Terminology:

The Inbox
It is personal to a given user, that represents the current status of the conversations of such user. It's the front-page of the chat feature.

Inbox entry
It is a specific conversation, that _the_ user can identify by the recipient jid, that is, the user jid in case of a one-to-one chat, or the room jid in case of a group-chat.

Box (also referred as "folder")
A category where entries can be classified. The default box is the active box, simply called _inbox_. There is a second box, called _archive_, where entries can be thrown into and not displayed by default. No more boxes are expected.

Properties of an entry
Given an entry, certain properties are defined of such entry:

Archived
Beekeeper has two different boxes for the inbox: the regular one, simply called the inbox, and an archive mailbox, where clients can manually throw conversations they don't want displayed in the default UI.

It is expected that entries will reside in the archive until they're either manually moved back to the active box, or they receive a new message: in such case the entry should jump back to the active box.

Read
Entries keep a count of unread messages that is incremented automatically upon receiving a new message, and (in the current implementation) set to zero upon receiving either a message by one-self, or an appropriate chat marker as defined in [XEP-0333](https://xmpp.org/extensions/xep-0333.html) (which markers reset the count is a matter of configuration, see [doc](../../modules/mod_inbox/#modulesmod_inboxreset_markers)).

This property can also be manually set to zero or to one using the appropriate requests as explained below.

Muted
Entries can be muted for given periods of time, and likewise, unmuted. This changes the UI representation, and also, means that the user won't get PNs (Push Notifications) for this entry, until the time set expires or the user sets otherwise. Knowledge of this is necessary to help build the UI.

Expected times can be extended before the previous has expired, without the need to first unmutting. When muting a conversation, the final timestamp will be calculated by the server

by the current time plus the requested period, in seconds. When muting an already muted conversation, the timestamp is simply overridden following the previous specification.

Other properties
No more properties are expected, but I could envisage notions of flagging conversations with different colours, for example according to their urgency, or a client-specific category (work, personal, fitness, and whatnot), or pins to denote an entry should be always displayed (possibly in a special format, like on top of the box). The design of the protocol, and the implementation, aims to leave room for future extensions.

Getting properties
To fetch all supported properties, a classic Data Form is used. Upon the client sending an iq-get without a jid:
```xml
<iq id='some_unique_id' type='get'>
  <query xmlns='erlang-solutions.com:xmpp:inbox:0#conversation'/>
</iq>
```
The server would respond with:
```xml
<iq from='alicE@localhost' to='alicE@localhost/res1' id='some_unique_id' type='result'>
  <query xmlns='erlang-solutions.com:xmpp:inbox:0'>
    <x xmlns='jabber:x:data' type='form'>
      <field type='hidden'
var='FORM_TYPE'><value>erlang-solutions.com:xmpp:inbox:0</value></field>
      <field var='archive' type='boolean' value='false'/>
      <field var='read' type='boolean' value='false'/>
      <field var='mute' type='text-single' value='0'/>
    </x>
  </query>
</iq>
```

If the properties of a certain entry were to be fetch, it can easily be done with:
```xml
<iq id='some_unique_id' type='get'>
  <query xmlns='erlang-solutions.com:xmpp:inbox:0#conversation' jid='bob@localhost'/>
</iq>
```
To which the server will reply, just like before, with:
```xml
<iq id='some_unique_id' to='alice@localhost/res1' type='result'>
  <query xmlns='erlang-solutions.com:xmpp:inbox:0#conversation'>
    <archive>false</archive>
    <mute>0</mute>
    <read>true</read>
  </query>
```

```
</iq>
```

I see no benefit to implement querying a single property that outweights the cost of implementation and proper testing.

Setting properties

Setting properties are done using the standard XMPP pattern of `iq-query` and `iq-result`, as below:

```xml
<iq id='some_unique_id' type='set'>
  <query xmlns='erlang-solutions.com:xmpp:inbox:0#conversation' jid='bob@localhost'>
    <Property>Value</Property>
    <!-- Possibly other properties -->
  </query>
</iq>
```

Where `Property` and `Value` are a list of key-value pairs as follows:

- `archive`: `true` or `false`
- `mute`: number of seconds to mute for. Choose `0` for unmuting.
- `read` (adjective, not verb): `true` or `false`. Setting to true essentially sets the unread-count to `0`, `false` sets the unread-count to `1` (if it was equal to `0`, otherwise it lefts it unchanged). No other possibilities are offered, to reduce the risk of inconsistencies or problems induced by a faulty client.

