

R-Code SDK Tutorial

by Ricardo A. Téllez, v1.2

4th September 2004



Contents

1	Introduction	3
2	The R-Code SDK	5
2.1	Description	5
2.2	Installation	5
2.2.1	Preparation of the memory stick	5
2.2.2	Execution of an R-Code program	5
2.2.3	Setting up the wireless console	6
2.3	General considerations	6
2.4	Programming R-Code	8
2.5	R-Code Commands	9
2.6	R-Code relational operators	11
2.7	Special system variables	12
2.8	Aibo actions	14
2.9	Aibo recognised words	15
2.10	Debugging	16
3	Examples	18
3.1	The Greeting.R code example	18
3.2	The Maze.R code example	19
4	Creation and use of contents files	21
4.1	Creating contents files	21
4.1.1	Creating motion files	21
4.1.2	Creating audio files	22
4.1.3	Creating LED files	23
4.2	Converting to ODA files	23
4.3	Create the MWC file	23
4.4	Copying files to the stick	25

1 Introduction

The R-Code SDK conforms a set of tools released by Sony in order to allow programming Aibo using the R-Code language. R-Code is a scripting language for programming Aibo robots. By script it is meant a simple text file containing simple commands. Those commands allow the programmer to send instructions to Aibo in a higher level manner than typical C++ programming environment (like for example the OPEN-R environment for Aibo). Scripting languages have the benefits of being simple to learn and to use and require no compilation, but they have the drawback of allowing less control to the programmer on how to do things. You can see an R-Code example here:

```
:1000
PLAY:ACTION:STAND
IF:Face:=:1:THEN
WAIT
PLAY:ACTION:CLIFF_DETECT_OFF
WAIT
```

R-Code allows to program complicated things (like walking or dancing) with just a few commands in a text file. Think of R-Code instructions like macros and OPEN-R programs like the code of those macros. With the use of the macros you cannot obtain more precision than the one coded in it by the OPEN-R code. Use R-Code when you do not require a precise control of the dog, just to perform actions, and start predefined movements or responses. But do not misunderstand: R-Code is a powerful tool that allows the implementation of real complicated behaviours.

To be able to use R-Code with Aibo, you will need two main things: first, a properly configured memory stick, and second, your R-Code program. In order to have the memory stick ready to work, you will need to have an empty one and save into it some special files provided by Sony (the Redist7¹ files). Those files are called the *base system*, and are part of the R-Code SDK. Once you have your memory stick with the base system installed on it, you can transfer your R-Code program to the stick.

Even that the R-Code SDK contains some specific files for Microsoft Windows based systems (the RTool and NsmAuth files), development of R-Code programs can be done in any operating system having a text editor. R-Code programs are plain text files that need no compilation, so any computer system providing such an editor will work fine with R-Code.

What you can do with R-Code:

- Put AIBO into its three basic postures (sit, stand and sleep)
- Make AIBO walk, turn around, kick, touch, move its head, and track its pink ball
- Make AIBO find a pink ball, face, and AIBOne.
- Make AIBO recognise 53 verbal commands, 35 media link sounds, and 68 tonal scales

¹Redist7 is the R-Code package for Aibo model ERS-7. Other models of Aibo require other R-Code base system, also provided by Sony. We will focus here in the ERS-7 version of R-Code.

- Execute 600 built-in contents (motions, sound, LED patterns) which are included in commercial AIBOware (AIBO MIND)
- Execute motions created using AIBO Motion Editor
- Execute MIDI, WAV sound created by users
- Execute LED patterns created by users
- Acquire data about obstacle detection (distance recognition), and data from the head, back, and chin touch sensors;
- Use control statements including IF and FOR statements
- Run subroutines
- Use variables (32-bit integers only)
- Perform arithmetic operations
- Perform stack PUSH and POP operations
- Use the Wireless LAN capability to send commands to AIBO and for debugging.

2 The R-Code SDK

2.1 Description

The R-Code SDK is the group of files released by Sony that allows users to program in R-Code. It contains several files:

- Redist7: is the set of files that will compose the base system of the memory stick. That base system will be needed in order to execute your R-Code program. Redist7 contains both Japanese and English versions, so you will need to select the one correct for you.
- Rtool: a windows application that converts MTN files (files describing Aibo motions, created by using Medit or another motion editor) into a format that can be recognised by R-Code and then used by it (this is called the ODA format). This utility also creates a special file called ERS-7.MWC which contains a list of motions, and lights and sound. The generation of such files will be discussed later. An R-Code script can invoke any of the contents of the ODA file by using the following command: `PLAY:MWCID:<number>`
- NsmAuth: is a library to be linked with windows C++ programs that want to connect to Aibo's R-Code wireless remote command line² and use authentication.

2.2 Installation

Go to the Sony Open-R web page (<http://openr.aibo.com>) and download the R-Code files. These include Redist7_ver1.zip, Rtool_ver1.zip and NsmAuth_ver1.zip. Nevertheless, you will only need now the first one, since the others are some support files for more complicated environments.

2.2.1 Preparation of the memory stick

First step is to prepare the memory stick with the correct file structure. In this way, the memory stick will be able to accueil your R-Code program. Unpack the Redist7_ver1.zip file. This will generate several files and directories. Now copy the Redist7/Eng/OPEN-R folder generated into an empty memory stick. Now the stick is ready to accommodate your program.

2.2.2 Execution of an R-Code program

Generate your program using any text editor. Save it with the name R-CODE.R (it must have that name). Then install this file in the /OPEN-R/APP/PC/AMS/ directory of your memory stick. Insert the memory stick into Aibo and switch it on. Aibo will automatically execute your program.

