

URBI doc for LEGO Mindstorms NXT

(book compiled from Revision 418M)

**Gostai™
Benjamin Renoust
Benoit Pothier**

URBI doc for LEGO Mindstorms NXT: (book compiled from Revision 418M)

by Gostai™, Benjamin Renoust, and Benoit Pothier

Published

Copyright © 2006-2008 Gostai™

This document is released under the Attribution-NonCommercial-NoDerivs 2.5 Creative Commons licence (<http://creativecommons.org/licenses/by-nc-nd/2.5/deed.en>).

Table of Contents

1. Introduction	1
2. Launching URBI for Mindstorms NXT	2
.....	2
.....	2
3. First steps with URBI to control Mindstorms NXT	3
Make basic movements	3
Improving the movements	3
Reading sensors	4
Tagging commands	4
Playing sounds	5
Cyclic moves	5
Parallelism	6
Using functions	6
Loading files	7
Conclusion	7
4. Default layout reference	8
Introduction	8
Motors	8
Sensors	8
Bumper	8
Sonar	8
Decibel	8
Light	9
Battery	9
Beeper	9
Command	9
5. How to make its own layout	11
Introduction	11
Instancing Motors	11
Instancing sensors	12
Other devices	12
A. Available UObject Devices	14
Servo	14
UltraSonicSensor	14
SoundSensor	15
LightSensor	15
Switch	16
Battery	17
Beeper	17
Command	17
Instances	18
Groups	19
B. Copyright	20

List of Tables

A.1. Servo's attributes:	14
A.2. UltraSonic Sensor's attributes:	15
A.3. Sound sensor's attributes:	15
A.4. Light sensor's attributes:	16
A.5. Switch's attributes:	16
A.6. Battery's attributes:	17
A.7. Beeper's methodes:	17
A.8. Command's methodes:	18
A.9. TriBot.ini layout	18
A.10. Group's list	19

Chapter 1. Introduction

This documentation contains informations about the Lego Mindstorms NXT URBI engine.

The first two chapters provide a short tutorial for using URBI to control the TriBot robot.

The remainder of the documentation contains an exhaustive reference for the Lego Mindstorms NXT URBI engine devices.

Chapter 2. Launching URBI for Mindstorms NXT

First you should build the *tribot* model from the Lego Mindstorms NXT basic model.

Then install the NXT driver you can find on the official NXT CD to your PC.

Synchronize your robot with your computer if you want to control your NXT by bluetooth, or make it recognized by your computer if you want to control it by USB.

Attention: in case you want to use bluetooth make sure you use bluetooth as recommended on the LEGO NXT website [<http://mindstorms.lego.com/Overview/Bluetooth.aspx>]. This is a known issue in LEGO fantom library (low COMx number allowed only). If you don't use this bluetooth driver, the Urbi engine for mindstorm NXT will not work.

To install the URBI engine for Mindstorms NXT, run the installer and follow instructions.

Then launch the URBI server (through the start menu) which will interpret your commands and control the robot accordingly.

Finally, launch a program to be able to send commands to and receive information from the URBI server. Different programs may be used for that, for example you will find `URBI_Console` in the `gostai` menu.

Then click on the `Connect to LEGO Mindstorms NXT` button of the `URBI_Console`.

You are now ready to start programming with URBI in the `URBI_Console`. You can try to launch single commands through the command line at the bottom of the interface or load files using the specific button.

Chapter 3. First steps with URBI to control Mindstorms NXT

This section is a short tutorial for using URBI with the Lego TriBot model with the default configuration provided with the URBI server. Users familiar with URBI should look directly to the next section for an extensive list of the available devices.

Make basic movements

We will first move the wheels of the robot.

In URBI, all the motor's speed and the sensor's value in a robot are associated with variables. You can set the motor's speed or get the sensor's value by assigning or reading the values of the corresponding variables.

