Testing

Quick links in this document

Related appendices

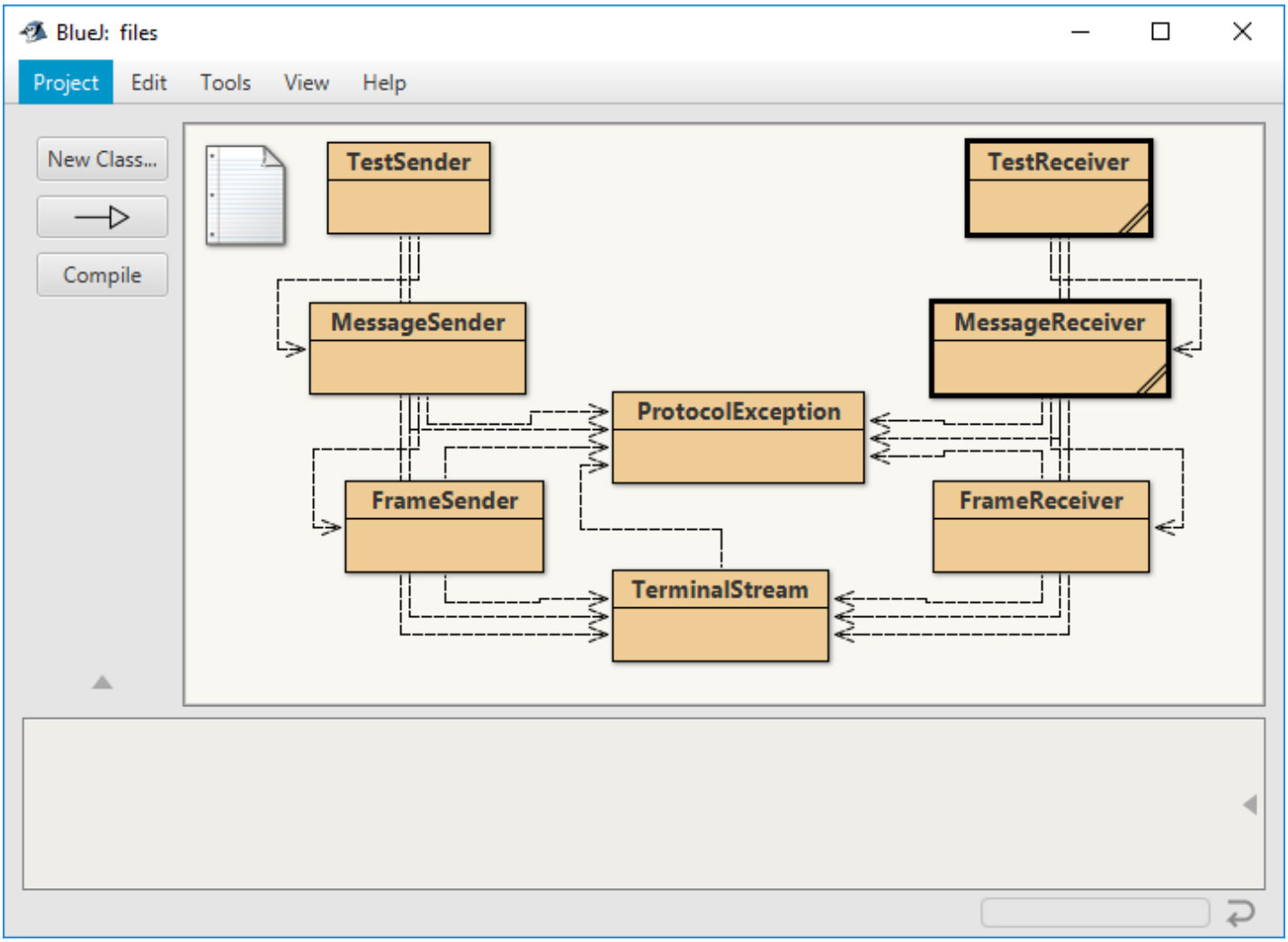
- ZIP archive of source files
- TestSender class documentation
- TestReceiver class documentation
- Frame format specification