If the query was successful, the server will answer with two stanzas, following the classic pattern of broadcasting state changes. First, it would send a message with a `<x>` children containing all new configuration, to the bare-jid of the user: this facilitates broadcasting to all online resources and to successfully synchronise their interfaces.

```xml
<message from='alice@localhost' to='alice@localhost' id='some_unique_id'>
  <x xmlns='erlang-solutions.com:xmpp:inbox:0#conversation' jid='bob@localhost'>
    <archive>false</archive>
    <mute>0</mute>
    <read>true</read>
  </x>
</message>
```

where `<mute>` may contain either a zero, to denote unmuted, or a RFC3339 timestamp, as in `2021-02-25T08:44:14.323836Z`.

To the requesting resource, a simple iq-result would be then send to notify of success, as required by the iq directives of the XMPP RFCs:

```xml
<iq id='some_unique_id' to='alice@localhost/res1' type='result'/>
```

If the request was not successful, the server will then answer as in:
```xml
<iq to='alice@localhost/res1' type='error'>
  <error type='Type'>
    <Condition xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```
Where `Type` will usually be `modify` or `cancel`, as explained in <https://xmpp.org/rfcs/rfc6120.html#stanzas-error-syntax>, and `Condition` is as explained in <https://xmpp.org/rfcs/rfc6120.html#stanzas-error-conditions>, being `bad-request` the most common.

This final syntax for the protocol has been chosen as it allows for better pipelining of requests, and it remains consistent with how, for example, rooms are configured for muclight.

Examples: archiving an entry
To archive an entry, the client can send:
```xml
<iq id='some_unique_id' type='set'>
  <query xmlns='erlang-solutions.com:xmpp:inbox:0#conversation' jid='bob@localhost'>
    <archive>true</archive>
  </query>
</iq>
```
On success, the server would return (considering the entry has no unread messages and is not muted):
```xml
<iq id='some_unique_id' to='alice@localhost/res1' type='result'>
  <query xmlns='erlang-solutions.com:xmpp:inbox:0#conversation' jid='bob@localhost'>
    <archive>true</archive>
    <mute>0</mute>
    <read>true</read>
  </query>
</iq>
```
If the client had sent an invalid number (negative, or NaN), the server would answer:
```xml
<iq to='alice@localhost/res1' type='error'>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

Examples: muting an entry
To mute an entry for a full day (86400 seconds in a day, 604800 in a week, for example), a client can send:
```xml
<iq id='some_unique_id' type='set'>
  <query xmlns='erlang-solutions.com:xmpp:inbox:0#conversation' jid='bob@localhost'>
    <mute>86400</mute>
  </query>
</iq>
```
On success, the server would return (considering the server receives the timestamp on "2021-02-26T09:11:05.634232Z", and the entry is on the active box and completely read):
```xml
<iq id='some_unique_id' to='alice@localhost/res1' type='result'>
  <query xmlns='erlang-solutions.com:xmpp:inbox:0#conversation' jid='bob@localhost'>
    <archive>false</archive>
    <mute>2021-02-27T09:11:05.634232Z</mute>
    <read>true</read>
  </query>
</iq>
```
If the client had sent an invalid number (negative, or NaN), the server would answer:
```xml
<iq to='alice@localhost/res1' type='error'>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```
To unmute, similarly, the client can send:
```xml
<iq id='some_unique_id' type='set'>
  <query xmlns='erlang-solutions.com:xmpp:inbox:0#conversation' jid='bob@localhost'>
    <mute>0</mute>
  </query>
</iq>
```
And server responses will be similar.

Examples: reading an entry
To set an entry as read, the client can send:
```xml
<iq id='some_unique_id' type='set'>
  <query xmlns='erlang-solutions.com:xmpp:inbox:0#conversation' jid='bob@localhost'>
    <read>true</read>
  </query>
</iq>
```

On success, the server would return (considering the entry is not archived and not muted):
```xml
<iq id='some_unique_id' to='alice@localhost/res1' type='result'>
  <query xmlns='erlang-solutions.com:xmpp:inbox:0#conversation' jid='bob@localhost'>
    <archive>false</archive>
    <mute>0</mute>
    <read>true</read>
  </query>
</iq>
```
On error, as usual, the client would get:
```xml
<iq to='alice@localhost/res1' type='error'>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```
And similarly, to set a conversation as unread:
```xml
<iq id='some_unique_id' type='set'>
  <query xmlns='erlang-solutions.com:xmpp:inbox:0#conversation' jid='bob@localhost'>
    <read>false</read>
  </query>
</iq>
```

