

URBI Script Quick reference for URBI 1.5 version

Book based on urbi kernel version 1.5

SHA1: 51997e, Jun-28-2008

(Book compiled from 682M)

Gommard Teddy

URBI Script Quick reference for URBI 1.5 version : Book based on urbi kernel version 1.5: SHA1: 51997e, Jun-28-2008 : (Book compiled from 682M)

by Gommard Teddy

Publication date

Copyright © 2007-2008 Gostai

This document is released under the Attribution-NonCommercial-NoDerivs 2.0 Creative Commons licence (<http://creativecommons.org/licenses/by-nc-nd/2.0/deed.en>).

Table of Contents

1. Introduction	1
2. Descriptions	2
Generality	2
Time Operation	2
Modifier	2
Tag, channels and flags	3
Variables	3
Groups	5
Property	6
Operation	6
Derivate, Normalize	7
Operators	7
Tests and loop	8
Event	10
File Access	11
Commands	12

List of Tables

2.1. Time Operation	2
2.2. Modifier	3
2.3. Tag and Flags	3
2.4. Variables types	4
2.5. Variables of type List	4
2.6. Variables of type String	4
2.7. Variables of type Array	5
2.8. Other commands for variables	5
2.9. Defined constants	5
2.10. Groups	5
2.11. Property	6
2.12. Operation	6
2.13. Derivate, Normalize	7
2.14. operators	7
2.15. Tests	8
2.16. isxxxx();	8
2.17. Control and Loops	9
2.18. Events	10
2.19. Events Managment	11
2.20. File Access	11
2.21. Commands	12

Chapter 1. Introduction

Introduction

This documentation is a Quick reference to all Urbi commands supported by Urbi Engines.

You may be interested in reading other Urbi documentation here: <http://www.gostai.com/doc> [<http://www.gostai.com/doc>] if you are not familiar with it yet.

All these commands need to be entered in a "connection", a client of an Urbi server. The server itself can have a connection ID if it is run with `-i` option, giving ability to be **I**nteractive.

Goal of this document is to concentrate all the command you can enter in an Urbi connection.

This document is just a memo of all commands, find more description of commands on *Official Urbi documentation page* [<http://www.gostai.com/doc>] or *forum.gostai.com* [<http://forum.gostai.com>].

Chapter 2. Descriptions

Generality

Default port for Urbi engine is TCP port 54000.

Standard Header for a connection is :

```
*** *****
*** URBI Language specif 1.0 - Copyright (C) 2005-2008 Gostai SAS
*** URBI Kernel version 1.5 rev. 2bec015
***
***   URBI Engine version 1.5 rev. 1cf8ce9
***     (C) 2006-2007 Gostai SAS
***
*** URBI comes with ABSOLUTELY NO WARRANTY;
*** This software can be used under certain conditions;
*** see LICENSE file for details.
***
*** See http://www.urbiforge.com for news and updates.
*** *****
*** ID: U135518192
```

Time Operation

Table 2.1. Time Operation

Urbi command	description	comment
A ; B	execute first A, then B	B starts after A (no time constraint, see)
A , B	begin executing A, then B	B starts as soon as it is possible after A (A can be not fully ended)
A & B	A and B start at the same time	this command returns when A AND B are ended
A B	B starts immediately after A	There is no delay between the end of A and B
{A ; B , C & { D E}};	group commands is available with "{" and "}"	
noop;	do nothing	usable if you need to give time for System
time();	return time in ms	returns the server server execution time

Modifier

Modifier applied on a command

```
motor.val = 5 time:500ms;
```

```
motor.val = 34 time:400ms;
```

Table 2.2. Modifier

Urbi command	description	comment
time:200ms	command executed in 200ms	default unit is millisecond (ms), you can enter second (s)
time:200ms adaptative:3	calcul is done again every step (here 3)	this helps if trajectory is not followed correctly
smooth:200ms	smooth profile for a trajectory	example: x=0; x=100 smooth:100ms;
speed:20	set speed for the command	unit is: unit/second
accel:5	set acceleration for the command	unit is: unit/second/second
sin:50ms ampli:a phase:p getphase:g,	sin, cos... modifier	motor.val = myCenterValue sin:myPeriod ampli:myAmplitude,.
off	change load property to 0	motors off; is same as motors.load = 0;
on	change load property to 1	motors on; is same as motors.load = 1;
switch	change load property to the inverse value	motors switch; is same as motors.load = ! motors.load ;

