Version 2.0

XML Objectifier Documentation

This documentation covers API for a dual-mode component that converts an XML Document or XML String into a JavaScript object.

Modes of operation

If jQuery is present on the page then component automatically gets added to jQuery namespace and become a plug-in (Ex.: $.xmlToJSON(xmlObj)). In case jQuery is not present, XMLObjectifier namespace will be created (Ex.: XMLObjectifier.xmlToJSON(xmlObj)). In any mode, component will work exactly the same way and will use same exact code base.

Principles of operation

XML Objectifier is capable of processing either XML objects or String. An example of XML object would be an XML response from an AJAX request, or any of it's' child components such as Node, CDATA, Comments, Text Node, etc. In case you're trying to convert a string, that string will be parsed into an XML object first and then converted.

XML Objectifier recursively works its way through XML DOM tree and builds out a JavaScript data structure that inherits named components and hierarchy of original document.

All XML Nodes are stored as Arrays that are assigned to the node name:
`<root><item /><item /></root> -> root.item = [obj, obj]`
All processed attributes are named starting "@" sign:
`<node attr="test"/> -> node["@attr"] = "test"`
Text nodes and CDATA elements become "Text" property:
`<node>Test | <![CDATA[Test]]> </node> -> node.Text = "Test"`
Comments get added to "$comments" Array:
`<node><!-- Test --></node> -> node["$comments"] = ["Test"]`

Building Blocks

Output JavaScript structure is built out of objects such as Root, NodeSet and Node. These objects provide a variety of methods that make accessing data fast and easy. Let me describe what each object type is:

- XMLObjectifier Factory – Singleton class that provides conversion functionality
- **`Root`** – represents a root node of an XML Document
- **`NodeSet`** – represents a sequence of one or more Node objects (**`<Array>`**)
- **`Node`** – represents a single node that may have children nodes, attributes, comments, value, etc.

Document Conventions

- **`Objects – Classes and Objects of XML Objectifier component`**
- **`<Types> - JavaScript types`**
- **`Object Members – Public methods, functions and properties`**
- `Function Arguments – Function arguments. [] = optional`
- **`Reserved Words – JavaScript reserved words`**
- `Code Examples – Inline code examples`
- `Code – JavaScript code`

API Description

`XMLObjectifier` – Singleton object that exposes functions for converting XML to JSON, Text to XML, JSON to XML and more.

- **`xmlToJSON`**(`xml`, `opt`) – converts XML to JSON
  - `xml` **`<Object>`** – XMLDocument | XMLDocumentFragment | XMLNode | Text Node, CDATA Node | Comment Node | XML String.
  - `opt` **`<Map>`** – Map of optional processing instructions
    - `noComments` – flag to include comment nodes. Values: `true` | **`false`**
    - `decorator` – callback function that runs in a scope of a Node object. This allows you to modify or skip nodes that go into a JSON output. To skip a node entirely just return false. Look at Node structure to apply proper filtering logic.
  - **`returns`** – instance of Root object.
- **`textToXML`**(`xmlstr`) – converts properly formatted XML string to XML Document object.
  - `xmlstr` **`<String>`** - properly formatted XML string (i.e. `"<node attr='test'/>"`)
  - **`returns`** **`<XMLDocument>`** - if properly parsed, returns an instance of `XMLDocument` object. Throws an exception if a string can not be parsed.
- **`xmlObjectifier`** **`<Map>`** - exposes two main class interfaces in case you may want to extend functionality through prototype.
  - `RootClass` – reference to a Root Class
  - `NodeClass` – reference to a Node Class

