SaxonJS 3.0
Major new functionality!

1 Aug 2024

Norm Tovey-Walsh

Debbie Lockett

Balisage, 2024

SAXONICA.COM
XSLT AND XQUERY PROCESSING

Table of Contents

Who are we?

- Saxonica is a small UK software company. We build open source and commercial XPath, XSLT, and XQuery processors for Java, C#, C, C++, Python, and PHP on Linux, macOS, and Windows.
- Norm Tovey-Walsh is the CEO and a lead developer.
- Debbie Lockett is a senior developer and has been the primary implementor on SaxonJS for many years.

What is SaxonJS?

- SaxonJS is an XSLT 3.0 processor that runs in the browser and on Node.js
- More technically: it’s a JavaScript interpreter for XSLT stylesheets compiled into the Saxonica “Stylesheet Export Format”

Why use SaxonJS?

- Easy of use
  - Many markup-related tasks are easier in XSLT than JavaScript
  - Especially for this audience!
- Cross-platform development

Why SaxonJS 3.0?

- To implement (some of) the ideas in Michael Kay’s Asynchronous XSLT paper from Balisage 2020.
- Improved APIs for cross-language processing.
- JavaScript has evolved.

SaxonJS example

- You’re looking at one: SlidesJS!
- I’ve also made a small one for this talk

Demo HTML

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Hello, Balisage</title>
  <script src="js/saxon-js/SaxonJS3.rt.js"></script>
  <script src="js/start.js"></script>
  ...
</head>
<body>
<header id="hello">
  <h1>Loading…</h1>
  <p>This page uses SaxonJS for progressive enhancement.</p>
</header>
<main></main>
</body>
</html>
```

Demo JavaScript

There are four lines of JavaScript in the demo:

```
window.onload = function() {
  SaxonJS.transform({
    "stylesheetLocation": "xslt/stylesheet.sef.json",
    "initialTemplate": "Q{}demo" }, "async"); }
```

The demo named template

```
<xsl:template name="demo">
  <xsl:result-document href="#hello" method="ixsl:replace-content">
    <h1><xsl:copy select="ixsl:page()/html/head/title/text()"/></h1>
    <hr/>
  </xsl:result-document>
  <xsl:apply-templates select="ixsl:page()/html/body/main"/>
</xsl:template>
```

1. Uses `ixsl:page()` to run XPath expressions over the HTML page
2. Uses `xsl:result-document` to update the page
3. Runs `xsl:apply-templates` on `main`

The main template

```
<xsl:template match="main">
  <xsl:result-document href="?." method="ixsl:replace-content">
    <span id="message"/><br/>
    <button x-count="0">
      <img width="400px" src="/img/button.png"/>
    </button>
  </xsl:result-document>
  <ixsl:promise select="ixsl:sleep(4000)"
                on-completion="f:aww-go-on#0"/>
</xsl:template>
```

1. Uses `xsl:result-document` to update *itself*
2. Adds a button to the page
3. A sneak peek at promises, coming up later

Event handlers in XSLT

Here’s the handler for the first button press:

```
<xsl:template mode="ixsl:onclick" match="button[@x-count='0']">
  <ixsl:set-attribute name="x-count" select="'1'"/>

  <xsl:result-document href="#message" method="ixsl:replace-content">
    <xsl:text>You pressed me!</xsl:text>
  </xsl:result-document>

  <ixsl:promise
      select="ixsl:http-request(map{'method': 'GET',
                                    'href': '/xml/strapline.xml'})
              => ixsl:then(f:update-strapline#1)
              => ixsl:catch(f:error#1)"/>
</xsl:template>
```

Updating the strapline

Here’s the `update-strapline` function:

```
<xsl:function name="f:update-strapline" ixsl:updating="true">
  <xsl:param name="response" as="map(*)"/>