Tag, channels and flags

Table 2.3. Tag and Flags

Urbi command	description	comment
myTag : sleep (2s),	tag a command with myTag	This allows to freeze or stop the command
myChannel << a;	Send command output in given channel	Useful for message filtering (liburbi)
myTag +begin : command;	display "*** begin at begin of command	useful for debugging tag
myTag +end : command;	display "*** end" at end of command	
myTag +report : command;	display "*** begin" at begin and "*** end" at end	same as +begin + end
myTag +error : command;	notify for errors	
myTag +connection(s) : command;	run command on the connection "s", "all", "other", "Uxxxxxx"	's' is a string, this flag will deasappear in futur versions !
myTag +bg : command;	run in background	
taglist;	return the list of tags	format: !!! for error, ### for warning, *** for information

Variables

You can force the declaration of variable with the **strict** word:

```
strict;
```

and disable it by **unstrict** word:

```
unstrict;
```

Variables are declared with the **var** word:

```
var myVariable = 5;
```

Undefining a variable is enable by **undef** word:

```
undef myVariable;
```

Deleting a variable is enable by **delete** word:

```
delete myVariable;
```

undef and **delete** commands have the same behavior. They will be unified in future versions.

Table 2.4. Variables types

description	type
var i = 48.0000;	float type
var myList = [1,2,"fe"];	list type
var myStr = "hello"; myStr = myStr + "world";	string type
var myBinary = <params>BIN <size><data>;	binary type
myObj = OBJ [x:1, y:2]; //DO NOTHING IN URBI 1.0	object type
myObj = new myConstructorObj(param);	object type

Table 2.5. Variables of type List

command	description
var emptyList=[];	load variable with an empty list
myList = [11,22] + [32,44,11] + "hello";	add elements to a list
head(myList);	return the 1st element
tail(mList);	return the last elements of the list
size(myList);	return number of elements of the list
getIndex(list, index);	return element a given index in given list

Table 2.6. Variables of type String

command	description
strlen(myString);	return integer size of the string
var mySubString = strstr(myString, pos, 3);	return a string from position "pos", of length 3

Table 2.7. Variables of type Array

command	description
<code>var myArray[12] = 4;</code>	load 12th element of array with 4
<code>var myArray__12 = 4;</code>	same as above, load 12th element of array with 4
<code>var myMultyArray[12][5] = 4;</code>	bidimensionnal array are available
<code>var myArray["name"] = "Ted";</code>	Array indexed by string instead of integer, same as <code>myArray_name = "Ted";</code>
<code>var s = "hello \$name" name:"John";</code>	string with parameter

Table 2.8. Other commands for variables

command	description
<code>vars;</code>	return list of all variables
<code>uservars;</code>	return list of variables for user
<code>var global.myUVar = 5;</code>	declare a global variables, for all
<code>\$("global.myUVar");</code>	return the variable
<code>%global.myUVar;</code>	return a string with the variable name
<code>myVarB = copy myVarA;</code>	do a hard copy of variable, not a pointer to it. Useful for binaries.
<code>(static x == 1)</code>	variable is locally static, if changed in another location, stay unmodified
<code>var mybin = bin 2048 wav 2 16000 16 1; #####2048 data#####</code>	declaration of a binary UVar.

Table 2.9. Defined constants

var	constant
<code>pi;</code>	return 3.14159
<code>true;</code>	return 1
<code>false;</code>	return 0
<code>on;</code>	return 1
<code>off;</code>	return 0

Groups

Table 2.10. Groups

type	description
<code>group myGroup;</code>	create an empty group named "myGroup"
<code>group myGroup { a1, a2 };</code>	create a group named "myGroup" with a1 and a2 inside
<code>addgroup myGroup { c, d };</code>	Add c and d elements to group named "myGroup"
<code>delgroup myGroup { a1, d };</code>	remove a1 and d elements from group named "myGroup"
<code>for m in group myGroup { echo (m); };</code>	parse a group named "myGroup" for list of its elements

Property

Properties are accessible by "->" operator.

Table 2.11. Property

command	description
<code>myVar->rangemin = -30;</code>	set minimal value for variable
<code>myVar->rangemax = 128.9;</code>	set maximal value for variable
<code>myVar->speedmin = 2.5;</code>	set minimal speed for change in variable
<code>myVar->speedmax = 5.9;</code>	set maximal speed for change in variable
<code>myVar->unit = "cm/s";</code>	set unit information for variable. This is for information purpose only. Will be implemented in Urbi 2.
<code>myVar->delta = 1.0;</code>	set delta for tolerance in the <code>=~</code> operator
<code>myVar->blend = "normal";</code>	set blend mode; normal, mix, add,queue, discard, cancel are possible values.