Running test harnesses in BlueJ

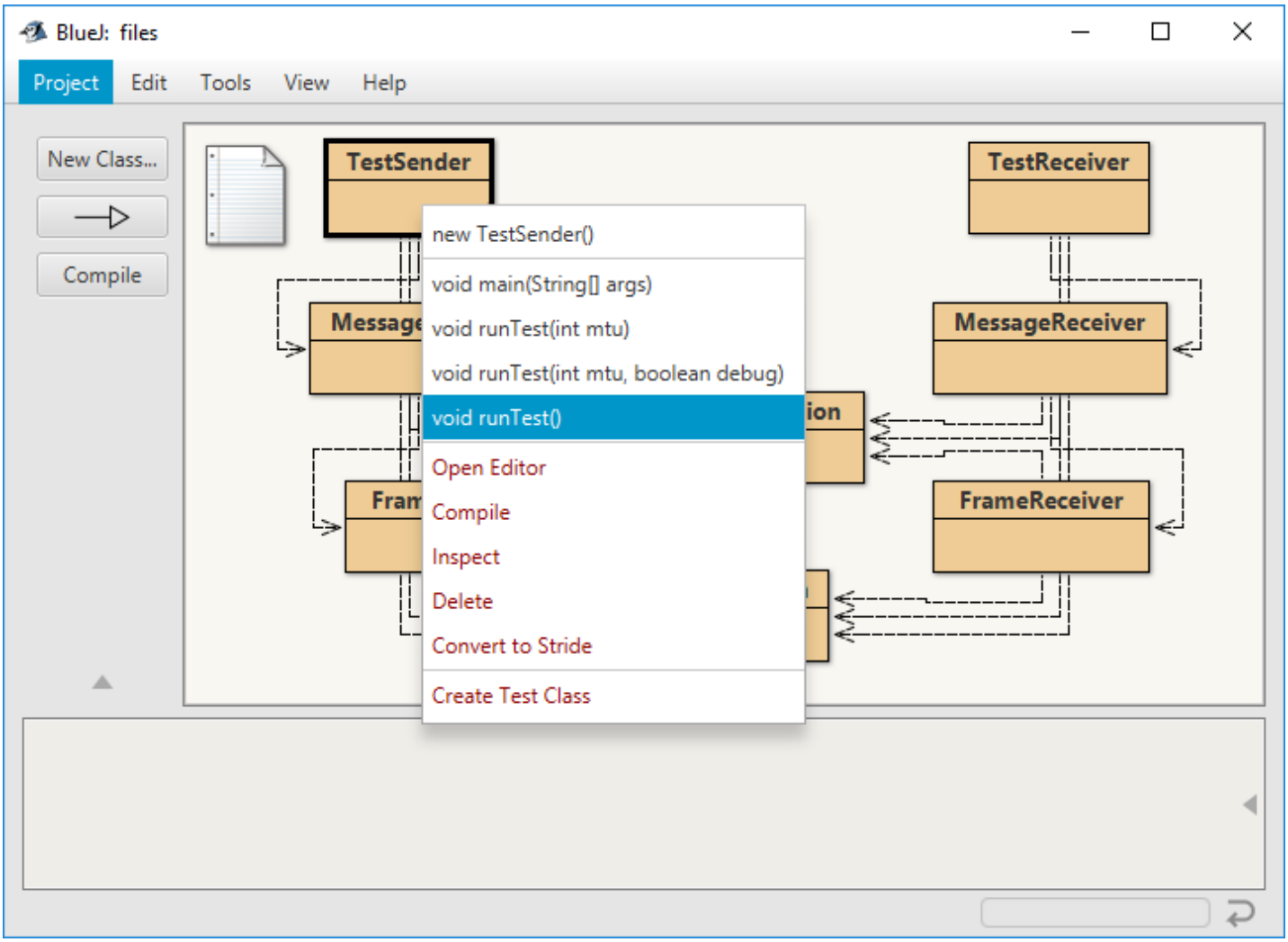
This appendix demonstrates the TestSender and TestReceiver test harness programs in operation under BlueJ. The test harnesses can also be used directly from a command line such as a Windows command prompt or Linux shell window. Formal descriptions are given in the class interface documentation and also in the supplied Java files. All files can be found in the ZIP archive.

