From JSON to SURF in Java

SeaJUG / 2017-05-16

Garret Wilson <garret@globalmentor.com>

GlobalMentor, Inc.

<http://www.globalmentor.com/>
info@globalmentor.com

SURF Format

JSON replacement for rich, readable data interchange.

SURF

Simple URF

URF

Uniform Resource Framework

http://urf.io/surf/

SURF Serializes ***Resources***

Categories of SURF resource serializations:

- literals
- objects
- collections

But why SURF?

Isn’t JSON simple and easy?

```json
{
  "name": "Jane Doe",
  "id": "bb8e7dbe-f0b4-4d94-a1cf-46ed0e920832",
  "email": "jane_doe@example.com",
  "phone": "+12015550123",
  "aliases": ["jdoe", "janed"],
  "homePage": "http://www.example.com/jdoe/",
  "joined": "2016-01-23",
  "balance": 0.90,
  "charge": 0.30
}
```

Isn't JSON good enough?

Want a date?

As of 2011, there are some de facto standards, e.g., converting from Date to String, but none universally recognized.

JSON , Wikipedia, retrieved 2017-05-14.

JSON Dates in the Wild

- `"\"\\/Date(1335205592410)\\/\""`
- `"\"\\/Date(1335205592410-0500)\\/\""`
- `"2012-04-23T18:25:43.511Z"`
- `"@1335205592410@"`

The "right" JSON date format, Stack Overflow. / Bertrand Le Roy.

But these are still all JSON strings, not dates!

How can one distinguish a *date* string from a *string* string? And what about local dates? Times? Durations?

Have a Date @ SURF

```
{
  "name": "Jane Doe",
  "id": "bb8e7dbe-f0b4-4d94-a1cf-46ed0e920832",
  "email": "jane_doe@example.com",
  "phone": "+12015550123",
  "aliases": ["jdoe", "janed"],
  "homePage": "http://www.example.com/jdoe/",
  "joined": @2016-01-23,
  "balance": 0.90,
  "charge": 0.30
}
```

SURF Likes Dates!

SURF provides over a dozen date/time/duration types.

- Instant: `@2017-02-12T23:29:18.829Z`
- ZonedDateTime: `@2017-02-12T15:29:18.829-08:00[America/Los_Angeles]`
- LocalDate: `@2017-02-12`
- LocaleTime: `@15:29:18.829`
- Year: `@2017`
- …

Aren't JSON numbers OK?

Calculating a JSON balance...

```
{"balance": 0.90, "charge": 0.30}
```

...using Java

```
double balance = 0.90;  //starting account balance of $0.90
double charge = 0.30;  //monthly charge of $0.30
while(balance > 0) {
  balance -= charge;  //subtract the charge each month
  System.out.println("balance: " + balance);
}
```

```
balance: 0.6000000000000001
balance: 0.3000000000000001
balance: 1.1102230246251565E-16
balance: -0.2999999999999999
```

❗ **Never use *IEEE 754* floating point numbers to represent money!**

❌ But that's all JSON has!

💡 Should you put money in JSON strings instead?

💡 Should you store integer cents as JSON numbers?

SURF Has a $ Decimal Type

```
{
   "name": "Jane Doe",
   "id": "bb8e7dbe-f0b4-4d94-a1cf-46ed0e920832",
   "email": "jane_doe@example.com",
   "phone": "+12015550123",
   "aliases": ["jdoe", "janed"],
   "homePage": "http://www.example.com/jdoe/",
   "joined": @2016-01-23,
   "balance": $0.90,
   "charge": $0.30
}
```

📌 Native language support: Java (`BigDecimal`), C#, Python, Ruby, …

SURF Has Lots of Types

- Regular Expression: `/a?b+c*/`
- IRI/URI/URL/URN: `<http://example.com/>`
- UUID: `&5623962b-22b1-4680-ae1c-7174a46144fc`
- Email Address: `^jdoe@example.com`
- Telephone Number: `+12015550123`
- Binary: `%Zm9vYmFy`

Rich Types with SURF

```
{
   "name": "Jane Doe",
   "id": &bb8e7dbe-f0b4-4d94-a1cf-46ed0e920832,
   "email": ^jane_doe@example.com,
   "phone": +12015550123,
   "aliases": ["jdoe", "janed"],
   "homePage": <http://www.example.com/jdoe/>,
   "joined": @2016-01-23,
   "balance": $0.90,
   "charge": $0.30
}
```