Operation

Table 2.12. Operation

Urbi command	description
<code>sin(pi);</code>	sinus
<code>asin(pi/5);</code>	arcsinus
<code>cos(2*pi/3);</code>	cosinus
<code>acos(0.4);</code>	arccosinus
<code>tan(0.4);</code>	tangent
<code>atan(0.4);</code>	arctangent
<code>atan2(y,x);</code>	arctangent with 2 variables
<code>exp(3.4);</code>	exponential
<code>log(3);</code>	logarithme
<code>round(3.5);</code> returns 4	round to up value
<code>trunc(3.9);</code> returns 3	trunc a number
<code>random(8);</code> returns a number between 0 and 8	return a random number
<code>sqr(5);</code> returns 25	square value
<code>sqrt(49);</code> returns 7	square root
<code>abs(-49);</code> returns 49	absolute value

Derivate, Normalize

Table 2.13. Derivate, Normalize

Urbi command	description
<code>x'n;</code>	normalize value between 0 to 1 (you need to set rangemin and rangemax properties first)
<code>x';</code>	1st theoretical derivate
<code>x'';</code>	2nde theoretical derivate
<code>x'd;</code>	1st real derivate
<code>x'dd;</code>	2nde real derivate
<code>x'e;</code>	error estimated between theory and reality
<code>x'in;</code>	inout property, on a USensor, the measured value
<code>x'out;</code>	inout property, on a USensor, the desired value

Operators

Table 2.14. operators

type	description	example
<code>&&</code>	logical AND	
<code> </code>	logical OR	
<code>!</code>	not	<code>!1;</code> return 0
<code>+</code>	add	<code>3 + 2;</code> return 5
<code>-</code>	substract	<code>3 - 2;</code> return 1
<code>=</code>	affection of value	<code>a = 5;</code> var 'a' as the value 5
<code>--</code>	decrement by 1	<code>a = 5; a--;</code> a is 4
<code>++</code>	increment by 1	<code>a = 5; a++;</code> a is 6
<code>+=</code>	increment by value	<code>a = 5; a += 6;</code> a is 11, equivalent to <code>a = a + value;</code>
<code>-=</code>	decrement by value	<code>a = 5; a -= 6;</code> a is -1, equivalent to <code>a = a - value</code>
<code>*</code>	multiply	<code>3 * 2;</code> return 6
<code>/</code>	divide	<code>3 / 2;</code> return 1.5
<code>**</code>	power, equivalent to math: ^	<code>3**4;</code> return 81, <code>3*3*3*3</code>
<code>%</code>	modulo, return the entire part of divide	<code>15 % 4;</code> return 3
<code>~=</code>	Difference between operands is # system.epsilon (defaults to 0.0001)	
<code>sgn()</code>	<code>sgn(value);</code> give sign of value; return 1 if positive, -1 if negative	<code>sgn(-4);</code> <code>sgn(5);</code>

Tests and loop

Table 2.15. Tests

type	description	example
A B	OR between command	if (A B) {};
A && B	AND between command	if (A && B) {};
A =~ B	return 1 if A is env equal to B, need delta property (A->delta and B->delta)	if (A =~ B) {};
A == 3 ~ 5s	return 1 if A equal 3 during 5s mini	if (A == 3 ~ 5s) {};
==	test an equality, useful for if test	1 == 2; return 0, 1 == 1; return 1
!=	test a non equality, useful for if test	1 != 2; return 1, 3 != 3; return 0
>	test a sup, useful for if test	1 > 2; return 0, 3 > 2; return 1
>=	test a sup or equal, useful for if test	1 >= 2; return 0, 3 >= 3; return 1
<	test a inf, useful for if test	1 < 2; return 1, 3 < 3; return 0
<=	test a inf or equal, useful for if test	1 <= 2; return 1, 3 <= 3; return 1

isxxxx() function enable to test type of an UVar.

