to expression language parser? [eal, acataldo] * ArrayToken
  1. ArrayToken(Token[]): In code, when looking for first non-nil token to get the type, is this already in the expression parser? [eal]
  2. ArrayToken(Token[]): This should be able to take a null argument. In this case, it should then create a nil token of type ArrayToken. (Under current design). Right now this doesn't support "being a nil", rather than "containing a nil". [eal]
  3. ArrayToken(Token): This code will give a null pointer exception (assigns token of length 0, then tries to address the first (zero'th) element). Delete 3 lines (if statement). [eal] * MatrixToken
  1. method name: description of error [login] * DoubleMatrixToken
  1. method name: description of error [login]

Related issues

```
1. Is a nil token equal to another nil token? What does Java do about NaN?
[actaldo/cxh]
   2. Why not make a static public member for nil token instead of nil token
constructor (the copy constructor)? Advantage: zero overhead in memory because
don't need boolean to identify whether is boolean. [eal]
   3. Do we have sparse array storage in Pt? [sprinkle]
   4. Do we need a nil type to be at the bottom of the type lattice (it would be
convertible to any type.)? Maybe it already does? Need expression language to
support it. [eal]
   5. More static public member suggestions:

DoubleToken.convert(Token t) {
        if (t.isNil()) {
          return NIL;  //NIL is a const.
        }
      }

[eal]
   6. Should we just not support nil tokens in MatrixToken? Because of
efficiency reasons. MatrixToken is supposed to support Java base types
efficiently. If you want an array of tokens where some of them are nil, use
ArrayToken. We don't want to mess with matrix manipulation algorithms to deal
with nil. [eal]
```

#9 - 04/13/2006 12:03 PM - Christopher Brooks

I'm marking this bug as fixed, because I think the work here is done.
If there are problems, it can be reopened.

One thing that bothers me is that now we have a number of places where
we check to see if the token is nil when doing operations on tokens.
Will this slow things down? The way we check for nil is by checking
to see if the token in question is to the NIL token, so this should be
very fast. I tried to think of different ways around this, such
as creating a subclass of each token that was NIL (IntTokenNil).
However, ScalarToken._doAdd() would still need to check for Nil-ness,
so no gain there. I dunno . .

Also, currently equals() returns false called on a NIL Token.
In otherwords, IntToken.NIL.equals(IntToken.NIL) returns false.
I'm not sure if this is right. IntToken.NIL.isEqualTo(IntToken.NIL)
should return false, since the values of NIL tokens are not comparable.
But, should equals()?

Below is a summary of some of the design decisions that I sent in email
to kepler-dev.

Many thanks to Dan, Jing, Tristan, Edward and everyone else for helping with
the Nil Token work.

_Christopher

> [Tristan wrote]
>
> > You mentioned in a previous email that you used to have nil-ness as
> > a property of the token rather than a state like the current
> > implementation. What were the problems with this system?
>
> When we reviewed the code, Edward suggested a NIL instance of
> the token instead of using a method to set a flag.
> The advantage of using a NIL instance is that we don't add a boolean
> to each Token. Another advantage is that to check if a token is
> nil, we check to see if it is equal to NIL instead of checking
> a boolean, which is probably faster.
>
> One disadvantage is that the XXXToken(String) constructor can no
> longer be used to create a nil token because the constructor can't
> return a different instance. We'd like something like:
>
> public FooToken(String init) {
> if (init.equals("nil") {
> return Token.NIL;

}
}

but that won't work because the constructor can't return a different instance. Thus, the String constructors don't support constructing nil tokens.

Also, each Token needs to have its own NIL because the convert() method returns a more specific type than the type of the argument, so we have to have IntToken.NIL instead of Token.NIL:

public static IntToken convert(Token token) throws IllegalActionException\

{

...
if (token null || token.isNil()) {
return IntToken.NIL;
}

Also, I had to introduce a type called niltype, (to differentiate it from nil). Token.NIL is of the niltype, which is losslessly convertible to UnsignedByteToken, BooleanToken, IntToken, LongToken and DoubleToken. The reason this is safe is because any operation that is performed on a NIL token usually results in a NIL. Some operations, such as average, could choose to ignore the NIL token. In general, if you have a NIL, it can be seen as being of any type and thus it can be losslessly converted in to other types that have NILs.

One other restriction is that the MatrixTokens do not support nil elements. The reason is that MatrixTokens are to be used for places where speed is needed. The underlying element type in a MatrixToken does not have support for nil types. For example IntMatrixToken has an int[][] matrix behind it. To support nil tokens here, we would need to add sparse matrix that would note which elements are nil. This could be done, but it points to creating classes that extend XXXMatrixToken rather than modifying the base XXXMatrixToken and getting a speed hit.

Another issue is that NIL StringTokens have the toString() value "nil", yet constructing a StringToken with new StringToken("nil") results in a StringToken with three letters, n, i and l, which is not a NIL token. Also, for historical reasons, new StringToken(null) returns a StringToken of length 0, aka the empty string. Currently, new StringToken(null) could not return StringToken.NIL anyway, but it seems odd.

#10 - 03/27/2013 02:19 PM - Redmine Admin

Original Bugzilla ID was 2240