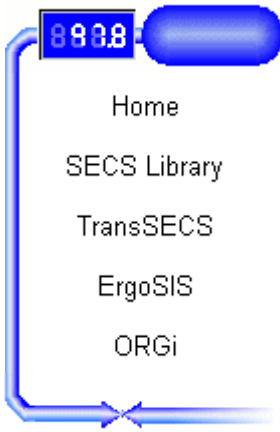

Host Example

This code example is distributed with the SECS Library. It demonstrates how a simple host application can be constructed. This simple example iterates through a few tests, sending S1F1s, S1F13s, S2F13s, and S2F15s.

```
/*Title:       SECS Example Host
 *Copyright:   Copyright (c) 2002
 *Author:      Drake Woodring
 *Author:      Jim Redman
 *Company:     ErgoTech Systems, Inc.
 *Description: This is the example host application for SECS/GEM

# SECS-GEM: True
# Test: True
# Not-To-JavaDoc: True

*/
/* The contents of this file are confidental property of ErgoTech Systems, Inc.
 * as described in the file "Ownership.txt". If you did not recieve a copy
 * of that file please contact ErgoTech at +1 505 662 5156 or info@ergotech.com.
 */
import com.ergotech.secs.*;
import java.util.Vector;


/** This is an example Host for either SecsI or HSMS connection. If the port
 * (the first argumen) is less then 128, then it will create a SecsI connection
 * otherwise it will create a HSMS connection.
 */
public class ExampleHost {


  static public int messagePaceTime = 250; //in msec
  public static final String cvsRev =  "CVS Info:$Revision: 1.10 $ $Date: 2003/08/06 16:16:25 $";

  static public void main (String args[]) {
   boolean HSMSTest = false;
   boolean runForever = true; // if true the simulator host runs forever, until stopped
   String hostname =  "localhost";
   int portNumber = 5500;
   int deviceId = 1;

   SecsISessionManager sm;

   // Find the port Number and optionally the deviceId
   if (args.length > 0) {
     try {
       portNumber = Integer.valueOf(args[0]).intValue();
       if (args.length > 1) {
         deviceId = Integer.valueOf(args[1]).intValue();
       }

       if (args.length > 2) {
         hostname = args[2];
       }
     } catch (Throwable e) {
```

```
      System.out.println("Using the default port of "+portNumber);
    }
  }

  System.out.println("Connecting to port " + portNumber + " on device " + deviceId);
  WrapperInterface connectionWrapper;

  // Find out if we are SecsI or HSMS based on port number
  // This is a completely arbitrary distinction - we assume that any port
  // number less than 128 is SECSI
  if (portNumber < 128) {
    connectionWrapper =  new SecsIWrapper(portNumber, deviceId);
  } else {
    connectionWrapper =  new HSMSActiveWrapper(portNumber, deviceId, hostname);
  }

  // Configure the logger, and setting it to the wrapper
  LoggerInterface logger = new Logger("Equipment on " + String.valueOf(portNumber),
"Equipment On " + String.valueOf(portNumber));
  //   logger.setDisplayBytes(true);
  logger.setDisplayMessages(true);
  logger.setDisplayEvents(true);
  connectionWrapper.setLogger(logger);

  // Make a connection to the Equipment
  if (connectionWrapper.connect()) {
    // At this point we can start sending messages
    try {
      // send out an S1F13 and wait until there is a response.
      // this is a GEM requirement.  Our simulated equipment will return something.
      S1F14 s1f14 =  null;

      do {
        System.out.println("Sending S1F13 Message");
        //        S1F13 s1f13 =  new S1F13(new MDLN("Java"), new SOFTREV("1.1"));
        S1F13 s1f13 =  new S1F13();
        s1f14 =  (S1F14)s1f13.sendMessageAndWait();
      } while (s1f14 == null);

      System.out.println("Sending S1F1 Messages, without waiting for a reply");
      // send out some messages without waiting for a reply.
        sendS1F1s();

      //send out a set of S2F13s
      sendS2F13s();

      //send out a set of S2F15s
        sendS2F15s();

      while  (runForever) {

      //send out an S1F1 without waiting for a reply
        System.out.println("Sending S1F1 Message");
        S1F1 s1f1 =  new S1F1();
        s1f1.sendMessage();

        //send out a set of S2F13s (waiting for a reply)
      sendS2F13s();

      //send out a set of S2F15s (waiting for  a reply)
        sendS2F15s();

      }
```

```
    } catch (Throwable e) {
      System.out.println(e);
      e.printStackTrace();
    }
  } else {
    System.out.println("Couldn't make connection to Equipment");
  }
 }


/** this sends some S1F1s without waiting for a reply */
  public static void sendS1F1s()
        throws SecsException {
      for (int scount =  0; scount < 10; scount++) {
        //pace the messages
        Thread myThread = Thread.currentThread();
try {
          Thread.sleep(messagePaceTime);
          } catch (InterruptedException e){
          // the VM doesn't want us to sleep anymore,
          // so get back to work
          }
        System.out.println("Sending S1F1 Message");
        S1F1 s1f1 =  new S1F1();
        s1f1.sendMessage();
      }
  }

/** this sends out S2F15s, waiting for a reply for each */
   public static void sendS2F15s()
     throws SecsException {
     SecsMsg msg, reply;
//pace the messages
for (int x = 0; x < 100; x++) {
       Thread myThread = Thread.currentThread();
try {
          Thread.sleep(messagePaceTime);
          } catch (InterruptedException e){
          // the VM doesn't want us to sleep anymore,
          // so get back to work
          }
        System.out.println("Sending S2F15 Message number " + x + " to equipment");
        msg =  new S2F15(new ECID("test" + x), new ECV(Math.random()));
        reply =  msg.sendMessageAndWait();
    }
 }

/** this sends out S2F13s, waiting for a reply for each */
   public static void sendS2F13s()
     throws SecsException {
     SecsMsg msg, reply;
     //pace the messages
 for (int x =  0; x < 100; x++) {
       Thread myThread = Thread.currentThread();
 try {
          Thread.sleep(messagePaceTime);
          } catch (InterruptedException e){
          // the VM doesn't want us to sleep anymore,
          // so get back to work
          }
       System.out.println("Sending S2F13 Message number " + x + " to equipment");
       msg = new S2F13(new ECID("test" + x));
     reply =  msg.sendMessageAndWait();
    }
}
```

```
}
```