### JSON object format

### Basics

fjåge's JSON protocol objects are typically shallow JSON objects. The first level of attributes are typically used by the containers to hold metadata and perform housekeeping tasks such as agent directory service. The attribute message in the JSON object contains the actual message that is exchanged between agents residing in different containers. We describe the JSON message format below which when sent to UnetStack (which runs on fjåge) will execute the task requested and respond with relevant notification JSON message, which the developer must look for and parse carefully. Next, we describe in detail the JSON message format which fjåge understands.

### JSON Message request/response attributes

A JsonMessage class is defined in [fjåge](http://org-arl.github.io/fjage/doc/html/index.html) which support a list of attributes. The attributes supported at the top level of the JSON object are listed below:

* id : **String** - A [UUID](https://en.wikipedia.org/wiki/Universally_unique_identifier) assigned to to each object.
* action : **String** - Denotes the main action the object is supposed to perform. Valid JSON message actions supported are listed here:
  + agents - Request for a list of all agents running on the target container.
  + containsAgent - Request to check if a specific container has a agent with a given AgentID running.
  + services - Request for a list of all services running on the target container.
  + agentForService - Request for AgentID of an agent that is providing a specific service.
  + agentsForService - Request for AgentID of all agents that is providing a specific service.
  + send - Request to send a payload to the target container.
  + shutdown - Request to shutdown the target container.
* inResponseTo : **String** - This attribute contains the action to which this object is a response to. A response object will have the exact same id as the original action object.
* agentID : **String** - An AgentID. This attribute is populated in objects which are responses to objects requesting the ID of an agent providing a specific service "action" : "agentForService". This field may also be used in objects with "action" : "containsAgent" to check if an agent with the given AgentID is running on a target container.
* agentIDs: **Array** - This attribute is populated in objects which are responses to objects requesting the IDs of agents providing a specific service with "action" : "agentsForService", or objects which are responses to objects requesting a list of all agents running in a container.
* service : **String** - Used in conjunction with "action" : "agentForService" and "action" : "agentsForService" to query for agent(s) providing this specific service.
* services: **Array** - This attribute is populated in objects which are responses to objects requesting the services available with "action" : "services".
* answer : **Boolean** - This attribute is populated in objects which are responses to query objects with "action" : "containsAgent".
* relay : **Boolean** - This attribute defines if the target container should relay (forward) the message to other containers it is connected to or not.
* message : **Object** - This holds two main attributes and is responsible for carrying the main payload. The first field is clazz and the second data. Note that the ordering of clazz and data fields is crucial. The developer must make sure that the clazz field comes ahead of data field. The structure and format of this object is discussed here:
  + clazz : **String** - A string identifier that identifies the type of the message. This is usually a fully qualified Java class name of the class of that type of message.
  + data : **Object** - The main payload containing data and message attributes. This holds the contents of the payload in objects with "action" : "send". The structure and format of this Object is discussed here :
    - data : **Object** - The main payload containing the data and type of data. **NOTE**: While receiving the JSON messages from UnetStack running on modem, this attribute will follow the structure as describe below. However, when the developer sends a JSON message, the developer can choose to wither follow this format as converting the data to Base64 and specifying the equivalent clazz or the data can be directly inserted as an array of numbers without specifying the clazz or data fields as explained later in the examples section.
      * clazz : **String** - This attribute contains the string to identify the type of data being carried by the JSON object. The types that are identified and supported are listed here:
        + "[F" - Float
        + "[I" - Integer
        + "[D" - Double
        + "[J" - Long
        + "[B" - Bytestring
      * data or signal: **Base64 String** - The data is encoded as a Base64 string and populated in this attribute. Either the data or signal attribute is used depending on the message that is being received or sent.
  + msgID : **String** - A [UUID](https://en.wikipedia.org/wiki/Universally_unique_identifier) assigned to to each message.
  + perf : **String** - Performative. Defines the purpose of the message. Valid performatives are :
    - REQUEST - Request an action to be performed.
    - AGREE - Agree to performing the requested action.
    - REFUSE - Refuse to perform the requested action.
    - FAILURE - Notification of failure to perform a requested or agreed action.
    - INFORM - Notification of an event.
  + recipient : **String** - An AgentID of the UnetStack/fjåge agent this message is being addressed to.
  + sender : **String** - An AgentID on the UnetStack/fjåge this message is being sent by.
  + inReplyTo : **String** - A UUID. Included in a reply to another object to indicate that this object is a reply to a object with this id.

