rosbridge v2.0 Protocol Specification

This document outlines the rosbridge v2.0 protocol. The v2.0 protocol incorporates a number of requirements which have arisen since the first version of rosbridge was released, and makes a small number of modifications to facilitate greater extensibility to the protocol. At its core, the protocol still contains the same operations with the same semantics as the prior versions of rosbridge. The main change is to the structure of messages, separating control information from message information. The main new additions are fragmentation, compression, and logging.

This document outlines the protocol specification, but also touches upon the intended direction for the rosbridge server implementation. The rosbridge v2.0 server implementation is architected in a way to make it easy to add and modify protocol operations. Furthermore, the rosbridge v2.0 server decouples JSON-handling from the websockets server, allowing users to arbitrarily change the specific websockets server implementation they are using.

The message transport of rosbridge is JSON objects. The only required field is the 'op' field, which specifies the operation of that message. Each 'op' then specifies its own message semantics.

The rosbridge protocol is a set of 'op' codes which define a number of operations, along with the semantics for each operation.

The rosbridge server is a server which accepts websockets connections and implements the rosbridge protocol.

The full source code of rosbridge is located in the rosbridge_suite package. The package is located at https://github.com/robotwebtools/rosbridge_suite, and the full breakdown of the stack and its packages is detailed in section 4.5 of this document.

1. The rosbridge transport

A rosbridge message is, in the base case, a JSON object with a string field called "op". For example:

```
{ "op": "Example" }
```

The op field indicates the type of message that this is. Messages with different values for op may be handled differently.

So long as the message is a JSON object with the op field, it is a valid rosbridge message.

Optionally, a message can also provide an arbitrary string or integer ID:

```
{ "op": "Example",
  "id":"fred"
}
```

If an ID is provided with a message to the server, then related response messages will typically contain that ID as well. Log messages caused by this operation will also contain the ID.

Semantically, the ID is not an identifier of the specific message that it is in, but instead is an identifier for an interaction which may consist of a number of operations in back-and-forth messages. Thus, the ID may be used by multiple messages referring to the same transaction.

2. The rosbridge protocol

The rosbridge protocol defines a number of different operations. They are as follows:

Message compression / transformation:

- **fragment** - a part of a fragmented message
- **png** - a part of a PNG compressed fragmented message

Rosbridge status messages:

- **set_status_level** - a request to set the reporting level for rosbridge status messages
- **status** - a status message

Authentication:

- **auth** - a set of authentication credentials to authenticate a client connection

ROS operations:

- **advertise** – advertise that you are publishing a topic
- **unadvertise** – stop advertising that you are publishing topic
- **publish** - a published ROS-message
- **subscribe** - a request to subscribe to a topic
- **unsubscribe** - a request to unsubscribe from a topic
- **call_service** - a service call

- **service_response** - a service response

In general, actions or operations that the client takes (such as publishing and subscribing) have opcodes which are verbs (subscribe, call_service, unadvertise etc.).

Response messages from the server are things that the client is giving back, so they are nouns (fragment, status, service_response etc.)

(The only slight exception to this naming convention is publish)

3. Details of the rosbridge protocol

Following is the specification of operations in the rosbridge protocol, supported by the rosbridge server. Anything marked with [experimental] may be subject to change after review.

3.1 Data Encoding and Transformation

The rosbridge protocol provides the ability to fragment messages and to compress messages.

3.1.1 Fragmentation (*fragment*) [experimental]

Messages may be fragmented if they are particularly large, or if the client requests fragmentation. A fragmented message has the following format:

```
{ "op": "fragment",
  "id": <string>,
  "data": <string>,
  "num": <int>,
  "total": <int>
}
```

id - an id is required for fragmented messages, in order to identify corresponding fragments for the fragmented message:

- **data** - a fragment of data that, when combined with other fragments of data, makes up another message
- **num** - the index of the fragment in the message
- **total** - the total number of fragments

To fragment a message, its JSON string is taken and split up into multiple substrings. For each substring, a fragment message is constructed, with the data field of the fragment populated by the substring.

To reconstruct an original message, the data fields of the fragments are concatenated, resulting in the JSON string of the original message.

3.1.2 PNG compression (*png*) [experimental]

Some messages (such as point clouds) can be extremely large, and for efficiency reasons we may wish to transfer them as PNG-encoded bytes. The PNG opcode duplicates the fragmentation logic of the FRG opcode (and it is possible and reasonable to only have a single fragment), except that the data field consists of ASCII-encoded PNG bytes.

```
{ "op": "png",
  (optional) "id": <string>,
  "data": <string>,
  (optional) "num": <int>,
  (optional) "total": <int>
}
```

- **id** – only required if the message is fragmented. Identifies the fragments for the fragmented message.
- **data** – a fragment of a PNG-encoded message or an entire message.
- **num** – only required if the message is fragmented. The index of the fragment.
- **total** – only required if the message is fragmented. The total number of fragments.