²The wireless command line will be described in section 2.5

2.2.3 Setting up the wireless console

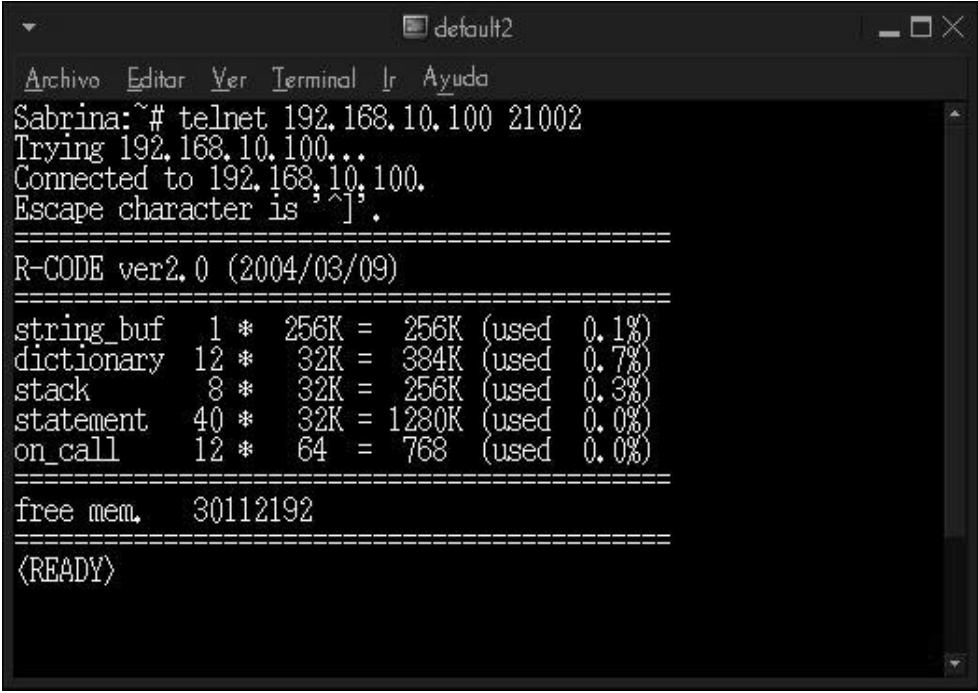
The wireless LAN console for R-Code SDK allows the user to send commands directly from a telnet session with Aibo. This has several advantages like online debugging or testing of routines and commands. To set up the console follow the steps:

1. Enable the wireless LAN. This step is explained in the OPEN-R web site (<http://openr.aibo.com>) or in the *Aibo quick-start guide* of the same author as this document (it can be found at <http://www.ouroboros.org/~rt71592>). Basically you will have to change the network parameters of the file in the memory stick path OPEN-R/SYSTEM/CONF/WLANDFLT.TXT. Please refer to the texts named for any problem you may have.
2. Disable authentication in R-Code. By disabling authentication neither ID nor password will be required when connecting to the Aibo console. To do this, just delete the file in the memory stick /OPEN-R/APP/DATA/P/OWNER.TXT and create the file /OPEN-R/APP/PC/AMS/NOAUTH.CFG (with size zero).
3. Insert the memory stick into Aibo and start it.
4. Connect to the robot by telnet protocol. Connections are accepted at port 21002, so the command would be *telnet Your_Aibo_IP 21002*. If connection succeeds, you will see an R-Code header on your screen and after some information the console prompt will be displayed (see figure 1).
5. You can send commands directly to Aibo by just typing in the console.
6. You can send whole programs directly by typing first the command EDIT. Then write your program on the console. Once you finish it, type END to specify that your program is finished and ready to execute. To execute the program type RUN. Any program sent by this way will replace the present program in Aibo's memory (not in the memory stick).
7. To disconnect the console type the *@DISS* command.

2.3 General considerations

Before starting coding into R-Code some general considerations must be set:

1. R-Code scripts are composed of lines of text specifying commands. Commands are words separated by colons. Only ASCII characters and underscores can be used in words, and lines can not be longer than 127 bytes long, including comments and the newline character. Indentation characters (Tabs and spaces at the beginning of the line) are ignored.
2. Lines beginning with a colon are considered label rows. These are used to jump between lines of code by using the GOTO function. Lines beginning with a character that is not an alphanumeric one or a colon are treated as comment lines. A comment can also be specified with a double slash. After the double slash everything is treated as a comment.



```
Sabrina:~# telnet 192.168.10.100 21002
Trying 192.168.10.100...
Connected to 192.168.10.100.
Escape character is '^]'.
=====
R-CODE ver2.0 (2004/03/09)
=====
string_buf   1 * 256K = 256K (used 0.1%)
dictionary  12 * 32K = 384K (used 0.7%)
stack        8 * 32K = 256K (used 0.3%)
statement    40 * 32K = 1280K (used 0.0%)
on_call      12 * 64  = 768  (used 0.0%)
=====
free mem.    30112192
=====
<READY>
```

Figure 1: A telnet session with Aibo in R-Code

3. R-Code is case sensitive. It means that *Value* and *VALUE* are different variables. Also, words completely capitalised are reserved for R-Code system use, so it is better not to give capitalised names to your own variables. For example, *Value* and *value* are good names for your variables, but *VALUE* is a bad name.
4. Words starting by plus, minus or a number are treated as numbers. Numbers starting by 0x or 0X are treated as hexadecimal. Numbers starting by 0o or 0O are treated as octal numbers. Numbers starting by 0b or 0B are treated as binary ones.
5. Only 32 bits integers can be used.