#1 Sender

TestSender allows the MessageSender class to be tested independently of the receiver classes. An example execution is illustrated below. This example assumes MessageSender has been completed correctly. Load up the project and ensure all classes have been compiled. The main BlueJ window should look like this (the arrows may differ):



Right-click on the **TestSender** class icon and select method runTest(). These are static methods so you don't need to create an object first. Several versions of runTest are available. The one with no parameters is easiest to use.



The BlueJ terminal window should now appear with initialization messages from the test harness. This version of runTest() sets the MTU to 20, which leaves room for 10 message characters in each frame. The test harness has a diagnostic mode which when enabled causes each method to report its action to assist with testing and debugging.

BlueJ Terminal Window

```
TestSender     : test rig starting (mtu = 20, debug = true)
FrameSender    : physical layer ready
MessageSender : data link layer ready (mtu = 20)
TestSender     : message entry loop starting
TestSender     : enter one message per line (no "quotes" required)
TestSender     : enter "." to stop test

TestSender     : enter message >
```

The test harness is now prompting for a message to be entered in the terminal window. Type a short message (less than 10 characters). The test harness will then invoke sendMessage passing this message as a parameter. The message fits in one frame so sendMessage should only need to invoke sendFrame once, which will in turn output the frame to the terminal window. For example, if you enter "Hi" the window should look like this (message and frame characters shown in blue for clarity):

BlueJ Terminal Window

```
TestSender     : enter message > Hi
TestSender     : calling sendMessage...
MessageSender :   sendMessage starting (message = "Hi")
FrameSender    :     sendFrame called (frame = "<E-02-Hi-79>")
MessageSender :   sendMessage finished
TestSender     : sendMessage returned normally
```

The test harness will automatically prompt for another message. Try entering a longer message (more than 10 characters). In this case sendMessage will need to break the message into more than one frame and invoke sendFrame separately for each. Here are some examples of messages requiring more than one frame:

BlueJ Terminal Window

```
TestSender    : enter message > Who Framed Roger Rabbit
TestSender    : calling sendMessage...
MessageSender :   sendMessage starting (message = "Who Framed Roger
Rabbit")
FrameSender   :     sendFrame called (frame = "<D-10-Who Framed-25>")
FrameSender   :     sendFrame called (frame = "<D-10- Roger Rab-52>")
FrameSender   :     sendFrame called (frame = "<E-03-bit-22>")
MessageSender :   sendMessage finished
TestSender    : sendMessage returned normally

TestSender    : enter message > The Byte Count of Monte Cristo
TestSender    : calling sendMessage...
MessageSender :   sendMessage starting (message = "The Byte Count of
Monte Cristo")
FrameSender   :     sendFrame called (frame = "<D-10-The Byte C-24>")
FrameSender   :     sendFrame called (frame = "<D-10-ount of Mo-19>")
FrameSender   :     sendFrame called (frame = "<E-10-nte Cristo-88>")
MessageSender :   sendMessage finished
TestSender    : sendMessage returned normally

TestSender    : enter message > This is a longer message and will
require eight frames with the default MTU
TestSender    : calling sendMessage...
MessageSender :   sendMessage starting (message = "This is a longer
message and will require eight frames with the default MTU")
FrameSender   :     sendFrame called (frame = "<D-10-This is a -21>")
FrameSender   :     sendFrame called (frame = "<D-10-longer mes-04>")
FrameSender   :     sendFrame called (frame = "<D-10-sage and w-06>")
FrameSender   :     sendFrame called (frame = "<D-10-ill requir-17>")
FrameSender   :     sendFrame called (frame = "<D-10-e eight fr-10>")
FrameSender   :     sendFrame called (frame = "<D-10-ames with -30>")
FrameSender   :     sendFrame called (frame = "<D-10-the defaul-78>")
FrameSender   :     sendFrame called (frame = "<E-05-t MTU-99>")
MessageSender :   sendMessage finished
TestSender    : sendMessage returned normally
```

