Boris

NetLogo thin agent-to-agent communications link reference guide

---

using the extension

The extension is named "nlboris", to use the extension include the following line at the start of your model code...

- `extensions [nlboris]`
  this loads the Boris extension

setting up the communications hub

The extension uses a communications hub. Using Boris terminology these hubs are typically called "portals" hence our choice of variable name in the examples (but you can choose another name if you prefer). You need to declare a variable for the hub in the declarations section and then make a hub in the setup stage. Hubs are given a name, in the example below the hub is called "HUB1"...

- `globals [portal]`
  declares a variable to hold the portal (the communications hub)

- `set portal nlboris:make "HUB1"`
  make a new portal (communications hub) called "HUB1" and assign it to the *portal* variable.

registering agents

In order to send and/or receive messages, NetLogo turtles must be registered with a portal, they do this by providing a name. eg:

- `nlboris:register portal "follower"`
  This command should be given a *turtle context* (ie: within an ask-turtle block or something similar). It will register a turtle with the name "follower" with the portal, the arguments provided are (i) the comms link (ii) a name for the turtle – this will be the name used for sending & receiving messages.

It is often necessary to register multiple agents but we must use different names for each of them. One way to manufacture unique names is to use their *who* number, eg:

- `nlboris:register portal (word "follower" who)`

If agents/turtles try to send messages without being registered the system will generate an exception.

deregistering agents

In many cases, there is no need to deregister turtles (ie: remove their details from a portal). If you try to send messages to a turtle that either (i) has never been registered or (ii) was registered but has since died nothing will happen, it will not cause an error. However, with long-running models with turtles that die & spawn often, deregistering turtles can reduce the volume of data that a portal needs to manage.

Deregistering a turtle must occur within a turtle context, it deregisters the current turtle from the portal specified, eg:

- `nlboris:deregister portal`

Turtles cannot send or receive messages after they are deregistered.

sending messages

Messages can be sent by one registered turtle to another (see restrictions about the use of multiple portals). All messages in NetLogo are sent as lists. Messages are queued so they will be received in the order they are sent.

- `nlboris:send-msg portal "follower"` *[...message-list...]*
  use the current agent (the one in context) to send the agent registered with the name "follower" a message.

broadcasting messages

Messages can be *broadcast* to all members of a breed. The syntax for broadcasting messages is similar to that for sending messages except that breed names are used in place of agent names as the recipients of messages, eg:

- `nlboris:broadcast portal "FOLLOWERS"` *[...message-list...]*
  send a message to all turtles in the "FOLLOWERS" breed

Notice breed names must be plural and in upper-case.

checking if messages are waiting

If an agent attempts to retrieve a message when there is no message waiting it will generate an exception. *nlboris:s-msg-waiting* is used to check if there are any messages available. It returns a Boolean value.

- `nlboris:is-msg-waiting portal`
  see if there is a message waiting for this turtle

This type of check is typically used in a code fragment like the one that follows (also see section on receiving messages)...

```
if nlboris:is-msg-waiting portal
[ let m nlboris:get-msg portal
  ...process message...
]
```

receiving messages

Messages are retrieved with the reporter *nlboris:get-msg*, eg:

- `let m nlboris:get-msg portal`
  get the next message for the current agent & store it in a (local) variable ***m***

All messages are retrieved in the form of structured lists where the first element is the name of the agent who sent the message and the second element is the message that was sent. *get-msg* is often used in a code fragment which destructures the list...

```
let m nlboris:get-msg portal
let from item 0 m
let msg  item 1 m
```

Note: if an agent attempts to retrieve a message when there is no message waiting it will generate an exception – see section on checking messages. Note also that messages are queued so they will be received in the order they are sent.

restrictions

Extensions are maintained & expanded and restrictions change so check the Boris web pages for updates.

Current restrictions are as follows...

- only turtles may register themselves as communicating agents (not patches);
- the nlboris extension does not allow communication with external Boris agents (to do this use the full Boris NetLogo subsystem with any additional Lisp/Java/C# subsystems required);