`Root` – Class component that represents a root element in the output JSON

- **`nodeName`** **`<String>`** - property that contains a node name of the root element
- **`ns`** **`<String>`** - property that contains an XML namespace prefix.
- **`typeOf`** **`<String>`** - constant value of "xmlObjectifier" to identify an object
- **`Text`** **`<String>`** - contains Text or CDATA value of the node
- **`hasCDATA`** **`<Boolean>`** - flag that indicates that node value originates from a CDATA node.
- **`attr`**(`name` [, `value`]) – setter/getter method to set or retrieve a node attribute
  - `name` **`<String>`** - attribute name
  - `value` **`<Object>`** - attribute value
  - **`returns`** **`<Object>`** – if value argument is provided - returns a reference to **`this`** object, else if named attribute found – returns its value, else returns undefined.
- **`find`**(`selector`) – tries to find child nodes based on selector expressions similar to XPATH
  - `selector` **`<String>`** - XPATH like expression to select child nodes
  - **`returns`** – may return a reference to a Node or NodeSet
- **`appendChild`**(`node`) – appends new child Node
  - `node` **`<Node>`** - Instance to a Node object
  - **`returns`** – reference to **`this`** object
- **`addComment`**(`comment`) – adds a comment to a node
  - `comment` **`<String>`** - comments value
  - **`returns`** – reference to **`this`** object
- **`val`**([`value`]) – setter/getter method for a node value
  - `value` **`<String>`** - Node string value
  - **`returns`** – if value argument is not provided – returns node value, else returns a reference to **`this`** object.
- **`toXML`**() – converts JSON to XMLDocument. Output will reflect any changes made to JSON object.
  - **`returns`** – XMLDocument representative of JSON object
- **`toString`**() – converts JSON to XML String
  - **`returns`** – XML formatted string

Node – Class component that represents an XML Node element.

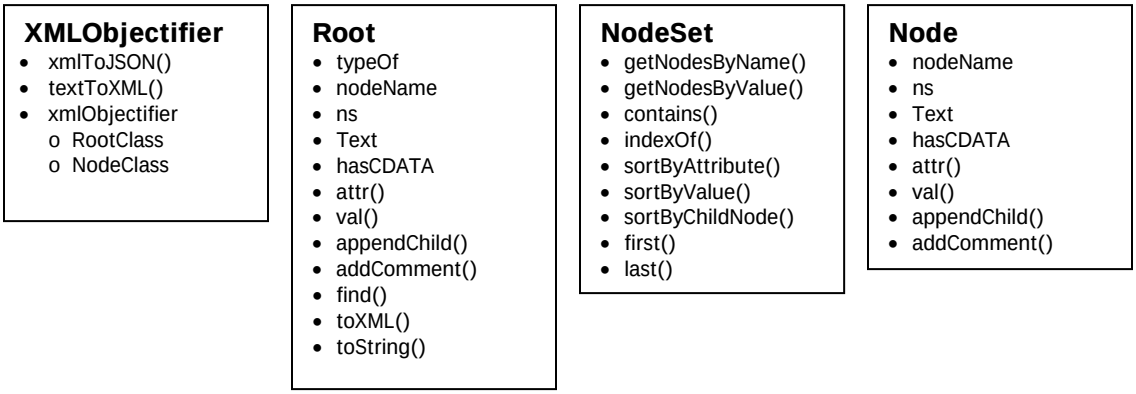
- Object Constructor (`name` | [`parent`, `name`, `value`]) – creates a new instance of the Node object.
  - `name` **<String>** - node name
  - `parent` **<Root | Node>** - reference to a parent element
  - `value` **<String>** - Node value
- **parent** **<Root | Node>** - property that holds a reference to a parent element
- **Text** **<String>** - contains Text or CDATA value of the node
- **hasCDATA** **<Boolean>** - flag that indicates that node value originates from a CDATA node.
- **nodeName** **<String>** - property that contains a node name of the root element
- **ns** **<String>** - property that contains an XML namespace prefix.
- **attr**(`name` [, `value`]) – setter/getter method to set or retrieve a node attribute
  - `name` **<String>** - attribute name
  - `value` **<Object>** - attribute value
  - `returns` **<Object>** – if value argument is provided - returns a reference to **this** object, else if named attribute found – returns its value, else returns undefined.
- **appendChild**(`node`) – appends new child Node
  - `node` **<Node>** - Instance to a Node object
  - **returns** – reference to **this** object
- **addComment**(`comment`) – adds a comment to a node
  - `comment` **<String>** - comments value
  - **returns** – reference to **this** object
- **val**([`value`]) – setter/getter method for a node value
  - `value` **<String>** - Node string value
  - **returns** – if value argument is not provided – returns node value, else returns a reference to **this** object.

