send-message

The most basic means of communication between agents uses a simple send-message call. In Lisp this is done using...

```
( send-message  self  to  msg )
```

where *self* is the agent sending the message, *to* is the name of the agent who will receive the message (a symbol) and *msg* is the message itself (any Lisp form).

To send the message '(*have a nice day*) from *agent1* to an agent named '*sid* the call would be...

```
( send-message  agent1  'sid  '(have a nice day) )
```

Note: send-message can take additional arguments, these are described in other parts of this document. send-message returns a value, this is also described elsewhere.

message-received

Agents receive messages through a message handler method. In Lisp this is provided as an argument to the agent constructor. This function should take the following arguments...

self	the agent sending the message (the agent itself, not just its name)
from	the name of the agent which sent the message;
to	the name of the agent that the message was addressed to. This is normally only useful if the same MessageListener is shared between agents;
msg	the text of the message;
msgId	a unique (and faceted) identifier for the message, MsgIds are described later in this document.

To supply an anonymous lambda function within the constructor you need to use something similar to the code below.

```
(make-instance 'agent :name 'agent1
               :receiver #'(lambda (self from to msg msg-id)
                            ; message handling code
                             ))
```

sending messages / clones & non-clones

Agents can be *cloneable* or *non-cloneable*. *Cloneable* agents process their messages as soon as they receive them. If a cloneable agent is busy with an existing task a new clone is created to process the new message. *Non-cloneable* agents have their messages queued when they are busy.

In practice Boris handles clones by creating a new process thread each time their receiver functions are triggered. Clones share global variables (including instance variables for agents which extend the agent class) but local method variables are not shared.

sessions

Messages are often sent in a series of exchanges which together form some kind of dialog. These exchanges are known as sessions. In Boris sessions are initiated with send-message but then continued by sending replies to earlier messages (or earlier replies).

send-reply

Replies are sent (in Lisp) using...

(send-reply self mfor reply)

The first argument for send-reply is the agent sending the reply; the second is the MsgId of the message that is being replied to. This is most often a message that has been recently received.

The code below defines a simple agent which sends a reply '(*thanks for the message*) for any message it receives...

```
(make-instance 'agent :name 'agent1
               :receiver #'(lambda (self from to msg msg-id)
                             (send-reply self msg-id
                                  '(thanks for the message))
                           ))
```