2.4 Programming R-Code

Any R-Code program is a list of commands for the Aibo robot. Programs look very similar to Basic programs and implement all the typical instructions of such a programming language like conditionals, subroutines, flow control, use of variables, etc, but also, R-Code provides the implementation of a stack that works in a similar way of that of the assembler language. The list of available commands is specified in the section 2.5 and is very self descriptive.

Usually, the R-Code program is generated using any ASCII text editor. The program should be written on the editor, and once finished, it must be saved into the memory stick (directory /OPEN-R/APP/PC/AMS/) with the name R-CODE.R. Then the memory stick is inserted into Aibo and the program will run when Aibo is switched on. There exists the possibility of creating the program directly into Aibo by using the wireless console. This approach has the advantage of been more flexible and faster when debugging errors (see section 2.10).

R-Code defines a set of variables that allow the programmer to identify the status of the robot. A complete set of the variables available is described in section 2.7. There are variables that correspond to the values of internal states but also to values sensed by the sensors. The programmer can read and set the status of those variables at any time, and make Aibo act consequently. For example, a variable called *Face*, indicates when Aibo has detected a face (it is set to value 1). You can use that value to see if Aibo is in front of someone. Other variables exist that show for example the value of the paw sensors or the detection of the ball or the AIBOne. The programmer can also create his own variables and use them to make calculations or pass them as parameters to subroutines (by using the stack). Just put the parameters in the stack by using the PUSH command, then call the subroutine (using the CALL command), and inside it, retrieve the parameters in the same order by using the ARG command.

R-Code can also make Aibo play actions. An action is a predefined movement (or group of actions) identified by a name. Those movements have been pre-done by the Sony engineers, so you do not have to design them, just to use them. You can specify which action you want the robot to perform by just indicating it, and the robot will do it. Just type PLAY:ACTION:<the action> to execute an action. For example, you can order Aibo to SIT by typing PLAY:ACTION:SIT. Some actions require the use of several parameters, that you may add to the action command. For example, if you want

Aibo to turn some degrees, you must specify the number of degrees as a parameter (PLAY:ACTION:TURN:30). A list of all the available predefined actions is provided in section 2.8 but you can create your own actions by using the MEdit and RTool tools (see section 4 about how to create new actions).

It is also possible to make Aibo recognise some spoken words by using R-Code. Aibo implements a speech recognition system that the programmer can make use of it. It also has a tone and a sound recognition system, that recognises tones and sounds produced by other Aibos (making it possible to make Aibos communicate easily). The variable *AU_Voice* indicates when a voice has been detected, and the variable *AU_Voice_ID* indicates the ID of the recognised word. A list of the words that can be recognised by Aibo is provided in section 2.9.

Summarising, any R-Code program will consist of a series of commands that will make Aibo hear, move and perform actions. Most of the actions and recognitions are already built inside R-Code so the programmer has just to make use of them. R-Code programs can be as complicated as you want!.

2.5 R-Code Commands

The following list contains an alphabetical list of all the R-Code commands and a brief description of their workings³:

- !* Break (force stop). Usage: *!*
- :* Label. Usage: *:<label name>*
- ADD* Add two values. Usage: *ADD:<var>:<value>* which performs *<var> = <var> + <value>*
- AND* Logical product. Usage: *AND:<var>:<value>* which performs *<var> = <var> & <value>*
- ARG* Retrieve subroutine argument from the stack into the variable. Note that arguments are retrieved first-in-first-out. Usage: *ARG:<var>*
- BREAK* Break out of loop. Usage: *BREAK*
- CALL* Call subroutine. Usage: *CALL:<label>{:<argc>}*. *<argc>* specifies the number of variables pushed onto the stack.
- CASE* Multiway-branching (Conditional execution). Usage 1: *CASE:<constant>:<command>*
Usage 2: *CASE:ELSE:<command>*
- CLR* Clear sensor variable. Usage: *CLR:SENSORS*
- CSET* Multiway-branching (Context value setting). Usage: *CSET:<value1>:<operator>:<value2>:<value3>*
- DIV* Divide. Usage: *DIV:<var>:<value>* which performs *<var> = <var> / <value>*
- DO* DO loop. Usage: *DO{:WHILE | UNTIL:<value1>:<operator>:<value2>}*. Requires the use of *LOOP*
- DUP* Copy the stack's top element and *PUSH* it onto the stack. Usage: *DUP*
- EDIT* Load program. Usage: *EDIT*
- END* Program load end point. Usage: *END*
- EQ* Checks if the two first values on the stack are equal, and pushes the result onto the stack. Usage: *EQ*

³A deeper description including examples can be found in the *rcode-ers7-cmdref-xxx_E.txt* file provided with the *Redist7.zip* file.