To construct a PNG compressed message, take the JSON string of the original message and read the bytes of the string into a PNG image. Then, ASCII-encode the image. This string is now used as the data field. If fragmentation is necessary, then fragment the data and set the ID, num and total fields to the appropriate values in the fragments. Otherwise these fields can be left out.

3.2 Status messages

rosbridge sends status messages to the client relating to the successes and failures of rosbridge protocol commands. There are four status levels: info, warning, error, none. By default, rosbridge uses a status level of error.

A rough guide for what causes the levels of status message:

- **error** – Whenever a user sends a message that is invalid or requests something that does not exist (ie. Sending an incorrect opcode or publishing to a topic that doesn't exist)
- **warning** – error, plus, whenever a user does something that may succeed but the user has still done something incorrectly (ie. Providing a partially-complete published message)
- **info** – warning, plus messages indicating success of various operations

3.2.1 Set Status Level (*status_level*) [experimental]

```
{ "op": "set_level",
  (optional) "id": <string>,
  "level": <string>
}
```

- **level** – one of 'info', 'warning', 'error', or 'none'

Sets the status level to the level specified. If a bad string is specified, the message is dropped.

3.2.2 Status message (*status*) [experimental]

```
{ "op": "status",
  (optional) "id": <string>,
  "level": <string>,
  "msg": <string>
}
```

- **level** – the level of this status message
- **msg** – the string message being logged
- **id** – if the status message was the result of some operation that had an id, then that id is included

3.3 Authentication message

Optional authentication information can be passed via the rosbridge protocol to authenticate a client connection. This information should come from some trusted third-party authenticator.

Authentication is based on the MAC (message authentication code) scheme. The key to using MAC is that it does not tie users to a single "user database." It simply requires some trusted third-party to provide the hash-keys.

3.3.1 Authenticate (*auth*)

To send authentication credentials, use the auth command.

```
{ "op": "auth",
  "mac": <string>,
  "client": <string>,
  "dest": <string>,
  "rand": <string>,
  "t": <int>,
  "level": <string>,
  "end": <int>
}
```

- **mac** – MAC (hashed) string given by the client
- **client** – IP of the client
- **dest** – IP of the destination
- **rand** – random string given by the client
- **t** – time of the authorization request
- **level** – user level as a string given by the client
- **end** – end time of the client's session
    - Any server that enabled authentication should wait for this request to come in first before accepting any other `op` code from the client.
    - Once the request comes in, it would verify the information (in a ROS system, using `rosauth` ; however, the verification method is not tied to ROS).
    - If the authentication is good, the connection would be kept and rosbridge would function as normal. If the authentication is bad, the connection would be severed.
    - In the case that authentication is not enabled on the server, this `op` code can be ignored.

3.4 ROS messages

These rosbridge messages interact with ROS, and correspond roughly to the messages that already exist in the current version of rosbridge.

3.4.1 Advertise (*advertise*)

If you wish to advertise that you are or will be publishing a topic, then use the advertise command.

```
{ "op": "advertise",
  (optional) "id": <string>,
  "topic": <string>,
  "type": <string>
}
```

- **topic** – the string name of the topic to advertise
- **type** – the string type to advertise for the topic
    - If the topic does not already exist, and the type specified is a valid type, then the topic will be established with this type.
    - If the topic already exists with a different type, an error status message is sent and this message is dropped.
    - If the topic already exists with the same type, a warning status message is sent and this message is dropped.
    - If the topic doesnt already exist but the type cannot be resolved, then an error status message is sent and this message is dropped.

3.4.2 Unadvertise (*unadvertise*)

This stops advertising that you are publishing a topic.

```
{ "op": "unadvertise",
  (optional) "id": <string>,
  "topic": <string>
}
```

- **topic** – the string name of the topic being unadvertised
    - If the topic does not exist, a warning status message is sent and this message is dropped
    - If the topic exists but rosbridge is not advertising it, a warning status message is sent and this message is dropped

3.4.3 Publish (*publish*)

The publish message is used to send data on a topic.

```
{ "op": "publish",
  (optional) "id": <string>,
  "topic": <string>,
  "msg": <json>
}
```

The publish command publishes a message on a topic.