In the default server layout for the TriBot model, the left wheel is assigned to variable `wheelL` and the right wheel is assigned to `wheelR`¹. The values of these variables control the corresponding wheel speed. First put the TriBot upside-down to avoid any accident, then you can move the left wheel just doing:

```
wheelL.speed = 50;
```

Don't omit the semicolon at the end of the line, or the command will not be executed. You can also move the right wheel (a negative value is a move backward) :

```
wheelR.speed = -50;
```

A *group* is also defined so that you can give orders to several motors at the same time. In our case, the group `wheels` contains both robot wheels. Setting `wheels` to a value will set both wheels to this value :

```
wheels.speed = 0;
```

will stop both wheels.

The third motor of the TriBot is associated with the `claw` variable.

Improving the movements

The commands given so far simply set a speed to the wheels, but you can use URBI to make precise sequences of movements. It is for example possible to set a speed for a given time, then stop.

To achieve that, URBI provides a command to wait for a given time : `wait(duration)`. And you can make sequences of commands separated with semicolon that will be executed one after another. So the following line :

```
wheels.speed = 50; wait(10s); wheels.speed = 0;
```

will make the robot to go forward during 10 seconds then stop.

It is also possible with URBI to control the way the values will change. In the previous examples, the motor speed goes from 0 to 50 as fast as possible. You can use a *modifier* to the variable assignation to control the way the variable value will change. For example, if you want to reach a value in a given time, use the `time:modifier` :

¹ To be precise, sensors and motors are associated with *objects*. The left motor is associated with the `wheelL` object and the left motor speed is associated with the `wheelL.speed` attribute of this object.

```
wheels.speed = 50 time:3s; wheels.speed = 0 time:3s;
```

Using this code, the robot will accelerate to reach the speed of 50 after 3 seconds, then slow down to stop after another 3 seconds.

Other modifiers are available in URBI, they are described in the URBI tutorial available at <http://www.gostai.com/doc.php>.

Reading sensors

In all the previous steps the commands don't take the environment into account. To do that, you need to get the sensor values. In URBI, you just have to type the associated variable name. For example when you type :

```
sonar.val;
```

The server returns a message with the ultrasonic sensor value :

```
[146711:notag] 48.000000
```

In this message, 146711 is the time (from the server clock) at which the value was read, `notag` is the tag of the command, which will be described in the next section. 48.00000 is the value of the sonar sensor, in this case the distance to the obstacle in front of it.

You also need a way to react to the sensor values. In URBI, this is achieved using *event catching commands*. These commands check some condition and launch other commands when these conditions are verified. For example the following command :

```
at (sonar.val < 50) wheels.speed = 0;
```

will stop the robot if an obstacle is detected at less than 50 centimeters. Note that this command remains active in the server. If you move the robot and set a forward speed :

```
wheels.speed = 50;
```

the robot will stop again when another obstacle is encountered.

The other sensors available for the TriBot are the sound sensor `decibel`, the bumper sensor `bumper` and the light sensor `light`. The following command :

```
at (bumper.val == 1) wheels.speed = -50;
```

will make the robot go backward if something hits the bumper.

Note that the `at` command reacts only when the condition *becomes* true. If you want to loop the execution of an action *whenever* the condition is true, use the `whenever` command (see an example in the section called "Playing sounds").

Tagging commands

The `at` commands you entered before are still active in the server. You need a way to stop them if you want to do something else. The tagging mechanism will enable you to do that.

A tag in a name associated to a command. To tag a command with the name `mytag`, simply prefix the command. For example :

```
mytag: at (decibel.val > 0.7) wheels.speed = 0;
```

will launch the event catching command using the decibel sensor with tag `mytag`.

Later you will be able to stop that command :

```
stop mytag;
```

will stop the previous command and the robot will not react to sound anymore.

The default tag when you don't specify one is `notag`. Therefore :

```
stop notag;
```

will stop all the commands you type since the beginning of this tutorial.

It is also possible to freeze temporarily a command :

```
freeze mytag;
```

will stop the commands tagged with `mytag`, but it will not be deleted, and

```
unfreeze mytag;
```

will resume the command.

Playing sounds

The Lego Mindstorms NXT is able to play sounds. The object `beeper` is associated with this capacity. However, you can't just assign a value to the variable `beeper`, because playing a sound requires several parameters. You therefore have to call a *method* of the beeper object :

```
beeper.play(200, 3s);
```

This will play a beep at 200Hz during 3 seconds (beep's frequency must be between 200Hz and 14000Hz).