EXIT Terminate execution. Usage: EXIT

FOR FOR loop. Usage: FOR:<var>:<from>:<to>{:<step>}. Requires the use of NEXT command

GE Checks if the second value on the stack is greater or equal to the first one, and pushes the result onto the stack. Usage: GE

GET Displays variable value on the console (for debugging). Usage: GET:<var>

GLOBAL Global variable declaration. Usage: GLOBAL:<var>{:<init_value>}

GO Jump to the label. Usage: GO:<label>

GT Checks if the second value on the stack is greater than the first one, and pushes the result onto the stack. Usage: GT

HALT Terminate program and shutdowns Aibo. Usage: HALT

IF Conditional test. Usage 1: IF:<value1>:<operator>:<value2>:THEN ... ENDIF
Usage 2: IF:<value1>:<operator>:<value2>:CALL:<label> Usage 3: IF:<value1>:<operator>:<value2>:BREAK
Usage 4: IF:<value1>:<operator>:<value2>:<jump_to_if>{:<jump_to_ifnot>}

INIT Initialise R-Code. Usage: INIT:<init_level>

IOR Logical sum. Usage: IOR:<var>:<value> which performs <var> = <var> | <value>

JF POP the stack, and if the value is false, jump to label. Usage: JF:<label>

JT POP the stack, and if the value is true, jump to labelJump if stack top is true. Usage: JT:<label>

LAND Logical product (Boolean operator). Usage: LAND:<var>:<value> which performs <var> = <var> logical AND <value>

LE Checks if the second value on the stack is less or equal to the first one, and pushes the result onto the stack. Usage: LE

LET Assign (simple assignment). Usage: LET:<var>:<value>

LIOR Inclusive OR (Boolean operator). Usage: LIOR:<var>:<value> which performs <var> = <var> logical OR <value>

LNOT Negation (Boolean operator). Usage: LNOT:<var>:<value> which performs <var> = logical NOT <value>

LOCAL Local variable declaration. Usage: LOCAL:<var>{:<init_value>}

LOOP DO loop termination. Usage: LOOP{:WHILE | UNTIL:<value1>:<operator>:<value2>}

LT Checks if the second value on the stack is less than the first one, and pushes the result onto the stack. Usage: LT

MOD Remainder. Usage: MOD:<var>:<value> which performs <var> = <var> % <value>

MUL Multiply. Usage: MUL:<var>:<value> which performs <var> = <var> * <value>

NE Checks if the two first values on the stack are not equal, and pushes the result onto the stack. Usage: NE

NEXT FOR loop end point. Usage: NEXT

NONE No operation. Usage: NONE

NOT Negation. Usage: NOT:<var>:<value> which performs <var> = ~ <value>

ONCALL Register/Cancel interrupt routine . Usage 1: ONCALL:<v1>:<op>:<v2>:<label>[:<resume_type>:<res>]
Usage 2: ONCALL:<-n>

PLAY Play action. Usage: PLAY:ACTION:<action>{:<optional arguments>}. A second usage is PLAY:MWCID:<mwcid>{:<optional arguments>} where mwcid is

the ID of a command created by the user and specified in an ODA file (see chapter 4). Aibo has some MWCID actions already defined on it (check the /OPEN-R/APP/PC/AMS/ACTION.MS file for a complete list of them).

POP Remove element from stack into a variable. If var is not specified, the value is discarded. Usage: POP{:<var>}

PRINT Print for online debugging. Usage: PRINT:<format>{:<vars_to_print>}. See more on this command on section 2.10

PUSH Add element to stack. Usage: PUSH:<var>

QUIT Emergency stop. Usage: QUIT

REPEAT REPEAT loop. Usage: REPEAT. Requires the use of UNTIL

RESUME Return from interrupt routine . Usage: RESUME

RET Return from subroutine (context version). Usage: RET:<context>

RETURN Return from subroutine. Usage: RETURN{:<return_value>}. Returned value is retrieved by using the POP command

RND Generate a random number. Usage: RND:<var>:<from>:<to>. The random number seed is specified by SET:Seed:<seed>

RUN Begin execution of in memory program. Usage: RUN

SET Assign variables (with special functions) values. Usage: SET:<var>:<value>

STOP Normal stop. Usage: STOP

SUB Subtract. Usage: SUB:<var>:<value> which performs <var> = <var> - <value>

SWITCH Multiway-branching (Context value setting) . Usage: SWITCH:<var> Requires the use of CASE

SYNC External synchronisation. Stops execution until SYNC command is received. Usage: SYNC

UNTIL REPEAT loop termination. Usage: UNTIL:<value1>:<operator>:<value2>

VDUMP Display variable value (see section 2.10). Usage: VDUMP:<var>

VLOAD Load variable value from a file on the memory stick in the file /OPEN-R/APP/PC/AMS/<var name>.SAV. Usage: VLOAD:<var>

VSAVE Save variable value in a file on the memory stick in the file /OPEN-R/APP/PC/AMS/<var name>.SAV. Usage: VSAVE:<var>

WAIT Wait until end of preceding action or milliseconds. Usage: WAIT{:<ms>}

WEND WHILE loop end point. Usage: WEND

WHILE WHILE loop. Usage: WHILE:<value1>:<operator>:<value2> Requires the use of WEND

XOR Exclusive OR. Usage: XOR:<var>:<value> which performs <var> = <var> ^ <value>

2.6 R-Code relational operators

List of relational operators. They can be used when performing conditionals.

= Equals

== Is equal to

<> Not equal to

!= Not equal to

< Less than

<= Less than or equal to

> Greater than
 >= Greater than or equal to
 & Bitwise AND (logical multiplication)
 | Bitwise OR (logical addition)
 ^ Bitwise exclusive OR
 && AND/logical multiplication (operands treated as Boolean values: 0 treated as FALSE / other than 0 treated as TRUE)
 || OR/logical addition (operands treated as Boolean values: 0 treated as FALSE / other than 0 treated as TRUE)

2.7 Special system variables

Next is an exhaustive list of the system variables. They describe the status of the robot at any time, and can be checked or set to acknowledge the robot status and act consequently.

