XQuery in the Browser

XQIB documentation

Version 0.7.0

Getting started

Download the ZIP file with the build results from the web site. Place the contents on your own web server.

Basic principles

XQIB allows you to execute XQuery code, just like you would do with JavaScript. You can create a script tag in the header, with type "text/xquery" and put your code inside. The engine is implemented in Javascript, you can include it by placing a script tag pointing to the library.

The XQuery engine behind XQIB is MXQuery (http://mxquery.org). Hence, most of the MXQuery documentation applies for XQIB as well.

Building a Hello World application

Create the following file (helloworld.html) and open it with your browser:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>XQIB: Sample page</title>
    <script type="text/javascript" src="mxqueryjs/mxqueryjs.nocache.js"></script>
    <script type="application/xquery">
         b:alert("Hello World")
    </script>
  </head>
  <body>
    <h1>Hello world page.</h1>
  </body>
</html>
```

A window should pop up as soon as the page is loaded.

The b:alert() function is one of the functions which the plugin makes available to you for programming the browser. It is sequential. The prefix b is predeclared.

Navigating the HTML document (DOM)

You can query the HTML content of the page containing your script. The function `b:dom()` provides access to the HTML document For example:

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>XQIB: Sample page</title>
    <script type="text/javascript" src="mxqueryjs/mxqueryjs.nocache.js"></script>
    <script type="application/xquery">

        b:alert(b:dom()//h1/data(.))

    </script>
  </head>
  <body>
    <h1>This is a title</h1>
  </body>
</html>
```

The default element/path namespace is the XHTML namespace, so that you can directly refer to HTML elements without any prefix.

Updating the DOM

You can update the DOM using updating expressions.

For example:

```
delete node b:dom()//INPUT[@id="myButton"]

insert node <p>Hello World</p> into b:dom()//BODY
```

This can, of course, only be done in updating or sequential functions.

Binding events

Interaction with code in the browser is typically done by reacting to events, such as clicking on a button. You can attach an XQuery function to an event (click on a button, …) using the b:addEventListener() function.

For example:

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>XQIB: Sample page</title>
    <script type="text/javascript"
src="mxqueryjs/mxqueryjs.nocache.js"></script>
    <script type="application/xquery">
      declare sequential function local:listener($loc, $evtObj) {
        b:alert("Hello, World")
      };

        b:addEventListener(b:dom()//input[@id = "myButton"], "onclick",
local:listener#2)

    </script>
  </head>
  <body>
    <h1>Onclick Event</h1>
    <input id="myButton" type="button" value="Click me"/>
</html>
```

The b:addEventListener functions takes three parameters: the element which triggers the event, the type of event, and the listener.

The listener function is provided as a function item (according to XQuery 3.0 Higher Order Functions) and can be defined inline or as a reference to an existing function, the latter as a QName#NumberOfParameters literal

An event handler function has two parameters, location and event object. The location contains the node on which the event was raised, the event object contains information e.g. about the mouse button which was pressed ($evtObj/button).

Navigating other Web sources

The doc function is not supported by XQIB. Instead, to query other sources from the Web, you should use the EXPath function module, documented at http://expath.org/modules/http-client/

It returns a sequence of two items:

1) The request metadata, such as header data
2) The payload. Depending on the data type, the payload is presented as XML document, String or Base64 Binary.

For example:

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>XQIB: Sample page</title>
    <script type="text/javascript"
src="mxqueryjs/mxqueryjs.nocache.js"></script>
    <script type="application/xquery">
      import module namespace http-client = "http://expath.org/ns/http-
client";

      declare updating function local:clickListener($loc, $evtLoc) {
        for $x in http-client:send-request(
          <http-client:request href="news.xml" method="get"/>
        )[2]//*:item
        return
        insert node <p>{
            $x/data(*:title)
        }</p> as last into b:dom()//body
      };

        b:addEventListener(b:dom()//input[@name="click"], "onclick",
local:clickListener#2)

    </script>
  </head>
  <body>
    <h1>D-INFK RSS</h1>
    <input type="button" name="click" value="Click!"/>
  </body>
</html>
```

There are two caveats:

1) Note that since XHTML is the default namespace, you need to use the joker (*) symbol to access elements which are in no namespace (it actually selects all possible namespaces). Unfortunately, there is no other known workaround about this in XQuery.
2) Since XQIB needs to follow the same-origin-policy as any other in-browser application, requests are only possible to the original server. There are some workarounds, such as CORS (http://en.wikipedia.org/wiki/Cross-Origin_Resource_Sharing). The easiest approach to integrate multiple sources is probably to set up a proxy on a server.

Asynchronous REST calls (AJAX)

XQIB also allows you to make asynchronous REST calls. For this purpose, the EXPath send-request function is split in two halves:

1) Send request with a listener
2) Call listener when request "comes back"

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
```

```
    <title>XQIB: Sample page</title>
    <script type="text/javascript" src="mxqueryjs/mxqueryjs.nocache.js"></script>
    <script type="application/xquery">
      import module namespace http-client = "http://expath.org/ns/http-client";

      declare updating function local:weatherResult($result){
              let $city := b:dom()//input[@name="city"]/data(@value)

            …….

            delete node b:dom()//div
      };

      declare updating function local:getWeather($loc, $evtObj) {
        http-client:async-request(<http-client:request href="forecast2.xml"
method="get"/>,
      local:weatherResult#1
          )
      };


         b:addEventListener(b:dom()//input[@name="button"], "onclick",
local:getWeather#2)

    </script>
  </head>
  <body>
    <h1>Weather forecast</h1>
    <input type="text" value="" style="font-size: 30pt; font-weight: bold"
name="city"/>
    <input type="button" value="Go!" style="font-size: 30pt; font-weight: bold"
name="button"/>
    <div><img src="JupiterZeus.gif"/></div>
  </body>
</html>
```

CSS Styles

You can access and edit CSS styles using b:getStyle (simple) and b:setStyle (sequential). For example:

```
b:getStyle(//BODY, "color")
b:setStyle(//BODY, "color", "blue")
```

Importing modules

You can import modules from the Web, just like otherwise in XQuery:

```
import module namespace
        m = "http://www.xqib.org/module"
        at "http://www.xqib.org/samples/module.xquery";
```

Samples

Samples demonstrating the various features of XQIB are available under http://www.xqib.org/js/