Disclaimer

This document is for informational purposes only and is subject to change at any time without notice. The information in this document is proprietary to Actian and no part of this document may be reproduced, copied, or transmitted in any form or for any purpose without the express prior written permission of Actian.

This document is not intended to be binding upon Actian to any particular course of business, pricing, product strategy, and/or development. Actian assumes no responsibility for errors or omissions in this document. Actian shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials. Actian does not warrant the accuracy or completeness of the information, text, graphics, links, or other items contained within this material. This document is provided without a warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, or non-infringement.
OpenROAD
Current Status and Futures

Bodo Bergmann
Principal Software Engineer
• Current Status
• Futures
• Q & A
Current Status

- OpenROAD Product Availability
- New Features since general availability
OpenROAD Product Availability

**OpenROAD 6.2**

- General Availability with Enterprise Support on all supported platforms:
  - Windows 32-bit
    - Runs on both 32-bit and 64-bit Windows
  - Windows 64-bit
  - Linux
  - AIX
  - Solaris
- Loadnrun
  - Client available for Windows 32-bit and Windows 64-bit
  - Server runs on all supported OpenROAD Server platforms

**Older OpenROAD Releases**

- Out of Enterprise support
- Extended Support or Obsolescence Support available

See [http://supportservices.actian.com/support-services/support](http://supportservices.actian.com/support-services/support) for more information.
New Features since general availability

Many new features have been added since GA:

▪ Features developed during or started at Code Sprints
▪ Features requested by customers
▪ Featured required for future development
▪ Features to improve usability and performance
▪ Features to improve tracing capabilities

Session OpenROAD New Features Since OR 6.2
Futures

- JSON support
  - General JSON support for OpenROAD
  - JSON-RPC support for OpenROAD

- Other new features
  - Web enablement using JSON-RPC
  - Customer requests
  - Code Sprints
  - As needed
JSON Support

- **JSON = JavaScript Object Notation**
  - compact, text-based data format for data exchange
  - Data objects consisting of attribute-value pairs
  - Language-independent
    - Directly usable in JavaScript
    - JSON support for many other languages (e.g. Java, C#, Python, etc.)
  - Most common data format used for browser/server communication
- **http://json.org**

```json
{
  "aint": 1,
  "afloat": 123.456,
  "astring": "Abc\tDef\nGhi\u20ac",
  "isbig": true,
  "isold": false,
  "obj1": null,
  "obj2": {},
  "obj2": {"attr1": "x"},
  "arr1": [123, true, null, "Blabla", 5.6e10, [], {"o":false}],
  "arr2": [],
  "arr3": null
}
```
Generic JSON Support for OpenROAD

New JsonHandler class

- Parse JSON from StringObject or file intoJsonValue subclass objects
- Write JSON from JsonValue subclass objects into StringObject or file

New classes to represent JSON types
- JsonValue and subclasses

New JsonHandler class
- Provide conversion between JsonObject and OpenROAD objects (both directions)
### JsonValue ↔ OpenROAD Type Mappings

<table>
<thead>
<tr>
<th>JsonValue subclass</th>
<th>OpenROAD types</th>
</tr>
</thead>
<tbody>
<tr>
<td>JsonNumber</td>
<td>float (any precision), integer (any size), decimal, money</td>
</tr>
<tr>
<td>JsonString</td>
<td>varchar, nvarchar, date</td>
</tr>
<tr>
<td></td>
<td>In JsonString absolute dates are specified in ISO8601 format: &quot;yyyy-mm-ddThh:mi:ssZ&quot;</td>
</tr>
<tr>
<td>JsonBoolean</td>
<td>any numeric type (values: true=1, false=0). There is no boolean type in OpenROAD (yet).</td>
</tr>
<tr>
<td>JsonNull</td>
<td>NULL (for both nullable scalar types and references)</td>
</tr>
<tr>
<td>JsonObject</td>
<td>Object</td>
</tr>
<tr>
<td>JsonArray</td>
<td>Rows of an ArrayObject. The items of the JsonArray must be JsonObject instances.</td>
</tr>
</tbody>
</table>
Handling Multiple and Circular References

- Object referenced multiple times gets "$id" member in addition to members for the attributes
- References to same object: Object with the "$ref" member only

OpenROAD 4GL code:

```
// with circular reference
person1.name = 'Joseph';
person2.name = 'Mary';
person1.spouse = person2;
person2.spouse = person1;
```

JSON representation:

```
{
    "$id": "1",
    "name": "Joseph",
    "spouse": {
        "name": "Mary",
        "spouse": { "$ref": "1" }
    }
}
```
JSON-RPC Support

- JSON-RPC is a JSON based Remote Procedure Call protocol standard
  - http://www.jsonrpc.org/specification

Request:
{"jsonrpc": "2.0", "method": "subtract", "params":{"subtrahend": 23, "minuend": 42}, "id": 3}

