The News Channel example using David

April 12, 2002

This document describes step by step how the News channel application has been developped with the David personality on top of Jonathan.

1 Introduction

The News channel example shows how to use the `EventChannel`[1] provided by Jonathan to build a one-to-many communication channel between a producer of news messages and an arbitrary number of news consumers. The example scenario proceeds as follows:

- a producer of news (`NewsSource`[2]) is started, it creates an event channel and registers it to a name server with a given symbolic name;
- new consumers (`NewsConsumer`[3]) can then be started with the name of the event channel to subscribe to.

`NewsTicker`[4] is the type of the event channel used in this example.

`NewsTicker` defines the interface through which news are delivered. To be used as the type of an `EventChannel`, this interface must only contain oneway operations (and in particular, no out or inout parameters, no return value, no exceptions).

```
31 interface NewsTicker {

33    typedef sequence<string> message;
```

This operation sends several news as a array of strings.

```
36    oneway void latestNews(in message msg);
37 };
```

2 On the producer side

`NewsSource.java` contains the code for the source:

[1] see the package `org.objectweb.david.libs.contexts.orbs.echannel`
[2] located in `examples/david/newsChannel/source`
[3] located in `examples/david/newsChannel/consumer`
[4] located in `examples/david/newsChannel`

```
25
26 import org.objectweb.david.libs.binding.orbs.echannel.*;
27
28 import idl.*;
29 import org.omg.CORBA.ORB;
30 import org.objectweb.david.apis.services.naming.NameServer;
31 import org.objectweb.david.apis.services.naming.NameServerHelper;
32
33 import java.util.Random;
```

`NewsSource` is the class used to implement a source of news messages which will be delivered on an `EventChannel` to an arbitrary (and unknown) number of interested news consumers.

```
38 public class NewsSource {
```

In this simple example, the actual news messages are picked at random from the following list:

```
42    static String news[]=
43       { "Bill Gates  about Jonathan: \"a serious competitor\"",
44         "\"Jonathan is Y2K-compliant\" says Jonathan Inc. CEO B.A.M. Dumant",
45         "Jonathan Inc. up 10 points on NASDAQ",
46         "PGA Championship begins with rain",
47         "Pippen asks Rockets for a trade to Lakers",
48         "Shooting suspect returned to L.A. to face charges",
49         "Microsoft takes wraps off Embedded NT ",
50         "Customer satisfaction survey puts Dell on top ",
51         "NASA investigates space station sickness ",
52         "Sun to take wraps off Java-appliances chip",
53         "Sony developing plastic hard  drive disks",
54         "China to allow U.S. Air Force plane to land in Hong Kong"
55  };
```

A NewsSource is provided at creation time the interface of the `EventChannel` on which it will produce news messages.

```
59  NewsTicker ticker;
60
61  public NewsSource(NewsTicker ticker) {
62  this.ticker=ticker;
63  }
```

This operation is used to start the periodic sending of random news on the event channel

```
67  public void produce() {
68
69  String msgs[]=new String[1];
70  int newsCounter=0;
71  Random random=new Random(System.currentTimeMillis());
72
73  while(true) {
74  int index=random.nextInt();
75  if(index<0) index=-index;
76  msgs[0]=newsCounter++ + "\t" + news[index%news.length];
```

This operation is invoked on the `EventChannel` and will trigger the invocation of the method for each consumer (if any) which has registered itself to the channel.

```
 81  ticker.latestNews(msgs);
 82  try {
 83  Thread.currentThread().sleep(1500);
 84  } catch(Exception ex) {}
 85  }
 86  }
```

A `NewsSource` expects as arguments upon initialization:

- the IP multicast address and port number to be used by the event channel;
- the name of the `EventChannel` which will be used to register it to a Name server;

```
 96  public static void main(String args[]) {
 97  if(args.length!=3) {
 98  System.err.println("Usage: NewsSource <address> <port> <channel name>");
 99  System.exit(1);
100  }
101
102  String address="";
103  int port=0;;
104  address=args[0];
105  try {
106  port=Integer.parseInt(args[1]);
107  } catch(NumberFormatException ex) {
108  System.err.println("Wrong port");
109  System.exit(1);
110  }
111
112  try {
```

We first retrieve an `EventChannelFactory` instance:

```
114  EventChannelFactory channel_factory =
115             EventChannelFactoryFactory.newEventChannelFactory(NewsSource.class);
```

We request the creation of a new event channel. We need to provide the type (i.e., class name) of the channel.

```
119          NewsTicker channel = (NewsTicker)
120             channel_factory.newEventChannel(address,port,"idl.NewsTicker");
```

We attempt to register the event channel with the name server under the name provided in the main arguments.

```
124          ORB orb = ORB.init(args,null);
125          NameServer name_server =
126             NameServerHelper.narrow(orb.resolve_initial_references("NameServer"));
127          name_server.put(args[2],channel,true);
128          System.out.println("Ready...");
```

The channel object is supplied to the `NewsSource` object.

```
132  NewsSource producer = new NewsSource(channel);
```

We start the emission of news messages

```
135  producer.produce();
136  } catch(Exception e ) {
137          e.printStackTrace();
138          System.exit(1);
139  }
140  }
141 }
```

3 On the consumer side

`NewsConsumer.java` contains the code for consumers:

```
25
26 import org.objectweb.david.libs.binding.orbs.echannel.*;
27 import idl.*;
28 import org.omg.CORBA.ORB;
29 import org.objectweb.david.apis.services.naming.NameServer;
30 import org.objectweb.david.apis.services.naming.NameServerHelper;
```

`NewsConsumer` is the class representing a consumer of news: It implements the `NewsTicker` interface.

```
34 public class NewsConsumer implements NewsTickerOperations {
```

On receipt of incoming events, `NewsConsumer` just prints them on the screen.

```
38  public void latestNews(String msgs[])   {
39  for(int i=0;i<msgs.length;i++)
40  System.out.println(msgs[i]);
41  }
```

a NewsConsumer expects one argument: the name of the news channel it should subscribe to.

```
45  static public void main(String args[]) {
46  if(args.length!=1) {
47  System.err.println("Usage: NewsConsumer <existing channel name>");
48  System.exit(1);
49  }
50
51  String channelName=args[0];
52
53  try {
```

The ORB initialization, and the retrieval of the name server reference are performed in a standard way.

```
56          ORB orb = ORB.init(args,null);
57          org.omg.CORBA.Object ns_ref =
58             orb.resolve_initial_references("NameServer");
59          NameServer name_server =
60             NameServerHelper.narrow(ns_ref);
```

We look up the event channel name in the name server.

```
63          NewsTicker channel =
64             NewsTickerHelper.narrow(name_server.get(channelName));
```

The channel might not be available, i.e., no news source created and registered it to the name server.

```
68          if(channel==null) {
69             System.err.println("Channel "+ channelName + " not found");
70             System.exit(1);
71          }
```

The returned channel might be used directly by a news source to publish news by invoking the ”latestNews()” method on the obtained channel. But here we want to implement a consumer: We create a new consumer object and bind it as a consumer to the event channel. If this step is successful, news should be delivered to the created `NewsConsumer` object.

```
79          EventChannelFactory.bindConsumer(new NewsConsumer(),channel);
80  } catch(Exception e) {
81  e.printStackTrace();
82  }
83  }
84 }
```

4 To compile and run your source and consumer

- `make` compiles the code;
- `make naming` starts the name server;
- `make source` starts the news channel source;
- `make consumer` starts a consumer (you may start several consumers).