- **topic** - the string name of the topic to publish to
- **msg** - the message to publish on the topic
    - If the topic does not exist, then an error status message is sent and this message is dropped
    - If the msg does not conform to the type of the topic, then an error status message is sent and this message is dropped
    - If the msg is a subset of the type of the topic, then a warning status message is sent and the unspecified fields are filled in with defaults

Special case: if the type being published has a 'header' field, then the client can optionally omit the header from the msg. If this happens, rosbridge will automatically populate the header with a frame id of "" and the timestamp as the current time. Alternatively, just the timestamp field can be omitted, and then the current time will be automatically inserted.

3.4.4 Subscribe

```
{ "op": "subscribe",
  (optional) "id": <string>,
  "topic": <string>,
  (optional) "type": <string>,
  (optional) "throttle_rate": <int>,
```

```
  (optional) "queue_length": <int>,
  (optional) "fragment_size": <int>,
  (optional) "compression": <string>
}
```

This command subscribes the client to the specified topic. It is recommended that if the client has multiple components subscribing to the same topic, that each component makes its own subscription request providing an ID. That way, each can individually unsubscribe and rosbridge can select the correct rate at which to send messages.

- **type** – the (expected) type of the topic to subscribe to. If left off, type will be inferred, and if the topic doesn't exist then the command to subscribe will fail
- **topic** – the name of the topic to subscribe to
- **throttle_rate** – the minimum amount of time (in ms) that must elapse between messages being sent. Defaults to 0
- **queue_length** – the size of the queue to buffer messages. Messages are buffered as a result of the throttle_rate. Defaults to 1.
- **id** – if specified, then this specific subscription can be unsubscribed by referencing the ID.
- **fragment_size** – the maximum size that a message can take before it is to be fragmented.
- **compression** – an optional string to specify the compression scheme to be used on messages. Valid values are "none" and "png"

If queue_length is specified, then messages are placed into the queue before being sent. Messages are sent from the head of the queue. If the queue gets full, the oldest message is removed and replaced by the newest message.

If a client has multiple subscriptions to the same topic, then messages are sent at the lowest throttle_rate, with the lowest fragmentation size, and highest queue_length. It is recommended that the client provides IDs for its subscriptions, to enable rosbridge to effectively choose the appropriate fragmentation size and publishing rate.

3.4.5 Unsubscribe

```
{ "op": "unsubscribe",
  (optional) "id": <string>,
```

```
  "topic": <string>
}
```

- **topic** – the name of the topic to unsubscribe from
- **id** – an id of the subscription to unsubscribe

If an id is provided, then only the corresponding subscription is unsubscribed. If no ID is provided, then all subscriptions are unsubscribed.

3.4.6 Call Service

```
{ "op": "call_service",
  (optional) "id": <string>,
  "service": <string>,
  (optional) "args": <list<json>>,
  (optional) "fragment_size": <int>,
  (optional) "compression": <string>
}
```

Calls a ROS service

- **service** – the name of the service to call
- **args** – if the service has no args, then args does not have to be provided, though an empty list is equally acceptable. Args should be a list of json objects representing the arguments to the service
- **id** – an optional id to distinguish this service call
- **fragment_size** – the maximum size that the response message can take before it is fragmented
- **compression** – an optional string to specify the compression scheme to be used on messages. Valid values are "none" and "png"

3.4.7 Service Response

```
{ "op": "service_response",
  (optional) "id": <string>,
  "service": <string>,
  (optional) "values": <list<json>>
}
```

A response to a ROS service call

- **service** – the name of the service that was called
- **values** – the return values. If the service had no return values, then this field can be omitted (and will be by the rosbridge server)
- **id** – if an ID was provided to the service request, then the service response will contain the ID

4 Further considerations

Further considerations for the rosbridge protocol are listed below.

4.1 Rosbridge psuedo-services

Rosbridge no longer provides the ROS-api introspection pseudo services that it previously did. These are, for example rosbridge/topics and rosbridge/services. Instead, these services are provided as proper ROS services by the new rosapi package.

4.2 Sampling

It has been suggested that rosbridge may be extended to provide an operation to sample a single message from a topic.

4.3 Latching

Rosbridge will support messages that were latched to topics internally in ROS. It is possible that the publish opcode will be extended so that remote clients can latch messages too.

4.4 Advertising Services

Rosbridge currently does not support services advertised by the remote client. It could be extended to support this.

4.5 Rosbridge package structure

Rosbridge 2.0 resides in a package named rosbridge_suite, located at https://github.com/robotwebtools/rosbridge_suite.

The meta-package will contain the following packages:

- **rosbridge_library** – the core rosbridge JSON-to-ROS implementation. This is be a Python library.
- **rosbridge_server** – depends on the rosbridge library, and implements the WebSockets server, passing incoming messages to the API and outgoing messages back to the WebSockets connection. The default server uses tornado, a python server implementation.
- **rosapi** – provides ROS services for various master API calls, such as listing all the topics, services, types currently in ROS