Copyright

Copyright © 1999 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, ObjectBroker, TOP END, and Tuxedo are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Connect, BEA Manager, BEA MessageQ, BEA Jolt, M3, eSolutions, eLink, WebLogic, and WebLogic Enterprise are trademarks of BEA Systems, Inc.

All other company names may be trademarks of the respective companies with which they are associated.

Using Security

<table>
<thead>
<tr>
<th>Document Edition</th>
<th>Date</th>
<th>Software Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.0</td>
<td>December 1999</td>
<td>BEA WebLogic Enterprise 5.0</td>
</tr>
</tbody>
</table>
Contents

About This Document

What You Need to Know ................................................................. x
e-docs Web Site ............................................................................. x
How to Print the Document............................................................. xi
Documentation Conventions ......................................................... xii

1. Overview of WLE Security

WLE Security Features .................................................................. 1-2
Link-Level Encryption .................................................................. 1-3
How LLE Works ......................................................................... 1-3
Development Process ................................................................. 1-4
Username/Password Authentication ............................................. 1-4
How Username/Password Authentication Works ..................... 1-5
Development Process for Username/Password Authentication .... 1-7
The SSL Protocol ......................................................................... 1-9
How the SSL Protocol Works ...................................................... 1-9
Requirements for Using the SSL Protocol ................................. 1-11
Development Process for the SSL Protocol .............................. 1-12
Certificate-Based Authentication ................................................. 1-14
How Certificate-based Authentication Works .......................... 1-15
Requirements for Using Certificate-Based Authentication .......... 1-16
Development Process for Certificate-Based Authentication .... 1-17
Commonly Asked Questions about WLE Security .................. 1-20
Do I have to Change the Security in an Existing WLE Application? 1-20
Can I Use the SSL Protocol in an Existing WLE Application? .... 1-21
When Should I Use Mutual Certificate-Based Authentication? ... 1-22

Using Security iii
2. Managing Certificates and Keys
   Installing the WLE Security Pack ...................................................... 2-2
   Using the LDAP Directory Service with Your WLE Application .......... 2-2
   Editing the LDAP Search Filter File .................................................. 2-3
   Publishing a Certificate for the Certificate Authority ......................... 2-5
   Obtaining Digital Certificates and Private Keys for Principals .......... 2-6
   Storing the Private Keys in a Common Location .................................. 2-6
   Defining the Trusted Certificate Authorities ................................... 2-8
   Creating a Peer Rules File ......................................................... 2-10

3. Configuring the WLE Environment for the SSL Protocol
   Setting Parameters for the SSL Protocol ........................................ 3-2
   Defining a Port for SSL Communications ....................................... 3-2
   Enabling Certificate-based Authentication ........................................ 3-3
   Enabling Host Matching .................................................................. 3-3
   Setting the Encryption Strength .................................................... 3-5
   Setting the Interval for Session Renegotiation ................................. 3-8
   Defining Security Parameters for the IIOP Listener/Handler ............. 3-8
   Example of Setting Parameters on the ISL System Process ............... 3-9
   Example of Setting Command Line Options on the CORBA C++ ORB ... 3-10
   Example of Setting System Properties on the CORBA Java ORB ....... 3-10

4. Defining Security for a WLE CORBA Application
   Setting Parameters for Security in the UBBCONFIG File .................. 4-2
   Configuring the Authentication Server ............................................ 4-2
   Defining a Security Level .............................................................. 4-3
   Setting the Level of Encryption .................................................... 4-5
   Sample UBBCONFIG File for Username/Password Authentication .... 4-5
   Sample UBBCONFIG File for Certificate-Based Authentication ....... 4-7
   Defining Authorized Users ........................................................... 4-8

5. Writing a WLE CORBA Application That Implements Security
   Understanding the Address Formats of the Bootstrap Object .......... 5-2
   Using the Host and Port Address Format ....................................... 5-4
   Using the corbaloc URL Address Format ...................................... 5-5
Using Security v

Using Security

Using the corbalocs URL Address Format ................................................ 5-5
Using Username/Password Authentication ....................................................... 5-6
The Security Sample Application .............................................................. 5-6
Writing the Client Application ................................................................. 5-7
Using Certificate-based Authentication .......................................................... 5-15
The Secure Simpapp Sample Application................................................ 5-16
Writing the Client Application ................................................................. 5-18
Using the Invocations_Options_Required() Method....................................... 5-21

6. Building and Running the CORBA Sample Applications

Building and Running the Security Sample Application .................................. 6-2
Step 1: Copy the files for the Security sample application into a work directory. ............................................................................................. 6-3
Step 2: Verify the settings of the environment variables. .......................... 6-5
Step 3: Change the Protection on the Files for the Security Sample Application.......................................................... 6-7
Step 4: Set the Environment Variables ...................................................... 6-8
Step 5: Initialize the Database ........................................................................ 6-8
Step 6: Load the UBBCONFIG File .......................................................... 6-8
Step 7: Compile the Security Sample Application ..................................... 6-9
Step 8: Start the server application............................................................. 6-9
Step 8: Start the C++ client application ................................................... 6-10
Step 9: Start the Java client application. .................................................. 6-10
Building and Running the Secure Simpapp Sample Application............... 6-13
Step 1: Copy the Files for the Secure Simpapp Sample Application into a Work Directory .......................................................... 6-13
Step 2: Change the protection attribute on the files for the Secure Simpapp sample application.......................................................... 6-17
Step 3: Verify the settings of the environment variables. ........................ 6-18
Step 4: Execute the runme command. ...................................................... 6-20
Using the Secure Simpapp Sample Application ...................................... 6-26

7. Writing a WLE Enterprise JavaBean that Implements Security

Before You Begin.............................................................................................. 7-2
How Authentication Works with WLE EJBs.................................................... 7-2
Development Steps.......................................................................................... 7-2
Step 1: Define security roles for the methods of the WLE EJB. .......................... 7-3
Step 2: Specify security roles in the Deployment Descriptor of the EJB. ........ 7-4
Step 3: Define the JNDI environment properties. ........................................... 7-5
  WLEContext.INITIAL_CONTEXT_FACTORY Property .................................. 7-5
  WLEContext.PROVIDER_URL Property ................................................. 7-6
  WLEContext.SECURITY_AUTHENTICATION Property ....................... 7-7
Step 4: Establish the InitialContext. .............................................................. 7-9
Step 5: Use Home to get a WLE EJB ............................................................ 7-9
Step 6: Use the getCallerPrincipal Method to authenticate a WLE EJB. .... 7-10
Limitations and Restrictions ........................................................................ 7-10
Example of Using Security in a WLE EJB .................................................... 7-10

8. Troubleshooting
   Using ULOGS and ORB Tracing................................................................. 8-1
   CORBA::ORB_init Problems ..................................................................... 8-3
   Username/Password Authentication Problems ........................................... 8-4
   Certificate-Based Authentication Problems ............................................ 8-5
   Tobj::Bootstrap::
     resolve_initial_references Problems............................................... 8-6
   IIOP Listener/Handler Startup Problems ................................................. 8-7
   Configuration Problems ........................................................................ 8-8
   Problems with Using Callbacks Objects with the SSL Protocol ............... 8-9
   Troubleshooting Tips for Digital Certificates ........................................ 8-9

9. WLE Security Service APIs
   The WLE Security Model ......................................................................... 9-2
     Authentication of Principals ................................................................. 9-2
     Controlling Access to Objects .............................................................. 9-3
     Administrative Control ....................................................................... 9-3
   Functional Components of the WLE Security Service ............................. 9-4
   The Principal Authenticator Object ....................................................... 9-5
     Using the Principal Authenticator Object with Certificate-based
     Authentication .................................................................................. 9-6
     WLE Extensions to the Principal Authenticator Object ......................... 9-6
   The Credentials Object ........................................................................... 9-7
   The SecurityCurrent Object ................................................................. 9-9
10. Security Modules
   CORBA Module....................................................................................... 10-2
   TimeBase Module.................................................................................... 10-2
   Security Module....................................................................................... 10-4
   Security Level 1 Module.......................................................................... 10-6
   Security Level 2 Module.......................................................................... 10-7
   Tobj Module............................................................................................. 10-8

11. C++ Security Reference
   SecurityLevel2::Credentials ..................................................................... 11-9
   SecurityLevel2::PrincipalAuthenticator................................................. 11-17

12. Java Security Reference

13. Automation Security Reference
   Method Descriptions ....................................................................................... 13-2
   DISecurityLevel2_Current.......................................................................... 13-2
   DIRobj_PrincipalAuthenticator................................................................ 13-7
   DIISecurityLevel2_Credentials ............................................................... 13-17
   Programming Example.................................................................................. 13-20
About This Document

This document provides an introduction to concepts associated with the BEA WebLogic Enterprise (WLE) security features, a description of how to secure your WLE applications using the WLE security features, and a guide to the use of the application programming interfaces (APIs) in the WLE Security Service.

