Message Format Detail

Details as to Message Format

```
{
  "grouperHeader": {
    "version": "1", //Message format version
    "timestamp": "createtime of message",
    "sequence": "1", //message sequence number, integer, increments per-message
    "securityImplementation": "grouperMessageFormat:PlainMessageSecurity",
    "source": "someReturnHandle"
        "payloadFormat" : "grouperMessageFormat:esb"  /* or :scim or :aqmp etc */
  }
  "payload": {
    "eventType": "MEMBERSHIP_ADD",
    "fieldName": "members",
    "groupId": "4854cde794b34948911bfea5b2acb611",
    "groupName": "atlassian:jira:jira-users",
    "id": "e8497c14bd6346b1aacca3ac13a25246",
    "membershipType": "flattened",
    "sequenceNumber": "388",
    "sourceId": "jdbc",
    "subjectId": "test.subject.2"
  }
}
```

Interface for message formatting

```
/**
* @author mchyzer
* $Id$
*/
package edu.internet2.middleware.grouperClient.messaging.security;
import java.util.Collection;
import net.sf.json.JSONObject;

/**
* Implement this interface to provide security on messages since in Grouper
*/
public interface MessageSecurity {
  /**
   * encrypt (or not) or sign or whatever
   * @param sendFrom
   * @param sendTo
   * @param messageContainer
   * @return the json object with the container, and an unencrypted payload
   */
  public JSONObject processForSend(String sendFrom, String sendTo, JSONObject messageContainer);

  /**
   * encrypt (or not) or sign or whatever
   * @param sendFrom
   * @param sendTos
   * @param messageContainer
   * @return the json object to send
   */
  public JSONObject processForSend(String sendFrom, Collection<String> sendTos, JSONObject messageContainer);

  /**
   * encrypt (or not) or sign or whatever
   * @param sendFrom
   * @param sendTo
   * @param messageContainer
   * @return the json object to send
   */
  public JSONObject processForReceive(String sendFrom, String sendTo, JSONObject messageContainer);

}
```

Example of encrypting point-to-point

```
{
  "version": "1", //Message format version
  "timestamp": "createtime of message",
  "sequence": "1", //message sequence number, integer, increments per-message
  "securityImplementation": "grouperMessageFormat:SymmetricEncryptAesCbcPkcs5PaddingMessageSecurity",
  "edu.internet2.middleware.grouperClient.messaging.security.SymmetricEncryptAesCbcPkcs5PaddingMessageSecurity.
secretSha1First4": "c4h2",  //could have message security params, and namespaced
  "source": "someReturnHandle",
  "payload": " xRnr1VN1F0kWS4uWuSpo3l75uJWI...MKx1GzN8="
}
```

The encrypted payload is a JSON string

```
{
    "eventType": "MEMBERSHIP_ADD",
    "fieldName": ...
```

Basic message payload format

```
{
    "eventType": "MEMBERSHIP_ADD",
    "fieldName": "members",
    "groupId": "4854cde794b34948911bfea5b2acb611",
    "groupName": "atlassian:jira:jira-users",
    "id": "e8497c14bd6346b1aacca3ac13a25246",
    "membershipType": "flattened",
    "sequenceNumber": "388",
    "sourceId": "jdbc",
    "subjectId": "test.subject.2"
  }
```

SCIM message proposal

```
{
  "source": "someReturnHandle",
  "method": "PATCH",
  "resource": "/Groups/acbf3ae7-8463-4692-b4fd-9b4da3f908ce"
  "body": {
    "schemas": ["urn:scim:schemas:core:1.0"],
    "members": [
      {
        "display": "Babs Jensen",
        "value": "pennperson:12345678",
        "operation": "delete"
      }
    ]
  }
}
```

JOSE JWS / JWE (JSON Web Signing / Encryption) added in based on https://tools.ietf.org/html/draft-ietf-jose-json-web-signature-41

Note

Do not try to validate anything below. This is purely illustrative and not meant to be an example that can be developed against. Future documentation will contain real examples that can be used to build unit-tests against and validate code.

So, we have a random message.

```
eyJhbGciOiJFUzI1NiJ9.VGhpcyBpcyBzb21lIHRleHQgdGhhdCBpcyB0byBiZSBzaWduZWQu.GHiNd8EgKa-
2A4yJLHyLCqlwoSxwqv2rzGrvUTxczTYDBeUHUwQRB3P0dp_DALL0jQIDz2vQAT_cnWTIW98W_A
```

Let's break this apart. It's basically a triple of the header, payload, and signature separated by '.'. Each part is Base64'd and URL Encoded. This gives you a platform-neutral encapsulation. Let's decode these and look at them:

```
{
"alg": "HS256"
}
.
<MESSAGE PAYLOAD GOES HERE -- format TBD, could be SCIM, could be something else -- SEE ABOVE FOR EXAMPLES>
.
BASE64(URLEncode(Signature)) of payload goes here
```

Here's an example using Encryption. Again, it looks like the random message as it transits the messaging platform with a few more fields using dot-separated notation.

```
<header>.<encrypted_key>.<IV>.<ciphertext>.<authentication tag>
```

Here's an example decoded

```
{
"enc": "RSA-OAEP",
"alg": "A128CBC-HS256",
"kid": "7", //this would represent the Key ID of the public key used to create the symmetric message encryption key
}
.
<Symmetric Content Encryption Key Goes Here>
.
<Initialization Vector Goes Here>
.
<Encrypted Message goes here -- See Above for possible message formats>
.
<Authentication tag info goes here>
```

While I'm recommending we use the compressed serialization in order to minimize message size and minimize issues related to having to chunk overly large payloads, there are alternate ways of encapsulating the above in it's own JSON.

Here would be a JOSE complete message

```
{
  "version": "1", //Message format version
  "timestamp": "createtime of message",
  "sequence": "1", //message sequence number, integer, increments per-message
  "securityImplementation": "grouperMessageFormat:JoseMessageSecurity",
  "source": "someReturnHandle",
  "payload": "eyJhbGciOiJFUzI1NiJ9.VGhpcyBpcyBzb21lIHR"
}
```

See Also

Grouper Messaging Built-in