X3D JSON Loader Tutorial

with Object Model integration

John Carlson (yottzumm@gmail.com)

https://github.com/coderextreme/X3DJSONLD

https://github.com/coderextreme/pythonSAI

X3DJSONLD Purpose

- Original Purpose:
  - Render JSON to X3DOM, allow for near-interactive prototyping with X3D JSON. Also works with Cobweb through SAI.
- Approach:
  - Generic, convert certain types of JSON to DOM and XML
- Features:
  - DOM and XML rendering with X3DOM and Cobweb
  - Document Validation against JSON Schema 3.3
  - Loading of JSON documents into a web page with Jquery
  - Fuzz testing

X3DJSONLD Modules

- X3DJSONLD.js – convert JSON to DOM and XML, load URLs—supposedly independent of Jquery. Server and client side. Contains some client-side code which shouldn’t be used on server.
- loaderJQuery.js – jQuery and other integrations useful for a web page. Client-side
- convertJSON.js – convert JSON and send to serializer. Also contains JSON validator. Server-side.
- json2all.js – driver script for convertJSON.js
- PythonSerializer.js – serialize DOM to Python (non-chained methods).
- JavaSerializer.js – serialize DOM to Java (chained methods).
- JavaScriptSerializer.js – serialize DOM to JavaScript (chained methods).
- DOMSerializer.js – serialize DOM to XML (produces *.x3d.new)

X3DJSONLD Modules (cont.)

- PrototypeExpander.js – Expands JSON Prototypes to regular JSON.
- Script.js – sample way to convert VRMLscript to JavaScript (pre-alpha)
- Flattener.js – “flattens” JSON – gets rid of empty objects
- completeXMLProtoExpander.js – Expands XML Prototypes to regular XML (pre-alpha)
- exi.js – for Efficient JSON interchange
- PPP.js – driver script for Prototype Expander
- app.js – simple web server. Runs external prototype expander.
- convert.js – converts XML to JSON
- fuzz.js – fuzz testing for generating random X3D JSON documents.

X3DJSONLD Modules (cont.)

- fieldTypes.js – maps fields in nodes to types—derived from Obect Model
- mapToMethod.js – maps a parent child relationship to a method—derived from Object Model.
- json2all.js – converts JSON to * formats
- runppp.sh – runs the prototype expander and the XML prototype expander
- several.sh – run X3D XML files through conversion to JSON, serialization to Python, JavaScript, Java, and XML, compiles Java and runs Java and JavaScript
- local.sh – run several.sh on original .x3d files in src/main/data

X3DJSONLD Modules (cont.)

- all.sh – run X3D XML files in /c/x3d-code/www.web3d.org/x3d/content/examples (download from X3D sourceforge site) through several.sh
- don.sh – run X3D XML files through stylesheets, compiling, running (validating).
- donlocal.sh – run original .x3d files in src/main/data through don.sh

X3DJSONLD Main Routine loadX3DJS()

- Server and client versions.
- Both versions take json, url, xml log, and namespace, a way to load a JSON schema, a validate routine for JSON schema, and a callback, whose parameter is a returned DOM element or null for load failure.
- Client side does Inline processing of JSON since X3DOM and Cobweb don’t support JSON yet—some of the time. Some of the time, I use XML Inlines—this works better. I’m fudging it, I know. It may be that the URL is incorrect. More research is needed.

Client-side X3DJSONLD loaderJQuery.js (replace, append)

- replaceX3DJSON() takes a selector, json JS object, url, xml log, namespace and callback, passes them to loadX3DJS(), then appends the element passed in the callback to the emptied selector. Returns the child element in a callback, which you may use to load DOM into an X3D viewer. The xml returned may be used to load XML browsers. (see loadX3D in loaderJQuery.js). Tested with the user interface.
- appendX3DJSON2Selector() takes a selector, json JS object, url, xml log, namespace and callback, passes them to loadX3DJS(), then appends the element passed back in the callback to the selector. Returns an xml LOG (array of strings). [not tested – use at your own risk. See below]
- appendInline() takes an element,a JSON url, and a callback and validates against Schema and converts the URL to X3DOM and appends it to the element. Only works with X3D documents as inlines. (new feature)
- loadSubscene() takes a parent selector, a JSON url, and a callback and calls appendInline with the selected element from the document. The callback returns the selected element (not the subscene). Only works with X3D documents as Inlines, so they can be validated. Selected element MUST be loaded.