NodeSet – Interface that exposes a multitude of helper functions for traversing and manipulating nodes inside a NodeSet.

- **getNodesByAttribute**(`attribute`, `value`) – retrieves all nodes that match value of an attribute
  - `attribute` **<String>** - attribute name
  - `value` **<String>** - attribute value to match
  - **returns** **<Array>** - array of **Node** elements or an empty array if nothing is found
- **getNodesByValue**(`value`) – retrieves all nodes that match Text value
  - `value` **<String>** - Text value of the Node
  - **returns** **<Array>** - array of Node elements or an empty array if nothing is found
- **contains**(`attribute`, `value`) – returns a Boolean to flag a presence of a node that matches an attribute value
  - `attribute` **<String>** - attribute name
  - `value` **<String>** - attribute value to match
  - **returns** **<Boolean>** - true if node is present
- **indexOf**(`attribute`, `value`) – returns an index of a first occurrence of a match
  - `attribute` **<String>** - attribute name
  - `value` **<String>** - attribute value to match
  - **returns** **<Integer>** - Index if node is present, -1 if not.
- **sortByAttribute**(`attribute`, `dir`) – sorts nodes in the NodeSet by an attribute value
  - `attribute` **<String>** - attribute name
  - `dir` **<String>** - Sorting order. Values: “asc” | “desc”
  - **returns** – reference to **this** object
- **sortByValue**(`dir`) – sorts nodes by text value
  - `dir` **<String>** - Sorting order. Values: “asc” | “desc”
  - **returns** – reference to **this** object

- **sortByChildNode**(nodeName, dir) – sorts nodes by a child node text value. This is common for Microsoft XML structures where a row is represented as a row.column group.
    - nodeName **<String>** - name of the child node
    - dir **<String>** - Sorting order. Values: “asc” | “desc”
    - **returns** – reference to **this** object
- **first**() – returns a reference to a first Node element in a NodeSet
- **last**() – returns a reference to the last Node element in a NodeSet

Block Diagram



Conversion Analysis

Source XML:

```
<root>
      <item name=”a”>Test</item>
      <item name=”b”><![CDATA[Test 2]]></item>
      <!—This is my comment -->
      <rows>
            <row index=”0”>
                  <cell>A</cell><cell>B</cell><cell>C</cell>
            </row>
      </rows>
</root>
```

Output: (Note: Public method names are skipped**)**

```
<Root> {
  typeOf: “xmlObjectifier”,
  nodeName: “root”,
  ns: “”,
  Text: “”,
  <NodeSet> item: [
      <Node> { nodeName: “item”, ns:””, “@name”:”a”, Text: “Test”, parent: {<Root>}},
      <Node> { nodeName: “item”, ns:””, “@name”:”b”, Text: “Test 2”, hasCDATA: true,
        parent: {<Root>}}
  ],
  $comments: [“This is my comment”],
  <NodeSet> rows: [
      <Node> { nodeName: “rows”, ns: “”, Text: “”, parent: {<Root>},
        <NodeSet> row: [
            <Node> {nodeName: “row”, ns: “”, Text: “”, “@index”:”0”, parent: {<Node>},
             <NodeSet> cell: [
                  <Node> { nodeName:”cell”, ns: “”, Text: “A”, parent: {<Node>}},
                  <Node> { nodeName:”cell”, ns: “”, Text: “B”, parent: {<Node>}},
                  <Node> { nodeName:”cell”, ns: “”, Text: “C”, parent: {<Node>}}
             ]}
        ]}
  ]
}
```