The parameters of the method may also be the result of operations, or depend on other variables. For example, it is possible to play a sound which frequency depends on the obstacle distance :

```
mytag:whenever (sonar.val < 100) beeper.play(200+6*sonar, 3ms);
```

This plays beeps with a lower frequency as the obstacle comes closer (under 1 meter).

Cyclic moves

We now introduce a modifier in URBI that makes it possible to do cyclic moves of sinusoidal shape. This modifier has the particularity to make the assignation never terminate. To be able to enter new commands after this modifier is used, you need to put this command in *background* so that the URBI server continues processing new commands. This is achieved by terminating the command with a comma instead of a semicolon. More details about these command separators will be given in the section called "Parallelism"

The `sinus` modifier can be used this way :

```
mytag:wheels.speed = 0 sin:10s ampli:100,
```

This command will make the speed of the wheels oscillate around 0, between -100 et 100 with a period of 10s. Use a tag to be able to stop the command. To make the speed change from 0 to 100 with a period of 3 seconds, use the command:

```
tag2:wheels.speed = 50 sin:3s ampli:50,
```

Parallelism

URBI handles command parallelism in a very simple way. Parallelism is enforced through various command separators :

- As usual in programming languages, two commands separated by a semicolon ";" will be executed one after the other :

```
wheels.speed = 100; wait(2s); wheels.speed = 0;
```

will turn the wheels, wait 2 seconds and then stop.

- In URBI, two commands separated by "&" will be executed in parallel and will start *exactly at the same time* :

```
wheelL.speed = 50 & wheelR.speed = -50;
```

will move at the same time the left wheel forward and the right wheel backward (so the robot will turn).

- The comma "," makes the first action to go in background and starts the next one as soon as possible, without enforcing a simultaneous start of the two commands :

```
wheelL.speed = 50 time:10s, wheelR.speed = -40 time:10s,
```

will not wait that wheelL grows to 50 to make wheelR decrease to -40, but will not enforce that the commands terminate at the same time, which would be the case with the "&" separator.

This explain why the comma was necessary in the previous section. If we finish a command that last forever with a semicolon, the server will wait forever before starting a new command. If you happen to make this mistake, it is still possible to open a new connection to the URBI server using another client and using the `stop tag;` command.

Here is an example where we use parallelism to move the tribot awkwardly while playing sounds :

```
wheelR.speed = 0 sin:3s ampli:50 &  
wheelL.speed = 0 sin:4s ampli:30 &  
beeper.play(200, 3s),
```

Parallelism handling is a key feature of URBI, which has in fact four command separators. See the URBI tutorial available at <http://www.gostai.com/doc.php> for more information.

Using functions

As in most programming language, it is possible to write *functions* in URBI. The functions are useful for making more complex programs where the same commands are used several times.

The following example defines a function which moves the robot for a given time at a given speed :

```
function global.forward(speed, timer)
{
  wheels.speed = speed;
  wait(timer);
  wheels.speed = 0;
},
```

Once defined, this function can be called simply :

```
global.forward(100, 3000);
```

More details about functions, objects and variables in URBI are available in the URBI tutorial (<http://www.gostai.com/doc.php>).

Loading files

When making larger URBI programs, it is easier to save them in files and to command the server to load the files.

The files are simple text files in which you write URBI commands. These files should be saved in the `data/` subdirectory of your sever directory. The files should use the `.u` extension which is the extension used for URBI script files.

An example `demo.u` is provided with the server. To make the URBI server to execute this script, type the URBI command :

```
load("demo.u");
```

Conclusion

This chapter has highlighted the main features of URBI with the Mindstorms Tribot robot. The next chapter details all the devices available for the TriBot in the default layout. For more details about the URBI language, please refer to the URBI tutorial available at <http://www.gostai.com/doc.php>

Chapter 4. Default layout reference

Introduction

This chapter describes the objects defined in the `Tribot.ini` layout shipped with the URBI mindstorms NXT server.