Look Mom, no commas!

```
{
  "name": "Jane Doe"
  "id": &bb8e7dbe-f0b4-4d94-a1cf-46ed0e920832
  "email": ^jane_doe@example.com
  "phone": +12015550123
  "aliases": [
    "jdoe"
    "janed"
  ]
  "homePage": <http://www.example.com/jdoe/>
  "joined": @2016-01-23
}
```

Compact with Commas

```
{"name":"Jane Doe","id":&bb8e7dbe-f0b4-4d94-a1cf-46ed0e920
"email":^jane_doe@example.com,"phone":+12015550123,"aliases
["jdoe","janed"],"homePage":<http://www.example.com/jdoe/>,
"joined":@2016-01-23,"balance":$0.90,"charge": $0.30}
```

Clearer with ! Comments

```
{
  "name": "Jane Doe"
  "id": &bb8e7dbe-f0b4-4d94-a1cf-46ed0e920832
  "email": ^jane_doe@example.com
  "phone": +12015550123
  "aliases": ["jdoe", "janed"]
  "homePage": <http://www.example.com/jdoe/>
  "joined": @2016-01-23
  "balance": $0.90  !current account balance
  "charge": $0.30  !monthly charge
}
```

JSON *objects* aren't really.

JSON *objects* are *maps*.

```
{
   "name": "Jane Doe",
   "id": "bb8e7dbe-f0b4-4d94-a1cf-46ed0e920832",
   "email": "jane_doe@example.com",
   "phone": "+12015550123",
   "aliases": ["jdoe", "janed"],
   "homePage": "http://www.example.com/jdoe/",
   "joined": "2016-01-23",
   "balance": 0.90,
   "charge": 0.30
}
```

⚠ JSON only allows string for map keys.

SURF Has Real Objects

A SURF object looks like this:

```
*
```

💡 Think: *An object instance.*

A SURF object can have real property assignments.

```
1  *:
2    name = "Jane Doe"
3  ;
```

A SURF object can have a type.

```
1  *User:
2    name = "Jane Doe"
3  ;
```

💡 Think: *An instance of `User` described as follows: …*

❗ SURF requires * for every object instance.

📌 The separate TURF format allows object types themselves to be described, e.g. `User:…;`.

A Real `User` Object in SURF

```
*User:
  name = "Jane Doe"
  id = &bb8e7dbe-f0b4-4d94-a1cf-46ed0e920832
  email = ^jane_doe@example.com
  phone = +12015550123
  aliases = ["jdoe", "janed"]
  homePage = <http://www.example.com/jdoe/>
  joined = @2016-01-23
  balance = $0.90
  charge = $0.30
;
```

SURF Has Collections

- list
- set
- map

ℹ️ SURF parsers map collections to native programming language implementations.

✅ Perfect fit for Java collection types!

📌 Even JavaScript has real map and set types nowadays.

Lists Like You Like

```
1  *Rainbow:
2    colors = ["red", "orange", "yellow", "green",
3        "blue", "indigo", "violet"]
4  ;
```

- Just like JSON.

Maps Make a Return

```
1  *Person:
2   favoriteThings = {
3       5: "This person's favorite number."
4       "aliquot": "This person's favorite word."
5     }
6  ;
```

- Just like JSON… but without key type restrictions.

SURF Sets Make Sense

```
*User:
  name = "Jane Doe"
  id = &bb8e7dbe-f0b4-4d94-a1cf-46ed0e920832
  email = ^jane_doe@example.com
  phone = +12015550123
  aliases = ("jdoe", "janed")
  homePage = <http://www.example.com/jdoe/>
  joined = @2016-01-23
  balance = $0.90
  charge = $0.30
;
```

ℹ Here a set makes more semantic sense for the `aliases` because they are unique and unordered.

Reference Resources with SURF Labels

SURF Local Label |foo| for *Internal* Referencing

```
1  *Game:
2    players = [
3      |doe|*User:
4        name = "Jane Doe"
5      ;
6      |smith|*User:
7        name = "John Smith"
8      ;
9    ]
10   winner = |doe| !reference an existing instance
11 ;
```

Global Labels `|<IRI>|` for *External* Referencing

```
|<urn:uuid:bb8e7dbe-f0b4-4d94-a1cf-46ed0e920832>|*User:
  name = "Jane Doe"
  id = &bb8e7dbe-f0b4-4d94-a1cf-46ed0e920832
  email = ^jane_doe@example.com
  phone = +12015550123
  aliases = ("jdoe", "janed")
  homePage = <http://www.example.com/jdoe/>
  joined = @2016-01-23
  balance = $0.90
  charge = $0.30
;
```

- Short form: `|<&bb8e7dbe-f0b4-4d94-a1cf-46ed0e920832>|`

Reference Global Labels, Too

```
1  *Game:
2    players = [
3      |<&bb8e7dbe-f0b4-4d94-a1cf-46ed0e920832>|*User:
4        name = "Jane Doe"
5      ;
6      |<&d5207ff8-f64e-46b9-8fd9-d84ce8e0ff29>|*User:
7        name = "John Doe"
8      ;
9    ]
10   winner = |<&bb8e7dbe-f0b4-4d94-a1cf-46ed0e920832>|
11 ;
```

- Global label IRIs are accessible even *outside* of SURF documents.

