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              if( useMechanism(mech) ) {
46                  return mech;
47              }
48          }
49          throw new SecurityMechanismException("Cannot use any of the " +
50                                               "target's supported mechanisms");
51      }
52
53      private boolean useMechanism(CompoundSecMech mech) {
54          TLS_SEC_TRANS tls = getCtc().getSSLInformation(mech);
55
56          if (mech.sas_context_mech.supported_naming_mechanisms.length > 0 &&
57              !isMechanismSupported(mech.sas_context_mech)) {
58              return false;
59          } else if (mech.as_context_mech.client_authentication_mech.length > 0 &&
60                     !isMechanismSupportedAS(mech.as_context_mech)) {
61              return false;
62          }
```

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

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

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

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