Fetching the inbox
As in [mod_inbox], to request the currently supported form, the client can:
```xml
<!-- Client -->
<iq type='get' id='some_unique_id'>
  <query xmlns='erlang-solutions.com:xmpp:inbox:0'/>
</iq>

<!-- Server -->
<iq from='alicE@localhost' to='alicE@localhost/res1' id='some_unique_id' type='result'>
  <query xmlns='erlang-solutions.com:xmpp:inbox:0'>
    <x xmlns='jabber:x:data' type='form'>
      <field type='hidden'
var='FORM_TYPE'><value>erlang-solutions.com:xmpp:inbox:0</value></field>
      <field var='start' type='text-single'/>
      <field var='end' type='text-single'/>
      <field var='order' type='list-single'>
        <value>desc</value>
        <option label='Ascending by timestamp'><value>asc</value></option>
        <option label='Descending by timestamp'><value>desc</value></option>
```

```
      </field>
      <field var='hidden_read' type='text-single' value='false'/>
      <field var='archive' type='boolean' value='false'/>
    </x>
  </query>
</iq>
```

To fetch the regular inbox, as explained in
[mod_inbox/fetching](../../modules/mod_inbox/#filtering-and-ordering), the client sends:

```xml
<iq type='set' id='10bca'>
  <inbox xmlns='erlang-solutions.com:xmpp:inbox:0' queryid='b6'>
    <x xmlns='jabber:x:data' type='form'>
      <field type='hidden'
var='FORM_TYPE'><value>erlang-solutions.com:xmpp:inbox:0</value></field>
      <field type='list-single' var='order'><value>asc</value></field>
      <field type='text-single' var='hidden_read'><value>true</value></field>
      <field type='boolean' var='archive'><value>false</value></field>
    </x>
  </inbox>
</iq>
```
where the `archive` determines whether to query the archive inbox. A value of true means querying only the archive inbox, a value of false means querying only the active inbox, and if the flag is not set, it is assumed all entries are requested.

Then the client would receive as in:
```xml
<message from="alicE@localhost" to="alicE@localhost" id="9b759">
  <result xmlns="erlang-solutions.com:xmpp:inbox:0" unread="0" queryid="b6">
    <forwarded xmlns="urn:xmpp:forward:0">
      <delay xmlns="urn:xmpp:delay" stamp="2018-07-10T23:08:25.123456Z"/>
      <message xml:lang="en" type="chat" to="bOb@localhost/res1"
from="alicE@localhost/res1" id=”123”>
        <body>Hello</body>
      </message>
    </forwarded>
    <archive>false</archive>
    <mute>0</mute>
  </result>
</message>
```

All other fields in the form are as specified in [mod_inbox].

Limiting the query
It can happen that the amount of inbox entries is too big for a given user, even after filtering by `start` and `end` as already available in [mod_inbox]. Hence, we need to set a fixed limit of the number of entries that are requested. For this, we can use a `<max>` attribute as defined in [XEP-0059: #2.1 Limiting the Number of Items](https://xmpp.org/extensions/xep-0059.html#limit):
```xml
<iq type='set' id='10bca'>
  <inbox xmlns='erlang-solutions.com:xmpp:inbox:0' queryid='b6'>
    <x xmlns='jabber:x:data' type='form'>
      <field type='hidden'
var='FORM_TYPE'><value>erlang-solutions.com:xmpp:inbox:0</value></field>
      <field type='list-single' var='order'><value>asc</value></field>
      <field type='text-single' var='hidden_read'><value>true</value></field>
      <field type='boolean' var='archive'><value>false</value></field>
    </x>
    <set xmlns='http://jabber.org/protocol/rsm'>
      <max>Max</max>
    </set>
  </inbox>
</iq>
```
where `Max` is a non-negative integer.

[mod_inbox]: ../modules/mod_inbox.md