Saxon - Bug #4439

XQJ reads but ignores output: serialization parameters

2020-01-23 21:58 - Andreas Sewe

Status:	Closed	**Start date:**	2020-01-23
Priority:	Low	**Due date:**	
Assignee:		**% Done:**	0%
Category:	XQuery Java API	**Estimated time:**	0:00 hour
Sprint/Milestone:		**Spent time:**	0:00 hour
Legacy ID:		**Fixed in Maintenance Release:**	
Applies to branch:	9.9	**Platforms:**	
Fix Committed on Branch:			

Description

This bug occurs with Saxon 9.9.1 PE using the XQJ API.

I have attached a simple Maven project with which you can reproduce. Just run mvn clean verify exec:java

If I explicitly set serialization parameters as a java.util.Properties object passed to XQResultSequence.writeSequence, parameters like method are respected:

```
With output parameters passed as Java Properties
<!DOCTYPE html><html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
      <title>With output parameters passed as Java Properties</title>
   </head>
   <body></body>
</html>
```

If I use inline declare option output:method and output:html-version, however, they are read but then **ignored** during writeSequence:

```
With inline output: parameters
<html>
   <head>
      <title>With inline output: parameters</title>
   </head>
   <body/>
</html>
```

Likewise, when using an output:parameter-document., the document is read (and you get a static error if it cannot be located, relative to ${user.dir}), but its contents are **ignored** during writeSequence:

```
With output:parameter-document
<html>
   <head>
      <title>With output:parameter-document</title>
   </head>
   <body/>
</html>
```

I realize that *JSR 225* doesn't say anything about output: serialization parameters, as AFAIK *XSLT and XQuery Serialization* wasn't a thing back then, but I would expect them to be respected. After all, they do already cause static errors when malformed (e.g., unknown output:method or unresolvable output:parameter-document).

History

#1 - 2020-01-23 22:05 - Andreas Sewe

- File saxon-bug.zip added

Sorry, the attached saxon-bug.zip contained a problem:

For experimentation I tried a classpath: URI for output:parameter-document. Unfortunately, this doesn't work; I get an unknown protocol: classpath I/O error, although a StandardURIResolver is used which, according to its Javadoc, should support classpath:. But that is a different bug, I fear.

At any rate, I have attached a fixed sample project.

#2 - 2020-01-23 23:25 - Michael Kay

XQJ was designed for XQuery 1.0 and its architecture very much assumes that the query result is being returned to the application as a sequence of items, not as a serialized document. In addition, XQJ is subject to very strict licensing rules which demand 100% conformance to the test suite and no implementor extensions.

I'm really not sure how you would expect a serialized query result to be delivered by XQJ. The only methods for returning query results return the result as an XQSequence or similar, which is clearly a sequence of items rather than a serialized lexical XML document.

I would encourage you, if you want to take full advantage of facilities introduced later than XQuery 1.0, and full advantage of Saxon's capabilities, to move to the s9api interface instead. XQJ is a dead end; it isn't being further developed, and it bans implementors from developing their own extensions.

The XQuery 3.1 specification states "A processor that is not performing serialization may report errors if any serialization parameters are incorrect, or may ignore such parameters." The Saxon XQuery compiler validates the parameters because it is not known at query compile time whether serialization will be requested at query execution time.

#3 - 2020-01-25 23:02 - Michael Kay

- Status changed from New to Closed

In jour Java application code you are doing

```
            XQPreparedExpression expression = connection.prepareExpression(query);
            XQResultSequence results = expression.executeQuery();
            results.writeSequence(System.out, outputParameters);
```

Since the second step is executing the query to obtain a result sequence, and the third step is serializing the result sequence to produce serialized output, the only way the serialized output could depend on anything in the query would be if this were somehow held as hidden information within the XQResultSequence. I would personally consider this an acceptable extension to the semantics of the XQJ API, but the XQJ rules prohibiting such extensions are very strict and I am loathe to do this.

Another approach would be to interrogate the XQPreparedExpression object to determine the output parameters defined in the query. Because of the rules preventing XQJ extensions, Saxon doesn't expose any public methods to do this, but you could do it by reflection: the XQPreparedExpression is a SaxonXQPreparedExpression, which contains an XQueryExpression, which contains an Executable, which has a method getOutputProperties().

I'm going to close this issue with no action; the product is working as designed.

#4 - 2020-01-26 17:59 - Andreas Sewe

Thank you for the detailed explanation behind the reasoning to close this issue. Much appreciated.

I would appreciate it even more, however, if this ticket could be reopened. Here's why:

Leaving the issue aside as to why I have to use XQJ at the moment, I think these statements are false (except for the "designed for XQuery 1.0" bit):

> XQJ was designed for XQuery 1.0 and its architecture very much assumes that the query result is being returned to the application as a sequence of items, not as a serialized document.

> I'm really not sure how you would expect a serialized query result to be delivered by XQJ. The only methods for returning query results return the result as an XQSequence or similar, which is clearly a sequence of items rather than a serialized lexical XML document.

While it is true that XQSequence provides access to individual items via its XQSequence.getItem() method, it also offers methods like XQSequence.writeSequence(InputStream, Properties) which are clearly meant to output "a serialized lexical XML document" to a byte stream. In fact, the Javadoc for writeSequence even references the *XQuery 1.0 serialization* specification:

> Serializes the sequence starting from the current position to an OutputStream according to the XSLT 2.0 and XQuery 1.0 serialization. Serialization parameters, which influence how serialization is performed, can be specified. Refer to the XSLT 2.0 and XQuery 1.0 serialization and Section 12 Serialization, XQuery API for Java (XQJ) 1.0 for more information.

So these methods **are** concerned with serialization. Alas, they refer to the wrong version of the spec.

Now I think the argument can be made that, when the query being executed was a XQuery 3.x query rather than an XQuery 1.0 query, we are beyond the scope of JSR 225 anyway, so it cannot mandate if and how output: options should be interpreted. My assumption as a user would hence be that for queries written in a successor of XQuery 1.0, writeSequence would work in accordance with the matching successor version of *XSLT 2.0 and*

XQuery 1.0 serialization, too.

Implementation-wise, I think the following would work:

- In SaxonXQForwardSequence, depending on the version of the query being executed, take the output-properties of this.expression.getXQueryExpression().getExecutable().getOutputProperties().
- Merge these with any non-null Properties passed to writeSequence(…). (My assumption would be that Properties passed explicitly override output properties specified in the query.)
- The static SaxonXQSequence.setDefaultProperties(Properties) might be extended to take both “external” Properties and “internal” Properties (from the Executable).

What do you think?

#5 - 2020-01-26 19:40 - Andreas Sewe

- File basex-demo.zip added

Did a bit more experimentation: Like Saxon, BaseX 9.3.1 does **not** respect output: options that are part of the query. (Try with mvn clean verify exec:java --lax-checksums.)

So there is an argument to be made that simply serializing according to *XSLT and XQuery Serialization 3.1* on an XQSequence.writeSequence(…) is not as “natural” as I was arguing it was. At least BaseX doesn’t behave that way either.

Files

saxon-bug.zip	2.79 KB	2020-01-23	Andreas Sewe
saxon-bug.zip	2.79 KB	2020-01-23	Andreas Sewe
basex-demo.zip	2.73 KB	2020-01-26	Andreas Sewe