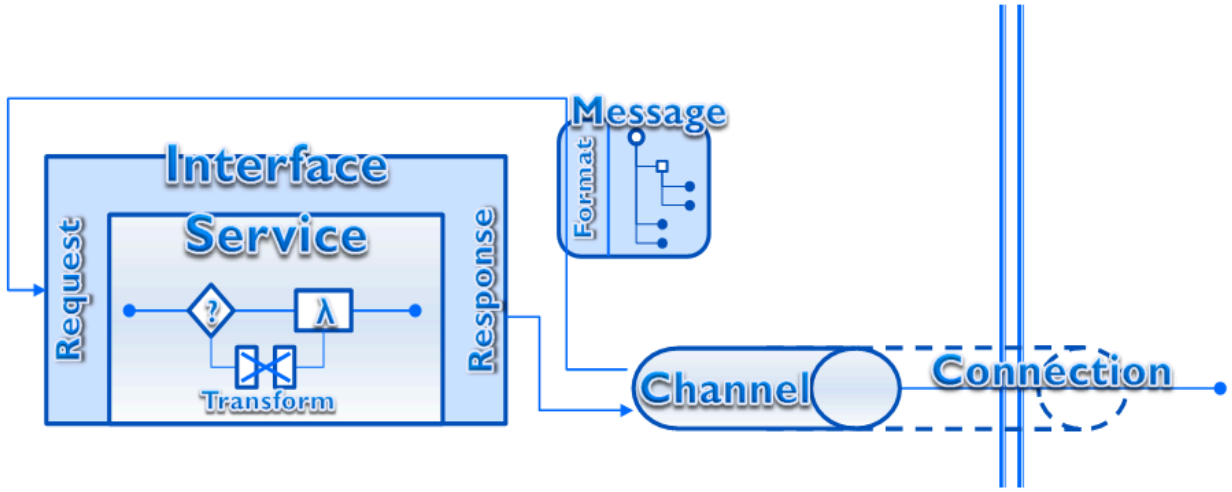
JSON Message Adapter

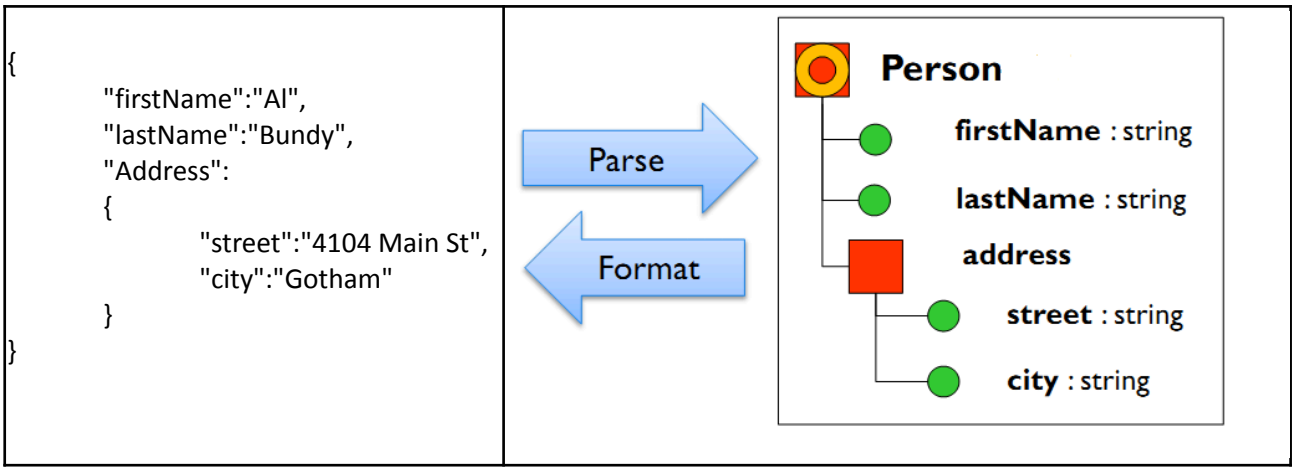
Overview

Enterprise-level integration requires one system to talk to another over networks with portable formats. The JSON Message Adapter will produce and consume JSON formatted messages.

NexJ Express Integration

The following diagrams show the relationship and significance of channels, message formats, interfaces and services.
When the channel receives a message, the JSON will be parsed into a canonical NexJ message, known as a TransferObject . A service may process the canonical message, possibly transforming it to another message definition. If so configured, the message may be sent back through the interface as a response to the same channel and same connection.





