

Introduction to libURBI for Urbi 1.x

(book compiled from Revision 411M)

Matthieu Nottale

Introduction to libURBI for Urbi 1.x: (book compiled from Revision 411M)

by Matthieu Nottale

Published

Copyright © 2006 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. Getting started	2
Connecting	2
Sending URBI commands	2
Sending binary data.	3
Sending a sound	3
3. Receiving	4
4. Synchronous operations	8
Reading the value of a device	8
Getting an image	8
Getting sound	8
5. Conversion functions	9
6. Putting it all together: some examples	10
7. Programming hints	11
8. Portability functions	12
A. Copyright	13

Chapter 1. Introduction

Liburbi-c++ is a library designed to encapsulate an URBI connection. It handles the TCP connection with the URBI server, and the dispatching of messages it sends. The library is thread-safe and reentrant.

The library consists of two C++ classes, `UClient` and `USyncClient`, and a few helpful functions.

We expect the reader to be a bit familiar with the URBI syntax.

Chapter 2. Getting started

Connecting

To connect to an URBI server, simply create a new instance of `UClient` (or `USyncClient` if you want to use the synchronous functions described bellow), passing the name or the address of the server as the first parameter, and optionnaly the port as the second parameter:

```
UClient * client = new UClient("myrobot.ensta.fr");

//a wrapper is also available in the urbi namespace:
UClient * client = urbi::connect("myrobot.ensta.fr");
```

The constructor will start an independant thread that will listen for incoming messages from the URBI server.

You can check if the connection was successfully established by calling the `error` function, which returns a zero value on success, or a nonzero error code in case of failure.

Sending URBI commands

The method `send` is the simplest way to send commands to the URBI server. It accepts a syntax similmar to the `printf` function. To send a sequence of commands without risk of having an other thread sending commands at the same time, the `lockSend` and `unlockSend` methods can be used to lock and then unlock the send buffer.

```
int sleeptime = 50;

client->send("motoron;");

client->lockSend(); //send() call by other threads will be blocked from this
                  //point until unlockSend is called

for (float val=0; val<=1; val+=0.05)
    client->send("neck.val = %f;wait (%d);", val, sleeptime);

client->unlockSend();
```

Alternatively, the `UClient` class inherits from `ostream`, so you can use the `<<` operator:

```
client << "headPan.val = " <<12 << urbi::comma;
```

The constants 'comma', 'semicolon', 'pipe' and 'parallel' are defined in the urbi namespace for ',', ';', '|' and '&' respectively.

A third possible way is to use the URBI macro, which uses the default connection (the first connection created with your program):

```
URBI((
  headPan.val = 12 ,
  echo "coucou" | speaker.play("test.wav") & leds.val = 1
));
//note the absence of double-quotes to delimit the URBI code
//the double-parenthesis are required
URBI() << "headPan.val = " << 12 << urbi::semicolon;
```

The function `urbi::setDefaultClientUClient *cl)` can be used to change the default client.

Sending binary data.

To send binary data to the robot, the method `sendBin` must be used. It takes as parameters the buffer to send and its size, and optionally a header.

```
client->sendBin(soundData, soundDataSize, "speaker.val = BIN %d raw 2 16000 16 1;",
soundDataSize);
```

Sending a sound

Although you could use `sendBin` to play a sound on the robot, a specific and efficient method has been written for this purpose: `sendSound`.

```
client->sendSound(sound, "endsound");
```

The first parameter is a `USound` structure describing the sound to send. The second is an optional tag that will be used by the server to issue a "stop" system message when the sound has finished playing. The function `convert` can be used to convert between various sound formats.

There is no limit to the size of the sound buffer, since it will be automatically cut into small chunks by the library. The data is copied by the library: the `USound` parameter and its associated data can be safely freed as soon as the function returns.