X3DJSONLD Client-side From the ground up

- Thanks to Andreas Plesch for starting a minimal example
- Cobweb DOM loader:
- <X3DCanvas cache="false">
- <X3D >
- <Scene>
- </Scene>
- </X3D>
- Your browser may not support all features required by Cobweb!
- </X3DCanvas>

X3DJSONLD client-side (continued—given a URL to load)

```
<script type="text/javascript" src="node/X3DJSONLD.js"></script>
  <script type="text/javascript" src="node/loaderJQuery.js"></script><!--for loadSchema and doValidate-->
<script type="text/javascript">

    var url = "data/ball.json";

    $.getJSON(url, importIntoCobweb);

    function importIntoCobweb(json) {
        var xml = [];
        loadX3DJS(json, url, xml, "", loadSchema, doValidate, function(jsDOM) {
            loadCobwebDOM(jsDOM);
        });
    }
```

X3DJSONLD client-side (continued—loading a DOM element into Cobweb, browser #0)

```
function loadCobwebDOM(element) {
        X3D(function(el) {
            var browser = X3D.getBrowser(el[0]);
            var importedScene = browser.importDocument(element);
            browser.replaceWorld(importedScene);
        });
    }
```

X3DJSONLD client-side, Prototype Expander on JS object.

- <script type="text/javascript" src="node/PrototypeExpander.js"></script>
- <script type="text/javascript" src="node/Flattener.js"></script> <!– gets rid of empty objects →
- <script type="text/javascript" src="node/test_json.js"></script> <!– for test_json variable →

-
- var url = "data/ExtrusionHeart.json"; // providing a fake path
- var xml = [], NS;
- test_json = prototypeExpander(url, test_json, "");
- test_json = flattener(test_json);
- loadX3DJS(test_json, url, xml, "", loadSchema, doValidate, function(json_element) {
- X3D(function(el){
- loadCobwebDOM(json_element, 0); // 0 is the 0th X3DCanvas element on the page, uses loaderJQuery.js version
- });
- });

Client-Side X3DOM and Cobweb together

```
<div id="x3domjson"></div>
function loadX3DJSON(selector, url) {
        $.getJSON(url, function(json) {
            replaceX3DJSON(selector, json, url, [], "", function(element) {
                if (element === null) {
                    alert("Couldn't load", url);
                } else {
                    loadCobwebDOM(element, 0);
                }
            });
        })
        .fail(function(jqXHR, textStatus, errorThrown) { alert('getJSON request failed! ' + textStatus + ' ' + errorThrown); });
    }
loadX3DJSON('#x3domjson','data/flipp.json');
```

Adding to X3DOM tree (not Cobweb)

- function ConvertToX3DOM(js_object_to_add (parent_key object value), parent_key, parent_element, path, containerField/*optional*/)
- DOES NOT DO VALIDATION OF JSON! We need a way to validate a partial JSON document!

Introducing loadSubscene() – new feature.

```
function loadX3DJSON(selector, url) {
        $.getJSON(url, function(json) {
            replaceX3DJSON(selector, json, url, [], "", function(element) {
                if (element === null) {
                    alert("Couldn't load", url);
                } else {
                    loadSubscene('Scene','data/abox.json', function(selected){
                        loadCobwebDOM(element, 0);
                    });
                }
            });
        })
        .fail(function(jqXHR, textStatus, errorThrown) { alert('getJSON request failed! ' + textStatus + ' ' + errorThrown); });
    }
    loadX3DJSON('#x3domjson','data/flipp.json');
```

loadSubscene() – continued.

- New feature, use with care.
- Requires parent DOM be loaded, use in a callback, or promise, or ensure selected element is loaded.
- Returns selected element, must traverse selected element to get children.
- Must be a complete X3D JSON file, including X3D object and Scene objects. Those objects, and the head object are removed.

Alternate to loadSubscene(). Direct JSON to Cobweb DOM.