SURF Names

Type names should use upper camelCase, e.g. `User`

Property names should use lower camelCase, e.g. `name`

SURF Namespaces (Optional)

Prevent name clashes by adding hyphen-minus characters to form namespaces:

`chem-salt` ***Salt (chemistry)***

```
*:chem-salt="monosodium glutamate";
```

`crypto-salt` ***Salt (cryptography)***

```
*:crypto-salt=%Zm9vYmFy;
```

ℹ No namespace declarations in SURF.

✔ SURF parsers need no knowledge of namespaces.

Mix Vocabularies Using Namespaces

```
|<&bb8e7dbe-f0b4-4d94-a1cf-46ed0e920832>|*example-User:
  name = "Jane Doe"
  email = ^jane_doe@example.com
  crypto-salt=%Zm9vYmFy
;
```

- When creating a namespace, try to use an identifier from a domain or second-level domain you control.

SURF and Semantics

SURF is based on the ***Uniform Resource Framework*** (URF), a semantic data model framework.

Uniform Resource Framework (URF)

- URF is like the ***Resource Description Framework*** (RDF)—only less complicated and more consistent.
- *SURF requires absolutely **no** knowledge of URF!*
- You can use SURF like a better JSON, ignoring URF.
- SURF gives you a semantic data model for free!

Every SURF datum represents an URF ***resource***.

- `example-User` (SURF type)
- `crypto-salt` (SURF property)
- `"foobar"` (SURF string literal)

Every URF resource can be identified by an IRI.

Identifying a SURF Object by Global Label IRI

```
1 |<urn:uuid:bb8e7dbe-f0b4-4d94-a1cf-46ed0e920832>|*example-U
2   name = "Jane Doe"
3 ;
```

Every SURF name represents an URF resource IRI.

Namespace components represent segments relative the IRI `https://urf.name/`.

`example-User` **(SURF type)**

`https://urf.name/example/User`

`crypto-salt` **(SURF property)**

`https://urf.name/crypto/salt`

Even SURF literals have URF resource IRIs.

SURF literals form resource URIs by their type URI, with the lexical representation as a fragment identifier.

`"foobar"` **(urf-String)**

`https://urf.name/urf/String#foobar`

`%Zm9vYmFy` **(urf-Binary)**

`https://urf.name/urf/Binary#Zm9vYmFy`

Every URF resource can be described.

Describing a SURF Object

```
1 |<urn:uuid:bb8e7dbe-f0b4-4d94-a1cf-46ed0e920832>|*example-U
2   name = "Jane Doe"
3 ;
```

Describing an URF Property in SURF

```
|<https://urf.name/crypto/salt>|*urf-Property:
  urf-description = "Cryptographic hash function extra inpu
  urf-domain = |<https://urf.name/urf/Resource>|
  urf-range = |<https://urf.name/urf/Binary>|
;
```

💡 Think: *`crypto-salt` is an `urf-Property` that can be assigned to any resource (its **domain**) and has values (its **range**) of type **urf-Binary**.*

Describing an URF Property in *TURF*

```
1  crypto-salt*urf-Property:
2    urf-description = "Cryptographic hash function extra inpu
3    urf-domain = urf-Resource
4    urf-range = urf-Binary
5  ;
```

ℹ️ TURF is a separate format for storing URF data.

📌 TURF is superset of SURF.

ℹ️ SURF prohibits describing named types directly to prevent confusion.

✅ *No need to know more about TURF for now.*

SURF has no schema.

There is no need to restrict the SURF syntax.

Define an URF ***ontology*** of the data model.

ontology

A semantic description of a vocabulary of types and properties; their meanings; and how they can be used.

Describing `example-User` in an URF Ontology

```
*urf-Ontology:
  urf-of = (
    |<https://urf.name/example/name>|*urf-Property:
      urf-description = "User name."
      urf-domain = |<https://urf.name/example/User>|
      urf-range = |<https://urf.name/urf/String>|
    ;
    …
  )
;
```

📌 Experimental.