AiboId AIBO ID (0-255) 0 when not connected via WLAN. When connected via WLAN, least significant byte of IP address

AiboType returns the Aibo Model (7 for ERS-7)

Year Year (2000 or later)

Month Month (1-12)

Day Date (1-31)

Hour Hour (0-23)

Min Minute (0-59)

Sec Second (0-59) resolution 2 seconds

Dow Day of week (Sun(0), Mon(1), ..., Sat(6))

Seed Random number seed (default is 1)

Power Power (0/1): 0 OFF 1 ON Has no meaning for versions 1.1 and later

Status Status: 0 Normal startup 1 Recovery. If AIBO recovers from falling, the program is started again from the beginning. At this time, the value is set to 1

Context Context value

Wait Number of actions being waited for completion

Clock Clock (incremented by 1 every 32 ms)

Brightness Ambient brightness (0-255)

Face Face was detected (0/1)

Pink_Ball Pink ball (0/1)

Pink_Ball_H Pink ball horizontal angle [degrees]

Pink_Ball_V Pink ball vertical angle [degrees]

Pink_Ball_D Distance to pink ball [mm]. The origin point of the position of Pink ball is a position of Image sensor.

AIBONE AIBONE (0/1)

AIBONE_H AIBONE horizontal angle [degrees]

AIBONE_V AIBONE vertical angle [degrees]

AIBONE_D Distance to AIBONE [mm]

AU_Voice Voice recognition (0/1)

AU_Voice_ID Voice ID (1-53) see [Voice ID List]

AU_AiboSound AiboSound detection (0/1)

AU_AiboSound_ID AiboSound ID (1-35) see [AiboSound ID List]
AU_AiboTone AiboTone detection (0/1)
AU_AiboTone_ID AiboTone ID (1-68) see [AiboTone ID List]
Temp_Hi Temperature at which operation is suppressed (0/1) (for safety, a forced shutdown will be executed in 20 seconds).
Head_Tilt Head: vertical (up-down) angle 1 [degrees]
Head_Tilt_2 Head: vertical (up-down) angle 2 [degrees]
Head_Pan Head: horizontal (left-right) angle [degrees]
Mouth Mouth: Amount open [degrees]
LFLeg_1 Left-front leg J1 (hip joint): angle in front/back direction [degrees]
LFLeg_2 Left-front leg J1 (hip joint): angle in left/right direction [degrees]
LFLeg_3 Left-front leg J2 (knee joint): angle in front/back direction [degrees]
LRLeg_1 Left-hind leg J1 (hip joint): angle in front/back direction [degrees]
LRLeg_2 Left-hind leg J1 (hip joint): angle in left/right direction [degrees]
LRLeg_3 Left-hind leg J2 (knee joint): angle in front/back direction [degrees]
RFLeg_1 Right-front leg J1 (hip joint): angle in front/back direction [degrees]
RFLeg_2 Right-front leg J1 (hip joint): angle in left/right direction [degrees]
RFLeg_3 Right-front leg J2 (knee joint): angle in front/back direction [degrees]
RRLeg_1 Right-hind leg J1 (hip joint): angle in front/back direction [degrees]
RRLeg_2 Right-hind leg J1 (hip joint): angle in left/right direction [degrees]
RRLeg_3 Right-hind leg J2 (knee joint): angle in front/back direction [degrees]
Tail_Pan Tail: horizontal (left/right) angle [degrees]
Tail_Tilt Tail: Vertical (up/down) angle [degrees]
Batt_Rest Battery charge remainder [%]
Batt_Temp Battery temperature [C]
Distance_Cliff Distance to cliff [mm]
Distance Distance to obstacle [mm]
Head_ON Head sensor pressed (0/1)
Head_LONG Head sensor pressed for 3 seconds or more (0/1)
BackF_ON Front back sensor was pressed (0/1)
BackM_ON Middle back sensor was pressed (0/1)
BackR_ON Rear back sensor was pressed (0/1)
BackFR_LONG Front and rear back sensor and was pressed for 3 seconds or longer (0/1)
BackF_Jaw_LONG Front back and chin back sensor was pressed for 3 seconds or longer (0/1)
Back_Pat Back sensors were pet (0/1)
BackR_Hit Back sensor was hit (0/1)
Jaw_ON Chin sensor was pressed (0/1)
RFLeg_ON Right front paw sensor was pressed (0/1)
RFLeg_OFF Right front paw sensor was released (0/1)
LFLeg_ON Left front paw sensor was pressed (0/1)
LFLeg_OFF Left front paw sensor was released (0/1)
RRLeg_ON Right hind paw sensor was pressed (0/1)
RRLeg_OFF Right hind paw sensor was released (0/1)
LRLeg_ON Left hind paw sensor was pressed (0/1)

LRLeg_OFF Left hind paw sensor was released (0/1)

2.8 Aibo actions

The following is a list of all the available actions followed by a short description. Play any of the actions by using the command `PLAY:ACTION:<action>:<arguments>`

ACTION Description

SIT Sit

STAND Stand

LIE Lie down

WALK Walk. Arguments: <horizontal angle>:<distance>

STOP_WALK Stop walking

TURN Turn around. Arguments: <horizontal angle>

KICK Kick. Arguments: <horizontal angle>:<distance>

TOUCH Touch. Arguments: <horizontal angle>:<distance>

MOVE_HEAD Look in the specified direction. Arguments: <horizontal angle>:<vertical angle>

TRACK_HEAD Track an object. Arguments: <target>

SEARCH Search for an object. Arguments: <target>

SEARCH.HEAD.NORMAL Search in the current head direction. Arguments: <target>

SEARCH.HEAD.SLOW Search slowly in the current head direction. Arguments: <target>

