

JSON and XML

This page covers converting between text and data with the JSON and XML objects, and reading XML documents.

Topics on this page:

- [JSON](#): [JSON.parse](#), [JSON.stringify](#), [example](#)
- [XML](#): [XML.parse](#), [XML.stringify](#), [XML node members](#), [querying lists of nodes](#)

See also: [Lists and Maps](#), [HTTP and Utilities](#).

JSON

JSON objects become [maps](#) and JSON arrays become [lists](#), so you can use all map and list features on parsed data.

JSON.parse(text)

Parses a JSON string and returns the resulting map, list, string, number, boolean, or null.

If text is not valid JSON, or is not a string, the result is null. No script error is raised, so check the result before you use it.

```
data = JSON.parse('{ "name": "Pump 1", "readings": [12.5, 13, 14.25], "ok": true }');
print(data.name, "\n");
print(data.readings[1], " ", data.readings.length, "\n");
print(JSON.parse("not json"), "\n");
return data.ok;
```

Print output:

```
Pump 1
13 3
null
```

Output: true

If you send a parsed map or list to an output property of a Script block, the value is converted to text when the property is a string property.

JSON.stringify(value, indent)

Converts a map, list, string, number, boolean, or null to a JSON string.

- `indent` — optional. A number of spaces, or a string, to indent each level. Without `indent` the result is on one line.

`value` can contain only JSON values. A `DateTime`, an XML node, a page model object, `NaN` or `Infinity` causes an error. Convert dates to strings first, for example with `dateFormat`.

```
obj = {a: 1, b: [1, 2], c: null};
print(JSON.stringify(obj), "\n");
print(JSON.stringify("say \"hi\"", "\n"));
return JSON.stringify({a: 1, b: [1, 2]}, 2);
```

Print output:

```
{"a":1,"b":[1,2],"c":null}
"say \"hi\""
```

Output:

```
{
  "a": 1,
  "b": [
    1,
    2
  ]
}
```

JSON Example

The following example uses a for loop to print the `id` of each menu item and prints dashes for null items. `print` does not add line breaks, so the output is one line.

```
json = {"menu": {
  "header": "SVG Viewer",
  "items": [
    {"id": "Open"},
    {"id": "OpenNew", "label": "Open New"},
  ]
}
```

```

        null,
        {"id": "ZoomIn", "label": "Zoom In"},
        {"id": "ZoomOut", "label": "Zoom Out"},
        {"id": "OriginalView", "label": "Original View"},
        null,
        {"id": "Quality"},
        {"id": "Pause"},
        {"id": "Mute"},
        null,
        {"id": "Find", "label": "Find..."},
        {"id": "FindAgain", "label": "Find Again"},
        {"id": "Copy"},
        {"id": "CopyAgain", "label": "Copy Again"},
        {"id": "CopySVG", "label": "Copy SVG"},
        {"id": "ViewSVG", "label": "View SVG"},
        {"id": "ViewSource", "label": "View Source"},
        {"id": "SaveAs", "label": "Save As"},
        null,
        {"id": "Help"},
        {"id": "About", "label": "About SVG Viewer..."}
    ]
};

print(json.menu.header);

for (var i = 0; i < json.menu.items.length; i++) {
    if (json.menu.items[i] == null) {
        print("----");
    } else {
        print(json.menu.items[i].id);
    }
}
return json.menu.items.length;

```

Print output:

```

SVG ViewerOpenOpenNew----ZoomInZoomOutOriginalView----QualityPauseMute----
FindFindAgainCopyCopyAgainCopySVGViewSVGViewSourceSaveAs----HelpAbout

```

Output: 22

XML

XML.parse(text)

Parses an XML string and returns its root element. If `text` is not well-formed XML, a script error occurs. If `text` is not a string, the result is `null`.

```
xml = XML.parse('<p id="a">b</p>');  
return xml.name;
```

Output: p

XML.stringify(element)

Converts an XML element back to an XML string. If the value is not an XML element, the result is `null`.

```
xml = XML.parse('<p id="a">b</p>');  
return XML.stringify(xml);
```

Output: <p id="a">b</p>

XML Node Members

Member	Description
<code>node.name</code>	The element name without a namespace prefix. <code>null</code> for text nodes
<code>node.getAttribute(name)</code>	The value of an attribute, or <code>null</code> if the attribute does not exist
<code>node.text</code>	The text of an element. This works only when the element contains exactly one text (or CDATA) node and nothing else; otherwise it is <code>null</code>
<code>node.data</code>	The text of a text or CDATA node, for example an item of children. <code>null</code> for elements
<code>node.children</code>	A list of all child nodes, including text nodes
<code>node.elements</code>	A list of the child elements only
<code>node.query(name)</code>	The first descendant element with the given name, or <code>null</code> . See below
<code>node.queryAll(name)</code>	A list of all descendant elements with the given name. See below
<code>node.remove()</code>	Removes the node from its parent

`query` and `queryAll` search the descendants of the node, depth first, by element name. The node itself is not checked. When an element matches, its own descendants are not searched, so `queryAll` does not return a matching element that is nested inside another matching element. The name is a plain element

name; paths and CSS-style selectors are not supported.

```
xml = XML.parse('<a><b>B 1<c>C 1</c></b><b att="2">B 2</b></a>');
print(xml.query("b").children.join(", "), "\n");
print(xml.query("b").elements.join(", "), "\n");
print(xml.query("b").text, "\n");
print(xml.query("c").text, "\n");
print(xml.query("b").children[0].data, "\n");
print(xml.queryAll("b").length, "\n");
print(xml.queryAll("b")[1].getAttribute("att"), "\n");
print(xml.query("missing"), "\n");
xml.query("c").remove();
return XML.stringify(xml);
```

Print output:

```
B 1,<c>C 1</c>
<c>C 1</c>
null
C 1
B 1
2
2
null
```

Output: <a>B 1<b att="2">B 2

Querying Lists of Nodes

A list of XML nodes returned by `children`, `elements` or `queryAll` also has `query(name)` and `queryAll(name)`. They search each node in the list in the same way. Unlike the element methods, a node in the list is itself checked against the name.

These methods return `null` when the list is empty or is not a list returned by an XML member (for example a list you built with `[ ]`).

```
xml =
XML.parse('<root><group><item>1</item><item>2</item></group><group><item>3</item></group></root>');
groups = xml.queryAll("group");
items = groups.queryAll("item");
values = items.map(function (n) { return n.text; });
print(values.join(", "), "\n");
return groups.query("item").text;
```

Print output:

```
1,2,3
```

Output: 1

Example: Read an RSS-style Feed

```
src = '<rss><channel><title>News</title>' +  
      '<item><title>First</title><link>http://a.example</link></item>' +  
      '<item><title>Second</title><link>http://b.example</link></item>' +  
      '</channel></rss>';  
doc = XML.parse(src);  
rows = [];  
doc.queryAll("item").forEach(function (item) {  
  rows.push({title: item.query("title").text, link: item.query("link").text});  
});  
print(doc.query("title").text, "\n");  
return JSON.stringify(rows);
```

Print output:

```
News
```

Output:

```
[{"title": "First", "link": "http://a.example"}, {"title": "Second", "link": "http://b.example"}]
```

From:

<http://wiki.dglogik.com/> - DGLogik

Permanent link:

http://wiki.dglogik.com/dglux5_wiki:dgscript:json_xml

Last update: **2026/10/22 17:28**