Motors

Three motors are defined, `wheelL`, `wheelR` and `claw`.

A group, `wheels`, is defined to group `wheelL` and `wheelR` :

```
wheels.speed = 10;
```

is equivalent to

```
wheelL.speed = 10 & wheelR.speed = 10;
```

The attribute `.speed` allows you to set the current speed of the motor. This value is set between -100 and 100.

Motor position (in degrees) can be accessed using the `.val` attribute. This attribute is set to 0 when you turn the robot on. We have added a PID control, so you can set a position :

```
claw.val = 360 ;
```

requests a full rotation of the `claw` motor. The PID parameters `claw.PGain`, `claw.IGain`, `claw.DGain` and `claw.Precision` can be set to change the PID behavior.

Sensors

Sensors are grouped in the `sensors` group and all hardware devices (`sensors` + `motors` + `battery` + `beeper`) are grouped in the `hardware` group.

Bumper

The switch device in front of the robot is called `bumper`. Its value is 1 if it is pressed, 0 otherwise.

Sonar

The ultrasonic sensor is called `sonar`.

Its value is the distance measured by the sensor in centimeters between 0 and 255. When the read operation fails, 255 is returned.

Decibel

The sound sensor is called `decibel`.

Its value relates the level of ambient sound. It is between 0 and 1.

Two different modes can be used by changing the `.mode` attribute to "DB" or "DBA".

- "DBA" is a mode measuring only frequencies between 200 and 14000Hz.
- "DB" measures a wider band.

Default value is "DBA".

Light

The light sensor is called `light`.

The value returned is between 0 and 1 representing the amount of light measured.

Three different modes are available by changing the `.mode` attribute.

- "Reflector" means the sensor lights on its led and measures the light reflected.
- "Ambiant" lights off the led and measures the ambient light.
- "Normal" lights off the led too and return the raw value measured.

Default value is "Reflector".

Battery

The battery device is called `battery`.

Its value is the current battery level between 0 and 1, 1 is full charge.

Beeper

The beeper device is called `beeper`.

You can request the beeper to play a sound using the `play` method :

```
play(frequency, time);
```

`frequency` is an integer between 200 and 14000, which is the frequency of the sound. `time` is an integer, which is the duration of the sound in milliseconds. A duration of 0 is infinite.

The command returns immediatly. If you want to wait until the end of the beep, use :

```
{ beeper.play(myFrequency, myDuration) & wait(myDuration) },
```

Command

The `Command UObject` is not a device. It makes it possible to send direct commands as you do with the NXT SDK. You will find more informations on <http://mindstorms.lego.com/Overview/NXTreme.asp>

This object has methods to send and receive data to the NXT :

```
Command.send(myBuff) ;
```

where :

- `myBuff` is a valid command buffer (list of integers between 0 and 255).

For example :

```
Command.send([3, 10, 10, 0 , 0]);
```

will play a beep.

```
answer = Command.request(myBuff, size) ;
```

where :

- `myBuff` is a valid command buffer (requiring an answer)
- `size` is the size of the return buffer (it must be the exact size)
- `answer` is a list containing the values of the return buffer ([] is returned when the request failed)

This command is designed for expert users. Using it is not recommended if you don't know what you are doing and might result in server crashes.

Here is an example:

```
answer = Command.request([7,0], 15);
```

that returns the informations on input port 1.

Chapter 5. How to make its own layout

Introduction

In this chapter, the method to build a custom layout for a different robot is explained. A good understanding of the objects in URBI in useful and the reading of the URBI tutorial is advised (available at <http://www.gostai.com/doc.php>).

There are three kind of devices provided by the Mindstorms NXT server : motors, sensors and other devices. There is one type of motor called *Servo*. There is four types of sensors: *Switch*, *SoundSensor*, *UltraSonicSensor* and *LightSensor*. Two other devices are available: *Battery* and *Beeper*. A complete description of these devices is available in Appendix A, *Available UObject Devices*

Writing a layout entails instancing particular objects from the devices above.