SEARCH.HEAD.NORMALCENT Look forward and search. Arguments: <target>

SEARCH.HEAD.SLOWCENT Look forward and search slowly. Arguments: <target>

SEARCH.HEAD.LOWCENT Look down and search. Arguments: <target>

PALONE.AUTO.EAR Move both ears

PALONE.AUTO.EARSTOP Stop moving ears

PALONE.AUTO.TAILV Wag tail up and down

PALONE.AUTO.TAILH Wag tail left and right

PALONE.AUTO.TAILROT Rotate tail

PALONE.AUTO.TAILD Lower tail

PALONE.AUTO.TAILSTOP Stop moving tail

MOVE.HEAD.FAST Look quickly in the specified direction. Arguments: <horizontal angle>:<vertical angle>

MOVE.HEAD.NORMAL Look in the specified direction. Arguments: <horizontal angle>:<vertical angle>

MOVE.HEAD.SLOW Look slowly in the specified direction. Arguments: <horizontal angle>:<vertical angle>

MOVE.TURN.NORMAL Turn around

MOVE.TURN.SLOW Turn around slowly

MOVE.MOVE.NORMAL Walk

MOVE.MOVE.SLOW Walk slowly

CHGPOS.WALK.NORMAL Change to walking posture

CONTACT.RIGHT.TOUCH Sit and touch with right paw

CONTACT.RIGHT.TOUCH2 Crouch and touch with right paw
CONTACT.FRONT.HEAD Head the ball
CHGPOS.STATR.NORMAL Lie down
SMESS.NOTICE.NOTICE1 Debug notice 1
SMESS.NOTICE.NOTICE2 Debug notice 2
SMESS.ERROR.ERROR1 Debug error notice 1
SMESS.ERROR.ERROR2 Debug error notice 2
SMESS.MODE.CLEAR Clear debug indication
CLIFF_DETECT_ON Cliff detect on (default)
CLIFF_DETECT_OFF Cliff detect off

2.9 Aibo recognised words

The following is a list of the words recognized by Aibo. When the AU_Voice variable has a value of 1, it means that Aibo has recognized a word. The ID of this list is the value returned in the AU_Voice_ID variable by the recognition system in order to show to the programmer which word was recognized.

ID Word
 1 AIBO
 2 What's your name?
 3 Say hello
 4 Shake paw
 5 Morning
 6 Hello sentences sentences
 7 Good night
 8 See you
 9 How are you?
 10 Hey AIBO
 11 Thanks
 12 Sorry
 13 Cheer up
 14 Banzai
 15 That's right
 16 That's wrong
 17 Good AIBO
 18 Don't do that
 19 Let's play!
 20 Sing a song
 21 Dance
 22 Show time
 23 Pose for me
 24 Clown around
 25 Show off
 26 Say message
 27 Let's be secret
 28 Open sesame

29 Happy day
30 Stand up
31 Lie down
32 Sit down
33 Turn right
34 Turn left
35 Go forward
36 Go backward
37 Go ahead
38 Stop
39 Faster
40 Slow down
41 Pink ball
42 Right leg kick
43 Right leg touch
44 Left leg kick
45 Left leg touch
46 Ready set go
47 You won
48 You lost
49 Action one
50 Action two
51 Action three
52 Action four
53 Action five

2.10 Debugging

A good procedure when creating R-Code programs is to create them by using the wireless console. It allows the easy test and debugging of the program without having to write all the time in the memory stick. Also, by using the wireless console, you can see messages posted by your R-Code program on it.

To create a program using the wireless console, connect to the console as specified in section 2.2.3. Then write your program into a text editor. Once you have it finished, type in the wireless console the EDIT command. Copy the text in the editor and paste it into the console. Type in the console the END command. At this point your new program is installed in Aibo's memory. To execute the program, type the RUN command in the console.

To send messages to the console, R-Code provides two commands:

1. VDUMP: this command displays a variable's value on the console. The use is VDUMP:<var name>. This will display on the screen the sentence: <var name> = <var value>
2. PRINT: it displays a message on the console in a similar way to the printf() command of C. The use is PRINT:<format>:<var1>:...:<var5> where <format>

defines the format of the string as %d for decimal or %x for hexadecimal, and <var> are the names of the variables. The <format> parameter can include some text that clarifies the sentence being displayed. The text follows the the typical C description. For example:

```
PRINT:"The value1 is %d and the value2 is %d":x:y
```

3 Examples

Here you will find describes the workings of the two examples included in the R-Code SDK by Sony. They are called *Greeting* and *Maze*.

3.1 The Greeting.R code example

The Greeting.R sample code makes Aibo recognise voice and perform a greeting sample.

```
//----- // GREETING //-----
:START
CALL:1001 // calls an initialisation subroutine called 1001 (see bellow)
DO // when it returns from the subroutine, it starts a loop (DO..LOOP)
WAIT:1 // waits for 1 ms
IF:AU_Voice:=1:THEN // if voice has been recognized then
WAIT:1
SWITCH:AU_Voice_ID // looks for which voice command has been said
CASE:1:CALL:1003 // it has been said AIBO?. Then call subroutine 1003
CASE:6:CALL:1005 // it has been said HELLO?. Then call subroutine 1005
CASE:ELSE:CALL:1007 // if none of them, call a routine to play angry sound
CALL:1001 // call the reset routine
ENDIF
WAIT:1000 // wait one second
LOOP // end of the loop (DO..LOOP)
// RESET
:1001
PLAY:ACTION:STAND // makes Aibo stand up
WAIT // waits until Aibo has finished the previous action (important!!)
SET:AU_Voice:0 // initialises voice recognition variable to zero
RETURN
// AIBO word has been said
:1003
PLAY:ACTION:WALK:0:100 // walks some steps
WAIT // waits until Aibo has finished the previous action (important!!)
PLAY:ACTION:MOVE_HEAD:0:-50 // moves the head
WAIT // waits until Aibo has finished the previous action (important!!)
PLAY:ACTION:MOVE_HEAD:0:0// moves the head
WAIT // waits until Aibo has finished the previous action (important!!)
RETURN
// GREET. HELLO word has been said
:1005
// L_PHUMAN.M_GREET.S_NORMAL
PLAY:MWCID:2750 // plays a greeting sound
WAIT // waits until Aibo has finished the previous action (important!!)
RETURN
// ANGRY. The word has not been understood
```

```

:1007
// L_EDSP.M_ANGRY.S_SOUNDLED
PLAY:MWCID:2471 // plays the angry sound and led combination
WAIT // waits until Aibo has finished the previous action (important!!)
RETURN

