oXigen is the brand name of the racing digital system for slot cars studied and developed by Slot.it.
Basically, it is a wireless system which allows digital slot racing with up of 20 cars on the same track.

oXigen system's communication protocol

A typical oXigen system is composed by the following items:

1. an oXigen digital cartridge compatible with the Slot.it SCP controller;
2. an oXigen digital chip for the car;
3. an oXigen lane changer;
4. an oXigen dongle.

Regarding the communication protocol, information is transmitted between devices as follows:

PC - dongle: USB
dongle - controllers: 2.4GHz RF
controller - car: 2.4GHz RF

The lane changer receives the information it needs from an IR LED inside the car.

Dongle - PC communication protocol

When the dongle is plugged into an USB port of the PC and the RMS software is running, the dongle receives, on the USB, data regarding the race status, according to the RMS set up, and sends through USB all the information collected from the controller/car pair. The dongle can only send the information about at most 5 controller/car pair for any given transmission frame.

Note: the PC recognises the dongle device as a virtual COM port. The ID vendor is 0x1FEE and the ID product is 0x0002.

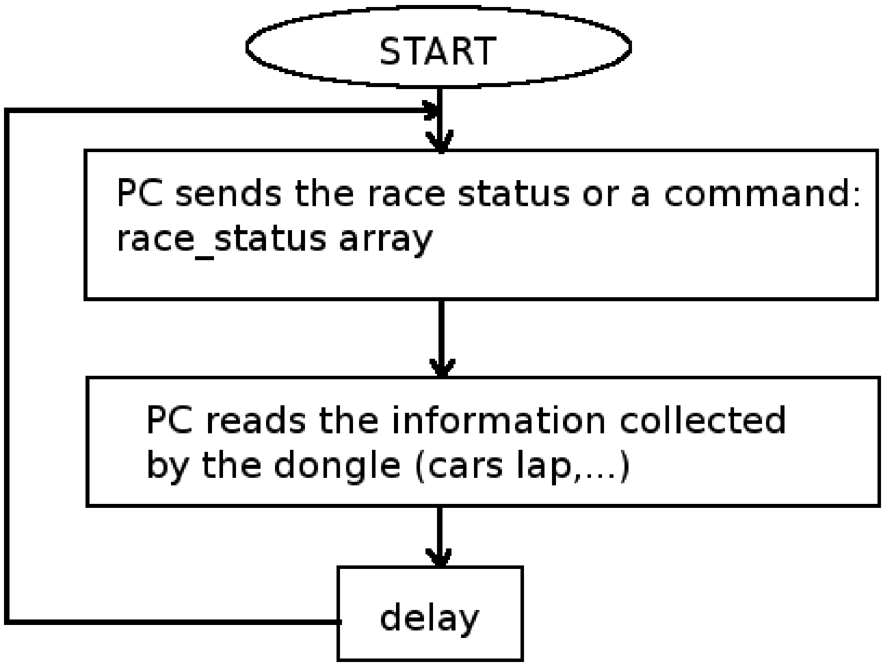
The race is managed through the RMS whose available features are summarized below:

1. START/STOP race. The user can start/stop the race and lap counting is therefore enabled/disabled. A START command, issued after a STOP command, resets lapcounting to zero. When the race is active, the user can adjust the maximum speed of the cars as well.
2. PAUSE race. The user can pause the race: lap counting is disabled and all cars are stopped. A START race command, issued after a PAUSE command, restarts lap counting from the current values; a STOP race command ends the race;
3. PACE race. The user can set the race in 'PACE car' (yellow flag) mode. The race is kept running and the lap counting is enabled and the maximum speed value of the cars can be

selected by the user. During this phase, the user can also enable/disable the lane changing.

4. specific info for single controller/car: The user can act on a single controller/car pair, changing the maximum speed, the maximum brake value or forcing a car to go into the pit-lane by setting a minimum speed and forcing the lane changing

As said earlier, the communication between PC and dongle is bidirectional as shown in the following flowchart below and complies to the protocol reported below.


START
PC sends the race status or a command: race_status array
PC reads the information collected by the dongle (cars lap,...)
delay

The speed with which data is refreshed on screen depends on the speed of the PC and on its load. In general, data is refreshed every 600ms.

TX protocol (PC→ Dongle):

payload: 7 bytes, array race_status[7].

race_status[0] and race_status[1] hold general information valid for all the couples controller/car

race_status[2], race_status[3] and race_status[4] hold specific information valid only for one controller/car pair. General race information have precedence over controller/car specific information.

In other words: each data frame contains 7 bytes, the first two of which contain information which is valid for all controller/cars, the rest (bytes 2 to 6) being specific to one controller/car pair only.

race_status[0]: race status: a value wrtitten in this byte has the meaning described in the table:

Value	Meaning
01h	STOP – max speed value is 0% (see race_status[1])
02h	Reserved – must not be used
03h	START – max speed value in range 0 ▯ 100% (see race_status[1])
04h	PAUSE - max speed value is 0% (see race_status[1])
05h	PACE CAR – Lane changing ENABLED max speed value in range 0 ▯ 100% (see race_status[1])
15h	PACE CAR – Lane changing DISABLED max speed value in range 0 ▯ 100% (see race_status[1])

race_status[1]: maximum speed value permitted for the oXigen cars. It is forced to zero in STOP and PAUSE status; in START and PACE CAR status can be chosen and modified in the range: 00h to FFh (equivalent to 0 to 100%).

race_status[2]: used to communicate specific information to a particular couple controller/car. It holds the controller/car ID in issue which has to be selected in the range 1 to 20.

race_status[3]:

- bit 7: command addressee:
  - 0: all couples controller/car;
  - 1: couple controller/car specified in race_status[2]
- bit 6...0: command value (see §Device commands);

race_status[4]: argument of the command specified in race_status[3], bit 6...0

race_status[5], race_status[6]: not used.