Instancing Motors

The *Servo* init constructor has the following syntax:

```
myServoObject = new Servo(myPort);
```

where *myPort* is either "A", "B" or "C". The created object makes it possible to control the motor connected to the port in parameter.

As an example, in the current *TriBot.ini*, three *Servo* objects are created :

```
wheelL = new Servo("C");  
wheelR = new Servo("A");  
claw = new Servo("B");
```

The wheel left is on port C, the wheel right on A and the claw is on port B.

A PID control is provided with this engine in order to be able to control a motor position. You can set your own PID parameters:

```
myServoObject.PGain      = 0.01;  
myServoObject.IGain      = 0.01;  
myServoObject.DGain      = 0.01;  
myServoObject.Precision = 0.01;
```

Changing these parameters will change the PID behaviour. The PID control is then launched by changing the *val* attribute:

```
myServoObject.val = myServoObject.val + 360;
```

Which means that the servo position will go forward of 360 degrees.

The PID control will stop when the precision you asked is reached, if you want to enable the control forever, just set the precision to 0.

Instantiating sensors

To create a Switch object, the syntax is the following :

```
mySwitchObject = new Switch(myPort);
```

where myPort can be 1, 2, 3 or 4.

In TriBot.ini the switch is used as a bumper plugged on the port 4 so the instancing is :

```
bumper = new Switch(4);
```

UltraSonicSensor objects are created the same way:

```
myUSSObject = new UltraSonicSensor(myPort);
```

myPort can be 1, 2, 3 or 4.

In TriBot.ini an ultrasonic sensor is connected on port 2:

```
sonar = new UltraSonicSensor(2);
```

The SoundSensor object has an additional parameter:

```
mySoundObject = new SoundSensor(myPort, myMode);
```

myPort can be 1, 2, 3 or 4. myMode sets the mode of the sensor. It can be "DB" or "DBA", meaning "decibel" or "decibel adjusted" (the last one records sounds in the frequency range 200-14000Hz).

In TriBot.ini, the sound sensor is on port 1 using decibel adjusted:

```
decibel = new SoundSensor(1, "DBA");
```

LightSensor also has two parameters :

```
myLightObject = new LightSensor(myPort, myMode);
```

myPort can be 1, 2, 3 or 4. myMode sets the sensor mode and it can be "Ambiant" (so the sensor measures the ambient light), "Reflector" (the sensor lights on its led and measures the reflected light) and "Normal" (the sensor returns its raw value).

In TriBot.ini, the light sensor is in reflector mode on port 1 :

```
light = new LightSensor(3, "Reflector");
```

Other devices

The Battery device makes it possible to get the current battery level.

```
myBattery = new Battery();
```

In TriBot.ini :

```
battery = new Battery();
```

The Beeper device makes it possible to emit customized beeps from the NXT speaker.

```
myBeeper = new Beeper();
```

In TriBot.ini:

```
beeper = new Beeper();
```

Appendix A. Available UObject Devices

This appendix exhaustively describes the UObjects available in the Lego Mindstorm NXT URBI server.

Servo

Servo describes a motor. A Servo instance contains the following attributes:

Table A.1. Servo's attributes:

Name	Description
val	Motor's position. Read only.
speed	Motor's speed, from -100 to 100.
PGain	Proportional gain for the PID control.
IGain	Integral gain for the PID control.
DGain	Derivative gain for the PID control.
Precision	Precision required for the PID control.
port	The port where the servo is plugged. It can be "A", "B" or "C". You can change the value while the server is running. Beware, if you change the port, that will free the old port (so you will be able to create another device on it) and will busy the new one (so you won't be able to create another device on it).
init(port_)	The UObject constructor. port_ is a port name. (See previous attribute port). Example: <pre>wheel = new Servo("A");</pre>

UltraSonicSensor

UltraSonicSensor describes a sonar sensor. A UltraSonicSensor instance contains the following attributes:

Table A.2. UltraSonic Sensor's attributes:

Name	Description
val	Distance in cm (from 0 to 255).
port	The port where the servo is plugged. It can be 1, 2, 3 or 4. You can change the value while the server is running. Beware, if you change the port, that will free the old port (so you will be able to create another device on it) and will busy the new one (so you won't be able to create another device on it).
init(port_)	The UObject constructor. port_ is a port name. (See previous attribute port). Example: <code>sonar = new UltraSonicSensor(1);</code>