This document covers the following topics:

- Chapter 1, “Overview of WLE Security,” introduces concepts associated with the WLE security features.
- Chapter 2, “Managing Certificates and Keys,” describes how to set up a public key infrastructure to interact with WLE applications that use the SSL protocol and certificate-based authentication.
- Chapter 3, “Configuring the WLE Environment for the SSL Protocol,” describes configuring the IIOP Listener/Handler, the CORBA C++ ORB, or the CORBA Java ORB so that it can be used with the Secure Sockets Layer (SSL) protocol and certificate-based authentication.
- Chapter 4, “Defining Security for a WLE CORBA Application,” explains the configuration tasks required when using security in a WLE application.
- Chapter 5, “Writing a WLE CORBA Application That Implements Security,” explains how the bootstrapping options work and describes implementing password-based authentication and certificate-based authentication in WLE CORBA applications.
- Chapter 6, “Building and Running the CORBA Sample Applications,” describes how to build and run the Security and Secure Simpapp sample applications.
- Chapter 7, “Writing a WLE Enterprise JavaBean that Implements Security,” describes implementing password-based and certificate-based authentication in WLE EJBs.
Chapter 8, “Troubleshooting,” provides troubleshooting tips that can be used when solving problems that occur with the security portion of a WLE application.

Chapter 9, “WLE Security Service APIs,” introduces the WLE Security model and the functional components of the security model.

Chapter 10, “Security Modules,” includes the Object Management Group (OMG) Interface Definition Language (IDL) for the modules used by the WLE Security service.

Chapter 11, “C++ Security Reference,” includes the C++ method descriptions.

Chapter 12, “Java Security Reference,” includes the Java method descriptions.


What You Need to Know

This document is intended for programmers who want to incorporate security into their WLE applications and system administrators who are responsible for setting up and maintaining the security infrastructure in an enterprise.

e-docs Web Site

The BEA WebLogic Enterprise product documentation is available on the BEA corporate Web site. From the BEA Home page, click the Product Documentation button or go directly to the “e-docs” Product Documentation page at http://e-docs.beasys.com.
How to Print the Document

You can print a copy of this document from a Web browser, one file at a time, by using the File→Print option on your Web browser.

A PDF version of this document is available on the WebLogic Enterprise documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Enterprise documentation Home page, click the PDF Files button, and select the document you want to print.

If you do not have the Adobe Acrobat Reader, you can get it for free from the Adobe Web site at http://www.adobe.com/.

Related Information

For more information about CORBA, Java 2 Enterprise Edition (J2EE), BEA TUXEDO, distributed object computing, transaction processing, C++ programming, and Java programming, see the WLE Bibliography in the WebLogic Enterprise online documentation.

Contact Us!

Your feedback on the BEA WebLogic Enterprise documentation is important to us. Send us e-mail at docsupport@beasys.com if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the WebLogic Enterprise documentation.

In your e-mail message, please indicate that you are using the documentation for the BEA WebLogic Enterprise 5.0 release.
If you have any questions about this version of BEA WebLogic Enterprise, or if you have problems installing and running BEA WebLogic Enterprise, contact BEA Customer Support through BEA WebSupport at www.beasys.com. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

### Documentation Conventions

The following documentation conventions are used throughout this document.

<table>
<thead>
<tr>
<th>Convention</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface text</strong></td>
<td>Indicates terms defined in the glossary.</td>
</tr>
<tr>
<td>Ctrl+Tab</td>
<td>Indicates that you must press two or more keys simultaneously.</td>
</tr>
<tr>
<td><em>italics</em></td>
<td>Indicates emphasis or book titles.</td>
</tr>
</tbody>
</table>
## Documentation Conventions

<table>
<thead>
<tr>
<th>Convention</th>
<th>Item</th>
</tr>
</thead>
</table>
| **monospace text** | Indicates code samples, commands and their options, data structures and their members, data types, directories, and file names and their extensions. Monospace text also indicates text that you must enter from the keyboard.  
  *Examples:*  
  #include <iostream.h> void main ( ) the pointer psz  
  chmod u+w * \tux\data\ap .doc tux.doc BITMAP float |
| **monospace boldface text** | Identifies significant words in code.  
  *Example:*  
  void **commit** ( ) |
| **monospace italic text** | Identifies variables in code.  
  *Example:*  
  String *expr* |
| **UPPERCASE TEXT** | Indicates device names, environment variables, and logical operators.  
  *Examples:*  
  LPT1 SIGNON OR |
| ( ) | Indicates a set of choices in a syntax line. The braces themselves should never be typed. |
| [ ] | Indicates optional items in a syntax line. The brackets themselves should never be typed.  
  *Example:*  
  buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]... |
<p>| | Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed. |</p>
<table>
<thead>
<tr>
<th>Convention</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>Indicates one of the following in a command line:</td>
</tr>
<tr>
<td></td>
<td>- That an argument can be repeated several times in a command line</td>
</tr>
<tr>
<td></td>
<td>- That the statement omits additional optional arguments</td>
</tr>
<tr>
<td></td>
<td>- That you can enter additional parameters, values, or other information</td>
</tr>
<tr>
<td></td>
<td>The ellipsis itself should never be typed.</td>
</tr>
<tr>
<td></td>
<td><em>Example:</em></td>
</tr>
<tr>
<td></td>
<td>buildobjclient [-v] [-o name ] [-f file-list]...</td>
</tr>
<tr>
<td></td>
<td>[-l file-list]...</td>
</tr>
<tr>
<td>.</td>
<td>Indicates the omission of items from a code example or from a syntax line.</td>
</tr>
<tr>
<td>.</td>
<td>The vertical ellipsis itself should never be typed.</td>
</tr>
</tbody>
</table>
CHAPTER 1

Overview of WLE Security

This topic includes the following sections:

- WLE Security Features
- Link-Level Encryption
- Username/Password Authentication
- The SSL Protocol
- Certificate-Based Authentication
- Commonly Asked Questions about WLE Security
Overview of WLE Security

WLE Security Features

The BEA WebLogic Enterprise (referred to as WLE) product enables you to integrate the following essential security features into your WLE applications:

- Authentication allows the two communicating parties to be certain that they are speaking with someone whom they trust. By using usernames and passwords or digital certificates and public-key technology, a WLE object and the application that invokes a request on the WLE object can each be authenticated to the other.

- Confidentiality is the ability to keep communication secret from parties other than the intended recipient. It is achieved by encrypting all data. The WLE product provides mechanisms that enable two communicating parties to negotiate an algorithm they both support and to agree on the keys with which to encrypt the data.

- Integrity is a guarantee that the data being transferred has not been modified in transit. The same handshake that allows two parties to agree on a method of encryption also provides a means of supplying integrity through the use of shared data integrity secrets. Shared data integrity secrets ensure that when data is received, any modifications made to the data while in transit will be detected.

The WLE product provides the security features:

- Link-Level Encryption
- Username/Password authentication
- The SSL protocol
- Certificate-based authentication

The remainder of this topic describes how the different security features work in the WLE product.
Link-Level Encryption

Link-level encryption (LLE) is the encryption of messages going across network links between machines in a WLE domain or between WLE domains. The objective of LLE is to ensure confidentiality so that a network-based eavesdropper cannot learn the content of WLE system messages or WLE application-generated messages. LLE is point-to-point, which means that data may be encrypted/decrypted as many times as it flows over network links.

How LLE Works

LLE works in the following way:

1. The system administrator sets a parameter to control the encryption strength.

2. The WLE domain receives the initial connection and starts to negotiate the encryption level to be used between the WLE applications.

3. The two WLE applications agree on the largest common key size supported by both.

4. The configured maximum key size parameter is reduced to agree with the installed software's capabilities. This step must be done at link negotiation time, because at configuration time it may not be possible to verify a particular machine’s installed encryption package.

