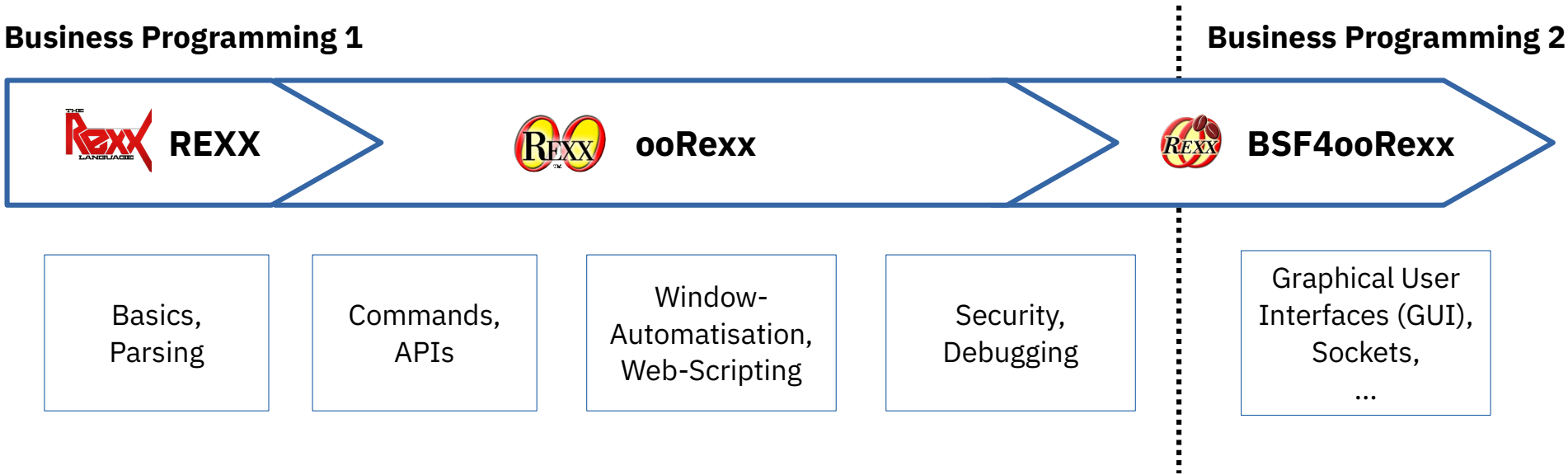
JSON (JavaScript Object Notation) with json.cls (Introduced with ooRexx 5.0)



Overview

- Introduction to **JSON**
- Introduce and demonstrate ooRexx' **json.cls**
  - First distributed in ooRexx 5.0.0
- Introduce and demonstrate BSF4ooRexx850' compatible **json-rgf.cls**
  - Updated **json.cls** with ooRexx 5.1.0beta (as of 2023-02-09)
- Roundup

JSON Encoded Data from Wikipedia

- Defined in the beginning of the 2000 to ease exchange of structured data via the Internet with JavaScript
- **JSON**
  - Acronym for "JavaScript Object Notation"
  - Datatypes
    - "Object" – a collection of comma separated name-value pairs (a Map) in curly brackets
    - "Array" – an ordered list of comma separated values in square brackets
    - "String" – a sequence of UTF-8 characters
    - "Boolean" – true or false
    - "Number" – any number
    - null (void) – null
  - cf. https://www.json.org/, https://en.wikipedia.org/wiki/JSON

Example of JSON Encoded Data

- Cf. Wikipedia https://en.wikipedia.org/wiki/JSON (2023-01-29)

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [
      "Catherine",
      "Thomas",
      "Trevor"
  ],
  "spouse": null
}
```

Unencode and encode JSON Data (1/4)

- Read JSON encoded data from file
  - Use Wikipedia's example
    - JSON encoded sample data is broken up into lines and indented for better legibility
    - Usually JSON encoded data is "minimized", i.e. does not contain ignorable whitespace meant for easying legibility and comprehensibility for humans
  - Result is an ooRexx *directory* object that contains all imported data
    - E.g. ontained JSON arrays are represented (stored) as ooRexx *array* objects
    - E.g. ontained JSON maps get represented (stored) as ooRexx *directory* objects
  - A dump routine will recursively iterate through all ooRexx data and display it
    - Sorting the names (keys) makes reading easier for humans
    - Indentation makes the nesting visible and easier to comprehend for humans
  - At the end the ooRexx *directory* object will be used to get a JSON rendering of it

Unencode and encode JSON Data (2/4)