The test harness will accept messages of any length containing any characters that can be entered at the keyboard. The only exception is a message consisting of a single dot (".") which is used to stop the test harness, viz:

BlueJ Terminal Window

```
TestSender    : enter message > .
TestSender    : end of input stream reached

TestSender    : test rig finished
```

To test how your implementation of sendMessage copes with different MTU values, invoke the version of runTest which includes the MTU as a parameter, such as runTest(int mtu). The following shows a few examples with the MTU set to 50.

BlueJ Terminal Window

```
TestSender    : test rig starting (mtu = 50, debug = true)
FrameSender   : physical layer ready
MessageSender : data link layer ready (mtu = 50)
TestSender    : message entry loop starting
TestSender    : enter one message per line (no "quotes" required)
TestSender    : enter "." to stop test
```

```
TestSender    : enter message > Flags of Our Fathers
TestSender    : calling sendMessage...
MessageSender :   sendMessage starting (message = "Flags of Our
Fathers")
FrameSender   :     sendFrame called (frame = "<E-20-Flags of Our
Fathers-31>")
MessageSender :   sendMessage finished
TestSender    : sendMessage returned normally

TestSender    : enter message > Noah's ARQ
TestSender    : calling sendMessage...
MessageSender :   sendMessage starting (message = "Noah's ARQ")
FrameSender   :     sendFrame called (frame = "<E-10-Noah's ARQ-05>")
MessageSender :   sendMessage finished
TestSender    : sendMessage returned normally

TestSender    : enter message > This is a longer message and will
require more than one frame
TestSender    : calling sendMessage...
MessageSender :   sendMessage starting (message = "This is a longer
message and will require more than one frame")
FrameSender   :     sendFrame called (frame = "<D-40-This is a longer
message and will requir-51>")
FrameSender   :     sendFrame called (frame = "<E-21-e more than one
frame-39>")
MessageSender :   sendMessage finished
TestSender    : sendMessage returned normally

TestSender    : enter message > .
TestSender    : end of input stream reached

TestSender    : test rig finished
```

#2 Receiver

TestReceiver allows the MessageReceiver class to be tested independently of the sender classes. Unfortunately, this is more complicated than the testing of MessageSender. Correctly formatted frames must now be entered, including checksums, instead of arbitrary strings. There are also many more potential error conditions. As a starting point, try entering the example frames shown above and below. The illustration below assumes MessageReceiver has been completed correctly.

Load up the project and ensure all classes have been compiled. Right-click on the **TestReceiver** class icon and select the runTest() method. Enter a correctly formatted frame for a short message, for example:

BlueJ Terminal Window

```
TestReceiver    : test rig starting (mtu = 20, debug = true)
FrameReceiver   : physical layer ready
MessageReceiver : data link layer ready (mtu = 20)
TestReceiver    : frame entry loop starting
TestReceiver    : enter one frame per line (no "quotes" required)
TestReceiver    : enter "." to stop test

TestReceiver    : calling receiveMessage...
MessageReceiver :   receiveMessage starting
FrameReceiver   :     receiveFrame starting
FrameReceiver   :     enter frame > <E-02-Hi-79>
FrameReceiver   :     receiveFrame returning "<E-02-Hi-79>"
MessageReceiver :   receiveMessage returning "Hi"
TestReceiver    : message received = "Hi"
```

When the test harness starts it invokes receiveMessage then outputs the message returned to the terminal window. receiveMessage should invoke receiveFrame one or more times until it detects the end of the message or input stream, extracting the message segment(s) and joining them to restore the full message string, which it then returns. Each receiveFrame call reads a line of text from the keyboard representing a single frame.

The next example shows what should happen when a message occupies several frames. In this case receiveMessage invokes receiveFrame three times. Each "enter frame >" prompt corresponds to a separate receiveFrame call.