5. The WLE applications exchange messages using the appropriate encryption level. Figure 1-1 illustrates these steps.
Development Process

The implementation of LLE is an administrative task. The system administrators for each WLE application set parameters in the UBBCONFIG file that control encryption strength. When the two WLE applications establish communication, they negotiate what level of encryption to use to exchange messages. Once an encryption level is negotiated, it remains in effect for the lifetime of the network connection.

Username/Password Authentication

The WLE product supports a username/password mechanism to provide authentication to existing WLE applications and to new WLE applications that are not prepared to deploy a full public key infrastructure (PKI). When using Username/Password authentication, the applications that initiate invocations on WLE objects authenticate themselves to the WLE domain using a defined username and password.
Username/Password Authentication

The WLE product utilizes a delegated trust authentication model. In this model, initiating applications authenticate to a trusted gateway process. In the WLE product, the trusted gateway process is the IIOP Listener/Handler. As part of successful authentication, a security association, called a security context, is established between the initiating application and the IIOP Listener/Handler that controls access to WLE objects.

Two levels of Username/Password authentication are provided:

- Application password—In this security scenario, the client application authenticates itself to the WLE domain. A user name or client application name, and application password are used to authenticate the client application.

- System authentication—In this security scenario, the client application provides the same information as when using application password authentication and additional authentication data that is verified by a WLE-provided authentication server (AUTHSRV) before access is granted to client application.

Username/Password authentication is available in both the base WLE product and the WLE Security pack. If you install the WLE Security pack and choose to use Username/Password authentication, the SSL protocol can be used to provide confidentiality to communication between different machines. When using Username/Password authentication, you have the option of using the 

\texttt{Tobj::PrincipalAuthenticator::logon()} or the 
\texttt{SecurityLevel2::PrincipalAuthenticator::authenticate()} methods.

How Username/Password Authentication Works

Username/Password authentication works in the following way:

1. The initiating application instantiates the Bootstrap object with any of the URL address formats. For more information about which URL address format to use with the Bootstrap object, see “Understanding the Address Formats of the Bootstrap Object” on page 5-2.

2. The initiating application obtains credentials for the user. The initiating application must provide proof material to be used by the WLE domain to authenticate the user. This proof material consists of the name of the user and a password.
The initiating application creates the security context using a PrincipalAuthenticator object. The request for authentication is sent to the IIOP Listener/Handler. The proof material in the authentication request is securely relayed to the authentication server, which verifies the supplied information.

If the verification succeeds, the WLE system constructs a Credentials object that is used by all future invocations. The Credentials object for the user is associated with the Current object that represents the security context.

3. The initiating application invokes a WLE object in the WLE domain using an object reference. The request is packaged into an IIOP request and is forwarded to the IIOP Listener/Handler that associates the request with the previously established security context.
   If the SSL protocol is used to secure the connection for confidentiality and integrity, the data will also be protected from eavesdropping.

4. The IIOP Listener/Handler receives the request from the initiating application.

5. The IIOP Listener/Handler forwards the request, along with the credentials of the initiating application, to the appropriate WLE object.

Figure 1-2 illustrates these steps.
Username/Password Authentication

Development Process for Username/Password Authentication

Defining Username/password authentication for a WLE application includes administration and programming steps. Table 1-1 and Table 1-2 list the administration and programming steps for Username/Password authentication. For a detailed description of the administration steps for Username/Password authentication, see “Defining Security for a WLE CORBA Application” on page 4-1. For a complete description of the programming steps, see “Writing a WLE CORBA Application That Implements Security” on page 5-1.
1 Overview of WLE Security

Table 1-1 Administration Steps for Username/Password Authentication

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Set the SECURITY parameter in the UBBCONFIG file to either APP_PW or USER_AUTH.</td>
</tr>
<tr>
<td>2</td>
<td>If you defined the SECURITY parameter as USER_AUTH, configure the authentication server (AUTHSRV) in the UBBCONFIG file.</td>
</tr>
<tr>
<td>3</td>
<td>Use the tpusradd and tpgrpadd commands to define lists of authorized users and groups.</td>
</tr>
<tr>
<td>4</td>
<td>Use the tmloadcf command to load the UBBCONFIG file. When the UBBCONFIG file is loaded, the system administrator is prompted for a password. The password entered at this time becomes the password for the WLE application.</td>
</tr>
</tbody>
</table>

Table 1-2 Programming Steps for Username/Password Authentication

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Write application code that uses the Bootstrap object to obtain a reference to the SecurityCurrent object.</td>
</tr>
<tr>
<td>2</td>
<td>Write application code that obtains the PrincipalAuthenticator object from the SecurityCurrent object.</td>
</tr>
<tr>
<td>3</td>
<td>Write application code that uses the Tobj::PrincipalAuthenticator::logon() or SecurityLevel2::PrincipalAuthenticator::authenticate() operation to establish a security context with the WLE domain.</td>
</tr>
<tr>
<td>4</td>
<td>Write application code that prompts the user for the password defined when the UBBCONFIG file is loaded.</td>
</tr>
</tbody>
</table>
The WLE product provides the industry-standard Secure Sockets Layer (SSL) protocol to establish secure communications between client and server applications. When using the SSL protocol, principals use digital certificates to prove their identity to a peer.

The default behavior of the SSL protocol in the WLE product is to have the IIOP Listener/Handler prove its identity to the principal who initiated the SSL connection using a digital certificate. The digital certificate is verified to ensure that the certificate has not been tampered with or expired. If there is a problem with the digital certificate in the chain, the SSL connection is terminated. In addition, the issuer of the digital certificate is compared against a list of trusted certificate authorities to verify the digital certificate received from the IIOP Listener/Handler has been signed by a certificate authority that is trusted by the WLE domain.

Figure 1-3 provides a conceptual overview of the SSL protocol.

**Figure 1-3  The SSL Protocol**

---

How the SSL Protocol Works

The SSL protocol works in the following manner:
1 Overview of WLE Security

1. The initiating application instantiates the Bootstrap object with a URL in the form of corbaloc://host:port or corbalocs://host:port.

   If you use the corbaloc://host:port URL address format, the bootstrapping process is unsecure. You can use the authenticate() method of the SecurityLevel2::Current interface and the invocations_options_required() method to secure the bootstrapping process and specify that certificate-based authentication is to be used.

2. The initiating application receives the digital certificate of the principal in this case the IIOP Listener/Handler. The security context is established as result of a Tobj_Bootstrap::resolve_initial_references() or a Tobj::PrincipalAuthenticator::Logon() method. This step is transparent to the user of the application.

3. If the verification succeeds, the WLE system constructs a Credentials object. The Credentials object for the principal represents the security context for the current thread of execution.

4. The initiating application invokes a WLE object in the WLE domain using an object reference.

5. The request is packaged into an IIOP request and is forwarded to the IIOP Listener/Handler that associates the request with the established security context.

   The request is digitally signed and encrypted before it is sent to the IIOP Listener/Handler. The WLE system performs the signing and sealing of requests.

6. The IIOP Listener/Handler receives the request from the initiating application. The digital signature of the request is verified and the request is decrypted.

7. The IIOP Listener/Handler forwards the request to the appropriate WLE object.

Figure 1-4 illustrates these steps.
Requirements for Using the SSL Protocol

To use the SSL protocol in a WLE application, you need to install the WLE Security Pack. Information about installing the WLE Security Pack can be found in the *BEA WebLogic Enterprise Installation Guide*.

The WLE implementation of the SSL protocol is flexible enough to fit into most public key infrastructures. The WLE product requires that certificates are stored in an LDAP-enabled directory. You can choose any LDAP-enabled directory service. You can also choose the certificate authority from which to obtain certificates and private keys used in a WLE application. You must have an LDAP-enabled directory service and a certificate authority in place before using the SSL protocol in a WLE application.
Overview of WLE Security

Development Process for the SSL Protocol

Using the SSL protocol in a WLE application is primarily an administration process. Table 1-4 lists the administration steps required to set up the infrastructure required to use the SSL protocol and configure the IIOP Listener/Handler for the SSL protocol. For a detailed description of the administration steps, see “Managing Certificates and Keys” on page 2-1 and “Configuring the WLE Environment for the SSL Protocol” on page 3-1.

Once the administration steps are complete, you can use either Username/Password authentication or Certificate authentication in your WLE application. For more information, see “Writing a WLE CORBA Application That Implements Security” on page 5-1. In addition, you can use the SSL protocol with Enterprise JavaBeans, for more information, see “Writing a WLE Enterprise JavaBean that Implements Security” on page 7-1.

