Kompics Programming Manual

For Kompics

Jim Dowling
Cosmin Arad

Kompics Programming Manual: For Kompics

by Jim Dowling and Cosmin Arad

Table of Contents

List of Figures

List of Tables

Chapter 1. Fundamental Concepts

This chapter is a brief informal introduction to Kompics. We begin with a discussion of the main concepts in Kompics, then describe how to download and install Kompics, and finally cover the build environment, based on Maven2.

Kompics: Components, Events, Ports and Channels

A component is a reactive unit of computation that communicates with other components asynchronously by passing data-carrying typed events. Events are passive immutable objects that can be serialized and communicated over network links and between different address spaces.

Components are loosely coupled in the sense that a component does not know the component type, availability or identity of any components with which it communicates. Rather, components are endowed with typed communication ports. A component sends events to and receives events from a local port. Ports of the same type on two different components can be connected by a bidirectional channel. This connection enables the two components to communicate.

As a reaction to received events, components execute event handlers, procedures specific to the types of events being received. During execution, event handlers may trigger new events, by sending them to one of the component's ports. An event handler is associated with events of a certain type, received though a certain port, by means of a subscription.

Table 1.1. Kompics Programming Abstractions

Entity	Description
Component	A *component* is a reactive unit of computation that communicates with other components asynchronously by passing data-carrying typed events over ports. Components contain handlers to execute received events, and components can be composed into *composite components*.
Event	An *event* is a passive immutable object that can be serialized and communicated over network links and between different address spaces.
Port	A port represents a bidirectional event interface of a component and it specifies the types of events that flow into or out of the component. The direction in which an event flows through a port is defined as either *positive* or *negative*. A negative event type flows towards the negative side of the port, while a positive event type flows towards teh positive side of the port. A port is illustrated in Figure 1.1, “Example port.”. By convention, the positive pole (+) of a port is understood to be the provided direction for events, while the negative pole (#) of a port is understood to be the required direction for events. When a component implements (or provides) a port, the port is oriented with its + pole to the outside of the component and its # pole inside. Conversely, a (required) port that is used by a component is oriented with its # pole outside and its + pole inside. The types of events flowing through the port from the # pole to the + pole are tagged with + and the types of events flowing from the + pole to the # pole are tagged with #.
Event Handler	An event handler is a procedure that a component executes as a reaction to receiving a certain event.
Subscription	A subscription binds an event handler to a port pole.
Channel	A channel is a first-class bidirectional connection between two ports of the same type. A channel can connect two ports of the same type and of different polarity.

In Figure 1.1, “Example port.”, we can see two Ports containing 2 events and 3 events, respectively. For `PortType1`, `e1` both goes "out" and comes "in". For `PortType2`, `e2` goes "out", while `e2` and `e3` come "in".

Figure 1.1. Example port.

A component with two ports (one outgoing, one incoming), two handlers and two subscriptions is illustrated in Figure 1.2, “Example component.”. The ports in this component are from Figure 1.1, “Example port.”. `PortType1` is provided by the component and `PortType2` is required by the component. We can see how the `subscriptions` map events from `Ports` to `handlers`, while handlers can trigger (or send) an `event` to a port (if the polarity of that port allows that event to be sent in that direction).

So, we can see for `PortType1`, the event `E1` can be both sent and received over this port. For `PortType2`, we can see that a handler inside the component can send either `E2` or `E3` to the port and handler inside the component can subscribe for `E2` (but not `E3`). For handlers or components outside this component (handlers would have to be in a parent component), they can send and receive events of the opposite type. So, for `PortType2`, they could subscribe for `E2` or `E3` events, and send `E2` events to the component.

Figure 1.2. Example component.

INPUt (Implements Negative, Positive Uses)

An easy way to remember whether a PortType refers to the client-side or server-side is to remember the idiom *INPUt* (Implements Negative, Positive Uses). INPUt reminds you that a negative PortType is one that is provided or implemented by a component (server-side), while a positive PortType is one that is used by a component (client-side).

Summary

We introduced a number of concepts for Kompics, and outlined the software requirements for downloading and installing Kompics.

Chapter 2. Installing Kompics

This chapter describes the software requirements for Kompics, and the steps required to install Kompics. These instructions cover the operating systems Windows (XP/Vista), Linux (all distros), Mac (OSX).

Install software required for Kompics

The minimal requirements for downloading and installing Kompics are *Java (JDK 5.0 update 6 and above)*, *subversion* and *maven2*. However, we recommend building Kompics from an Eclipse environment (Netbeans should also work fine, but isn't discussed here).

