Chapter 1. A Minimal Kompics Application

Table of Contents

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 ??? for a basic introduction to Kompics.

Ping (Example 1)

In this first example, see Figure 1.1, “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 ??? (Implements Negative, Positive Uses).

Figure 1.1. 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()
 {
 }
}
```

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 1.2. 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 1.3. 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 1.4, “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 1.4. A PingComponent and a PongComponent with two Ports each, inside a parent Root component.