Note that not all the above attributes need to be populated in a JSON message. The attributes depend on the task that needs to be executed on the modem using UnetStack software. Also, the message attribute may have additional attributes depending on the exact message that is being constructed. The details of which are available online as part of the [UnetStack](http://www.unetstack.net/javadoc/index.html) API documentation. Next, we describe some of the basic examples in order to let the developer understand what JSON messages to send and how to construct them for different use cases.

### Examples

### JSON Message to transmit a CONTROL frame

Equivalent JSON message:

|  |
| --- |
| {  "action": "send",  "message": {  "clazz": "org.arl.unet.phy.TxFrameReq",  "data": {  "type": 1,  "data": [  1,  2,  3  ],  "msgID": "a2fbff38-a0fb-4e3a-bf22-ae6cf4642e6b",  "perf": "REQUEST",  "recipient": "phy",  "sender": "MyCustomInterface"  }  }  } |

Note that the equivalent JSON message of a TxFrameReq message to transmit a CONTROL frame is as shown above. The message attribute contains the attributes specific to TxFrameReq message as described in the earlier section and can also be found online at UnetStack API documentation.

A JSON message sent by UnetStack running on modem in response to the JSON message sent is as given below:

|  |
| --- |
| {  "action": "send",  "message": {  "clazz": "org.arl.unet.phy.TxFrameNtf",  "data": {  "txTime": 3329986666,  "type": 1,  "msgID": "dc227a96-4d6e-4b64-9d55-bb108ea338b0",  "perf": "INFORM",  "recipient": "MyCustomInterface",  "sender": "phy",  "inReplyTo": "a2fbff38-a0fb-4e3a-bf22-ae6cf4642e6b"  }  },  "relay": false  } |

Note that there is an attribute inReplyTo populated in the response received which indicates that this JSON message was in reply to the JSON message with exact same msgID.

### JSON Message to transmit a DATA frame

Equivalent JSON message:

|  |
| --- |
| {  "action": "send",  "message": {  "clazz": "org.arl.unet.phy.TxFrameReq",  "data": {  "type": 2,  "data": [  1,  2,  3  ],  "msgID": "a2fbff38-a0fb-4e3a-bf22-ae6cf4642e6b",  "perf": "REQUEST",  "recipient": "phy",  "sender": "MyCustomInterface"  }  }  } |

Note that the equivalent JSON message of a TxFrameReq message to transmit a DATA frame is as shown above. The only thing that changes in the JSON Message is the type attribute whose value is 2 for DATA frame and 1 for CONTROL frame. This information can again be found online at the UnetStack API documentation. The response to this JSON message is again a equivalent JSON message of TxFrameNtf type as shown in the above example and hence we will not repeat it here.

### JSON Message to transmit a datagram

Now, let us try to send a string Hello World! as a bytestring as explained in the previous section.

Equivalent JSON message:

|  |
| --- |
| {  "action": "send",  "message": {  "clazz": "org.arl.unet.DatagramReq",  "data": {  "data": {  "clazz": "[B",  "data": "aGVsbG8gd29ybGQh"  },  "msgID": "8152310b-155d-4303-9621-c610e036b373",  "perf": "REQUEST",  "recipient": "phy",  "sender": "MyCustomInterface"  }  }  } |

Equivalent JSON message sent by UnetStack running on modem:

|  |
| --- |
| {  "action": "send",  "message": {  "clazz": "org.arl.unet.phy.TxFrameNtf",  "data": {  "txTime": 4550354666,  "type": 1,  "msgID": "fde91abf-68ac-4a93-b2ae-27d1cee01869",  "perf": "INFORM",  "recipient": "MyCustomInterface",  "sender": "phy",  "inReplyTo": "8152310b-155d-4303-9621-c610e036b373"  }  },  "relay": false  } |

### JSON Message to record a baseband signal

In order to record a baseband signal with 100 baseband samples the equivalent JSON message is following:

|  |
| --- |
| {  "action": "send",  "message": {  "clazz": "org.arl.unet.bb.RecordBasebandSignalReq",  "data": {  "recLen": 100,  "msgID": "28db3bd4-ad14-4d86-b4a0-a2d8ebb3cb65",  "perf": "REQUEST",  "recipient": "phy",  "sender": "MyCustomInterface"  }  }  } |

The specific attribute such as recLen is message specific which in this case is RecordBasebandSignalReq and the relevant supported attributes can be found online at the [UnetStack](http://www.unetstack.net/javadoc/index.html) API documentation.