SoundSensor

SoundSensor describes a sound sensor. A SoundSensor instance contains the following attributes:

Table A.3. Sound sensor's attributes:

Name	Description
val	The sound level measured (from 0 to 1).
mode	"DB" is decibel or "DBA" is decibel audible (measures only in the audible frequencies range).
port	The port where the servo is plugged. It can be 1, 2, 3 or 4. You can change the value while the server is running. Beware, if you change the port, that will free the old port (so you will be able to create another device on it) and will busy the new one (so you won't be able to create another device on it).
init(port_, mode_)	The UObject constructor. port_ is a port name. (See previous attribute port). mode_ is a mode name. (See previous attribute mode). Example: <code>decibel = new SoundSensor(1, "DB");</code>

LightSensor

LightSensor describes a light sensor. A LightSensor instance contains the following attributes:

Table A.4. Light sensor's attributes:

Name	Description
val	The light level measured (from 0 to 1).
mode	"Ambiant" measures the ambiant light. "Reflector" lights on a led and measures the reflection. "Normal" returns the raw value
port	The port where the servo is plugged. It can be 1, 2, 3 or 4. You can change the value while the server is running. Beware, if you change the port, that will free the old port (so you will be able to create another device on it) and will busy the new one (so you won't be able to create another device on it).
init(port_, mode_)	The UObject constructor. port_ is a port name. (See previous attribute port). mode_ is a mode name. (See previous attribute mode). Example: <code>light = new LightSensor(1, "Ambiant");</code>

Switch

Switch describes a touch sensor. A Switch instance contains the following attributes:

Table A.5. Switch's attributes:

Name	Description
val	The switch status (0 or 1).
port	The port where the servo is plugged. It can be 1, 2, 3 or 4. You can change the value while the server is running. Beware, if you change the port, that will free the old port (so you will be able to create another device on it) and will busy the new one (so you won't be able to create another device on it).
init(port_)	The UObject constructor. port_ is a port name. (See previous attribute port). Example: <code>bumper = new Switch(1);</code>

Battery

Battery describes a battery device. A Battery instance contains the following attributes:

Table A.6. Battery's attributes:

Name	Description
val	Available power of battery.
init()	The UObject constructor Example <code>battery = new Battery();</code>

Beeper

Beeper describes a speaker device. A Beeper instance contains the following methods:

Table A.7. Beeper's methods:

Name	Description
init()	The UObject constructor Example <code>beeper = new Beeper();</code>
play(frequency, duration)	Plays a beep of a custom frequency for a custom duration. frequency is the frequency of the beep (in Hz), between 200 and 14000. duration is the duration of the beep (in ms), 0 is infinite. Example: <code>beeper.play(200,1000);</code>

Command

Command allows you to use expert commands. The Command UObject contains the following methods:

Table A.8. Command's methodes:

Name	Description
send(buffer)	<p>Sends a command without requiering any answer.</p> <p>buffer must be a list of integers between 0 and 255.</p> <p>Example:</p> <pre>Command.send([3,10,10,0,0]);</pre> <p>that plays a beep.</p>
request(bufferIn, sizeOut)	<p>Sends a command requiering an answer, and returns the buffer out. This is only recommended to expert users. Beware of the sizeOut parameter. Indeed if it is too long, the request will normaly fail and when the size is too short, the command may work but all mindstorms internal buffers will be shift back (so this will bend all other requests).</p> <p>bufferIn is the buffer of the command.</p> <p>sizeOut is the size of the answer.</p> <p>Example:</p> <pre>answer = Command.request([7,0], 15);</pre> <p>that returns the informations on input port 0.</p>

Instances

Here are the different instances in the TriBot.ini layout.

Table A.9. TriBot.ini layout

Instance	UObject	Description
wheelL	Servo	Left wheel
wheelR	Servo	Right wheel
claw	Servo	Claw
sonar	UltraSoundSensor	Distance sensor
decibel	SoundSensor	Sound sensor
light	LightSensor	Light sensor
bumper	Switch	Touch sensor
beeper	Beeper	Emits beeps
battery	Battery	Battery

Groups

Here are the different groups provided in the current layout.