```
parse arg fn                -- get file name
s = .stream~new(fn)~~open("read")
jsonData = s~charIn(1,s~chars)   -- read entire file
s~close
j = .json~new               -- create a JSON instance
d =  j~fromJson(jsonData)   -- let it unencode the data
call dumpJsonData d         -- show data
say
say "---"
say j~toJson(d)             -- turn into JSON

::requires "json.cls"       -- get access to the JSON class

… continued on the right
```

```
… continued from the lieft

::routine dumpJsonData      -- dump the ooRexx collection
   use arg o, level=0, suffix=""
   if o~isA(.collection) then
   do
      items = o~items
      isMap = o~isA(.mapCollection)
      say "  "~copies(level) || isMap~?("{","[")
      indent="  "~copies(level+1)
      do counter c idx over o~allIndexes~sort
         suffix = (c=items)~?("",",")
         v=o[idx]
         if v~isA(.collection) then
         do
             if isMap then say indent || pp(idx)":"
             call dumpJsonData v,level+1, suffix
             say suffix
         end
         else say indent || isMap~?(pp(idx)"=","") || pp(v) || suffix
      end
      .output~charOut("  "~copies(level) || isMap~?("}","]") || suffix)
   end

::routine pp                -- enclose value in brackets
  if arg(1)~isNil then return "null"
  return  '"' || arg(1)~changeStr('"','\"') || '"'
```

Unencode and encode JSON Data (3/4)

rexx json_01.rxj wikipedia.json

Wikipedia.json

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [
      "Catherine",
      "Thomas",
      "Trevor"
  ],
  "spouse": null
}
```

Output:

```
{
  "address":
  {
    "city"="New York",
    "postalCode"="10021-3100",
    "state"="NY",
    "streetAddress"="21 2nd Street"
  },
  "age"="27",
  "children":
  [
    "Catherine",
    "Thomas",
    "Trevor"
  ],
  "firstName"="John",
  "isAlive"="1",
  "lastName"="Smith",
  "phoneNumbers":
  [
    {
      "number"="212 555-1234",
      "type"="home"
    },
    {
      "number"="646 555-4567",
      "type"="office"
    }
  ],
  "spouse"=null
}
```

```
---
{"age":27,"spouse":null,"address":{"city":"New
York","state":"NY","streetAddress":"21 2nd
Street","postalCode":"10021-3100"},"children":
["Catherine","Thomas","Trevor"],"firstName":"J
ohn","isAlive":1,"lastName":"Smith","phoneNumb
ers":[{"type":"home","number":"212 555-1234"},
{"type":"office","number":"646 555-4567"}]}
```

Unencode and encode JSON Data (4/4)

- Some observations on the ooRexx 5.0.0 version
  - Method toJson will always produce minimized JSON, i.e. without ignorable whitespace
  - Handling of JSON boolean values is not supported when creating JSON data
    - Cf. name "isAlive" with its value true in the Wikipedia JSON example
    - In ooRexx the Boolean true value is represented with 1 and therefore the JSON encoding uses the *number* 1 instead of the JSON true encoding
    - When correct support of JSON truth values is needed, one can use either the version of ooRexx 5.1.0 (beta or later) or BSF4ooRexx850' json-rgf.cls (from samples/JavaFX/fxml_99) instead

Using json-rgf.cls

- Part of the JavaFX samples in BSF4ooRexx850, updated to ooRexx 5.1 json.cls
  - Get from "BSF4ooRexx850/samples/JavaFX/fxml_99"
    - E.g. use the BSF4ooRexx850 menu and choose "samples → JavaFX → fxml_99"
- json-rgf.cls is compatible with ooRexx 5.0' json.cls
  - Simply replace "`::requires json.cls`" with "`::requires json-rgf.cls`"
- To use JSON Boolean values in ooRexx fetch .json~true or .json~false
  - These values can be used wherever ooRexx' .true and .false can be used
  - When creating a JSON encoding these values will create the proper JSON values true or false
- To make JSON legible (human readable) supply .true as an additional argument to
  - Method toJSON or method toJSONFile
- It is able to correctly process all ooRexx collections in addition to .Directory and .Array in the structure that gets JSON encoded

Unencode and encode JSON Data (1/3)

```
parse arg fn                -- get file name
s = .stream~new(fn)~~open("read")
jsonData = s~charIn(1,s~chars)   -- read entire file
s~close
j = .json~new               -- create a JSON instance
o =  j~fromJson(jsonData)   -- let it unencode the data
say j~toJson(o,.true)       -- encode as legible JSON (human-friendly)
say "---"
say j~toJson(o)             -- encode as minimized JSON (standard)

::requires "json-rgf.cls"   -- get access to the JSON class
```