Note: If you are using the BEA CORBA C++ or CORBA Java ORB as a server application, the ORB can also be configured to use the SSL protocol. For more information, see “Configuring the WLE Environment for the SSL Protocol” on page 3-1.

Table 1-3 Administration Steps for the SSL Protocol

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Install the WLE Security pack.</td>
</tr>
<tr>
<td>2</td>
<td>Set up an LDAP-enabled directory service.</td>
</tr>
<tr>
<td>3</td>
<td>Obtain a certificate and private key for the IIOP Listener/Handler from a certificate authority.</td>
</tr>
<tr>
<td>4</td>
<td>Publish the certificates for the IIOP Listener/Handler and the certificate authority in the LDAP-enabled directory service.</td>
</tr>
<tr>
<td>5</td>
<td>Define the SEC_PRINCIPAL_NAME, SEC_PRINCIPAL_LOCATION, and SEC_PRINCIPAL_PASSVAR parameters for the ISL server process in the UBBCONFIG file.</td>
</tr>
<tr>
<td>6</td>
<td>Define a port for secure communication on the IIOP Listener/Handler using the -a option of the ISL command.</td>
</tr>
</tbody>
</table>
The SSL Protocol

Table 1-3 Administration Steps for the SSL Protocol

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Create a Trusted Certificate Authority file (trust_ca.cer) that defines the certificate authorities trusted by the IIOP Listener/Handler.</td>
</tr>
<tr>
<td>8</td>
<td>Use the tmloadcf command to load the UBBCONFIG file.</td>
</tr>
<tr>
<td>9</td>
<td>Optionally, create a Peer Rules file (peer_val.rul) for the IIOP Listener/Handler.</td>
</tr>
<tr>
<td>10</td>
<td>Optionally, modify the LDAP Search filter file to reflect the directory hierarchy in place in your enterprise.</td>
</tr>
</tbody>
</table>

Figure 1-5 illustrates the configuration of a WLE application that uses the SSL protocol.

Figure 1-5  Configuration for Using the SSL Protocol in a WLE Application
Certificate-Based Authentication

Certificate-based authentication requires that each side of an SSL connection proves its identity to the other side of the connection. In the WLE product, the IIOP Listener/Handler presents its digital certificate to the principal who initiated the SSL connection. The initiator then provides a chain of digital certificates that are used by the IIOP Listener/Handler to verify the identity of the initiator.

Once a chain of digital certificates is successfully verified, the IIOP Listener/Handler retrieves the value of the distinguished name from the subject of the digital certificate. The WLE product uses the email address element of the subject’s distinguished name as the identity of the principal. The IIOP Listener/Handler uses the identity of the principal to impersonate the principal and establish a security context between the initiating application and the WLE domain.

Once the principal has been authenticated, the principal that initiated the request and the IIOP Listener/Handler agree on a cipher suite that represents the type and strength of encryption that they both support. They also agree on the encryption key and synchronize to start encrypting all subsequent messages.

Figure 1-6 provides a conceptual overview of the certificate-based authentication.

Figure 1-6 Certificate-Based Authentication
Certificate-Based Authentication

How Certificate-based Authentication Works

Certificate-based authentication works in the following manner:

1. The initiating application instantiates the Bootstrap object with a URL in the form of corbaloc://host:port or corbalocs://host:port and controls the requirement for protection by setting attributes on the SecurityLevel2::Credentials object returned as a result of the SecurityLevel2::PrincipalAuthenticator::authenticate operation.

   Note: You can also use the SecurityLevel2::Current::authenticate() method to secure the bootstrapping process and specify that certificate-based authentication is to be used.

2. The initiating application obtains the digital certificate and the private key of the principal. Retrieval of this information may require proof material to be supplied to gain access to the principal’s private key and certificate. The proof material typically is a pass phrase rather than a password.

   The security context is established as result of a Tobj_Bootstrap::resolve_initial_references() or a Tobj::PrincipalAuthenticator::Logon() method. This step is transparent to the user of the application.

3. If the verification succeeds, the WLE system constructs a Credentials object. The Credentials object for the principal represents the security context for the current thread of execution.

4. The initiating application invokes a WLE object in the WLE domain using an object reference.

5. The request is packaged into an IIOP request and is forwarded to the IIOP Listener/Handler that associates the request with the established security context.

6. The request is digitally signed and encrypted before it is sent to the IIOP Listener/Handler. The WLE system performs the signing and sealing of requests.

7. The IIOP Listener/Handler receives the request from the initiating application. The digital signature of the request is verified and the request is decrypted.

8. The IIOP Listener/Handler maps the principals certificate identity to a TUXEDO user identity.
Overview of WLE Security

9. The IIOP Listener/Handler forwards the request, along with the TUXEDO identity of the principal, to the appropriate WLE object.

Figure 1-7   How Certificate-Based Authentication Works

Requirements for Using Certificate-Based Authentication

Certificate-based authentication uses the SSL protocol so you need to install the WLE Security Pack. Information about installing the WLE Security Pack can be found in the BEA WebLogic Enterprise Installation Guide. You also need an LDAP-enabled directory. You can choose any LDAP-enabled directory service. You can also choose the certificate authority from which to obtain certificates and private keys used in a WLE application. You must have an LDAP-enabled directory service and a certificate authority in place before using certificate-based authentication in a WLE application.
Development Process for Certificate-Based Authentication

Using certificate-based authentication in a WLE application includes administration and programming steps. Table 1-4 and Table 1-5 list the administration and programming steps for certificate-based authentication. For a detailed description of the administration steps, see “Managing Certificates and Keys” on page 2-1 and “Configuring the WLE Environment for the SSL Protocol” on page 3-1.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Install the WLE Security pack.</td>
</tr>
<tr>
<td>2</td>
<td>Set up an LDAP-enabled directory service.</td>
</tr>
<tr>
<td>3</td>
<td>Obtain a certificate and private key for the IIOP Listener/Handler from a certificate authority.</td>
</tr>
<tr>
<td>4</td>
<td>Obtain a certificate and private key for the WLE application from a certificate authority.</td>
</tr>
<tr>
<td>5</td>
<td>Store the private key files for the WLE application in the Home directory of the user or in $TUXDIR/udataobj/security/keys.</td>
</tr>
<tr>
<td>6</td>
<td>Publish the certificates for the IIOP Listener/Handler, the WLE application, and the certificate authority in the LDAP-enabled directory service.</td>
</tr>
<tr>
<td>7</td>
<td>Define the SEC_PRINCIPAL_NAME, SEC_PRINCIPAL_LOCATION, and SEC_PRINCIPAL_PASSVAR for the ISL server process in the UBBCONFIG file.</td>
</tr>
<tr>
<td>8</td>
<td>Use the tpusradd command to define the authorized users of your WLE application.</td>
</tr>
<tr>
<td>9</td>
<td>Define a port for secure on the IIOP Listener/Handler using the -s option of the ISL command.</td>
</tr>
<tr>
<td>10</td>
<td>Enable certificate-based authentication in the IIOP Listener/Handler using the -a option of the ISL command.</td>
</tr>
<tr>
<td>11</td>
<td>Create a Trusted Certificate Authority file (trust_ca.cer) that defines the certificate authorities trusted by the IIOP Listener/Handler.</td>
</tr>
</tbody>
</table>
Overview of WLE Security

Table 1-4  Administration Steps for Certificate-Based Authentication

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Create a Trusted Certificate Authority file (trust_ca.cer) that defines the certificate authorities trusted by the WLE client application.</td>
</tr>
<tr>
<td>13</td>
<td>Use the <code>tmloadcf</code> command to load the UBBCONFIG file.</td>
</tr>
<tr>
<td>14</td>
<td>Optionally, create a Peer Rules file (<code>peer_val.rul</code>) for both the WLE client application and the IIOP Listener/Handler.</td>
</tr>
<tr>
<td>15</td>
<td>Optionally, modify the LDAP Search filter file to reflect the directory hierarchy in place in your enterprise.</td>
</tr>
</tbody>
</table>

Figure 1-8 illustrates the configuration of a WLE application that uses certificate-based authentication.
Table 1-5 lists the programming steps for using certificate-based authentication in a WLE application. For more information, see “Writing a WLE CORBA Application That Implements Security” on page 5-1. In addition, you can use certificate-based authentication with Enterprise JavaBeans, for more information see “Writing a WLE Enterprise JavaBean that Implements Security” on page 7-1.