For tutorials on how to use subversion and maven, we refer you to:

- Subversion red book [http://svnbook.red-bean.com/]
- Maven2 book [http://books.sonatype.com/maven-book/reference/public-book.html]

Download and Install Subversion and Maven2

The requirements for installing Kompics are:*maven2* and *subversion*. We present two ways of installing subversion and maven: *using the command-line* and *as eclipse plugins*. The easiest way to build Kompics is to use Eclipse plugins. We outline the steps required for installing subversion and maven2 using either approach in the table below, see Table 2.1, “Install Kompics”. Please refer to the subversion and maven2 books above for additional help.

Table 2.1. Install Kompics

Command-Line	Eclipse

For Ubuntu:
`$ sudo apt-get install subversion maven2`
`$ svn checkout svn://small.sics.se/kompics/tags/0.4`
`$ cd 0.4`
`$ mvn install`
For Windows:
- Go to http://maven.apache.org/ and download an install Maven2.
- Download and install a subversion client, such as tortoise/svn [http://tortoisesvn.tigris.org/], and then checkout the Kompics code from our subversion server using the address:
`svn://small.sics.se/kompics/tags/0.4`
. Once you have checked out the Kompics code, you can build Kompics using maven2, by running the following command from the Kompics source code root folder:
`c:\...\>mvn install`
- If you prefer Netbeans, you can generate a netbeans project using the following plugin Netbeans plugin for Maven [http://wiki.netbeans.org/MavenBestPractices].
	You will need to install the following Eclipse tools to build Kompics:
	- *Eclipse IDE* (Ganymede version is recommended). You now need to know how to install plugins for Eclipse. For the Ganymede version of Eclipse, you click on `Help->Software Updates->Available Software` . Then click the "Add site" box to add an update site for a plugin.
	- *subclipse* - an Eclipse Plugin for Subversion available at Eclipse update site `http://subclipse.tigris.org/update_1.4.x`. It is recommended that you check at least the following boxes for installation: "subclipse", "subclipse client adapter" and the "javahl adapter".
	- *m2eclipse* - an Eclipse Plugin for Maven2 available at Eclipse update site `http://m2eclipse.sonatype.org/update/`. It is recommended that you check at least the following boxes for installation: "maven embedder", "maven integration for eclipse", "Maven SCM handler", and the "Maven SCM handler for Subclipse".
	In Eclipse, import the maven project from the subversion repository: `File->Import->Other->Checkout Mvn Projects from SCM`.
	Then select 'svn' as SCM type, and enter as SCM URL: `svn://small.sics.se/kompics/tags/0.4/`
	You should now restart Eclipse, and you will be able to import and build Kompics.

Summary

We specified the software requirements for Kompics and described how to download and build Kompics.

Chapter 3. A Minimal Kompics Application

This chapter shows you how to build a simple Kompics Ping-Pong application. The goal of this chapter is to familiarize you with the basic steps required to build a minimal Kompics application. We do not explain very many details of the source code here, as these details will be introduced later chapters.

See Chapter 1, *Fundamental Concepts* for a basic introduction to Kompics.

Ping (Example 1)

In this first example, see Figure 3.1, “Ping component inside a Root component.”, a `Root` component will send a *ping* message to a `Host` component. `Root` contains the `public void static main` , where a component that sends a `Ping` event to the `Host` component. `Host` has registered a handler, `handlePing`, with its `PingPort` PortType, so when the `Ping` event arrives at `PingPort` it is forwarded to `handlePing`. Finally, `handlePing` prints a message saying received the `Ping` event.

Note

It helps immensely to draw a diagram of your components and their ports along with the ports' polarity. In particular, a diagram will aid you in understanding the polarity of the port based on your context. For example, when you are a client of a Port you have a reference to the Port the opposite polarity.

Also, you should keep in mind the INPUt (Implements Negative, Positive Uses) (Implements Negative, Positive Uses).

Figure 3.1. Ping component inside a Root component.

Ping Event

```
package sandbox.manual.example1;
import sandbox.se.sics.kompics.Event;

public class Ping extends Event {
 public Ping()
 {the
 }
}
```

PingPort Port

```
package sandbox.manual.example1;
```

```
import sandbox.se.sics.kompics.PortType;

public class PingPort extends PortType {

 {
  negative(Ping.class);
 }
}
```

Root Component

An alternative (and more common) way of starting this program is to write a `startHandler` for `Root`. When a `Root` component is constructed, its `startHandler` is automatically called.

```
package sandbox.manual.example1;
import sandbox.se.sics.kompics.Component;
import sandbox.se.sics.kompics.ComponentDefinition;
import sandbox.se.sics.kompics.Handler;
import sandbox.se.sics.kompics.Kompics;
import sandbox.se.sics.kompics.Start;

public class Root extends ComponentDefinition {

 public static void  main(String[] args)
 {
  Kompics.createAndStart(Root.class);
 }
 public Root() {
  subscribe(handleStart,control);
 }

 private Handler<Start> handleStart = new Handler<Start>() {
  public void handle(Start event) {
   Component hostComponent = create(Host.class);
   trigger(new Ping(), hostComponent.getPositive(PingPort.class));
  }
 };
}
```

