Chapter 1

A Corrected Code Version

In this appendix, a possible corrected version of the code is provided for only the assigned methods. Two important clarifications follow. First of all, the number of the lines is not the same for two main reasons: first, here the numeration starts from 1 at the first assigned method (without considering all previous code); second, the numeration cannot be the same due to we have modified several lines of codes.
Then, we have added a new method to implement the logger. In fact, the code used to log some events is often duplicated in the assigned methods with a few differences. Hence, a better way to write that code is to define a new private method that includes the duplicated lines of code. In this way the code is more clear and readable and a change can be perform speedily and easily by a change on the method.

```
1   private Subject getSubjectFromSecurityCurrent()
2           throws SecurityMechanismException {
3       com.sun.enterprise.security.SecurityContext securityContext;
4       securityContext = com.sun.enterprise.security.SecurityContext.getCurrent();
5       if(securityContext == null) {
6           fineLevelLog(" SETTING GUEST ---");
7           securityContext = com.sun.enterprise.security.SecurityContext.init();
8       }
9       if(securityContext == null) {
10          throw new SecurityMechanismException("Could not find " +
11                                               "security information");
12      }
13      Subject subject = securityContext.getSubject();
```

```
14          if(subject == null) {
15              throw new SecurityMechanismException("Could not find " +
16                                                   "subject information in the " +
17                                                   "security context.");
18          }
19          fineLevelLog("Subject in security current:" + subject);
20          return subject;
21      }
22
23      public CompoundSecMech selectSecurityMechanism(IOR ior)
24              throws SecurityMechanismException {
25          CompoundSecMech[] mechList = getCtc().getSecurityMechanisms(ior);
26          CompoundSecMech mech = selectSecurityMechanism(mechList);
27          return mech;
28      }
29
30      /**
31       * Select the security mechanism from the list of compound security
32       * mechanisms.
33       */
34      private CompoundSecMech selectSecurityMechanism(CompoundSecMech[] mechList)
35                                  throws SecurityMechanismException {
36          // We should choose from list of compound security mechanisms
37          // which are in decreasing preference order. Right now we select
38          // the first one.
39          if(mechList == null || mechList.length == 0) {
40              return null;
41          }
42          CompoundSecMech mech;
43          for(int i = 0; i < mechList.length; i++) {
44              mech = mechList[i];
45              boolean useMech = useMechanism(mech);
46              if(useMech) {
47                  return mech;
48              }
49          }
50          throw new SecurityMechanismException("Cannot use any of the " +
51                                               "target's supported mechanisms");
52      }
53
54      private boolean useMechanism(CompoundSecMech mech) {
55          boolean val = true;
56          TLS_SEC_TRANS tls = getCtc().getSSLInformation(mech);
57
58          if (mech.sas_context_mech.supported_naming_mechanisms.length > 0 &&
59              !isMechanismSupported(mech.sas_context_mech)) {
60              return false;
61          } else if (mech.as_context_mech.client_authentication_mech.length > 0 &&
62                     !isMechanismSupportedAS(mech.as_context_mech)) {
```

```
63              return false;
64          }
65
66          if(tls == null) {
67              return true;
68          }
69          int targetRequires = tls.target_requires;
70          if(isSet(targetRequires, EstablishTrustInClient.value)) {
71              if(! sslUtils.isKeyAvailable()) {
72                  val = false;
73              }
74          }
75          return val;
76      }
77
78      private boolean evaluateClientConformanceSsl(
79                          EjbIORConfigurationDescriptor iordesc,
80                          boolean  sslUsed,
81                          X509Certificate[] certchain) {
82
83          boolean sslRequired  = false;
84          boolean sslSupported = false;
85          int sslTargetRequires = 0;
86          int sslTargetSupports = 0;
87
88          try {
89              fineLevelLog("SecurityMechanismSelector.evaluate_client_" +
90                           "conformance_ssl->:");
91
92              /*********************************************************************
93               * Conformance Matrix:
94               *
95               * |---------------|-----------------|-----------------|-----------|
96               * | SSLClientAuth | targetrequires. | targetSupports. | Conformant|
97               * |               |       ETIC      |      ETIC       |           |