Table 1-5  Programming Steps for Certificate-Based Authentication

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Write application code that uses the corbaloc or corbalocs URL address formats of the Bootstrap object. Note that the CommonName in the Distinguished Name of the certificate of the IIOP Listener/Handler must match exactly the host name provided in the URL address format.</td>
</tr>
</tbody>
</table>
Overview of WLE Security

Commonly Asked Questions about WLE Security

The following sections answer some of the commonly asked questions about the WLE security features.

Do I have to Change the Security in an Existing WLE Application?

The answer is no. If you are using security interfaces from previous versions of the WLE product in your WLE application there is no requirement for you to change your WLE application. You can leave your current security scheme in place and your existing WLE application will work with WLE applications built with the WLE 5.0 product.

For example, if your WLE application consists of a set of server applications which provide general information to all client applications which connect to them, there is really no need to implement a stronger security scheme. If your WLE application has a set of server applications which provide information to client applications on an internal network which provides enough security to detect sniffers, you don’t need to implement the features in the WLE Security Pack.

Table 1-5  Programming Steps for Certificate-Based Authentication

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Write application code that uses the authenticate() method of the SecurityLevel2::Current interface to perform authentication. Specify Tobj::CertificateBased for the method argument and the pass phrase for the private key as the auth_data argument for Security::Opaque.</td>
</tr>
</tbody>
</table>

1-20  Using Security
Can I Use the SSL Protocol in an Existing WLE Application?

The answer is yes. You may want to take advantage of the extra security protection provided by the SSL protocol in your existing WLE application. For example, if you have a WLE server application which provides stock prices to a specific set of client applications, you can use the SSL protocol to make sure the client applications are connected to the correct WLE server application and that they are not being routed to a fake WLE server application with incorrect data. A username and password is sufficient proof material to authenticate the client application. However, by using the SSL protocol, the username and password will be encrypted adding an additional level of security.

The SSL protocol offers WLE applications the following benefits:

- Protection of the entire conversation including the initial bootstrapping process. The SSL protocol protects against man-in-the-middle attacks, replay attacks, tampering, and sniffing.

- Even if you only use the default settings, the SSL protocol provides signed and sealed protection since the default encryption settings are a minimum of 40 bit by default.

- Client verification of the connected IIOP Listener/Handler using the certificate of the IIOP Listener/Handler. The client application can then apply additional security rules to restrict access to the client application by the IIOP Listener/Handler. This protection also applies to IIOP Listener/Handlers connecting to remote server applications when using callback objects.

To use the SSL protocol in a WLE application, set up the infrastructure to use digital certificates, change the command line options on the ISL server process to use the SSL protocol, and configure a port for secure communications on the IIOP Listener/Handler. If your existing WLE application uses username/password authentication, you can use that code with the SSL protocol. If your WLE C++ CORBA client application does not already catch the InvalidDomain exception when resolving initial references to the Bootstrap object and performing authentication, write code to handle this exception. For more information, see “The SSL Protocol” on page 1-9.
Overview of WLE Security

Note: The Java implementation of the
Tobj_Bootstrap::resolve_initial_references() method does not throw an InvalidDomain exception. When the corbaloc or corbalocs URL address formats are used, the
Tobj_Bootstrap::resolve_initial_references() method internally catches the InvalidDomain exception and throws the exception as a COMM_FAILURE. The method functions this way in order to provide backward compatibility.

When Should I Use Mutual Certificate-Based Authentication?

You might be ready to migrate your existing WLE application to use Internet connections between the WLE application and web browsers and commercial web servers. For example, users of your WLE application might be shopping over the internet. The users must be confident that:

- They are in fact communicating with the server at the online store and not an impostor that mimics the store’s server to get credit card information.
- The data exchanged between the user of the WLE application and the online store will be unintelligible to network eavesdroppers.
- The data exchanged with the online store will arrive unaltered. An instruction to order $500 worth of merchandise must not accidently or maliciously become a $5000 order.

In these situations, the SSL protocol and certificate-based authentication offer WLE applications the maximum level of protection. In addition to the benefits achieved through the use of the SSL protocol, certificate-based authentication offers WLE applications:

- IIOP Listener/Handler verification of the client application that initiates a request using the certificate of the client application. In addition, the IIOP Listener/Handler can apply additional rules which restrict access to the client application based on the identity established by the certificate. A remote ORB acting as a server application can also be configured to allow mutual authentication and verify the identity of a client application based on a certificate.
Inside the WLE domain, the client application can still have a TUXEDO username and password. The IIOP Listener/Handler will perform maps the identity defined in a certificate to a TUXEDO username and password thus allowing existing WLE applications to have an identity in native WLE server applications.

For more information, see “The SSL Protocol” on page 1-9 and “Certificate-Based Authentication” on page 1-14
1 Overview of WLE Security
CHAPTER

2 Managing Certificates and Keys

This topic includes the following sections:

- Installing the WLE Security Pack
- Using the LDAP Directory Service with Your WLE Application
- Editing the LDAP Search Filter File
- Publishing a Certificate for the Certificate Authority
- Obtaining Digital Certificates and Private Keys for Principals
- Storing the Private Keys in a Common Location
- Defining the Trusted Certificate Authorities
- Creating a Peer Rules File

The WLE product requires you have an LDAP-enabled directory service and a certificate authority (either commercial or private) set up for your organization.

Perform the tasks in this topic only if you are using the SSL Protocol or certificate-based authentication.
Installing the WLE Security Pack

To use the SSL protocol or certificate-based authentication to protect communication between principals and the WLE domain, you need to install the WLE Security Pack. The WLE Security Pack contains the files necessary to enable the use of the SSL protocol. For complete information about installing the WLE Security Pack, see the BEA WebLogic Enterprise Installation Guide.

Using the LDAP Directory Service with Your WLE Application

The use of a global directory service is the most popular way to store certificates. A directory service simplifies the management of information that needs to be globally available to an ever-growing number of users. The Lightweight Directory Access Protocol (LDAP) provides access to a variety of directory services.

The WLE product retrieves digital certificates for principals and certificate authorities from an LDAP-enabled directory service, such as Netscape Directory Service or Microsoft Active Directory. Before you can use the SSL protocol or certificate-based authentication, you need to install an LDAP-enabled directory service and configure it for your organization. The WLE product requires that the digital certificates be stored in the directory service in Privacy Enhanced Mail (PEM) format.

Directory services define a hierarchy of object classes. While there are a number of different object classes, there is a small set associated with digital certificates. Figure 2-1 illustrates the object classes associated with digital certificates.
By default, the WLE product retrieves digital certificates from the following object classes:

- **certificationAuthority**—contains digital certificates for certificate authorities
- **strongAuthenticationUser**—contains digital certificates for principals and the IIOP Listener/Handler.

Refer to the *BEA WebLogic Enterprise Installation Guide* for information about integrating your LDAP-enabled directory service into the WLE environment.

---

**Editing the LDAP Search Filter File**

When configuring a WLE application to use the SSL protocol or certificate-based authentication, you may need to customize the LDAP search filter file to limit the scope of the search of the directory service. Customizing the LDAP search filter file can result in significant performance gains. The WLE Security Pack ships with the following LDAP search filters:

- A filter stanza that searches the directory service for digital certificates assigned to certificate authorities. The filter limits its search to instances of the `certificationAuthority` object class.
A filter stanza that searches the directory service for digital certificates assigned to principals. The filter limits its search to instances of the strongAuthenticationUser object class.

If the directory service scheme for your organization is defined to store digital certificates in object classes other than certificationAuthority and strongAuthenticationUser, the LDAP search filter file must be modified to specify those object classes.

If can specify a location of the LDAP search filter file during the installation of the WLE Security pack. For more information, see the BEA WebLogic Enterprise Installation Guide.

The LDAP search filter file is owned by the administrator account. BEA recommends that the file be protected so that only the owner has read and write privileges for the file and all other users have only read privileges for the file.

To limit the search of the directory service for certificates for principals and certificate authorities, you need to modify the following tags in the LDAP search filter file:

- BEA_person_lookup
- BEA_issuer_lookup

These tags identify the stanzas in the LDAP search filter file that contains the filter expression that will be used when looking up information in the directory service. These BEA-specific tags allow the stanzas of an LDAP search filter file to be stored in a common LDAP search filter file with stanzas used by other LDAP-enabled applications that might be found in your organization.

