Controlling the DACTA 70909 from JAVA

Timo Paukku Dinnesen (timo@daimi.au.dk) University of Aarhus
Aabogade 34 8200 Aarhus N, Denmark

March 8, 2006

1 Introduction

Before the LEGO Mindstorms was introduced LEGO's most advanced kit for controlling motors and getting input from sensors was the DACTA control box. This box exists in different versions but one of the newest is the DACTA 70909. The box can be connected to a RS232 port and controlled via this port. The box needs power to run. This power is supplied via a separate 10Volt power adaptor.
This control box is available at the LEGO Lab at DAIMI.

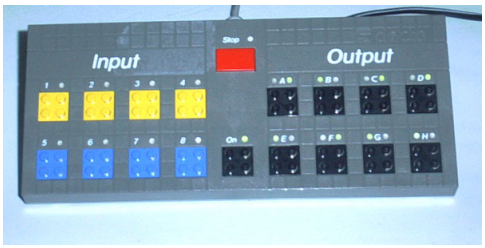

Figure 1: The Dacta 70909 control box.

2 Input ports and output ports

The DACTA 70909 has eight input ports and eight output ports. The first four of the eight inputs are passive and the other four are active. The active inputs will supply power to the sensor connected to it. For example the LEGO light sensors and the LEGO rotation sensors require an active port for full funtionality.

The eight output ports can be turned on and off and the polarity can be changed. On top of this each output port can be set to output nine different levels of power by means of *pulse width modulation*,[wik]. These range from

level zero which is the same as a turned off state, to level eight that is full power. Full power equals nine volts that is the typical voltage for newer LEGO motors.

3 Controlling the box

The box is supposed to be controlled by some software included with the LEGO set. Today this software is outdated and it does not seem easy to find it anywhere. It would be usefull to be able to use the control box from other programs and not have to rely on some specific software or platform. JAVA could be a good choice since it is not tied to any specific platform, and because there allready exist some JAVA frameworks for programming JAVA on the more modern LEGO Mindstorms RCX bricks. Since no JAVA controls exist for the box we will have to create our own.

4 Serial communication in JAVA

Getting access to RS232 ports using JAVA requires that JAVA Communications,[Com], is installled. Installing and using JAVA Communications is not trivial because a lot of files need to be copied manually. A quick description is to:

1. Download and unpack JAVA Communications.
2. Copy win32com.dll to your `<JRE>\bin` directory.
3. Copy comm.jar to your `<JRE>\lib` directory.
4. Copy javax.comm.properties to your `<JRE>\lib` directory.
5. Add `<JRE>\lib\comm.jar` to CLASSPATH.

In the above `<JRE>` indicates the directory of the Java Runtime Environment(JRE) that will be used to run the program that needs to use JAVA Communications. Some JAVA installations come with several JRE's and this makes this installation process more difficult. An easy solution is to copy the Java Communication files into the directories of all installed JRE's.

5 Reverse engineering the protocol

The protocol for communicating with the box is not official and LEGO does not provide any information on this. To figure out how the box worked some research was needed.

Other people have created their own software to controll the box. One of these is Anders Isaksson, who created a page describing his discoveries using a program to monitor transmission between the DACTA box and a

PC,[Isa]. This page combined with a little guessing was enough to get the box up and running from JAVA.

6 Initialization

To communicate with the box the RS232 port must be set up with the following properties.

- Baudrate: 9600
- Databits: 8
- Parity: NONE
- Number of stop bits: 1

When the port is set up the DACTA 70909 can be initialized by sending some special messages to it.

1. First send the letter p and a NULL character.
2. The DACTA 70909 will reply with some copyright messages.
3. The special "secret" LEGO message must be sent to the box:

```
###Do you byte, when I knock?$$$
```

4. The box will reply with:

```
###Just a bit off the block!$$$
```

The final message is probably sent to ensure that the device connected to the serial port is actually a DACTA 70909. It is good practice to listen for it to make sure the box is connected, but it is not really neccesary. The box will start sending sensor data as soon as it has sent the reply.

Since there is no specific request/reply protocol there is no need to wait for the copyright messages. An example of how simple it can be done in JAVA can be seen here:

```
String message = "p\0###Do you byte, when I knock?$$$";
outputstream.write(message.getBytes());
```

7 Communicating with the box

When the DACTA 70909 has been initialized, communicating with it is fairly easy and seems to go well with the typical stream communication that is used in JAVA.

When running the box will accepts commands as a stream of bytes. It will output a continuous stream of bytes that contains the state of the sensors inputs. When a command is sent to the box there is no reply telling if the command was performed or not. One must simply assume that executing the command went well.

7.1 Output port Commands

Table 1 shows the bitpatterns for the different commands and tells what each command does. Some letters have a special meaning in the table:

- An x will mean that the setting of that bit does not change the command.
- *nnn* specifies a number between 0 and 7 encoded into 3 bits.
- *oooooooo* specifies a byte that is used to tell which output ports will be affected by the command. The least significant bit is port 1 and the most significant bit is port 8.
- *BREAK* specifies the special break signal that is used by RS232 ports. In JAVA this special signal can be sent using the sendBreak(int millis) method on the SerialPort object.
- *tttttttt* specifies a byte containing a number indicating time in tenths of a second(Deciseconds.)