```

3.2 The Maze.R code example

The Maze.R code, makes Aibo escape from a maze

```

//-----// MAZE //-----

:1000
PLAY:ACTION:STAND // makes Aibo stand up
WAIT // waits until the action has been performed
PLAY:ACTION:CLIFF_DETECT_OFF // disconnects detection of cliff
WAIT
DO // starts a loop
PLAY:ACTION:MOVE_HEAD:0:0 // moved head to the centre
WAIT
PLAY:ACTION:WALK:0:10000 // walks at horizontal angle 0 for 10 meters
FOR:t:1:1000 // starts a FOR loop
IF:Distance:<:300:BREAK // If AIBO finds a wall in less than 300 mm or t=1000
then break
WAIT:1
NEXT // end of FOR loop
PLAY:ACTION:STOP_WALK // stops walking
WAIT
PLAY:ACTION:MOVE_HEAD:90:0 // moves head to one side
WAIT // waits until the action has been finished (very important!!)
PLAY:ACTION:MOVE_HEAD:-90:0 // moves head to the other side. In this case
it does not wait for the conclusion
// this implies that Aibo's head will perform a scanning of distances form one side
to the other
SET:dd:0
WHILE:Wait:>:0 // starts WHILE loop. It will continue until the last action has
been finished
SET:d:Distance // sets d variable with the distance sensed
SET:p:Head_Pan // sets p variable with the value of the head pan angle
IF:d:>:dd:THEN
SET:dd:d // dd contains the largest distance to a wall
SET:pan:p // pan contains the pan angle to that distance
ENDIF
WEND // end of WHILE loop
VDUMP:dd
VDUMP:pan

```

```
IF:dd:<:300:THEN // If dd is less than 300, then it means that there is nowhere to
go (turn 180)
PLAY:ACTION:TURN:180 // then Aibo turns 180 degrees
WAIT
ENDIF
PLAY:ACTION:MOVE_HEAD:pan:0 // If found an exit, then moves the head on
that direction
PLAY:ACTION:TURN:pan // and then moves the body on that direction
WAIT
LOOP // end of the loop
```

4 Creation and use of contents files

Even that the base R-Code environment comes with a lot of sequences of movements, LED patters and sounds, it is possible that you create your owns and to include them in your R-Code programs. For example, you can create your own dance movements using a motion editor, and then invoke that dance in your R-Code program using the PLAY command. All the motions, sounds and LED patterns used for your dance movement are called the contents files. This section describes how to create and use them.

The basic steps to create and use such contents is the following:

1. Create the contents files. This depends on what you want to use on your program. Motions are described in MTN files (*.mtn), LED patterns are created in LED files (*.led) and audio can have wave or midi files (*.wav,*.mid).
2. Convert the contents files to ODA files.
3. Edit a command list file, called MWC.CFG, that will contain a list of all the new contents.
4. Convert the command list to ERS-7.MWC.
5. Copy the ODA and ERS-7.MWC to the memory stick.
6. Invoke the contents you created from your R-Code program using the PLAY command.

There are several tools available for the creation of the content files. This text will concentrate on the use of the official tools released by Sony (when available), but points to the other third-part tools will be also given.

4.1 Creating contents files

There are three types of contents files to be created: motion, audio and LED patterns.

4.1.1 Creating motion files

Motion files describe sequences of movements of Aibo's joints and have *.mtn extension. These sequences must be designed by the programmer using a motion editor program. There are several motion editor for Aibo available, like for example AIBO Master Studio (released from Sony) or Skitter (that can be found at <http://www.dogsbodynet.com> for free). For the creation of motions for Aibo ERS-7, Sony released for free the Motion Editor (MEdit) program that can be downloaded from its web site.

To use the MEdit program, download it from openr.aibo.com and install it. It is only available for Windows systems, but it works well under Linux if started using Wine⁴ (see figure 2).

⁴Wine is a free Windows emulator for Linux that allows the execution of Windows programs under Linux. You can find it with instructions of use at <http://www.winehq.com>

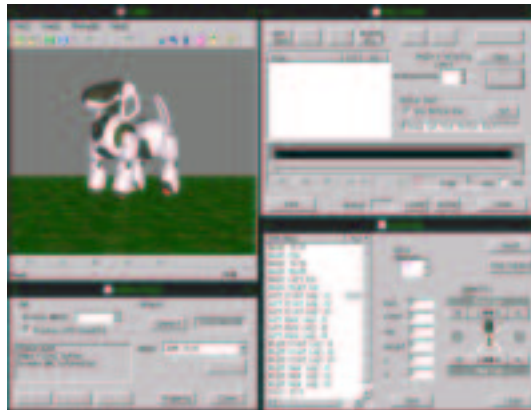


Figure 2: The MEdit program running under Linux

Once installed, use the MEdit program to create your motion contents. It is not described here how to use that program. You will find a very good tutorial included with the program.