Table 2.16. isxxxx();

command	description
isunknown();	
isnum();	
isstring();	
isbinary();	
isimage();	
issound();	
isvoid();	test if uvar is void or not : return 1 if void, 0 if value is valid
islist();	
isobject();	
isfunction();	
isdef(myUVar);	return 1 if myUVar is defined, 0 else
isfile();	
isvariable();	

Table 2.17. Control and Loops

command	description	note
if(test){ ... } else{ ... };	if structure	see tests table for "test" examples
while(test){ ... };	while structure	
while (test){ ... };	while structure	pipe all instructions in ONE cycle, quicker
for (i=0 ; i< 5 ; i++){ ... };	for structure	
for (i=0 ; i< 5 ; i++){ ... };	for structure	pipe all instructions in ONE cycle, quicker
for &(i=0 ; i< 5 ; i++){ ... };	for& structure	launch all the command in parallele, as a & between all of them
loop{ ... };	loop structure	infinite loop of instructions
loopn(10){ ... };	loopn structure	finite loop of 10 iterations in this case
loopn & (10){ ... };	loopn & structure	finite loop of 10 iterations in this case, anded
loopn (10){ ... };	loopn structure	finite loop of 10 iterations in this case, piped

Event

Table 2.18. Events

command	description	note
<code>emit myEvent;</code>	send an event that can be caught by at	
<code>emit(10s) myEvent;</code>	send an event that can be caught by whenever . This event has a 10 seconds duration	
<code>emit(5h3m25s) myEvent;</code>	same with event during 5 hours 3min 25s	
<code>at(event){ ... } onleave{ ... };</code>	at event is true, execute code	run once the code, when event is true, event can be a condition or an emitted event from emit
<code>whenever(event){ ... } else{ ... };</code>	whenever event	do instruction whenever event is true
<code>emit myEvent(1,3);</code>	send an event that can be caught by at or whenever with parameters	parameter can be caught by variables in at/whenever structure to read them
<code>at(myEvent(x,y)){ ... } onleave{ ... };</code>	at special event	entry once in the event with 2 parameters caught in 'x' and 'y' UVars
<code>waituntil(condition);</code>	lasts until condition is true	any following command will be delayed indefinitely
<code>every(Time){ ... };</code>	execute block of instruction every time	every(10s){ emit test;echo("send!"); }
<code>timeout(Time){ ... };</code>	abort instruction if not executed within given time	
<code>sleep(100ms);</code>	sleep	wait for 100 milliseconds

Table 2.19. Events Management

command	description
stop mytag;	stop code associated to tag "mytag"
stopif (condition){...};	run command and stop if condition is reached
block mytag;	block code associated to tag "mytag". Any "mytag" code is stoped, and future "mytag" code will be discarded
unblock mytag;	unblock code associated to tag "mytag, previously blocked by block comand
blockif(test) mytag;	if test is true, block tag ""mytag"
freeze tag;	freeze tag ""mytag", can be unfreeze by unfreeze
unfreeze tag;	unfreeze tag ""mytag" previously freezed by freeze command
freezeif(test){ ... };	run command and if test is true, freeze code
stopall;	stop all comannnds
inherits;	add an inheritance link to another class
disinherits;	remove inheritance from given class

File Access

Table 2.20. File Access

command	description	example
load(file);	load file with "file" name, and execute its code (.u files)	load("../myFile.u"); load("c:/test/myFile2.ini");
save(file,s);	save string s in file "file"	save("myFile.txt", "my data to be saved");
save(file,myVar);	save UVar myVar in file "file"	save("image.jpg", camera.val);
loadwav("./lorie.wav");	return a BIN with the sound	var mySound = loadwav("lorie.wav");

Commands

Table 2.21. Commands

command	description	example
eval(StringInstruction);	evaluate an expression and return result. Deprecated, use exec.	eval("a=1;");
exec(StringInstruction);	execute an expression	exec(" echo(\"test\"); ");
cpuload();	return cpu load, value between 0 and 1 (1 for full)	
freemem();	return free memory, value in bytes	
power();	return power status, 0 if empty, 1 when full	
ping;	return "pong" when executed	just test the connection alive state
connections;	list all connections to the server	
killall s;	Stop all code of the named connection	killall U12345678;
disconnect s;	disconnect connections named "s"	s = "Uxxxxxx";
quit;	close current connection	
shutdown;	server quit, this shutdown the server and so kill all connections	
reboot;	reboot robot	
reset;	reset the connection, by loading URBI.INI / CLIENT.INI	
commands;	return debug informations	
setpriority 20;	set priority of a connection, in URBIRT.INI file	