Response - result:
{"jsonrpc": "2.0", "result": 19, "id": 3}

Request:
{"jsonrpc": "2.0", "method": "foobar", "id": 1}  

Response - error:
{"jsonrpc": "2.0", "error": {"code": -32601, "message": "Method not found"}, "id": 1}
JSON-RPC Support for OpenROAD

- JSON-RPC 2.0 interface for calling 4GL procedures contained in OpenROAD Server applications
  - The "params" member within an JSON-RPC request has to be a JSON object containing members for the named parameters
    - Reason: Parameters to 4GL procedures are always passed by name (never by position)

- Makes the OpenROAD procedures available to many client applications/architectures
  - Via HTTP(S)
  - Can even use "curl" command line utility

- Removes existing limitations for OpenROAD server procedure calls (Call4GL/Callproc methods)
Calling OpenROAD 4GL Procedures from the "outside world"

JavaScript

C#

...

curl

HTTP POST request w/ JSON data

HttpServer w/ OpenROADJSONRPC servlet

OpenROAD Server

OpenROAD Server applications

C:\temp> curl -d "{"jsonrpc":"2.0","id":1,"method":"subtract","params":{"subtrahend":23.4,"minuend":42.8},}"
--header "Content-Type: application/json"
http://somehost:8080/orjsonrpc/myapp

{"result":19.4, "id":1, "jsonrpc":"2.0"}

**procedure** subtract(
  minuend = FLOAT NOT NULL,
  subtrahend = FLOAT NOT NULL
) =

```cpp
{
  return minuend-subtrahend;
}
```
JSON-RPC New Features

▪ Subclass objects/arrays can be passed to parameters of a defined class/array
  – A voids SCP proliferation
▪ Objects and arrays of system classes can be passed
▪ Language independent
  – Like the existing XML interface, but without its limitations
▪ 4GL procedures can have a return value
  – Will be represented in the "result" of the response message
  – No need to wrap procedure by an SCP with byref parameter
▪ Different ways to return values to the caller
  – Return value and/or ByRef parameters
JSON-RPC New Features (cont.)

▪ Multiple and circular references are preserved in objects passed as parameters and returned.
  – Represented using "$id" and "$ref" meta properties.

▪ A batch of requests is possible to be passed to the server with one call
  – JSON array containing request objects
  – Response will contain an array of response objects

▪ New RemoteServer class method to serve all server procedure calls:
  status = JsonRpcRequest(request, response);
  – StringObject (BSTR, string) parameters
    • response is ByRef
  – The actual procedure to be executed will be passed in the "method" parameter of the "request" string.

▪ Names of 4GL Procedures available for JSON-RPC requests and their parameters are configured via file
  – Use of BYREF for a parameter can also be determined
JSON-RPC config File

- Filename *ApplicationName*.json
- Location specified by environment variable `II_W4GL_JSON_CONFIG`
  - Default: "orjsonconfig" subdirectory of `%II_CONFIG%\%II_SYSTEM%\inges\files"

```json
{
  "registeredprocs" : {
    "rows" : [
      {
        "name" : "subtract", 
        "params" : {
          "rows" : [
            {
              "name" : "minuend"},
            {
              "name" : "subtrahend"}
          ]
        }
      },
      {
        "name" : "helloworld", 
        "params" : {
          "rows" : [
            {
              "name" : "hellostring", 
              "byref_use" : true},
            {
              "name" : "counter", 
              "byref_use" : true
            }
          ]
        }
      }
    ]
  }
}
```
JSON-RPC Support for OpenROAD - Examples

- Using batch of requests

Request:

```
[  
  {"jsonrpc": "2.0", "method": "subtract", "params": {"subtrahend": 23, "minuend": 42}, "id": 1},  
  {"jsonrpc": "2.0", "method": "xyz"},  
  {"jsonrpc": "2.0", "method": "dummy", "id": 2},  
  {"jsonrpc": "2.0", "method": "subtract", "params": {"minuend": 3, "subtrahend": 1}, "id": 3}  
]
```

Response:

```
[  
  {"jsonrpc": "2.0", "result": 19, "id": 1},  
  {"jsonrpc": "2.0", "error": {"code": -32601, "message": "Method not found"}, "id": 2},  
  {"jsonrpc": "2.0", "result": 2, "id": 3}  
]```
JSON-RPC Support for OpenROAD - Examples

- Using BYREF parameters - defined by pseudo-parameter "$byref_params"

```plaintext
// 4GL Procedure in OpenROAD Server application:
procedure helloworld(hellostring = VARCHAR(100) NOT NULL, counter = INTEGER NOT NULL)={...}
```

**JSON-RPC request:**

```json
{
   "jsonrpc": "2.0",
   "method": "helloworld",
   "params": {
      "hellostring": "HELLO",
      "counter": 0,
      "$byref_params": "hellostring,counter"
   },
   "id": 2
}
```

**JSON-RPC response:**

```json
{
   "result": {
      "result": null,
      "byref_results": {
         "hellostring": "Well \"HELLO\" to you too.",
         "counter": 1
      }
   },
   "id": 2,
   "jsonrpc": "2.0"
}
```
Q & A
Thank you!