Host Component

```
package sandbox.manual.example1;
import sandbox.se.sics.kompics.ComponentDefinition;
import sandbox.se.sics.kompics.Handler;
import sandbox.se.sics.kompics.Negative;

public class Host extends ComponentDefinition {

 Negative<PingPort> pingN = negative(PingPort.class);
 public Host() {
  subscribe(handlePing, pingN);
 }

 private Handler<Ping> handlePing = new Handler<Ping>() {
  public void handle(Ping event) {
   System.out.println("Received ping..");
  }
 };
}
```

Ping-Pong with 2 Ports (Example 2a)

In this example, a Root component will exchange *ping* and *pong* messages with a Host component. The `Root` component sends a `Ping` event to the `Host` component. `Host` has registered a handler, `handlePing`, with its `PingPort` PortType, so when the `Ping` event arrives at `PingPort` it is forwarded to `handlePing`. `handlePing` sends a `Pong` event to its PongPort, which is forwarded to handlePong in Root.

In the examples directory for this manual, you will find a reworking of this example (Example 2b), where we reverse the event directions for the pong port in `PongPortReversed`.

Figure 3.2. Ping-Pong component with two Ports inside a Root component.

Pong Event

```
package sandbox.manual.example2a;
import sandbox.se.sics.kompics.Event;

public class Pong extends Event {
 public Pong()
 {
 }
}
```

PongPort

```
package sandbox.manual.example2a;

import sandbox.se.sics.kompics.PortType;

public class PongPort extends PortType {

 {
  positive(Pong.class);
 }
 }
```

Root Component

```
package sandbox.manual.example2a;

import sandbox.manual.example1.Ping;
import sandbox.manual.example1.PingPort;
import sandbox.se.sics.kompics.Component;
import sandbox.se.sics.kompics.ComponentDefinition;
import sandbox.se.sics.kompics.Handler;
import sandbox.se.sics.kompics.Kompics;
import sandbox.se.sics.kompics.Start;

public class Root extends ComponentDefinition {

 public static void  main(String[] args)
 {
  Kompics.createAndStart(Root.class);
 }
 public Root() {
  subscribe(handleStart,control);
 }

 private Handler<Start> handleStart = new Handler<Start>() {
  public void handle(Start event) {
   Component hostComponent = create(Host.class);
   subscribe(handlePong, hostComponent.getPositive(PongPort.class));
   trigger(new Ping(), hostComponent.getPositive(PingPort.class));
  }
 };

 private Handler<Pong> handlePong = new Handler<Pong>() {
  public void handle(Pong event) {
   System.out.println("Pong received.");
  }
 };
}
```

Host Component

```
   package sandbox.manual.example2a;

 import sandbox.manual.example1.Ping;
 import sandbox.manual.example1.PingPort;
 import sandbox.se.sics.kompics.ComponentDefinition;
 import sandbox.se.sics.kompics.Handler;
 import sandbox.se.sics.kompics.Negative;
```

```
public class Host extends ComponentDefinition {

 Negative<PingPort> negPing = negative(PingPort.class);
 Negative<PongPort> negPong = negative(PongPort.class);

 public Host() {
  subscribe(handlePing, negPing);
 }

 private Handler<Ping> handlePing = new Handler<Ping>() {
  public void handle(Ping event) {
   System.out.println("Received ping, sending Pong..");
   trigger(new Pong(), negPong);
  }
 };
}
```

Ping-Pong with a Single Port (Example 3a)

We now refactor the section called “Ping-Pong with 2 Ports (Example 2a)” so that `Host` only has a single `PingPong Port`, instead of two ports. This example demonstrates the concept of "two-way event interfaces" (where events flow in and come out of a component).

In the examples directory for this manual, you will find a reworking of this example (Example 3b), where we reverse the event directions in a port called `PingPongPortReversed`.

Figure 3.3. Ping-Pong component with one Port inside a Root component.

PingPongPort

```
package sandbox.manual.example3a;

import sandbox.manual.example1.Ping;
import sandbox.manual.example2a.Pong;
import sandbox.se.sics.kompics.PortType;

public class PingPongPort extends PortType {

 {
  negative(Ping.class);
  positive(Pong.class);
 }
}
```

