IMPLEMENTING XFORMS USING INTERACTIVE XSLT 3.0

Declarative Amsterdam 2019

O'Neil Delpratt       Debbie Lockett
oneil@saxonica.com    debbie@saxonica.com
SAXON-FORMS

- Partial implementation of XForms
- Written in interactive XSLT 3.0
- Runs in Saxon-JS in the browser
SAXON-JS 2.0

- XSLT 3.0 run-time processor
- Written in JavaScript, runs in the browser
  + beta version to run in Node.js
- Executes compiled XSLT stylesheets (SEFs)
  generated by Saxon-EE
  or new alternative compiler written in XSLT
  (less optimised SEF, but open-source)
- Internal changes to improve performance
- More XSLT features e.g. higher-order functions,
  serialization
SAXON-JS KEY FEATURES

- XSLT 3.0 (e.g. XPath 3.1, xsl:evaluate)
- 'Interactive' XSLT extensions: event-handling template rules (for handling user input in XSLT); functions/instructions to access HTML page and other browser window objects, and edit the DOM
- Call JavaScript code from XSLT
- Dynamic generation of (X)HTML - modify page content using xsl:result-document
- HTTP client
GOOD FIT FOR AN IN-BROWSER XFORMS IMPLEMENTATION

- Rather than using existing implementations, a new implementation which runs in Saxon-JS allows for better integration within application
SAXON-FORMS IMPLEMENTATION DETAILS:

- Initialization:
  - XForm Controls: Transform the section with form controls into HTML forms elements
  - JavaScript global variables and functions to handle:
    - XForms instances
    - Actions (bind element)
    - Item properties
  - XForms function library: XSLT functions
• Processing:
  - Event handling, actions: interactive XSLT 3.0
  - Submission: XSLT 3.0 & JavaScript validate instance.
  - XForms functions
<html>
  <head>
    <script id="xforms-cache">
      var XFormsDoc;
      var initialInstanceDoc;
      var instanceDoc;
      var pendingUpdatesMap; /* XPath map*/
      var relevantMap; /* XPath map*/
      var actions;

      /*Getter/Setter Functions */

      var setInstance = function(doc) {
        instanceDoc = doc;
      }

      var getInstance = function() {
        return instanceDoc;
      }

      var addAction = function(name, value) {
        actions[name] = value;
      }
    </script>
  </head>
</html>
XSLT CODE TO ADD ACTION TO JSON OBJECT IN JAVASCRIPT SPACE

```xml
<xsl:variable name='action-map' select='map{
    "@ref": "Document/Shipments/Order/MaintenanceDays",
    "@event": "xforms-value-changed",
    "setvalue": [map{"@value": "if(xs:integer(.) > 0) then .
        "ref": '../..../Options/MaintenanceDate"},
        map{"value": "true",
            "ref": '../..../Options/Updated"}]
} />
<xsl:sequence select='js:addAction("d26aApDhDa", $action-map)'/>
```

Call JavaScript global function from interactive XSLT by using http://saxonica.com/ns/globalJS namespace
EVENT HANDLING

```xml
<xsl:template match="input[exists(@data-action)]"
    mode="ixsl:onchange">
    <xsl:variable name="refi" select="@data-ref"/>
    <xsl:variable name="refElement" select="@data-element"/>
    ...
    <xsl:variable name="xforms-value-change"
        select="js:getAction(string(@data-action))"/>
    <xsl:variable name="updatedInstanceXML"/>
    ...
</xsl:variable>
<xsl:sequence
    select="js:setInstance($updatedInstanceXML")/>
```
<xsl:for-each select="$xforms-value-change">
  <xsl:variable name="action-map" select="."/>

  <xsl:variable name="ref"
    select="map:get($action-map, '@ref')"/>

  <!-- if and while clause setup-->
  ...

  <xsl:variable name="instanceXML_Frag" as="node()">
    <xsl:evaluate xpath="$ref"
      context-item="$updatedInstanceXML"/>
  </xsl:variable>

  <!-- update form controls directly or add change to pendingUpdateMap to change instance later

THANK YOU FOR LISTENING

- Saxon-JS: https://www.saxonica.com/download/javas
- Saxon-Forms is available at https://github.com/Saxonica/Saxon-Forms
  Future goal: Full implementation?
  (With help from the community)
THANK YOU FOR LISTENING

QUESTIONS?