Chapter 3. Receiving

Most of the messages received from the URBI server are the results of a previously sent command. The mechanism of URBI tags enables to link a message to its reply: with each command is associated a tag, and this tag is repeated in the reply message. The `UClient` class handles the reception of those messages in the independant thread created by the constructor, parses them and fills a `UMessage` structure. Callback functions with an associated tag can be registered with the method `registerCallback`: each time a message with this tag is sent by the server, the callback function will be called with a `UMessage` structure as a parameter. The two basic forms of `registerCallback` are:

```
typedef UCallbackAction (*UCallback)          (const UMessage &msg);
typedef UCallbackAction (*UCustomCallback)    (void * callbackData, const UMessage &msg);

UCallbackID setCallback (UCallback cb, const char *tag)
UCallbackID setCallback (UCustomCallback cb, void *callbackData, const char *tag)
```

The first parameter is always a pointer to the function to call. `callbackData` is a pointer that will be given back to the callback function each time it is called. The callback function must return `URBI_CONTINUE`, or `URBI_REMOVE`, in which case the function will be unregistered.

A few examples:

```
UCallbackAction onImage(const UMessage &msg) {
//Test if someone has used your tag or an error message has been received
    if (msg.type != MESSAGE_DATA || ((UImage)msg).imageFormat == IMAGE_UNKNOWN)
        return URBI_CONTINUE;
    UImage img = (UImage)msg;
    msg.client.printf("Image of size (%d,%d) received from server at %d\n",img.width,
img.height);

    unsigned char *image = new unsigned char[img.width*img.height*3];
    int sz = img.width*img.height*3;

    if (img.imageFormat == IMAGE_JPEG)
        convertJPEGtoRGB((const byte *) img.data, img.size, (byte *) image, sz); //provided
    if (img.imageFormat == IMAGE_YCbCr)
        convertYCrCbtoRGB((const byte *) img.data, img.size, (byte *) image); //provided

    myDisplayRGBImage(image, img.width, img.height);
    delete image;
    return URBI_CONTINUE;
}

UCallbackAction onSound(const UMessage &msg) {
    if (msg.type != MESSAGE_DATA || USound(msg).soundFormat == SOUND_UNKNOWN)
        return URBI_CONTINUE;

    //convert the sound to a wav 16KHz 16bit.
    USound snd;
    snd.soundFormat = SOUND_WAV;
    snd.rate = 16000;
```

```

    snd.sampleSize = 16;
    snd.sampleFormat = SAMPLE_SIGNED;
    snd.channels = 0; //take the value from source
    snd.data = 0;
    snd.size = 0;
    convert((USound)msg, snd); //this function is provided by liburbi
    myPlayWAV(snd.data, snd.size);
    return URBI_CONTINUE;
}

UCallbackAction onJoint(const UMessage &msg) {
    if (msg.type != MESSAGE_DATA || ((UValue)msg).type != DATA_DOUBLE)
        return URBI_CONTINUE;
    msg.client.printf("The joint value is %lf\n", UValue(msg).val);
    return URBI_CONTINUE;
}

int main(int argc, const char * argv[]) {
    UClient * cl = new UClient(argv[1]);
    if (cl->error()) urbi::exit(1); //portability call explained below
    cl->setCallback(&onImage, "img");
    cl->setCallback(&onSound, "snd");
    cl->setCallback(&onJoint, "joint");
    cl->send("img: camera.val;");
    cl->send("loop snd: micro.val,");
    cl->send("joint: headPan.val;");
    urbi::execute(); //portability call explained below
}

```

UMessage

The UMessage structure is capable of storing the informations contained in any kind of URBI message by using a "type" field and an UValue (union of type-dependant structures). These two structures are defined as follows:

```

class UMessage
{
public:
    /// Connection from which originated the message.
    UAbstractClient &client;
    /// Server-side timestamp.
    int timestamp;
    /// Associated tag.
    std::string tag;

    UMessageType type;

    urbi::UValue *value;
    std::string message;
    /// Raw message without the binary data.
    std::string rawMessage;
};

```

UValue

```

class UValue
{
public:
    UDataType type;
    ufloat val; // value if of type DATA_DOUBLE
    union
    {
        std::string      *stringValue; // value if of type DATA_STRING
        UBinary          *binary;      // value if of type DATA_BINARY
        UList            *list;        // value if of type DATA_LIST
        UObjectStruct    *object;      // value if of type DATA_OBJ
    };
};

```

The type field `UMessageType` can be `MESSAGE_SYSTEM`, `MESSAGE_ERROR` or `MESSAGE_DATA`. If the type is `MESSAGE_DATA`, the message contains an `UValue`. The `UValue` itself contains an `UDataType` which can take the values: `DATA_DOUBLE`, `DATA_STRING`, `DATA_BINARY`, `DATA_LIST`, `DATA_OBJECT`, `DATA_VOID`. Depending of this field, the corresponding value in the union will be set. If the `UValue` is of the binary type, it contains an `UBinary` structure defined hereafter. The `UBinaryType` in the `UBinary` structure will give additional informations on the type of data (`BINARY_NONE`, `BINARY_UNKNOWN`, `BINARY_IMAGE`, `BINARY_SOUND`), and the appropriate sound or image structure will be filled.