98               * |---------------|-----------------|-----------------|-----------|
99               * |     Yes       |          0      |      1          |    Yes    |
100              * |     Yes       |          0      |      0          |    No     |
101              * |     Yes       |          1      |      X          |    Yes    |
102              * |     No        |          0      |      X          |    Yes    |
103              * |     No        |          1      |      X          |    No     |
104              * |---------------|-----------------|-----------------|-----------|
105              *
106              *********************************************************************/
107
108             // gather the configured SSL security policies.
109
110             sslTargetRequires = this.getCtc().getTargetRequires(iordesc);
111             sslTargetSupports = this.getCtc().getTargetSupports(iordesc);
```

```
112
113             if (isSet(sslTargetRequires, Integrity.value) ||
114                 isSet(sslTargetRequires, Confidentiality.value) ||
115                 isSet(sslTargetRequires, EstablishTrustInClient.value)) {
116                 sslRequired = true;
117             } else {
118                 sslRequired = false;
119             }
120
121             if ( sslTargetSupports != 0) {
122                 sslSupported = true;
123             } else {
124                 sslSupported = false;
125             }
126
127             /* Check for conformance for using SSL usage.
128              *
129              * a. if SSL was used, then either the target must require or
130              *    support SSL. In the latter case, SSL is used because of client
131              *    policy.
132              * b. if SSL was not used, then the target must not require it
133              *    either. The target may or may not support SSL (it is
134              *    irrelevant).
135              */
136             fineLevelLog("SecurityMechanismSelector.evaluate_client_" +
137                          "conformance_ssl:" +
138                          " " + isSet(sslTargetRequires, Integrity.value) +
139                          " " + isSet(sslTargetRequires, Confidentiality.value) +
140                          " " +
141                          isSet(sslTargetRequires,EstablishTrustInClient.value) +
142                          " " + sslRequired +
143                          " " + sslSupported +
144                          " " + sslUsed);
145
146             if (sslUsed) {
147                 if (! (sslRequired || sslSupported)) {
148                     return false;  // security mechanism did not match
149                 }
150             } else {
151                 if (sslRequired) {
152                     return false;  // security mechanism did not match
153                 }
154             }
155
156             /* Check for conformance for SSL client authentication.
157              *
158              * a. if client performed SSL client authentication, then the target
159              *    must either require or support SSL client authentication. If
160              *    the target only supports, SSL client authentication is used
```

```
161             *    because of client security policy.
162             *
163             * b. if SSL client authentication was not used, then the target must
164             *    not require SSL client authentcation either. The target may or may
165             *    not support SSL client authentication (it is irrelevant).
166             */
167
168            fineLevelLog("SecurityMechanismSelector.evaluate_client_" +
169                         "conformance_ssl:" +
170                         " " +
171                         isSet(sslTargetRequires,EstablishTrustInClient.value) +
172                         " " +
173                         isSet(sslTargetSupports,EstablishTrustInClient.value));
174
175            if (certchain != null) {
176                if ( ! ( isSet(sslTargetRequires, EstablishTrustInClient.value) ||
177                     isSet(sslTargetSupports, EstablishTrustInClient.value))) {
178                    return false; // security mechanism did not match
179                }
180            } else {
181                if (isSet(sslTargetRequires, EstablishTrustInClient.value)) {
182                    return false; // security mechanism did not match
183                }
184            }
185
186            fineLevelLog("SecurityMechanismSelector.evaluate_client_" +
187                         "conformance_ssl: true");
188
189            return true ; // mechanism matched
190        } finally {
191            fineLevelLog("SecurityMechanismSelector.evaluate_client_" +
192                         "conformance_ssl<-:");
193        }
194    }
195
196    //At the end of the class or into a specific class dedicated to the logger
197    private fineLevelLog (String s) {
198        if(_logger.isLoggable(Level.FINE)) {
199            _logger.log(Level.FINE, s);
200        }
201    }
```

Listing 1.1: "A corrected version of the code."