Byte pattern	Description
00100nnn	Turn port nnn on.
00101nnn	Turn port nnn off.
00100nnn	Reverse direction of port number nnn.
01000nnn	Set direction of port nnn to left.
01001nnn	Set direction of port nnn to right.
0111xxxx	Turn the box off.
1000xxx0 oooooooo	Set outputs to normal mode.
1000xxx0 oooooooo	Set outputs to flashing mode.
10010000 BREAK	Turn off all ports.
1001x000 oooooooo	Turn ports off.
1001x001 oooooooo	Turn ports on.
1001x010 oooooooo	Set power level to zero.
1001x011 oooooooo	Set direction of ports to left.
1001x100 oooooooo	Set direction of ports to right.
1001x101 oooooooo	Reverse directions of ports.
10110nnn oooooooo	Set power to nnn.
1100xnnn tttttttt	Turn port nnn on for specified time.
11101nnn tttttttt	Set ON cycle time for port.(Flashing mode)
11100nnn tttttttt	Set OFF cycle time for port.(Flashing mode)
11110000 tttttttt	Seems to flash output 2 after specified delay.

Table 1: Table of commands

8 Reading sensor inputs

When the DACTA 70909 has been initialized it will send frames containing the state of the input ports. The DACTA 70909 has a sampling rate of 20 milliseconds and will thus send an information frame 50 times a second. Each frame is 19 bytes long, and contains the following:

1. A header, two bytes long and containing a zero each.

0	1
0	0

2. A data part of 16 bytes containing the sensor data. Each sensor has a high byte and a low byte. Compared to the numbers on the DACTA 70909 the bytes are positioned in a somewhat strange manner.

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
4H	4L	8H	8L	3H	3L	7H	7L	2H	2L	6H	6L	1H	1L	5H	5L

3. The frame is ended with a checksum. The integrity of the frame is checked by adding the values of all bytes together and checking if the result logically "AND'ed" with 0xFF equals 0xFF.

18
CHK

Each sensor input port is represented with two bytes in the frame. These two bytes represent two values. A ten bit raw sensor value (R9-R0) and a status value (S5-S0).

High byte								Low byte							
R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	S5	S4	S3	S2	S1	S0

The raw value indicates the resistance in the sensor. A values of 0 will indicate zero Ohms and the max value of 1023 corresponds to about 2 Mega Ohms. The use of the 6 bit status value is a bit unclear. In 1994 Andrew Carol posted a few notes on usenet, describing how the status value change when using a LEGO angle sensor on a port,[Car]. Unfortunately this work seems to never have been finished. According to the post the status value indicates when changes occur in the raw value, and it also seems like the status value indicates how the raw value has changed. Since the raw value is allready available the status value is redundant and was not investigated further during the writing of this appendix.

9 Keeping the box alive

The box will shutdown when it has not received a message for a few seconds. No special command is needed and even non-commands will work to keep the box alive. In the JAVA Control class this was accomplished by a thread that would send a byte containing the number 2 every two seconds. This number was chosen because it was also used in other implementations using the DACTA 70909.

10 A JAVA control class

With this information about the DACTA box is was fairly easy to program a control class in JAVA. The code and documentation can be found at `http://timo.dk/Projects/DACTAController/` and on the CD included in this thesis[1].

Since some of the commands are redundant not all of them were used. Also the status value of the inputs are not used for anything in the current version.

The control class was created for easy use instead of trying to utilize all the features of the DACTA 70909. Basically only four methods are used:

- init(String port) Initializes the DACTA 70909 that is connected on the port specified by the name given in the string.
- getSensorValue(int sensornumber) Returns the raw value of a sensor input port.

[1] In the directory DACTA70909

- getStatusValue(int sensornumber) Returns the status value of a sensor input port.
- setOutPut(int port, boolean on, boolean right) Sets the output of port to on/off and left/right depending on the booleans passed along with the call.
- setPower(int outputnr, int value) Sets the power of the output port.

All methods return a status value. If an error occourred this value will be negative. The things that can go wrong when calling a method are usually not very critical and because of this returning errorcodes was chosen instead of creating exceptions.

11 Conclusion

The DACTA 70909 can now be controlled easily from a pc with JAVA installed. The class could be improved by implementing some of the commands that are currently supported by the box but not the JAVA class. This seems a waste of time since the current set of methods allow for complete control of the DACTA 70909.

Perhaps the DACTA box has even more features. For example the input status values may have some nice features. Unfortunately it seems that all information about these have dissapeared or is very hard to find. Discovering the bits of information and putting this information together to create the JAVA class was interesting, but it also leaves a question about if all the posibilities of the box has been discovered.

References

[Car] Andy Carol. `http://groups.google.dk/group/rec.toys.lego/msg/d72300a9f5cc8bfc`. (This link was checked and found working on 8. March 2006.).

[Com] JAVA Communications. `http://java.sun.com/products/javacomm/index.jsp`. (This link was checked and found working on 8. March 2006.).

[Isa] Anders Isaksson. `http://web.telia.com/~u16122508/dacta/`. (This link was checked and found working on 8. March 2006.).

[wik] `http://en.wikipedia.org/wiki/Pulse_width_modulation`. (This link was checked and found working on 8. March 2006.).