GEP 7 - JSON Support

Metadata

Number:	GEP-7
Title:	JSON Support
Version:	3
Type:	Feature
Target:	1.8
Status:	Implemented (in 1.8-beta-4)
Leader:	Guillaume Laforge
Contributors:	Andres Almiray
Created:	2010-10-25 by Andres
Last modification:	2011-02-02 by Guillaume

Abstract

Provide a builder/slurper combination for handling data in JSON format in a similar fashion as it's already done for XML.

Rationale

JSON has become ubiquitous to the web. RESTful services exchange data in both POX (Plain Old XML) and JSON formats. Groovy has excellent support for producing/consuming XML with `MarkupBuilder`, `XmlSlurper` and `XmlParser` but lacks this kind support for JSON. This GEP strives to remedy the situation, by providing a compatible builder approach.

Producing JSON

The following builder syntax is proposed

```
def builder = new groovy.json.JsonBuilder()
def root = builder.people {
    person {
        firstName 'Guillame'
        lastName 'Laforge'
        // Maps are valid values for objects too
        address(
            city: 'Paris',
            country: 'France',
            zip: 12345,
        )
        married true
        conferences 'JavaOne', 'Gr8conf'
    }
}

// creates a data structure made of maps (Json object) and lists (Json array)
assert root instanceof Map

println builder.toString()

// prints (without formatting)
{"people": {
    "person": {
        "firstName": "Guillaume",
        "lastName": "Laforge",
        "address": {
           "city": "Paris",
           "country": "France",
           "zip": 12345
        },
        "married": true,
        "conferences": [
            "JavaOne",
            "Gr8conf"
        ]
    }
}
```

Valid node values are: `Number`, `String`, `GString`, `Boolean`, `Map`, `List`. `null` is reserved for object references. Arrays can not be null but they can be empty. Anything else results in an IAE (or a more specialized exception) being thrown.

Special cases

There is a special case to be considered: when the top node results in an anonymous object or array. For objects a `call()` method on the builder is needed which takes a map as argument, for arrays `call()` takes a vararg of values. Here are some examples:

```
builder.foo "foo"
// produces
{foo: "foo"}

builder([{
  foo 'foo'
}])
// produces
[{"foo": "foo"}]

builder([[
  foo: 'foo'
]])
// produces, same as above
[{"foo": "foo"}]

builder {
    elem 1, 2, 3
}
// produces
{ "elem": [1, 2, 3] }
```

When a method is called on the builder without arguments, and empty JSON object is associated with the key:

```
builder.element()
// produces
{ "element": {} }
```

You can also pass a map and a closure argument:

```
builder.person(name: "Guillaume", age: 33) { town "Paris" }
// produces
{"name": "Guillaume", "age": 33, "town": "Paris}
```

Calls like the following, with a map and a value, don't have any meaningful representation in JSON (unlike in XML), and triggers a JsonException:

```
shouldFail(JsonException) {
    builder.elem(a: 1, b: 2, "some text value")
}
```

In case of overlapping keys in the map and the closure, the closure wins – a visual clue for this rule is that the closure appears "after" the map key/value pairs.

Consuming JSON

The proposal is for the creation of a JsonSlurper class that can read JSON from a string (in a non-streaming fashion) and produce a hierarchy of maps and lists representing the JSON objects and arrays respectively.

```
String json = '{"person": {"firstName": "Guillaume", "lastName": "Laforge",
"conferences": ["JavaOne", "Gr8conf"]}}'
def root = new JsonSlurper().parseText(json)
assert root instanceof Map
assert root.person.conferences instanceof List
assert root.person.firtsName == 'Guillaume'
assert root.person.conferences[1] == 'Gr8conf'
```

JsonSlurper's API should mirror closely what XmlParser/XmlSlurper offers in terms of its parse* method variants.

References

JSON Spec

- json.org
- RFC-4627

Java Implementations

- json.org
- json-lib

Mailing-list discussions

Built-in JSON support in Groovy 1.8

JIRA issues

GROOVY-4644