UBinary

```

class UBinary
{
public:
    UBinaryType type;
    union
    {
        struct
        {
            void *data; /// binary data
            int size;
        } common;
        UImage image;
        USound sound;
    };
};

```

USound

```

class USound {
public:
    char          *data; // pointer to sound data
    int           size;  // total size in byte
    int           channels; // number of audio channels
};

```

```

int          rate;          // rate in Hertz
int          sampleSize;   // sample size in bit
USoundFormat soundFormat;  // format of the sound data
                                // (SOUND_RAW, SOUND_WAV, SOUND_MP3...)
USoundSampleFormat sampleFormat; // sample format
};

```

UImage

```

class UImage {
public:
    char          *data;          // pointer to image data
    int           size;          // image size in byte
    int           width, height; // size of the image
    UImageFormat  imageFormat;   // IMAGE_RGB, IMAGE_YCbCr, IMAGE_JPEG...
};

```

Template versions of `registerCallback` are also defined. They allow to set callbacks on member functions, with from 0 to 4 custom parameters of any type (including pointers and references). The only constraint on the function signature is that it must return a `UCallbackAction`, and take a `const UMessage&` as its last parameter. A few examples:

```

class Test {
public:
    UCallbackAction onJoint(int value, const UMessage &msg);
};

UCallbackAction Test::onJoint(int value, const UMessage &msg) {
    msg.client.printf("got a message at %d with tag %s, our int is %d\n",msg.timestamp, value);
    return URBI_REMOVE; //unregister ourself
}

int main(int argc, const char * argv[]) {
    Test *a = new Test();
    UClient * cl= new UClient(argv[1]);
    if (cl->error()) urbi::exit(1);
    cl->setCallback(*a, &Test::onJoint, 12, "tag");
    cl->send("tag: headPan.val;");
    urbi::execute();
}

```

Chapter 4. Synchronous operations

The derived class `USyncClient` implements methods to synchronously get the result of URBI commands. You must be aware that these functions are less efficient, and that they are not easily portable.

Reading the value of a device

To get the value of a device, you can use the method `syncGetDevice` or `syncGetNormalizedDevice`. The first parameter is the name of the device (for instance, "neck"), the second is a double that is filled with the received value. The difference between the two methods is that `syncGetDevice` retrieves the value with a "val" command, whereas `syncGetNormalizedDevice` uses "valn" (see `urbidoc.html` for more details about "val" and "valn").

```
double neckVal;  
syncClient->syncGetDevice("neck", neckVal);
```

Getting an image

You can use the method `syncGetImage` to synchronously get an image. The method will send the appropriate command, and wait for the result, thus blocking your thread until the image is received.

```
client->send("camera.resolution = 0;camera.gain = 2;");  
int width, height;  
client->syncGetImage("camera", myBuffer, myBufferSize, IMAGE_JPEG, URBI_TRANSMIT_J
```

The first parameter is the name of the camera device. The second is the buffer (void*) which will be filled with the image data. The third must be an integer variable equal to the size of the buffer. The function will set this variable to the size of the data. If the buffer is too small, data will be truncated .

The fourth parameter is the format in which you want to receive the image data. Possible values are `IMAGE_RGB` for a raw RGB 24 bit per pixel image, `IMAGE_PPM` for a PPM file, `IMAGE_YCbCr` for raw YCbCr data, and `IMAGE_JPEG` for a jpeg-compressed file.

The fifth parameter can be either `URBI_TRANSMIT_JPEG` or `URBI_TRANSMIT_YCbCr` and specifies how the image will be transmitted between the robot and the client. Transmitting JPEG images increases the frame rate and should be used for better performances.

Finally the width and height parameters are filled with the width and height of the image on return.