  <xsl:for-each select="ixsl:page()//h1">
    <xsl:result-document href="?." method="ixsl:insert-after">
      <xsl:sequence select="$response?body"/>
    </xsl:result-document>
  </xsl:for-each>
</xsl:function>
```

Note `ixsl:insert-after` to add the strapline between the `h1` and the `hr`.

What’s interesting?

- The execution model
    - JavaScript is (cooperatively!) single threaded
- JavaScript and the XPath Data Model
    - A partial match at best

Old features

- No, I’m not going to show those, there isn’t time.

New features

Promises

A promise is a construct used for synchronizing program execution. They describe an object that acts as a proxy for a result that is initially unknown, usually because the computation of its value is not yet complete.
— Wikipedia (paraphrased)

- `ixsl:promise` is an alternative to `ixsl:schedule-action`
  - a richer mechanism for asynchronous processing
  - more closely aligned with JavaScript promises
- Asynchronous versions of XPath fetching functions: `ixsl:json-doc()`, `ixsl:unparsed-text()`, etc.
- Full set of promise-related functions: `ixsl:resolve()`, `ixsl:then()`, `ixsl:catch()`, `ixsl:finally()`, `ixsl:all()`, `ixsl:all-settled()`, `ixsl:any()`, `ixsl:race()`
- For more detail about the design, see Debbie Lockett’s excellent *Declarative Amsterdam* talk from last year: *Asynchrony with Promises in SaxonJS.*

Crossing the divide

- New APIs for calling JavaScript functions from XSLT and vice-versa.
- This is where data model alignment comes in
  - JavaScript has arrays, but so does XPath
  - XPath has sequences, which JavaScript doesn’t
    - All XPath singleton values are sequences of length 1
- Internally, SaxonJS uses nested arrays in…interesting ways.
  - Lazy evaluation uses iterators
  - It also uses objects for the XPath types: `XdmAtomicValue`, `XdmString`, `XdmInteger`, …

Can it just be simple, please?

- Convert function arguments and return types; `XdmInteger(5)` ⟺ `5`
    - Maps and arrays also converted
    - Nodes, function items, etc. don’t get converted
- Sequences are always passed as JavaScript iterators

Calling JavaScript functions (from XSLT)

- It just works. This returns 26:

```
<xsl:sequence select="my:js-implementation-of-square(5) + 1"/>
```

- Except…the *compiler* has to know the function signatures
    - There’s a new API for that and a new command-line option

Aside: Bubbling events

- An event handler responds to some event that occurs in the browser.
- Event handlers can be assigned on any element.
- Many events “bubble”

```
<body onclick="...">
  <main>
    <section>
      <p>
        <button>click me</button>
  …
```

Aside: Non-bubbling events

- Some events don’t bubble.

```
<body onfocus="...">
  <main>
    <section>
      <p>
        <input id="name"></input>
  …
```

- You’ll never see the focus event.

Calling XSLT functions (from JavaScript)

- The same conversion rules apply

- Here’s a useful trick:

```
SaxonJS.transform({stylesheetLocation: "main.sef.json"}, "async")
    .then(result => {
        fn = SaxonJS.xsltFunctionMapper
               .lookup("Q{http://example.com/xslt/functions}has-focus")
        document.querySelector("#f1").addEventListener("focus", fn)
    });
```

- That’s a mouthful, and yes, it’s on the JavaScript side of the divide, but it registers an *XSLT* function to respond to a *non-bubbling* browser event!

Miscellany

- SaxonJS 3.x the Fetch API
    - Replaces browser-native `XmlHttpRequest`
    - Allows SaxonJS on Node.js to perform `ixsl:http-request()`
- `ixsl:new()` to call JavaScript object constructors
- `ixsl:json-parse()`, etc
- New options to better control JavaScript/XDM conversions in existing APIs

SaxonJS/EE?

- Free runtime, license is in the compiler
- EE compiler includes support for
    - More builtin functions (EXPath, Saxon extensions, etc.)
        - Some are Node.js only, some are browser only
    - `SaxonJS.compile` (maybe?)

Can I have it now!?

- No.
- Saxonica does **a lot** with five developers.
- There’s been some amount of feature creep since we started.
- FYI:
    - SaxonJS 3.x will run SaxonJS 2.x SEF files.
    - SaxonJS3 will compile JS3 features.
    - SaxonJ 12.5+ required to compile JS3 features.

When can I have it?

- ¯_(ツ)_/¯
- Real Soon Now™

Thank you

See also:

- https://saxonica.com/