JXD

by markus@danubetech.com

v1 19 Sep 2016 - initial version
v2 20 Sep 2016 - some small bugfixes
v3 21 Sep 2016 - added introduction and some description/rules

INTRODUCTION

JXD is a JSON-based serialization format for the XDI graph model, designed to combine the simplicity of JSON with the semantic richness of XDI.

An XDI graph can sometimes be serialized to JXD in different ways (some more verbose, some more compact), but deserializing a JXD back to XDI always results in the same original XDI graph. Every XDI graph can be serialized to JXD, and every JXD document can be deserialized to a valid XDI graph.

An XDI graph is built from XDI context nodes, which form a semantic tree. In JXD, an XDI context node is represented as a JSON object, with an **@id** JSON object key set to the XDI context node's address.

A single XDI context node is described as a single JSON object:

```
//=markus

{
  "@id": "=markus"
}
```

Multiple XDI context nodes are described as a single top-level JSON array, consisting of multiple JSON objects:

```
//=markus
//=drummond

[
  { "@id": "=markus" },
  { "@id": "=drummond" }
]
```

The next few pages explain how to describe the contents of XDI context nodes in more detail, including XDI literals, XDI relations, nested XDI context nodes, and XDI inner roots. For describing them, we can use a JXD mapping block (using an **@xdi** JSON object key) that maps simple JSON object keys to XDI concepts. The JXD mapping block may be part of the JXD document, or it may be referenced at an external location. The part of the JXD document that is not the JXD mapping block is called the JXD body.

ATTRIBUTES AND LITERALS

An XDI attribute and XDI literal are described using a JSON object key with a value that is the value of the XDI literal. In the JXD mapping block, a simple JSON object key may be defined that maps the JSON object key to a description of the XDI attribute and literal.

EXAMPLE

```
=markus<#name>/&/"Markus Sabadello"
=markus<#email>/&/"markus@danubetech.com"
```

```
{
  "@id": "=markus",
  "<#name>": "Markus Sabadello",
  "<#email>": "markus@danubetech.com"
}
```

OR: (describing the XDI attributes in the JXD mapping block)

```
{
  "@xdi": {
    "name": "<#name>",
    "email": "<#email>"
  },
  "@id": "=markus",
  "name": "Markus Sabadello",
  "email": "markus@danubetech.com"
}
```

OR! (describing the XDI attributes in the JXD mapping block in a more verbose way)

```
{
  "@xdi": {
    "name": { "@id": "<#name>" },
    "email": { "@id": "<#email>" }
  },
  "@id": "=markus",
  "name": "Markus Sabadello",
  "email": "markus@danubetech.com"
}
```

RELATIONS

One or more XDI relations are described using a JSON object key with a value that is a JSON array containing the target addresses of the XDI relation(s). In the JXD mapping block, a simple JSON object key may be defined that maps the JSON object key to a description of the XDI relation. The target address(es) of the XDI relation(s) must have a **@type** declaration set to **@id**. This declaration may be done either in the JXD body or in the JXD mapping block.

EXAMPLE

```
=markus/#friend/=drummond
```

```
{
  "@id": "=markus",
  "#friend": [ { "@id": "=drummond", "@type": "@id" } ]
}
```

OR: (describing the XDI relation in the JXD mapping block)

```
{
  "@xdi": {
    "friend": { "@id": "#friend", "@type": "@id" }
  },
  "@id": "=markus",
  "friend": [ "=drummond" ]
}
```

OR: (also describing the target address of the XDI relation in the JXD mapping block)

```
{
  "@xdi": {
    "friend": { "@id": "#friend", "@type": "@id" },
    "drummond": { "@id": "=drummond", "@type": "@id" }
  },
  "@id": "=markus",
  "friend": [ "drummond" ]
}
```

NESTED CONTEXT NODES

A nested XDI context node is described using a JSON object key with a value that is a JSON object containing the contents of the nested XDI context node, following the rules in this document recursively. In the JXD mapping block, a simple JSON object key may be defined that maps the JSON object key to a description of the nested XDI context node. The nested XDI context node must have a **@type** declaration set to **@id**. This declaration may be done either in the JXD body or in the JXD mapping block. If an XDI context node's only purpose is to contain nested XDI context nodes, then multiple XDI context nodes can be "collapsed" and therefore some JSON objects can be "omitted". This "collapse" may be done either in the JXD body or in the JXD mapping block.

