Saxon - Patch #1420

Add saxon:closure extension function

2003-08-13 04:25 - Anonymous

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:	Internals	**Estimated time:**	0:00 hour
Sprint/Milestone:		**Spent time:**	0:00 hour
Legacy ID:	sf-787809	**Fixed in Maintenance Release:**	
Applies to branch:		**Platforms:**	
Fix Committed on Branch:			

Description

SourceForge user: gschadow

Hi Michael, it worked! I could indeed add a saxon:closure

() extension function that will force it's argument

Expression to be lazily evaluated. It simply creates a

Closure from the first argument in the current

XPathContext.

In order to make it happen I had to make numerous little

additions, most of them to get the name saxon:closure

known. I used the saxon:expression extension function

as an example and did everything the way you did it for

saxon:expression.

The only real **change** that you may want to review is

that I made SequenceValue implement Item (simply by

adding one trivial little missing function implementation.

Also required was a little change in Value.asValue().

Attached is a ZIP file that contains the complete diff

that should take care of every change needed. It's

against 7.6 though but the diff has not much context so

I don't think much would be rejected.

Also in the ZIP file is a little test case (to invoke as a

transform over any XML input). The test-case requires a

Java extension thingie with a little side-effect, because

that's my whole point.

The point that the test illustrates is how side-effects

can be carried out with lazy evaluation ONLY if the value

is actually referenced. The real example is a database

insert that should only be done for parent-records that

have dependent children. If you run the example without

the closure() function it's output is:

next 1 for insert a

next 2 for insert b parent 1

next 3 for insert c parent 1

next 4 for insert d

next 5 for insert e

next 6 for insert f parent 5

<?xml version="1.0" encoding="ASCII"?>

```
  <child id="2" parentId="1" name="b"/>
```

```
  <child id="3" parentId="1" name="c"/>
```

```
  <child id="6" parentId="5" name="f"/>
```

which is bad because it inserted "next 4 for indsert d" for

the parent named "d" that had no children. Once you

apply the patch and use the closure() function the

output is correctly:

next 1 for insert a

next 2 for insert b parent 1

next 3 for insert c parent 1

next 4 for insert e

next 5 for insert f parent 4

<?xml version="1.0" encoding="ASCII"?>

```
  <child id="2" parentId="1" name="b"/>
```

```
  <child id="3" parentId="1" name="c"/>
```

```
  <child id="5" parentId="4" name="f"/>
```

History

#1 - 2003-08-13 21:26 - Anonymous

SourceForge user: gschadow

Logged In: YES

user_id=575520

Oops, the diff file was somehow screwed up, I couldn't
quite apply it properly. Here is a resubmission of just that
diff file.

#2 - 2003-08-14 14:19 - Anonymous

SourceForge user: gschadow

Logged In: YES

user_id=575520

hmm, I'm wondering if calling this function "closure" is actually
appropriate. It's intent is more like "delay evaluation" or "force
lazy evaluation". So, lazy-evaluate() or delay-evaluation()
might be better names. You call the shots on this.

#3 - 2003-08-28 08:08 - Anonymous

SourceForge user: mhkay

Logged In: YES

user_id=251681

I've thought about this on and off. I can't say I'm
comfortable with it. It's going to be very hard to explain
to people what this function is designed to achieve, and I
suspect that it's going to be difficult to keep it tested
and maintained as the internal structure of the software
evolves. I can see a stronger case for the reverse function,
one which forces eager evaluation of an expression (this can
be written as a standard extension function with no special
support).

I can already see some problems, for example lazy evaluation
of an expression isn't safe if it depends on certain aspects
of the context, such as position() and last(). These
conditions could change in the future.

I would be much happier with a design that takes a proper
look at how the language can be extended cleanly to manage
side-effects, on the lines of monads in Haskell (or
otherwise!). I think many of the use-cases could be handled
more cleanly by using higher-order functions and I want to
explore extensions in that direction - though my priority at
the moment is to implement what's in the 2.0 spec, rather
than extensions to it.

I can't see why you want SequenceValue to implement Item. An
Item is defined in the language spec as a Node or an

AtomicValue, whereas a SequenceValue can hold multiple Items.

The name saxon:closure, incidentally, was used in the past

for a function that computes a transitive closure, and since

people sometimes upgrade from very old releases, I wouldn't

want to re-use the name. saxon:lazy-evaluate would be better.

Files

saxon-closure.zip	9.56 KB	2003-08-13	Anonymous
closure.diff	9.56 KB	2003-08-13	Anonymous