Getting sound

The method `syncGetSound` can be used to get a sound sample of any length from the server.

```
client->syncGetSound("micro", duration, sound);
```

The first parameter is the name of the device from which to request sound, the second is the duration requested, in milliseconds. Sound is a `USound` structure that will be filled with the recorded sound on output.

Chapter 5. Conversion functions

We also have included a few functions to convert between different image and sound formats. The usage of the image conversion functions is pretty straightforward:

```
int convertRGBtoYCrCb(const byte* source, int sourcelen, byte* dest);
int convertYCrCbtoRGB(const byte* source, int sourcelen, byte* dest);
int convertJPEGtoYCrCb(const byte* source, int sourcelen, byte* dest, int &size);
int convertJPEGtoRGB(const byte* source, int sourcelen, byte* dest, int &size);
```

The *size* parameter must be set to the size of the destination buffer. On return it will be set to the size of the output data.

To convert between different sound formats, the function `convert` can be used. It takes two `USound` structures as its parameters. The two audio formats currently supported are `SOUND_RAW` and `SOUND_WAV`, but support for compressed sound formats such as Ogg Vorbis and MP3 is planned.. If any field is set to zero in the destination structure, the corresponding value from the source sound will be used.

Chapter 6. Putting it all together: some examples

Have a look at the examples, in the "example" or "utils" directory of the liburbi distribution. It currently contains:

- **urbiimage**: Display the images taken by the camera in realtime, or save a snapshot to a file.
- **urbisound**: Play the sound from the robot's microphone on the computer speaker, or record it to a file.
- **urbisendsound**: Play a wav file from the computer, on the robot, converting it if necessary.
- **urbiping**: Send the URBI command 'ping' at regular intervals to measure latency.
- **urbibandwidth**: Measure the effective bandwidth.
- **urbisend**: Send a set of commands contained in a file to the robot.
- **urbirecord**: Record all the movements of the robot to a file.
- **urbiplay**: Play a file recorded with **urbirecord**, or dump it in a human-readable form.
- **urbimirror**: Copy the movements of a robot on another robot. Same as piping **urbirecord** to **urbiplay**, but with less latency.
- **urbiscale**: Change the speed of a file recorded with **urbirecord**.
- **urbireverse**: Reverse a file recorded with **urbirecord**.
- **urbicycle**: Detects and extract cycles in a file recorded with **urbirecord**.
- **urbiballtrackinghead**: Port of the OPEN-R ballTrackingHead example to URBI.

Each program when invoked with no option will display its command line syntax and additional informations when appropriate.

Chapter 7. Programming hints

- Except if what you are doing is trivial, try not to use the sync* functions. They are less efficient than the asynchronous ones.
- The callback functions should return as fast as possible, since all callbacks are called by the same thread. If you have time-consuming operations, you should spawn an other thread and use synchronisation mechanisms such as semaphores or mutexes.

Chapter 8. Portability functions

When other versions of the liburbi will be available, (in particular an OPEN-R version that will allow programs to run on the robot), it will be possible to compile the same code for both libraries, if a few rules are respected:

- Do not use `USyncClient`.
- Use the `printf` method of `UClient` instead of the standard version.
- Use the `getCurrentTime` method of `UClient` instead of functions from the `stdlib`.
- Use the `urbi::exit` function (in the "urbi" namespace) instead of `exit`.
- At the end of your `main`, call `urbi::execute`.
- Do not use threads, or any function call not implemented in the OPEN-R version of `stdlib`.

Appendix A. 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

1. "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. 2. "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. 3. "Licensor" means the individual or entity that offers the Work under the terms of this License. 4. "Original Author" means the individual or entity who created the Work. 5. "Work" means the copyrightable work of authorship offered under the terms of this License. 6. "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:

1. 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; 2. 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:

1. 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 reference to such Licensor or the Original Author, as requested.
2. 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.
3. If you distribute, publicly display, publicly perform, or publicly digitally perform the Work, You must keep intact all copyright notices for the Work and give the Original Author credit reasonable to the medium or means You are utilizing by conveying the name (or pseudonym if applicable) of the Original Author if supplied; 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. 4.

For the avoidance of doubt, where the Work is a musical composition:

1. 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.
2. 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.
5. 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

1. 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. 2. 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

1. 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. 2. 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. 3. 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. 4. 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.