Table A.10. Group's list

Group Name	Description
wheels	The two wheels.
motors	All the motors.
sensors	All the sensors.
hardware	All the devices.

Appendix B. Copyright

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions

- a. "Collective Work" means a work, such as a periodical issue, anthology or encyclopedia, in which the Work in its entirety in unmodified form, along with a number of other contributions, constituting separate and independent works in themselves, are assembled into a collective whole. A work that constitutes a Collective Work will not be considered a Derivative Work (as defined below) for the purposes of this License.
- b. "Derivative Work" means a work based upon the Work or upon the Work and other pre-existing works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which the Work may be recast, transformed, or adapted, except that a work that constitutes a Collective Work will not be considered a Derivative Work for the purpose of this License. For the avoidance of doubt, where the Work is a musical composition or sound recording, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered a Derivative Work for the purpose of this License.
- c. "Licensor" means the individual or entity that offers the Work under the terms of this License.
- d. "Original Author" means the individual or entity who created the Work.
- e. "Work" means the copyrightable work of authorship offered under the terms of this License.
- f. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a

previous violation.

2. Fair Use Rights.

Nothing in this license is intended to reduce, limit, or restrict any rights arising from fair use, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.

3. License Grant.

Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

- a. to reproduce the Work, to incorporate the Work into one or more Collective Works, and to reproduce the Work as incorporated in the Collective Works;
- b. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission the Work including as incorporated in Collective Works;

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats, but otherwise you have no rights to make Derivative Works. All rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights set forth in Sections 4(d) and 4(e).

4. Restrictions.

The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

- a. You may distribute, publicly display, publicly perform, or publicly digitally perform the Work only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy or phonorecord of the Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Work that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Work

itself to be made subject to the terms of this License. If You create a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any credit as required by clause 4(c), as requested.

- b. You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.
- c. If you distribute, publicly display, publicly perform, or publicly digitally perform the Work, You must keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or (ii) if the Original Author and/or Licensor designate another party or parties (e.g. a sponsor institute, publishing entity, journal) for attribution in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; the title of the Work if supplied; and to the extent reasonably practicable, the Uniform Resource Identifier, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work. Such credit may be implemented in any reasonable manner; provided, however, that in the case of a Collective Work, at a minimum such credit will appear where any other comparable authorship credit appears and in a manner at least as prominent as such other comparable authorship credit.
- d. For the avoidance of doubt, where the Work is a musical composition:
 - i. Performance Royalties Under Blanket Licenses. Licensor reserves the exclusive right to collect, whether individually or via a performance rights society (e.g. ASCAP, BMI, SESAC), royalties for the public performance or public digital performance (e.g. webcast) of the Work if that performance is primarily intended for or directed toward commercial advantage or private monetary compensation.
 - ii. Mechanical Rights and Statutory Royalties. Licensor reserves the exclusive right to collect, whether individually or via a music rights agency or designated agent (e.g. Harry Fox Agency), royalties for any phonorecord You create from the Work ("cover version") and distribute, subject to the compulsory license created by 17 USC Section 115 of the US Copyright Act (or the equivalent in other jurisdictions), if Your distribution of such cover version is primarily intended for or directed toward commercial advantage or private

monetary compensation.

- e. Webcasting Rights and Statutory Royalties. For the avoidance of doubt, where the Work is a sound recording, Licensor reserves the exclusive right to collect, whether individually or via a performance-rights society (e.g. SoundExchange), royalties for the public digital performance (e.g. webcast) of the Work, subject to the compulsory license created by 17 USC Section 114 of the US Copyright Act (or the equivalent in other jurisdictions), if Your public digital performance is primarily intended for or directed toward commercial advantage or private monetary compensation.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability.

EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

- a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Collective Works from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.
- b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

- a. Each time You distribute or publicly digitally perform the Work or a Collective Work, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.
- b. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.
- c. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.
- d. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.