Unencode and encode JSON Data (2/3)

rexx json_02.rxj wikipedia.json

Wikipedia.json

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [
      "Catherine",
      "Thomas",
      "Trevor"
  ],
  "spouse": null
}
```

Output:

```
{
    "address": {
        "city": "New York",
        "postalCode": "10021-3100",
        "state": "NY",
        "streetAddress": "21 2nd Street"
    },
    "age": 27,
    "children": [
        "Catherine",
        "Thomas",
        "Trevor"
    ],
    "firstName": "John",
    "isAlive": true,
    "lastName": "Smith",
    "phoneNumbers": [
        {
            "number": "212 555-1234",
            "type": "home"
        },
        {
            "number": "646 555-4567",
            "type": "office"
        }
    ],
    "spouse": null
}
```

```
---
{"address":{"city":"New
York","postalCode":"10021-
3100","state":"NY","streetAddress":"21 2nd
Street"},"age":27,"children":
["Catherine","Thomas","Trevor"],"firstName":"J
ohn","isAlive":true,"lastName":"Smith","phoneN
umbers":[{"number":"212 555-
1234","type":"home"},{"number":"646 555-
4567","type":"office"}],"spouse":null}
```

Unencode and encode JSON Data (3/3)

- Some observations
  - json-rgf.cls allows for round-trip decoding and encoding
    - JSON booleans get handled correctly
  - The legible argument makes it easy to have the toJSON method create a legible (human readable if optional legible argument is set to .true) or a minimized JSON encoding
  - The map's names (keys) are sorted alphabetically to ease analyzing the encoded data for humans

Creating a JSON Encoding from ooRexx Data (1/3)

- The toJSON method takes an ooRexx object to encode
- The example creates an ooRexx structure using an ooRexx relation object and an ooRexx array and demonstrates how to encode that data
  - As an encoding to JSON is the purpose of this exercise a JSON Boolean value gets employed for demonstration purposes
  - The toJSON method creates minimized encodings by default
  - An instance of the JSON class from json-rgf.cls encodes for humans if toJSON's optional legible argument got set to .true

Creating a JSON Encoding from ooRexx Data (2/3)

```
rel = .relation~new          -- create a relation (allows duplicates)
rel["WU"]="Vienna Business University"
rel["Wien"]= ("Vienna", "Vienne")    -- English, French
rel["historical districts"] = .list~of(1190, 1090, 1020)
rel["currently in district"] = 1020
rel["is older than Harvard"] = .json~false    -- a JSON false value

j = .json~new                -- create a JSON instance
say j~toJson(rel)            -- encode as minimized JSON (standard)
say "---"
say j~toJson(rel,.true)      -- encode as legible JSON (human-friendly)

::requires "json-rgf.cls"  -- get access to the JSON class
```

```
{"WU":"Vienna Business University","Wien":
["Vienna","Vienne"],"currently in
district":1020,"historical districts":
[1190,1090,1020],"is older than
Harvard":false}
---
{
   "WU": "Vienna Business University",
   "Wien": [
      "Vienna",
      "Vienne"
   ],
   "currently in district": 1020,
   "historical districts": [
      1190,
      1090,
      1020
   ],
   "is older than Harvard": false
}
```

Creating a JSON Encoding from ooRexx Data (3/3)

- Some observations
  - json-rgf.cls
    - JSON booleans get handled correctly
    - The legible argument makes it easy to control whether the toJSON method should create a minimized (default) or a human readable (if the argument legible is set to .true) JSON encoding
    - Any ooRexx *MapCollection* object can be used (a relation object in the example) and will be encoded as a JSON Object collection (a Map)
    - Any ooRexx *OrderedCollection* can be used (a list object in the example) and will be encoded as a JSON array

Roundup

- ooRexx 5.0 introduced the Rexx package json.cls
  - Implements an ooRexx class named JSON
    - The fromJSON method allows for turning JSON encoded string data into an ooRexx structure (collection)
    - The toJSON method allows for encoding any ooRexx structure (collection) into a JSON string
- BSF4ooRexx(850)' "json-rgf.cls" is compatible to "json.cls"
  - Get from BSF4ooRexx(850)' "samples/JavaFX/fxml_99"
  - Adds support for JSON Boolean values
  - The attribute legible, if set to .true will encode JSON in a human friendly form
  - Adds utility class methods to directly read (fromJsonFile(fileName)) from or write to files (toJsonFile(fileName, rexxObject, isLegible=.true))