SURF Reference Implementation

http://urf.io/surf/impl/ref

Source Code

https://bitbucket.org/account/user/globalmentor/pr

Issues

https://globalmentor.atlassian.net/projects/URF

Maven

`io.urf`

SURF Parser

pom.xml

```
<project>
  …
  <dependencies>
    …
    <dependency>
      <groupId>io.urf</groupId>
      <artifactId>surf-parser</artifactId>
      <version>x.x.x</version>
    </dependency>
  </dependencies>
</project>
```

Parse File `user.surf`

```
1  import io.urf.surf.parser.*;
2
3  …
4
5  SurfParser parser = new SurfParser();
6  Optional<Object> surf;
7  Path = Paths.get("user.surf");
8  try (final InputStream inputStream = newInputStream(path))
9    surf = parser.parse(inputStream);
10 }
```

Parsing Results

The returned SURF data will be one of the following, based upon the root resource in the file:

- `Optional.empty()` if the file was empty.
- A value object (e.g. `java.lang.Integer`) if the root resource was a ***SURF literal***.
- A implementation of a collection type (e.g. `java.util.Set<>`) if the root resource was a ***SURF collection***.
- An instance of `SurfObject` if the root resource was a ***SURF object***.

Access SURF Data

```
1  SurfObject user = (SurfObject)surf
2      .orElseThrow(() -> new IOException("User file empty."))
3  URI userIri = user.getIri()
4      .orElseThrow(() -> new IOException("Missing user IRI."))
5  String userName = (String)user.getPropertyValue("name")
6      .orElseThrow(() -> new IOException("Missing user name."
7  Optional<Object> userJoined = user.getPropertyValue("joined
8  userJoined.ifPresent(joined -> {
9    System.out.println(String.format("User joined %d days ago
10       DAYS.between((LocalDate)joined, LocalDate.now()));
11 });
```

SURF Serializer

pom.xml

```
<project>
  …
  <dependencies>
    …
    <dependency>
      <groupId>io.urf</groupId>
      <artifactId>surf-serializer</artifactId>
      <version>x.x.x</version>
    </dependency>
  </dependencies>
</project>
```

Create `SurfObject`

```
1  import io.urf.surf.parser.*;
2  import io.urf.surf.serializer.*;
3
4  …
5
6  userIri = URI.fromString("urn:uuid:bb8e7dbe-f0b4-4d94"
7      +"-a1cf-46ed0e920832");
8  SurfObject user = new SurfObject(userIri, "example-User");
9  user.setPropertyValue("name", "Jane Doe");
10 LocalDate joined = LocalDate.of(2016, Month.JANUARY, 23);
11 user.setPropertyValue("joined", joined);
```

Serialize File `user.surf`

```
1  SurfSerializer serializer = new SurfSerializer();
2  serializer.setFormatted(true);
3
4  Path = Paths.get("user.surf");
5  try (final OutputStream outputStream = newOutputStream(path
6    serializer.serialize(outputStream, user);
7  }
```

SURF Use Cases

Apache Commons Configuration

`SurfConfiguration`

https://bitbucket.org/globalmentor/urf-apache-commons-configuration2-surf

Maven

`io.urf:apache-commons-configuration2-surf`

📌 `SurfConfiguration` uses the SURF Reference Implementation.

`SurfConfiguration` Dependency

pom.xml

```
<project>
  …
  <dependencies>
    …
    <dependency>
      <groupId>io.urf</groupId>
      <artifactId>apache-commons-configuration2-surf</arti
      <version>x.x.x</version>
    </dependency>
  </dependencies>
</project>
```

`config.surf` Configuration Builder

```
1  import java.nio.file.*;
2  import org.apache.commons.configuration2.*;  //etc.
3  …
4  import io.urf.apache.commons.configuration2.surf.*;
5  …
6  Path configFile = Paths.get("config.surf");
7  BuilderParameters configBuilderParams = new Parameters()
8      .fileBased().setFile(configFile.toFile());
9  ConfigurationBuilder<SurfConfiguration> configurationBuilde
10     new FileBasedConfigurationBuilder<>(SurfConfiguration.c
11     .configure(configBuilderParams);
```

Access Configuration

```
1  Configuration config = configurationBuilder.getConfiguratio
2
3  LocalDate lastLogin = config.get(LocalDate.class, "lastLogi
4  System.out.println(String.format("Last login %d days ago.",
5      DAYS.between(lastLogin, LocalDate.now())));
```

Next SURF Implementations

Languages

- JavaScript
- Python

Tools

- Atom Editor
- Prism.js

More Information on SURF

http://urf.io/surf/

info@globalmentor.com