Now, in response to the the JSON message to record baseband samples, the UnetStack sends a JSON message equivalent to the RxBasebandSignalNtf message containing the recorded data and is as shown here:

Equivalent JSON message sent by UnetStack running on modem:

|  |
| --- |
| {  "action": "send",  "message": {  "clazz": "org.arl.unet.bb.RxBasebandSignalNtf",  "data": {  "rxTime": 4905996833,  "rssi": -43.190178,  "adc": 1,  "signal": {  "clazz": "[F",  "data": ""  },  "fc": 12000,  "fs": 12000,  "channels": 1,  "preamble": 0,  "msgID": "7720595f-3512-4f12-8168-6b55da613766",  "perf": "INFORM",  "recipient": "MyCustomInterface",  "sender": "phy",  "inReplyTo": "28db3bd4-ad14-4d86-b4a0-a2d8ebb3cb65"  }  },  "relay": false  } |

Again it can be observed from the inReplyTo attribute that the above JSON message is in reply to the JSON message corresponding to the RecordBasebandSignalReq. Also note that this JSON message contains the data recorded as a base64 encoded string and the clazz attribute indicates that the actual values are floats. The developer/user can utilize this information and decode the recoded data from this data attribute to a usable format. The other attributes that are added to the JSON message in response can be found in the UnetStack online documentation for the RxbasebandSignalNtf message.

### JSON Message to transmit a signal

A signal can be transmitted using TxbasebandSignalReq message in one of the two ways:

* Without base64 encoding
* With base64 encoding

Equivalent JSON message without base64 encoded signal:

|  |
| --- |
| {  "action": "send",  "message": {  "clazz": "org.arl.unet.bb.TxBasebandSignalReq",  "data": {  "signal": [  1,  1,  1  ],  "preamble": 1,  "msgID": "24078a7f-0054-42c9-a578-99eb7f4c0c07",  "perf": "REQUEST",  "recipient": "phy",  "sender": "MyCustomInterface"  }  },  "relay": true  } |

Equivalent JSON message with base64 encoded signal:

|  |
| --- |
| {  "action": "send",  "message": {  "clazz": "org.arl.unet.bb.TxBasebandSignalReq",  "data": {  "signal": {  "clazz": "[F",  "data": "P4AAAD+AAAA/gAAA"  },  "preamble": 1,  "msgID": "7774ae54-cb34-44c5-b5d0-4de12e2afcba",  "perf": "REQUEST",  "recipient": "phy",  "sender": "MyCustomInterface"  }  }  } |

A TxFrameNtf is sent in response by UnetStack, the equivalent JSON message of which is as shown below:

|  |
| --- |
| {  "action": "send",  "message": {  "clazz": "org.arl.unet.phy.TxFrameNtf",  "data": {  "txTime": 6903128000,  "type": 0,  "msgID": "586fb281-8891-4308-8130-74563a8a7365",  "perf": "INFORM",  "recipient": "MyCustomInterface",  "sender": "phy",  "inReplyTo": "24078a7f-0054-42c9-a578-99eb7f4c0c07"  }  },  "relay": false  } |

The above response is shown when the signal is transmitted without the base64 encoding of the signal. The reader can compare the msgID and inReplyTo attributes of the corresponding message.

### JSON Message for measuring range to other node

Equivalent JSON message to request range:

|  |
| --- |
| {  "action": "send",  "message": {  "clazz": "org.arl.unet.phy.RangeReq",  "data": {  "to": 2,  "msgID": "b413a17f-fed6-4bba-ba17-e617a8e4648f",  "perf": "REQUEST",  "recipient": "ranging",  "sender": "MyCustomInterface"  }  }  } |

Equivalent JSON message sent by UnetStack running on modem is :

|  |
| --- |
| {  "action": "send",  "message": {  "clazz": "org.arl.unet.phy.RangeNtf",  "data": {  "from": 2,  "to": 1,  "range": 999.7,  "timeOffset": -1733855560,  "isValid": true,  "msgID": "37ebaa15-92f2-48b5-912b-70c7f29d83fb",  "perf": "INFORM",  "recipient": "#ranging\_\_ntf",  "sender": "ranging"  }  },  "relay": false  } |

which is a RangeNtf message in UneStack and the relevant attributes populated can be found in the online UnetStack API documentation.