BlueJ Terminal Window

```
TestReceiver    : calling receiveMessage...
MessageReceiver :   receiveMessage starting
FrameReceiver   :     receiveFrame starting
FrameReceiver   :     enter frame > <D-10-The Checks-14>
FrameReceiver   :     receiveFrame returning "<D-10-The Checks-14>"
FrameReceiver   :     receiveFrame starting
FrameReceiver   :     enter frame > <D-10-um of All -16>
FrameReceiver   :     receiveFrame returning "<D-10-um of All -16>"
FrameReceiver   :     receiveFrame starting
FrameReceiver   :     enter frame > <E-05-Fears-02>
FrameReceiver   :     receiveFrame returning "<E-05-Fears-02>"
MessageReceiver :   receiveMessage returning "The Checksum of All
Fears"
TestReceiver    : message received = "The Checksum of All Fears"
```

The example below shows what should happen if the stream contains several messages. Each call to receiveMessage fetches the next message in turn. In this case the messages happen to be short but in general they could be any length.

BlueJ Terminal Window

```
TestReceiver    : calling receiveMessage...
MessageReceiver :   receiveMessage starting
FrameReceiver   :     receiveFrame starting
FrameReceiver   :     enter frame > <E-09-Message 1-99>
FrameReceiver   :     receiveFrame returning "<E-09-Message 1-99>"
MessageReceiver :   receiveMessage returning "Message 1"
TestReceiver    : message received = "Message 1"

TestReceiver    : calling receiveMessage...
MessageReceiver :   receiveMessage starting
FrameReceiver   :     receiveFrame starting
FrameReceiver   :     enter frame > <E-09-Message 2-00>
FrameReceiver   :     receiveFrame returning "<E-09-Message 2-00>"
MessageReceiver :   receiveMessage returning "Message 2"
TestReceiver    : message received = "Message 2"

TestReceiver    : calling receiveMessage...
MessageReceiver :   receiveMessage starting
FrameReceiver   :     receiveFrame starting
FrameReceiver   :     enter frame > <E-09-Message 3-01>
FrameReceiver   :     receiveFrame returning "<E-09-Message 3-01>"
MessageReceiver :   receiveMessage returning "Message 3"
TestReceiver    : message received = "Message 3"
```

The example below illustrates what should happen if an invalid frame is entered. The checksum of 66 is incorrect. The text following "receiveMessage threw an exception" will depend on the message your implementation used when creating the ProtocolException. It does **not** need to match the string shown here. Note the test harness will always terminate after an exception is thrown.

BlueJ Terminal Window

```
TestReceiver    : calling receiveMessage...
MessageReceiver :   receiveMessage starting
FrameReceiver   :     receiveFrame starting
FrameReceiver   :     enter frame > <E-02-Hi-66>
FrameReceiver   :     receiveFrame returning "<E-02-Hi-66>"
TestReceiver    : receiveMessage threw an exception "Checksum
calculated differs from that recorded"
```

```
TestReceiver    : test rig finished
```

TestReceiver should stop when the user enters a single dot. When receiveFrame reads a line containing a single dot it returns **null** to signal that the end of the input stream has been reached. receiveMessage should do the same. If entering a single dot doesn't lead to the "test rig finished" message it means your implementation of receiveMessage isn't handling this situation correctly.

BlueJ Terminal Window

```
TestReceiver    : calling receiveMessage...
MessageReceiver :   receiveMessage starting
FrameReceiver   :     receiveFrame starting
FrameReceiver   :     enter frame > .
FrameReceiver   :     receiveFrame returning null (end of input
stream)
MessageReceiver :   receiveMessage returning null (end of input
stream)
TestReceiver    : end of input stream reached

TestReceiver    : test rig finished
```

Running test harnesses from command line

Both test harnesses can be run directly from the command line. Options such as the MTU value can be set using command line arguments. This is covered in the documentation for TestSender / TestReceiver and can also be found at the start of the relevant source files. Please also read the "additional notes when running from a command line" section of the documentation/comments.

Input and output can be redirected to/from files. However, most prompt messages are sent to standard output and so won't be visible on the screen if output has been redirected to a file.