`NetRadio` specifications

Emile Rolley Antoine Liu Hugo Thomas

2020/2021

Entities

Definition

A *message* is defined as:

```
object Message =
 { owner_id: (Multicaster.id | Client.id)
 ; num_mess: Nat < 9999
 ; content: Char[140]
 }
```

A *multicaster* is defined as:

```
object Multicaster =
 { id: Char[8]
 ; comport: Nat < 9999
 ; castport: Nat < 9999
 ; castaddr: @
 ; messages: List[Message]
 }
```

A *manager* is defined as:

```
object Manager =
 { comport: Nat < 9999
 ; max_multicaster: Nat < 99
 ; multicasters: List[Multicaster]
 }
```

A *client* is defined as:

```
object Client = { id: Char[8] }
```

The protocol

All following connections will be in TCP except for the multicaster to the world.

Note

Each type is part of a “category” which is following a particular structure. The categories are separated by the number of field(s) accepted by the types.

Note that `END` is not part of the calculation in the number of field.

From *multicaster*..

..to the *world*

The broadcast is done by sending to the port `Multicaster.castport` at the address `Multicaster.castaddr` a message of the form:

```
 0  1  2  3  0  1  2  3  0  1  2  3  0  1  2  3   (bytes)
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|   "DIFF"   |  |  NumMess  |  |        Id       :
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
:     |  |         Mess (140 bytes)       | End |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

NumMess := to_str Message.num_mess
(* to_str 120 -> "0120" *)

Id := complete_with_hashes Message.owner_id
(* complete_with_hashes "RADIO" -> "RADIO###" *)

Mess := complete_with_hashes Message.content
```

After each sending:

```
let increment num : Nat -> Nat = (num + 1) mod 9999

Message.num_mess <- increment Message.num_mess
```

.. to *manager*

A multicaster can register itself to a manager by sending to `Manager.comport`:

```
 0  1  2  3  0  1  2  3  0  1  2  3  0  1  2  3   (bytes)
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|   "REGI"   |  |          Id           |  | Ip1 :
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
:                                |  |   Port1   |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                     Ip2                    |  |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|   Port2    | End |
+--+--+--+--+--+--+

Id := complete_with_hashes Message.owner_id
(* complete_with_hashes "RADIO" -> "RADIO###" *)

Ip1 := addr_to_str Multicaster.castaddr
(* addr_to_str "127.0.0.1" -> "127.000.000.001" *)

Port1 := Multicaster.castport

Ip2 := addr_to_str @Multicaster

Port2 := Multicaster.comport
```

If the **registration succeeds**, the manager sends back **without closing the connection**:

```
 0  1  2  3  0  1   (bytes)
+--+--+--+--+--+--+
|   "REOK"   | End |
+--+--+--+--+--+--+
```

Otherwise, **before closing the connection**, the manager responds:

```
 0  1  2  3  0  1   (bytes)
+--+--+--+--+--+--+
|   "RENO"   | End |
+--+--+--+--+--+--+
```

From *manager*..

..to *multicaster*

A manager can test if a multicaster is still alive by sending:

```
 0  1  2  3  0  1   (bytes)
+--+--+--+--+--+--+
|   "RUOK"   | End |
+--+--+--+--+--+--+
```

The multicaster needs to responds:

```
 0  1  2  3  0  1   (bytes)
+--+--+--+--+--+--+
|   "IMOK"   | End |
+--+--+--+--+--+--+
```

If after a delay the multicaster doesn't respond or sends back a wrong answer, the manager removes it from its `Manager.multicasters` and closes the connection.

From *client*..