4.1.2 Creating audio files

Audio files contain the sounds and music that Aibo can reproduce, and are saved with *.wav and *.mid extensions. You can create two types of audio files: MIDI or WAV. Sony doesn't provide any special tool for the creation of those contents, but you can find lots of free applications on the Internet to create those files. Just keep in mind the following considerations when designing WAV and MIDI files:

1. WAV files can be coded in the following formats:
 - 8k 8bits MONO PCM
 - 16k 16bits MONO PCM
 - 8k 4bits MONO IMA ADPCM
 - 16k 4bits MONO IMA ADPCM
2. MIDI files must be coded in the following format:
 - SMF Format0

If you don't have an audio program to generate your files, you can use the Skitter motion editor program. This motion editor program has a built in editor for the handling of Aibo sounds.

4.1.3 Creating LED files

LED files have sequences of LEDs that Aibo can reproduce, and they are saved with *.led extension. LED files are saved in the same format as MIDI files, so you can use your MIDI generator application to generate your LED patterns. LED files must be saved in SMF Format0 with only track 1 as effective. You can also use the Skitter performance editor to create your LED files, since it has a built in LED editor.

4.2 Converting to ODA files

This step will convert your MTN, WAV, MIDI and LED files to MOTION.ODA, AUDIO.ODA and LED.ODA files.

1. In order to convert the contents files to ODA files you must create first a directory for each type of content: a MOTION directory, an AUDIO directory and a LED directory. Store in each directory the contents files related, and then run the RTool.exe program, downloaded from the OPEN-R web site.
2. Select the ODA tab and select the MOTION, AUDIO, LED and Output Path directories to the ones created by you on the previous step. The Output Path is the place where the final ODA files will be placed by RTool.
3. Select which ODA files do you want to create and click *Make ODA* to create the files.

4.3 Create the MWC file

The MWC file is the file that will describe all the new motions, sounds and LED patterns that you have created by assigning to each one an identifier (ID). This ID will be the number to use in your R-Code program when trying to play an action (PLAY:MWCID:<your ID>). The MWC file required is called ER-7.MWC. To create that binary file, first you must create a text file called MWC.CFG (this is the command list) and then convert it to the binary one by using RTool. An example of the MWC file is the following:

```
#MWC 1.0
1 3
26000 3
cmagentMOTIONPERFORMER a_stand#stand_so0r_greet 1 1 0x0 0x0 0
cmagentSOUNDPERFORMER soc_d00greetso0r_x1x 1 1 0x0 0x0 0
cmagentFACELIGHT sol2_d00greetso0r_l1f 1 1 0x0 0x0 0
```

The steps to create such file are the following:

1. Creation of a MWC.CFG file. You can create a new one or modify an existing one. Follow the steps below about how to create a new one:

- (a) Each new command should have a unique ID to use with the PLAY command of R-Code. New commands should have an ID starting from 26000 and up to 27999.
- (b) The MWG.CFG file will have the following format:

```
#MWC 1.0
Num1 Num2
Num3 Num4
Text1 Text2 Num5 Num6 Num7 Num8 Num9
```

where NumX and TextX have the following values:

- Num1: is the number of MWCommands included in the file
- Num2: is the number of CMAgent Commands included in the file
- Num3: is the ID of the following command defined in the file (from 26000 to 27999)
- Num4: is the number of CMAgent Commands in the same MWCommand
- Text1: is the type of CMAgent Performer. It depends on the type of contents that is going to reproduce, and follows the next list
 - cmagentMOTIONPERFORMER All body motion
 - cmagentMOUTHPERFORMER Mouth motion
 - cmagentHEADPERFORMER Head motion
 - cmagentLEGPFORMER Legs motion
 - cmagentTAILPERFORMER Tail motion
 - cmagentEARPERFORMER Ear motion
 - cmagentSOUNDPERFORMER Sound
 - cmagentMODELIGHT Mode LED
 - cmagentFACELIGHT Face LEDs
 - cmagentEARLIGHT Head LEDs
 - cmagentBACKLIGHT Back LEDs
 - cmagentLIVELIGHT Wireless LAN LED
- Text2: is the filename of the CMAgent command. Usually is the content file name without the filename extension
- Num5: number of times the CMACommand must repeat. Usually is 1
- Num6: Synchronous CMACommand. If one command has one content, it should be 0. (No Sync) If one command has two or more contents, it should be 1. (Sync)
- Num7: parameter (Internal use). It should be 0x0
- Num8: parameter 0x01=this command can be stopped at anytime. 0x0=this command cannot be stopped until the end. Recommend 0x0. You should be careful when making stoppable motions.

- Num9: it should be 0
2. To start conversion, initiate RTool and select the *Input file* (MWC.CFG), the *Output file* (ERS-7.MWC) and the *Base file* (this is a file called BASE-7.MWC provided by RTool located under the MWC/ directory).
 3. Click the Make MWC button. The ERS-7.MWC binary file will be created.

4.4 Copying files to the stick

Once all the files have been created, they must be saved onto the memory stick in order to be able to use them in your programs. Copy MOTION.ODA, LED.ODA and AUDIO.ODA to the OPEN-R/MW/DATA/P/ directory of the memory stick. Copy the ERS-7.MWC file to the OPEN-R/MW/CONF/ directory of the memory stick.

Now you can use your new content created by calling it with the PLAY:MWCID:<your ID> command inside your R-Code program or from a telnet console.