EXAMPLE

```
+danubetech=markus<#work><#email>/&/"markus@danubetech.com"
```

```
{
  "@xdi": {
    "email": "<#email>"
  },
  "@id": "+danubetech",
  "=markus": {
    "@type": "@id",
    "<#work>": {
      "@type": "@id",
      "email": "markus@danubetech.com"
    }
  }
}
```

OR: (describing a nested XDI context node in the JXD mapping block)

```
{
  "@xdi": {
    "work": { "@id": "<#work>", "@type": "@id" },
    "email": "<#email>"
  },
  "@id": "+danubetech",
  "=markus": {
    "@type": "@id",
    "work": {
      "email": "markus@danubetech.com"
    }
  }
}
```

OR: (describing an additional nested XDI context node in the JXD mapping block)

```
{
  "@xdi": {
    "=markus": { "@type": "@id" },
    "work": { "@id": "<#work>", "@type": "@id" },
    "email": "<#email>"
  },
  "@id": "+danubetech",
  "=markus": {
    "work": {
      "email": "markus@danubetech.com"
    }
  }
}
```

OR: (collapsing two XDI context nodes in the JXD body)

```
{
  "@xdi": {
    "work": { "@id": "<#work>", "@type": "@id" },
    "email": "<#email>"
  },
  "@id": "+danubetech=markus",
  "work": {
    "email": "markus@danubetech.com"
  }
}
```

OR: (collapsing two XDI context nodes in the JXD mapping block)

```
{
  "@xdi": {
    "=markus": { "@type": "@id" },
    "workemail": "<#work><#email>"
  },
  "@id": "+danubetech",
  "=markus": {
    "workemail": "markus@danubetech.com"
  }
}
```

OR: (collapsing XDI context nodes both in the JXD body and in the JXD mapping block)

```
{
  "@xdi": {
    "workemail": "<#work><#email>"
  },
  "@id": "+danubetech=markus",
  "workemail": "markus@danubetech.com"
}
```

INNER ROOTS

An XDI inner root is described using a JSON object key with a value that is a JSON object containing the contents of the XDI inner root, following the rules in this document recursively. In the JXD mapping block, a simple JSON object key may be defined that maps the JSON object key to a description of the XDI inner root. The XDI inner root must have a **@type** declaration set to **@graph**. This declaration may be done either in the JXD body or in the JXD mapping block.

EXAMPLE 1 (MESSAGE)

```
(=markus[$msg]*!:uuid:1234$do/$set)=markus<#name>/&/"Markus Sabadello"
(=markus[$msg]*!:uuid:1234$do/$set)=markus<#email>/&/"markus@danubetech.com"
```

```
{
  "@xdi": {
    "name": "<#name>",
    "email": "<#email>",
    "friend": { "@id": "#friend", "@type": "@id" },
    "set": { "@id": "$set", "@type": "@graph" }
  },
  "@id": "=markus[$msg]*!:uuid:1234$do",
  "set": {
    "=markus": {
      "name": "Markus Sabadello",
      "email": "markus@danubetech.com"
    }
  }
}
```

EXAMPLE 2 (LINK CONTRACT)

```
(=markus/=drummond)$do/$get/=markus<#email>
(=markus/=drummond)($do$if$and/$true){$from}/$is/=drummond
(=markus/=drummond)($do$if$and/$true){$msg}<$sig><$valid>/&/true
```

```
{
  "@xdi": {
    "do": { "@id": "$do", "@type": "@id" },
    "if": { "@id": "$if", "@type": "@id" },
    "and": { "@id": "$and", "@type": "@id" },
    "or": { "@id": "$or", "@type": "@id" },
    "true": { "@id": "$true", "@type": "@graph" },
    "false": { "@id": "$false", "@type": "@graph" },
    "get": { "@id": "$get", "@type": "@id" },
    "from": { "@id": "{$from}", "@type": "@id" },
    "msg": { "@id": "{$msg}", "@type": "@id" },
    "is": { "@id": "$is", "@type": "@id" },
    "sigvalid": { "@id": "<$sig><$valid>" }
  },
  "@id": "=markus",
  "=drummond": {
    "@type": "@graph",
    "do": {
      "get": [ "=markus<#email>" ],
      "if": {
        "and": {
          "true": {
            "from": {
              "is": [ "=drummond" ]
            },
            "msg": {
              "sigvalid": true
            }
          }
        }
      }
    }
  }
}
```