..to *multicaster*

Register a new message

A client can sends a message to broadcast by sending to `Multicaster.recvport`:

```
 0  1  2  3  0  1  2  3  0  1  2  3  0  1  2  3   (bytes)
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|   "MESS"   |  |          Id          |  |     :
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
:              Mess (140 bytes)           | End |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

Id := complete_with_hashes Client.id
(* complete_with_hashes "RADIO" -> "RADIO###" *)
```

Then, the multicaster sends back:

```
 0  1  2  3  0  1   (bytes)
+--+--+--+--+--+--+
|   "ACKM"   | End |
+--+--+--+--+--+--+
```

Get the last broadcasted messages

A client can ask to a multicaster a list of its last messages by sending to `Multicaster.recvport`:

```
 0  1  2  3  0  1  2  3  0  1   (bytes)
+--+--+--+--+--+--+--+--+--+--+
|   "LAST"   |  | NbMess | End |
+--+--+--+--+--+--+--+--+--+--+

NbMess := to_str (length Multicaster.messages)
(* to_str 3 -> "003" *)
```

Then, the multicaster sends back `nb-mess` following the form:

```
 0  1  2  3  0  1  2  3  0  1  2  3  0  1  2  3   (bytes)
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|   "OLDM"   |  |  NumMess  |  |        Id        :
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
:     |  |         Mess (140 bytes)       | End |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

NumMess := to_str Message.num_mess
(* to_str 120 -> "0120" *)

Id := complete_with_hashes Message.owner_id
(* complete_with_hashes "RADIO" -> "RADIO###" *)

Mess := complete_with_hashes Message.content
```

In order to signal the end of the messages sending, the multicaster sends **before closing the connection**:

```
 0  1  2  3  0  1   (bytes)
+--+--+--+--+--+--+
|   "ENDM"   | End |
+--+--+--+--+--+--+
```

Get the weather of a location

A client can ask to a multicaster the current weather of a location by sending to `Multicaster.recvport`:

```
 0  1  2  3  0  1  ... 50   0  1   (bytes)
+--+--+--+--+--+--+--+--+--+--+--+
|   "RWET"   |  | Location  | End |
+--+--+--+--+--+--+--+--+--+--+--+

Location := complete_with_hashes (Message.content)
```

The response of the multicaster is done by sending as much as needed messages of the following format :

```
 0  1  2  3  0                          0  1   (bytes)
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|   "WETC"   |  |  Content (140 bytes)  | End |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

Content := complete_with_hashes (resp_buffer)
```

In order to signal the end of the file, the multicaster sends **before closing the connection**:

```
 0  1  2  3  0  1   (bytes)
+--+--+--+--+--+--+
|   "PEOF"   | End |
+--+--+--+--+--+--+
```

At any moment, if something goes wrong on the multicaster side, the multicaster sends **before closing the connection**:

```
 0  1  2  3  0  1   (bytes)
+--+--+--+--+--+--+
|   "WERR"   | End |
+--+--+--+--+--+--+
```

..to *manager*

A client can ask to a manager a list of its multicasters by sending to `Manager.comport`:

```
 0  1  2  3  0  1    (bytes)
+--+--+--+--+--+--+
|   "LIST"   | End |
+--+--+--+--+--+--+
```

To which, the manager responds:

```
 0  1  2  3  0  1  2  3  0 (bytes)
+--+--+--+--+--+--+--+--+--+
|   "LINB"   |  | Ndf | End |
+--+--+--+--+--+--+--+--+--+

Ndf := to_str (length Manager.multicasters)
(* to_str 5 -> "05" *)
```

Followed by `length Manager.multicasters` messages of the form (**before closing the connection**):

```
 0  1  2  3  0  1  2  3  0  1  2  3  0  1  2  3   (bytes)
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|   "ITEM"   |  |          Id            |  | Ip1 :
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
:                                 |  |   Port1    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                     Ip2                     |  |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|   Port2    | End |
+--+--+--+--+--+--+

Id := complete_with_hashes Message.owner_id
(* complete_with_hashes "RADIO" -> "RADIO###" *)

Ip1 := addr_to_str Multicaster.castaddr
(* addr_to_str "127.0.0.1" -> "127.000.000.001" *)

Port1 := Multicaster.castport

Ip2 := addr_to_str @Multicaster

Port2 := Multicaster.comport
```

Message