Root Component

```
package sandbox.manual.example3a;
```

```
import sandbox.manual.example1.Ping;
import sandbox.manual.example2a.Pong;
import sandbox.se.sics.kompics.Component;
import sandbox.se.sics.kompics.ComponentDefinition;
import sandbox.se.sics.kompics.Handler;
import sandbox.se.sics.kompics.Kompics;
import sandbox.se.sics.kompics.Start;

public class Root extends ComponentDefinition {

 private Component hostComponent;

 public static void  main(String[] args)
 {
  Kompics.createAndStart(Root.class);
 }
 public Root() {
  hostComponent = create(Host.class);
  subscribe(handleStart,control);
  subscribe(handlePong, hostComponent.getPositive(PingPongPort.class));
 }

 private Handler<Start> handleStart = new Handler<Start>() {
  public void handle(Start event) {
   trigger(new Ping(), hostComponent.getPositive(PingPongPort.class));  }
 };

 private Handler<Pong> handlePong = new Handler<Pong>() {
  public void handle(Pong event) {
   System.out.println("Pong received.");
  }
 };
}
```

Host Component

```
package sandbox.manual.example3a;

import sandbox.manual.example1.Ping;
import sandbox.manual.example2a.Pong;
import sandbox.se.sics.kompics.ComponentDefinition;
import sandbox.se.sics.kompics.Handler;
import sandbox.se.sics.kompics.Negative;

public class Host extends ComponentDefinition {

 Negative<PingPongPort> negPingPong = negative(PingPongPort.class);

 public Host() {
  subscribe(handlePing, negPingPong);
 }

 private Handler<Ping> handlePing = new Handler<Ping>() {
  public void handle(Ping event) {
   System.out.println("Received ping, sending Pong..");
   trigger(new Pong(), negPingPong);
  }
 };
}
```

Ping-Pong with a HostPing and a HostPong component (Example 4)

The diagram in Figure 3.4, “A PingComponent and a PongComponent with two Ports each, inside a parent Root component.” shows the same Ping-Pong example factored as two different components, `HostPing` and `HostPong`. The application starts by `Root` sending a start event to `HostPong`, which then sends a `Ping` event to `HostPing`, which then replies to `HostPong` with a `Pong` event.

In the code fragment below, we connect the *positive side* of `PingPort` on `pingHost` to the *negative side* of `PingPort` on `pongHost`, which returns a *Channel* object `x1`.

The code for this example can be found in the examples directory for this manual.

```
Positive<PingPort> pingPosPort = pingHost.getPositive(PingPort.class);
Negative<PingPort> pingNegPort = pongHost.getNegative(PingPort.class);
Channel<PingPort> x1 = connect(pingNegPort, pingPosPort);
```

Figure 3.4. A PingComponent and a PongComponent with two Ports each, inside a parent Root component.

Appendix A. Copyright

Authors: Jim Dowling, Cosmin Arad.

A summary of the license is given below, followed by the full legal text.

You are free:

- to copy, distribute, display, and perform the work
- to make derivative works
- to make commercial use of the work

Under the following conditions:

Attribution. You must give the original author credit.

- For any reuse or distribution, you must make clear to others the license terms of this work.

- Any of these conditions can be waived if you get permission from the author.

Your fair use and other rights are in no way affected by the above.

The above is a summary of the full license below.

==

Creative Commons Legal Code
Attribution 2.0

CREATIVE COMMONS CORPORATION IS NOT A LAW FIRM AND DOES NOT PROVIDE LEGAL SERVICES. DISTRIBUTION OF THIS LICENSE DOES NOT CREATE AN ATTORNEY-CLIENT RELATIONSHIP. CREATIVE COMMONS PROVIDES THIS INFORMATION ON AN "AS-IS" BASIS. CREATIVE COMMONS MAKES NO WARRANTIES REGARDING THE INFORMATION PROVIDED, AND DISCLAIMS LIABILITY FOR DAMAGES RESULTING FROM ITS USE.

License

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 create and reproduce Derivative Works;

   c. 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;

   d. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission Derivative Works.

   e.

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

      i. Performance Royalties Under Blanket Licenses. Licensor waives 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.

      ii. Mechanical Rights and Statutory Royalties. Licensor waives 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).

   f. Webcasting Rights and Statutory Royalties. For the avoidance of doubt, where the Work is a sound recording, Licensor waives 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).

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. All rights not expressly granted by Licensor are hereby reserved.

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 reference to such Licensor or the Original Author, as requested. If You create a Derivative Work, upon notice from any Licensor You must, to the extent practicable, remove from the Derivative Work any reference to such Licensor or the Original Author, as requested.

   b. If you distribute, publicly display, publicly perform, or publicly digitally perform the Work or any Derivative Works or Collective Works, 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; 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; and in the case of a Derivative Work, a credit identifying the use of the Work in the Derivative Work (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). Such credit may be implemented in any reasonable manner; provided, however, that in the case of a Derivative Work or 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.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO 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, MERCHANTIBILITY, 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 Derivative Works or 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. Each time You distribute or publicly digitally perform a Derivative Work, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

   c. 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.

d. 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.

e. 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.

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, neither party will use the trademark "Creative Commons" or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons' then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time.

Creative Commons may be contacted at http://creativecommons.org/.

==