RX protocol(Dongle→ PC):

The dongle transmits to the PC all the information collected between two consecutive USB pack transmission. Payload 45 bytes (9 bytes per car).

rf_data_x [0] : 80h. header of a valid packet for the couple controller/car with ID equivalent to rf_data_x [1];
rf_data_x [1] : controller/car ID in hexadecimal format (it certifies the controller is powered up);
rf_data_x [2] : last lap time (high byte);
rf_data_x [3] : last lap time (low_byte) (see Note1);
rf_data_x [4] : not used;
rf_data_x [5] : total lap number (low_byte);
rf_data_x [6] : total lap number (high_byte) (see Note 2);
rf_data_x [7] : car fuel and feedback

- bit 7: car presence on the track.
  - 0: the car is not on the track;
  - 1: the car is on the track. Info available only if the paired controller is powered up.
- bit 6...0: trigger mean value (see Note 3);

rf_data_x [8] : software release car/controller, pit-lane information.

- bit 7: device software release owner:
  - value 0: controller software release;
  - value 1: car software release;
- bit 6: car pit-lane status.
  - value 0: car is not in pit-lane;
  - value 1: car is in pit-lane;
- bit 5,...,bit 4: main software release of the device specified by bit 7;
- bit 3, …, bit 0: sub software release of the device specified by bit 7

Note 1:

last lap time = [(rf_data_x [2]*256) + (rf_data_x [3])] / 99,25

The result is the last lap time in [s].
The size of last lap time variable is equal or grater than 16 bit.

Note 2:

lap number = (rf_data_x [6] * 256) + (rf_data_x [5])

The size of lap number variable is equal or grater than 16 bit.

Note 3: this data can be used to calculate the fuel consumption, whose value on the RMS page is calculated this way:

- tmp_value = ((rf_data_x [6] AND 7Fh) * 2)
- if (tmp_value > 253) then tmp_value = 255;
- RMSshown_value = (tmp_value / 255)*10).

The dongle keeps track of all lap counting received from the controllers. If a SCP controller is reset during normal operation, when the car crosses the finish line for the first time after reset, the dongle sends the correct lap number to the PC while the lap time value will be set equal to zero. This not only avoids displaying a wrong lap time after a controller reset (since the lap time is calculated by SCP) but informs the RMS that a controller was reset.

Devices' commands

Two command types are currently available:

- commands for controller/s;
- commands for the dongle.

Commands for controller/s

Command description	race_status[3] value	race_status[4] value
Neutral command	00h (80h)	00h
Pit lane speed	01h (81h)	New pit-lane speed value
Maximum speed limitation	02h (82h)	Maximum speed value
Minimum speed limitation	03h (83h)	bit 7: 1/0 force exchange such as arrow ↑ SCP button bit 6: 1/0 force exchange such as arrow ↓ SCP button bit 5..0: minimum speed limitation divided by 2
RF transmission power changing	04h (84h)	Transmission power value 0 : -18dBm 1 : -12dBm 2 : - 6dBm 3 : 0dBm
Maximum brake value	05h (85h)	Maximum brake value

The second table column describes the command type, while the third column the command argument. In the second table column there are two values for every command type because the user can send a command to a specific couple controller/car or to all the couples controller/car as describes below:

- copy in race_status[3] the hexadecimal value in brackets to transmit the command to a specific couple controller/car. In this case it's necessary to write the receiver's couple controller/car ID in race_status[2] ;
- copy in race_status[3] the hexadecimal value not in brackets to transmit the command to all the couples controller/car

Examples of possible valid commands:

- Maximum speed limitation (upper speed limit 7Fh), for couple controller/car ID 3:
  - race_status[2] = 03h; → controller/car number
  - race_status[3] = 82h; → command type
  - race_status[4] = 7Fh; → upper speed limit value

- Minimum speed limitation (minimum speed value 40h), for all controllers:
  - race_status[2] = not important in this case type;
  - race_status[3] = 03h; → command type
  - race_status[4] = 20h; → minimum speed value. The final value will be 40h.

It's possible to cancel a sent command. To do this it's necessary to transmit the default value, as shown in the table below

Command description	race_status[3] value	race_status[4] default value
Maximum speed limitation	02h (82h)	ffh
Minimum speed limitation	03h (83h)	00h
Maximum brake value	05h (85h)	ffh

Some command cancellation examples:

- Maximum speed limitation cancellation for controller number 3:
  - race_status[2] = 03h; → controller/car number
  - race_status[3] = 82h; → command type
  - race_status[4] = ffh; → upper speed limit value

- Minimum speed limitation cancellation for all controllers:
  - race_status[2] = not important in this case type;
  - race_status[3] = 03h; → command type
  - race_status[4] = 00h; → minimum speed default value (speed 0 and no lane exchange forced).

Commands for dongle

Currently, we have developed only the dongle software release query, with the following command:

race_status[0]: 06h;
race_status[1]: 06h;
race_status[2]: 06h;
race_status[3]: 06h;
race_status[4]: XXh (any value);
race_status[5]: XXh (any value);

The dongle will answer to the query sending two bytes: the first one is the main software release, while the second one is the sub software release