The following is an example of the stanzas of an LDAP search filter file used by the WLE product for the SSL protocol and certificate-based authentication:

```
"BEA_person_lookup"
 ".* " "((objectClass=strongAuthenticationUser) (mail=%v))"
 "email address"
 "((objectClass=strongAuthenticationUser) (mail=%v))"
 "start of email address"

"BEA_issuer_lookup"
 ".* " "(& (objectClass=certificationAuthority)
 (cn=%v))") "exact match cn"
 (sn=%v))" "exact match sn"
```

- BEA_person_lookup specifies to search the LDAP directory service for principals by their email addresses.
BEA_issuer_lookup specifies to search the LDAP directory service for principals by their common names (cn).

See the documentation for your LDAP-enabled directory service for additional information about LDAP Search File filters.

---

**Publishing a Certificate for the Certificate Authority**

During the authentication process, the identity of a principal depends on the integrity of the public key value in the principal’s digital certificate and the private key associated with the digital certificate. A certificate authority is a trusted entity that confirms the integrity of a digital certificate. All digital certificates must be signed by a certificate authority. In order to use the SSL protocol or certificate-based authentication in a WLE application, you need to set up a certificate authority from which to obtain digital certificates.

When setting up security for a WLE application, it is important to choose a suitable certificate authority, make the digital certificate for the certificate authority available, and then use the certificate authority to sign digital certificates for your WLE application. You can use a commercial or private certificate authority of your choice with the WLE product. The certificate authority must be in place before using the SSL protocol or certificate-based authentication.

Once you have chosen a certificate authority, you need a digital certificate for the certificate authority you are using. Refer to the documentation for the certificate authority you are using for instructions on obtaining a digital certificate for the certificate authority. Load the digital certificate for the certificate authority in the certificationAuthority object class of the LDAP-enabled directory service you are using.
Obtaining Digital Certificates and Private Keys for Principals

When using the SSL protocol and certificate-based authentication, the IIOP Listener/Handler and any principal that will use your WLE application require a digital certificate and private key to prove their identity to initiators of an SSL connection.

Refer to the documentation for the certificate authority you are using for instructions on obtaining a digital certificate for a principal. Load the digital certificates for the principals in the strongAuthenticationUser object class of the LDAP-enabled directory service you are using.

Storing the Private Keys in a Common Location

When a principal gets a digital certificate from a certificate authority, they also get a file with a private key. Principals need this private key file to verify their identity in the authentication process. Store the private key file in a local directory structure that is accessible to remote applications. Assign the private key file protections so that only the owner of the private key file has write privileges and all other users only have read privileges for the file.

The WLE system uses the email address of the principal to construct a name for the private key file as follows:

1. The @ character in the name is replaced by an underscore (_) character.
2. All characters after the dot (.) character are deleted.
3. A .PEM file extension is appended to the file.
For example, if the name of the principal is milozzi@bigcompany.com the resulting private key file is milozzi_bigcompany.pem. This naming convention allows an enterprise to have multiple principals that share a common username but are in different email domains.

The WLE software looks in the following directories for private key files:

**UNIX**

```
$HOME
```

**Window NT**

```
%HOMEDRIVE% %HOMEPATH%
```

The WLE software also looks in the following directory for private key files:

```
$TUXDIR/udataobj/security/keys
```

The `/keys` directory should be protected so that only the administrator has read and write privileges for the directory and all other users should only have read privileges for the directory.

Listing 2-1 provides an example of a private key file.

**Listing 2-1** Example of Private Key File

```
-----BEGIN ENCRYPTED PRIVATE KEY-----
MIICoDAaBgkqhkiG9w0BBQgstartEnter/3wIwMjAkBgkqhkiG9w0BBQgstartEnter/3wIwMjAkBgkqhkiG9w0BBQgstartEnter/3wIwMjAkBgkqhkiG9w0BBQgstar
-----END ENCRYPTED PRIVATE KEY-----
```
Managing Certificates and Keys

Defining the Trusted Certificate Authorities

When establishing an SSL connection, the WLE processes (client applications and the IIOP Listener/Handler) check the identity of the certificate authority and certificates from the peer’s digital certificate chain against a list of trusted certificate authorities to ensure the certificate authority is trusted by the organization. This check is similar to the check done in Web browsers. If the comparison fails, the initiator of the SSL connection refuses to authenticate the target and drops the SSL connection. It is typically the job of the system administrator to define a list of trusted certificate authorities.

Retrieve from the LDAP-enabled directory service the digital certificates for the certificate authorities that are to be trusted. Cut and paste the PEM formatted digital certificates into a file named `trust_ca.cer` which is stored in `$TUXDIR/udataobj/security/certs`. The `trust_ca.cer` can be edited with any text editor.

The `trust_ca.cer` file should be owned by the administrator account. BEA recommends that the file be protected so that only the owner has read and write privileges for the file and all other users have only read privileges for the file.

Listing 2-2 provides an example of a Trusted Certificate Authority file.

Listing 2-2   Example of Trusted Certificate Authority File

```
-----BEGIN CERTIFICATE-----
MIIExzCCBCSgAwIBAgIQKtZuM5AOzS9dZaIAATJxIuDANBgkqhkiG9w0BAQQFADCB
zDEXMBUAgAUEChMOVaVNAp224sIEluYy4xHzAdBgNVBAsTF1Z1cm1TauduIFRy
dxNOIE51dHdvmxsrYJBEbgNVdT52XJpc21nb15jB20vcmWwb3NpdG9y
es9SUEgSw5j3JWliBcBzS2WYuExJQiiuTFREkGMPoTGxSDBGBgNVBAMr1ZIcmlTaWdu
mlTaWduIENsYxNzIDEgQ0EgSw5kaXZpZVhAdCBTdzJ3YIpVYyLVBlcnNvbmEg
Tm90IFZhbGlkYXRlZDAeFw05OTA2MTQwMDBaFw0wMDA2MTMyMzU5NT1aMIIB
GJEXMBUAgAUEChMOVaVNAp224sIEluYy4xHzAdBgNVBAsTF1Z1cm1TauduIFRy
dxNOIE51dHdvmxsrYJBEbgNVdT52XJpc21nb15jB20vcmWwb3NpdG9ye
es9SUEgSw5j3JWliBcBzS2WYuExJQiiuTFREkGMPoTGxSDBGBgNVBAMr1ZIcmlTaWdu
mlTaWduIENsYzNzIDEgWzEgSw5kaXZpZVhAdCBTdzJ3YIpVYyLVBlcnNvbmEg
Tm90IFZhbGlkYXRlZDAeFw05OTA2MTQwMDBaFw0wMDA2MTMyMzU5NT1aMIIB
GJEXMBUAgAUEChMOVaVNAp224sIEluYy4xHzAdBgNVBAsTF1Z1cm1TauduIFRy
dxNOIE51dHdvmxsrYJBEbgNVdT52XJpc21nb15jB20vcmWwb3NpdG9ye
es9SUEgSw5j3JWliBcBzS2WYuExJQiiuTFREkGMPoTGxSDBGBgNVBAMr1ZIcmlTaWdu
mlTaWduIENsYzNzIDEgWzEgSw5kaXZpZVhAdCBTdzJ3YIpVYyLVBlcnNvbmEg
Tm90IFZhbGlkYXRlZDAeFw05OTA2MTQwMDBaFw0wMDA2MTMyMzU5NT1aMIIB
GJEXMBUAgAUEChMOVaVNAp224sIEluYy4xHzAdBgNVBAsTF1Z1cm1TauduIFRy
dxNOIE51dHdvmxsrYJBEbgNVdT52XJpc21nb15jB20vcmWwb3NpdG9ye
es9SUEgSw5j3JWliBcBzS2WYuExJQiiuTFREkGMPoTGxSDBGBgNVBAMr1ZIcmlTaWdu
mlTaWduIENsYxNzIDEgQ0EgSw5kaXZpZVhAdCBTdzJ3YIpVYyLVBlcnNvbmEg
Tm90IFZhbGlkYXRlZDAeFw05OTA2MTQwMDBaFw0wMDA2MTMyMzU5NT1aMIIB
GJEXMBUAgAUEChMOVaVNAp224sIEluYy4xHzAdBgNVBAsTF1Z1cm1TauduIFRy
dxNOIE51dHdvmxsrYJBEbgNVdT52XJpc21nb15jB20vcmWwb3NpdG9ye
es9SUEgSw5j3JWliBcBzS2WYuExJQiiuTFREkGMPoTGxSDBGBgNVBAMr1ZIcmlTaWdu
mlTaWduIENsYxNzIDEgQ0EgSw5kaXZpZVhAdCBTdzJ3YIpVYyLVBlcnNvbmEg
Tm90IFZhbGlkYXRlZDAeFw05OTA2MTQwMDBaFw0wMDA2MTMyMzU5NT1aMIIB
GJEXMBUAgAUEChMOVaVNAp224sIEluYy4xHzAdBgNVBAsTF1Z1cm1TauduIFRy
dxNOIE51dHdvmxsrYJBEbgNVdT52XJpc21nb15jB20vcmWwb3NpdG9ye
es9SUEgSw5j3JWliBcBzS2WYuExJQiiuTFREkGMPoTGxSDBGBgNVBAMr1ZIcmlTaWdu
mlTaWduIENsYxNzIDEgQ0EgSw5kaXZpZVhAdCBTdzJ3YIpVYyLVBlcnNvbmEg
Tm90IFZhbGlkYXRlZDAeFw05OTA2MTQwMDBaFw0wMDA2MTMyMzU5NT1aMIIB
GJEXMBUAgAUEChMOVaVNAp224sIEluYy4xHzAdBgNVBAsTF1Z1cm1TauduIFRy
dxNOIE51dHdvmxsrYJBEbgNVdT52XJpc21nb15jB20vcmWwb3NpdG9ye
es9SUEgSw5j3JWliBcBzS2WYuExJQiiuTFREkGMPoTGxSDBGBgNVBAMr1ZIcmlTaWdu
mlTaWduIENsYxNzIDEgQ0EgSw5kaXZpZVhAdCBTdzJ3YIpVYyLVBlcnNvbmEg
Tm90IFZhbGlkYXRlZDAeFw05OTA2MTQwMDBaFw0wMDA2MTMyMzU5NT1aMIIB
GJEXMBUAgAUEChMOVaVNAp224sIEluYy4xHzAdBgNVBAsTF1Z1cm1TauduIFRy
dxNOIE51dHdvmxsrYJBEbgNVdT52XJpc21nb15jB20vcmWwb3NpdG9ye
es9SUEgSw5j3JWliBcBzS2WYuExJQiiuTFREkGMPoTGxSDBGBgNVBAMr1ZIcmlTaWdu
mlTaWduIENsYxNzIDEgQ0EgSw5kaXZpZVhAdCBTdzJ3YIpVYyLVBlcnNvbmEg
Tm90IFZhbGlkYXRlZDAeFw05OTA2MTQwMDBaFw0wMDA2MTMyMzU5NT1aMIIB
GJEXMBUAgAUEChMOVaVNAp224sIEluYy4xHzAdBgNVBAsTF1Z1cm1TauduIFRy
```