Design Philosophy

1. JSON format should be clean with the preference that no meta-information is included. (For example, where there are no circular references, meta-information for circular references should be omitted)

2. Should support circular references for TransferObject's. (refer to nexj.core.rpc.json)

3. Should make use of existing JSON classes

4. To achieve better performance, anything that can be pre-computed, should be pre-computed and stored in the message part mapping

5. Should support message inheritance for basic message reuse

6. If possible, should support polymorphic messages

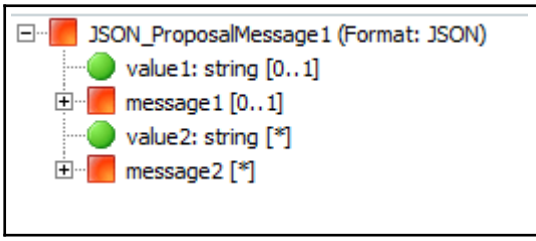
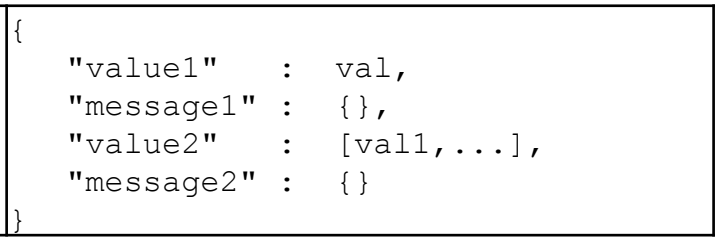
Technical

The following are the minimum number of classes to be created.

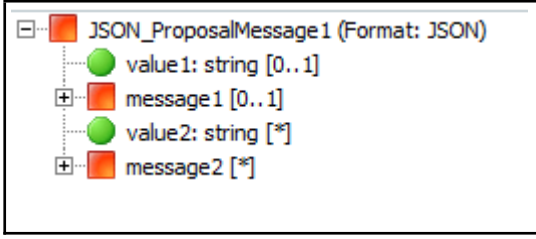
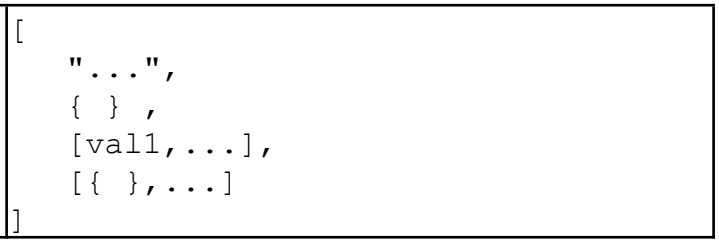
JSONMessageFormatter	Formatter for JSON messages. Responsible for taking a TransferObject containing the data and writing it to JSON format to the Output, as configured by the Message
JSONMessageParser	Parser for JSON messages. Responsible for taking a JSON stream and converting it into NexJ format (Transfer objects)
JSONMessagePartMapping	Used to configure JSON-specific characteristics of the message mapping. For example, may allow a node defined in the .message file to have a name such as "firstName", but the attribute in the JSON data may be named "fname".
XMLJSONMessageMappingLoader	Responsible for creating mappings for JSON message parts.

JSON Integration Encoding Rules

Message as Object (Default)

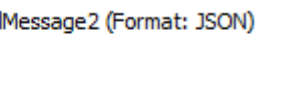
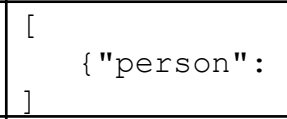
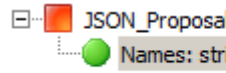
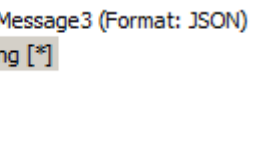
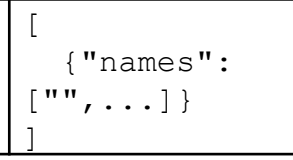
Message as Array

Message as "Typed" Array

The encodings mentioned above are limited because they cannot be use to produce "Typed Arrays" i.e. array of same Object or array of same Primitive.

To solve this we propose a "Typed Array" option, which will only be available for messages with one node under the root.

	Root as Object	Root as Typed Array
		
		

Implementation details

- Does not support channels with multiple messages.
- For message parts of type "ANY",
  - timestamp has limited supported because JSON has no way of differentiating between a number and a timestamp .
  - binary has limited supported because JSON has no way of differentiating between a regular string and a base 64 encodes string.
- Throws integration exception if required part is null.

Tentative Schedule

May 20th Proposal submitted to NexJ for review

June	Writing Code
Late July	First code review/code submission