- getJSON from: https://developers.google.com/web/fundamentals/getting-started/primers/promises
- Requires Cobweb/X3DJSONLD. Available soon after testing is complete I hope. I may have to create an alternate distribution under GPLv3.
- getJSON(suburl).then(function(subjson) {
- for (var c in subjson["X3D"]["Scene"]["-children"]) {
- json["X3D"]["Scene"]["-children"].push(subjson["X3D"]["Scene"]["-children"][c])
- }
- loadCobwebJS(json, 0); // reloads cobweb element 0 on page.
- }).catch(function(err) {
- alert("Subscene Load failed");
- console.log(err);
- });

Server X3DJSONLD

- Purpose:
  - Convert JSON to XML, Python, Java and Nashorn JavaScript
  - Do prototype expansion, command line and extern
  - Verify JSON input with Java output.
- Features:
  - ConvertJSON.js to load the JSON into DOM and call a serializer
  - 4 serializers, called by json2all.js and runjson.sh, based on Object Model inputs.
  - test runner for .x3d files (several.sh). Use local.sh for an example (processes files in src/main/data/*.x3d).
- Results:
  - Very few files converted back to exact original (meta nodes, containerFields added). Currently researching and exchanging bug reports. More work needed.

Server-side code (node.js)

App.js:

- var X3DJSONLD = require('./src/main/node/X3DJSONLD.js');
- var loadURLs = X3DJSONLD.loadURLs;
- var PE = require('./src/main/node/PrototypeExpander')
- PE.setLoadURLs(loadURLs);
- var externPrototypeExpander = PE.externPrototypeExpander;
- var FL = require('./src/main/node/Flattener')
- var flattener = FL.flattener;
- var runAndSend = require('./src/main/node/runAndSend');

Server-side example (continued)

```
function convertX3dToJson(res, infile, outfile, next) {
    console.error("Calling converter on "+infile);
    runAndSend(['---overwrite', '---./', infile], function(json) {
        console.error("Calling extern proto expander");
        json = externPrototypeExpander(outfile, json); <!– handle ExternProtoDeclare →
        json = flattener(json);
        // console.error("Json", json);
        send(res, json, "text/json", next);
    });
}
```

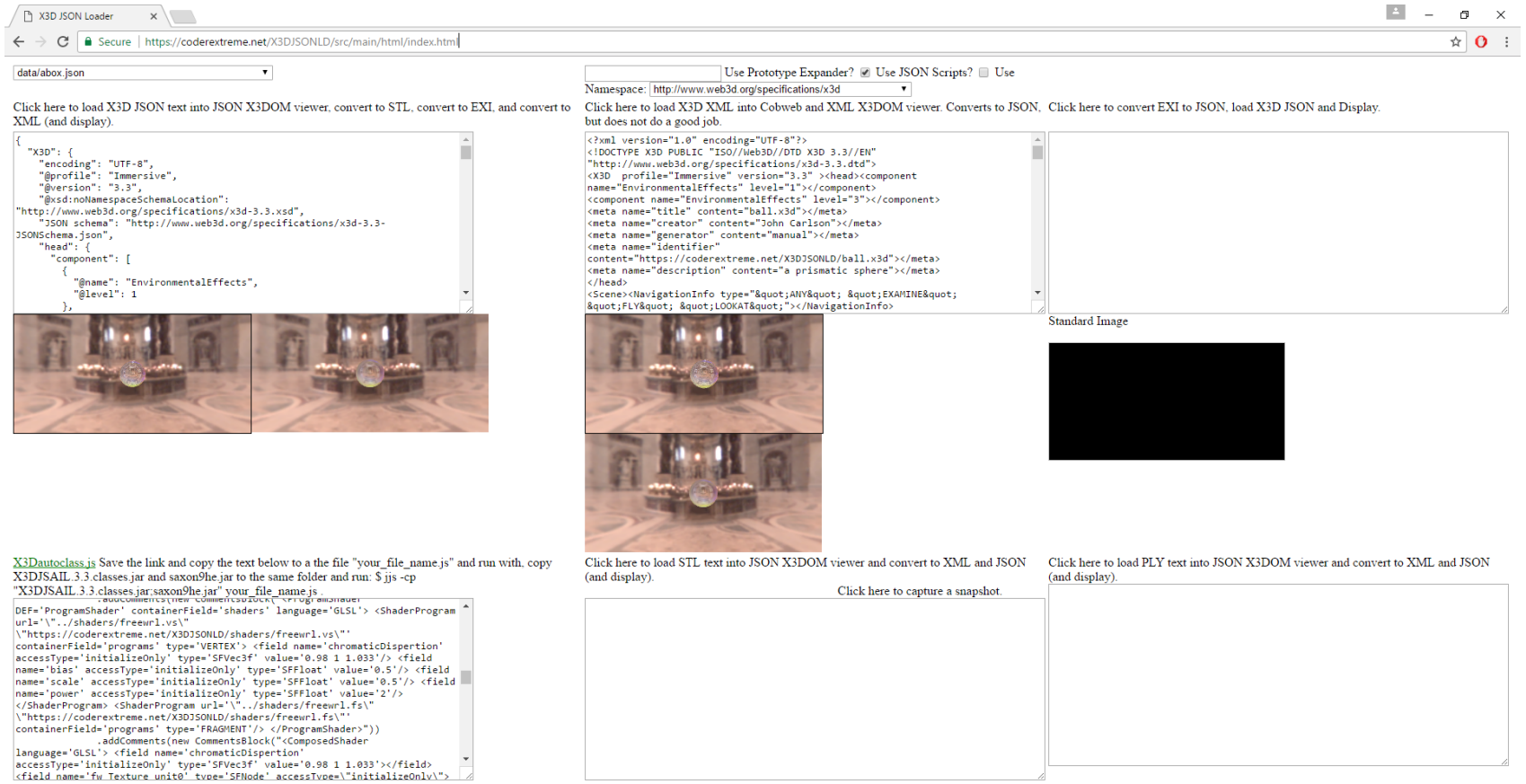
Server-side example (continued)

```
if (fs.existsSync(outfile)) {
        var data = fs.readFileSync(outfile);
        var json = JSON.parse(data.toString());
        json = externPrototypeExpander(outfile, json); <!– run the ExternProtoDeclare handler →
        json = flattener(json);
        send(res, json, "text/json", next);
    } else {
        var infile = file.substr(0, file.lastIndexOf("."))+".x3d";
        infile = www + "/" + infile;
        convertX3dToJson(res, infile, outfile, next);
    }
});
```