-----END CERTIFICATE-----
Defining the Trusted Certificate Authorities

Using Security
Creating a Peer Rules File

When communicating across network links, it is important to validate the peer to which you are connected is the intended or authorized peer. Without this check, it is possible to make a secure connection, exchange secure messages, and receive a valid chain of digital certificates but still be vulnerable to a man-in-the-middle attach. You perform peer validation by verifying a set of specified information contained in the peer digital certificate against a list of information that specifies the rules for validating peer trust. The system administrator maintains the Peer Rules file.

The peer rules are maintained in an ASCII file named peer_val.rul. Store the peer_val.rul file in the following location in the WLE directory structure:

$TUXDIR/udataobj/security/certs

Listing 2-3 provides an example of a Peer Rules file.

Listing 2-3   Example of Peer Rules File

# This file contains the list of rules for validating if
# a peer is authorized as the target of a secure connection
#
O=Ace Industry
O=“BEA Systems, Inc.”; OU=Enterprise Engineering; L=Nashua; S=NH
O=“Netscape Communications, Corp.”, C=US
O=Ace Industry, ou=QA, cn=www.ace.com

Each rule in the Peer Rules file is comprised of a set of elements that are identified by a key. The WLE product recognizes the key names listed in Table 2-1.

Table 2-1  Supported Keys for Peer Rules File

<table>
<thead>
<tr>
<th>Key</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>CN</td>
<td>CommonName</td>
</tr>
<tr>
<td>SN</td>
<td>SurName</td>
</tr>
<tr>
<td>L</td>
<td>LocalityName</td>
</tr>
</tbody>
</table>
Creating a Peer Rules File

Each key is followed by an optional white space, the character =, an optional white space, and finally the value to be compared. The key is not case sensitive. A rule is not a match unless the subject’s distinguished name contains each of the specified elements in the rule and the values of those elements match the values specified in the rule, including case and punctuation.

Each line in the Peer Rules file contains a single rule that is used to determine if a secure connection is to be established. Rules cannot span lines; the entire rule must appear on a single line. Each element in the rule can be separated by either a comma (,) or semi-colon (;) character.

Lines beginning with the pound character (#) are comments. Comments cannot appear on the same line as the name of an organization.

A value must be enclosed in single quotation marks if one of the following cases is true:

- Strings contain any of the following characters:
  , + = " " <CR> < > # ;

- Strings have leading or trailing spaces

- Strings contain consecutive spaces

By default, the WLE product verifies peer information against the Peer Rules file. If you do not want to perform this check, create an empty Peer Rules file.

<table>
<thead>
<tr>
<th>Key</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>StateOrProvinceName</td>
</tr>
<tr>
<td>O</td>
<td>OrganizationName</td>
</tr>
<tr>
<td>OU</td>
<td>OrganizationalUnitName</td>
</tr>
<tr>
<td>C</td>
<td>CountryName</td>
</tr>
<tr>
<td>E</td>
<td>EmailAddress</td>
</tr>
</tbody>
</table>

Table 2-1  Supported Keys for Peer Rules File
CHAPTER 3

Configuring the WLE Environment for the SSL Protocol

This topic includes the following sections:

- Setting Parameters for the SSL Protocol
- Defining a Port for SSL Communications
- Enabling Certificate-based Authentication
- Enabling Host Matching
- Setting the Encryption Strength
- Setting the Interval for Session Renegotiation
- Defining Security Parameters for the IIOP Listener/Handler
- Example of Setting Parameters on the ISL System Process
- Example of Setting Command Line Options on the CORBA C++ ORB

Perform the tasks in this topic only if you are using the SSL Protocol or certificate-based authentication.
Setting Parameters for the SSL Protocol

To use the SSL protocol or certificate-based authentication with the IIOP Listener/Handler, the CORBA C++ object request broker (ORB), or the CORBA Java ORB, you need to:

- Specify the secure port on which SSL connections will be accepted.
- Enable certificate-based authentication.
- Specify the strength that will be used when encrypting data.
- Optionally, set the interval for session renegotiation (IIOP Listener/Handler only).

The following sections detail how to use the options of the ISL command, the command line options of the CORBA C++ ORB, or the system properties of the CORBA Java ORB to set these SSL parameters.

Defining a Port for SSL Communications

To define a port for SSL communications:

- Use the \(-s\) option of the ISL command to specify which port of the IIOP Listener/Handler will listen for secure connections using the SSL protocol. You can configure the IIOP Listener/Handler to allow only SSL connections by setting the \(-s\) option and \(-n\) option of the ISL command to the same value.

- If you are using a remote CORBA C++ or CORBA Java ORB, use the \(-ORBsecurePort\) command line option or system property on the ORB to specify which port of the ORB will listen for secure connections using the SSL protocol. You should set this command line option or system property when using callback objects or the WLE Notification Service.

**Note:** If you are using the SSL protocol with a joint client/server application, you must specify a port number for SSL communications. You cannot use the default.
Enabling Certificate-based Authentication

Defining a secure port for SSL communication requires the WLE Security Pack to be installed. If the -S option or the -ORBsecurePort command line option or system property is executed and a license to enable the use of the SSL protocol does not exist, the IIOP Listener/Handler, CORBA C++ ORB, or CORBA Java ORB will not start.

Enabling Certificate-based Authentication

To enable certificate-based authentication:

- Use the -a option of the ISL command to specify that certificate-based authentication must be used by applications connecting to the IIOP Listener/Handler.

- Use the -ORBmutualAuth command line option or system property on the ORB to specify that certificate-based authentication must be used by applications connecting to the CORBA C++ or CORBA Java ORB.

Enabling certificate-based authentication requires the WLE Security Pack to be installed. If the -a option or the -ORBmutualAuth command line option or system property is executed and a license to enable the use of the SSL protocol does not exist, the IIOP Listener/Handler, CORBA C++ ORB, or CORBA Java ORB will not start.