Results

- Varied. Simple files work. Many prototypes work. Inlines are troublesome.
- Scripts work in Cobweb, not X3DOM. The JSON version of X3DOM show scripts on the page (yikes). I need an option to remove the script if not going to cobweb.
- Cobweb has a way of stopping working. Requires a reload—Too many WebGL contexts? I think X3DOM has a way of reclaiming old ones.
- Client-side XSLT 2.0 processing with Saxon CE works poorly. Must use server side.
- Not much experience adding to a scenegraph through a script.

https://coderextreme.net/X3DJSONLD/



X3DJSONLD is a page for X3D JSON testing

- Convert X3D JSON -> XML (XML -> JSON hopefully soon, available with node app.js server on github—use X3D-Edit for now).
- JSON <-> EXI
- X3D JSON -> JavaScript
- X3D JSON <-> STL (beginnings)
- X3D JSON <- PLY (beginnings)
- Plans: JSON <-> GPG signed and encrypted files. “Working” prototype accomplished at: https://github.com/coderextreme/x3dserve/

Object Model Integration (src/main/python)

- **The Object Model provides data for X3DJSONLD Serializers and schema generators.**
- Uses Beautiful Soup 4 and lxml (bs4 and lxml python modules) to parse Object Model XML
- mapToMethod.py (X3DJSONLD) – produces mapToMethod.js.
  - Lists Java X3DJSAIL methods for parent node/child node relationships.
  - Disadvantages:
    - Produces duplicate keys
    - Does not cover everything, serializer must add additional methods.
    - Still needs work with duplicate keys, chaining methods.
- fieldtypes.py (X3DJSONLD) – produces fieldTypes.js
  - Lists types of attributes in a JSON object
- classes.py – produces X3DJSAIL autoclass imports for python (pythonSAI) and Nashorn X3DJSAIL “classes” (X3DJSONLD).
- etgenerateJSONSchema.py (X3DJSONLD) (early prototype, generates JSON Schema).

Credits

- Cubemap by Paul Debevec. http://www.pauldebevec.com/Probes/
- Thanks to Andreas Plesch for minimal X3DJSONLD example (modified).