Enabling Host Matching

The SSL protocol is capable of encrypting messages for confidentiality; however, the use of encryption does nothing to prevent a man-in-the-middle attack. During a man-in-the-middle attack, a principal masquerades as the location from which an initiating application retrieves the initial object references used in the bootstrapping process.

To prevent man-in-the-middle attacks, it is necessary to perform a check to ensure that the digital certificate received during an SSL connection is for the principal for which the connection was intended. Host Matching is a check that the host specified in the object reference used to make the SSL connection matches the common name in the
subject in the distinguished name specified in the target’s digital certificate. Host Matching is performed only by the initiator of an SSL connection, and confirms that the target of a request is actually located at the same network address specified by the domain name in the target’s digital certificate. If this comparison fails, the initiator of the SSL connection refuses to authenticate the target and drops the SSL connection. Host Matching is not technically part of the SSL protocol and is similar to the same check done in Web browsers.

The domain name contained in the digital certificate must match exactly the host information contained in the object reference. Therefore, the use of DNS host names instead of IP addresses is strongly encouraged.

By default Host Matching in enabled in the IIOP Listener/Handler and in the CORBA C++ and CORBA Java ORBs. If you need to enable Host Matching, do one of the following:

- In the IIOP Listener/Handler, specify the `-v` option of the ISL command.
- In the CORBA C++ or CORBA Java ORBs, specify the `-ORBpeerValidate` command line option or system property.

The values for the `-v` option and the `-ORBpeerValidate` command line option or system property are as follows:

- `none`—No host matching is performed.
- `detect`—If the object reference used to make the SSL connection does not match the host name in the target’s certificate, the IIOP Listener/Handler or the ORB does not authenticate the target and drops the SSL connection. The `detect` value is the default value.
- `warn`—If the object reference used to make the SSL connection does not match the host name in the target’s certificate, the IIOP Listener/Handler or the ORB sends a message to the user log and continues processing.

If there is more than one IIOP Listener/Handler in a WLE domain configured for SSL connections (for example, in the case of fault tolerance), BEA recommends using DNS alias names for the IIOP Listener/Handlers or creating different digital certificates for each IIOP Listener/Handler. The `-H` switch on the IIOP Listener can be used to specify the DNS alias name so that object references will be created correctly.
Setting the Encryption Strength

To set the encryption strength:

- Use the `-z` and `-Z` options of the ISL command to set the encryption strength in the IIOP Listener/Handler.
- Use the `-ORBminCrypto` and `-ORBmaxCrypto` command line option or system property on the ORB to set the encryption strength in the CORBA C++ or CORBA Java ORB.

The `-z` option and the `-ORBminCrypto` command line option or system property set the minimum level of encryption used when an application establishes an SSL connection with the IIOP Listener/Handler, the CORBA C++ ORB, or the CORBA Java ORB. The valid values are 0, 40, 56, and 128. 0 means the data is signed but not sealed while 40, 56, and 128 specify the length (in bits) of the encryption key. If this minimum level of encryption is not met, the SSL connection fails. The default is 40.

The `-Z` option and the `-ORBmaxCrypto` command line option or system property set the maximum level of encryption used when an application establishes an SSL connection with the IIOP Listener/Handler, the CORBA C++ ORB, or the CORBA Java ORB. The valid values are 0, 40, 56, and 128. 0 means that data is signed but not sealed while 40, 56, and 128 specify the length (in bits) of the encryption key. The default minimum value is 40. The default maximum value is whatever capability is specified by the license.

The `-z` or `-Z` options and the `-ORBminCrypto` and `-ORBmaxCrypto` command line options or system properties are available only if the WLE Security pack is installed.

To change the strength of encryption currently used in a WLE application, you need to shut down the IIOP Listener/Handler or the ORB.

The combination in which you set the encryption values is important. The encryption values set in the initiator of an SSL connection need to be a subset of the encryption values set in the target of an SSL connection.

Table 3-1 lists combinations of encryption values and describes the encryption behavior.
### Table 3-1 Combinations of Encryption Values

<table>
<thead>
<tr>
<th>ORBminCrypto</th>
<th>ORBmaxCrypto</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No value specified</td>
<td>No value specified</td>
<td>If the use of the SSL protocol is specified by some other command line option or system property but no values are specified for ORBminCrypto and ORBmaxCrypto, these command line options or system properties are assigned their default values.</td>
</tr>
<tr>
<td>0</td>
<td>No value specified</td>
<td>Maximum encryption defaults to the maximum value specified in the license. Tamper/replay detection and privacy protection are negotiated.</td>
</tr>
<tr>
<td>No value specified</td>
<td>0</td>
<td>Tamper/replay detection is negotiated. Privacy protection is not provided.</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Tamper/replay detection is negotiated. Privacy protection is not provided.</td>
</tr>
<tr>
<td>40, 56, 128</td>
<td>No value specified</td>
<td>Maximum encryption defaults to the maximum value specified in the license. Privacy protection can be negotiated to the maximum allowed by the SSL license.</td>
</tr>
<tr>
<td>No value specified</td>
<td>40, 56, 12</td>
<td>Privacy protection can be negotiated to the value specified by the -z option as long as it is less than the maximum allowed by the SSL license. The -z option defaults to 40.</td>
</tr>
<tr>
<td>40, 56, 128</td>
<td>40, 56, 128</td>
<td>Privacy protection can be negotiated between the values specified by the -z option up to the value specified by the -Z option as long as the values are less than the maximum allowed by the SSL license.</td>
</tr>
</tbody>
</table>

**Note:** In all combinations listed in Table 3-1, the value of the SSL license controls the maximum bit strength. If a bit strength is specified beyond the maximum licensed value, the IIOP Listener/Handler or ORB will not start and an error will be generated indicating the bit strength setting is invalid. Stopping the
IOP Listener/Handler or ORB from starting, instead of lowering the maximum value and giving only a warning, protects against an incorrectly configured application running with less protection than was expected.

If a cipher that exceeds the maximum licensed bit strength is somehow negotiated, the SSL connection is not established.

The WLE product supports the cipher suites described in Table 3-2.

Table 3-2  SSL Cipher Suites Supported by the WLE Product

<table>
<thead>
<tr>
<th>Cipher Suite</th>
<th>Key Exchange Type</th>
<th>Symmetric Key Strength</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL_RSA_WITH_RC4_128_SHA</td>
<td>RSA</td>
<td>128</td>
</tr>
<tr>
<td>SSL_RSA_WITH_RC4_128_MD5</td>
<td>RSA</td>
<td>128</td>
</tr>
<tr>
<td>SSL_RSA_WITH_DES_CBC_SHA</td>
<td>RSA</td>
<td>56</td>
</tr>
<tr>
<td>SSL_RSA_EXPORT_WITH_RC4_40_MD5</td>
<td>RSA</td>
<td>40</td>
</tr>
<tr>
<td>SSL_RSA_EXPORT_WITH_DES40_CBC_SHA</td>
<td>RSA</td>
<td>40</td>
</tr>
<tr>
<td>SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5</td>
<td>RSA</td>
<td>40</td>
</tr>
<tr>
<td>SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA</td>
<td>Diffie Hellman</td>
<td>40</td>
</tr>
<tr>
<td>SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA</td>
<td>Diffie Hellman</td>
<td>40</td>
</tr>
<tr>
<td>SSL_RSA_WITH_3DES_EDE_CBC_SHA</td>
<td>RSA</td>
<td>112</td>
</tr>
<tr>
<td>SSL_RSA_WITH_NULL_SHA</td>
<td>RSA</td>
<td>0</td>
</tr>
<tr>
<td>SSL_RSA_WITH_NULL_MD5</td>
<td>RSA</td>
<td>0</td>
</tr>
</tbody>
</table>
Configuring the WLE Environment for the SSL Protocol

Setting the Interval for Session Renegotiation

Note: You set the interval for session renegotiation only in the IIOP Listener/Handler.

Use the -R option of the ISL command to control the time between session renegotiations. Periodic renegotiation of an SSL session refreshes the symmetric keys used to encrypt and decrypt information which limits the time a symmetric key is exposed. You can keep long-term SSL connections more secure by periodically changing the symmetric keys used for encryption.

The -R option specifies the renegotiation interval in minutes. If an SSL connection does renegotiate within the specified interval, the IIOP Listener/Handler will request the application to renegotiate the SSL session for inbound connections or actually perform the renegotiation in the case of outbound connections. The default is 0 minutes which results in no periodic session renegotiations.

You can not use session renegotiation when enabling certificate-based authentication using the -a option of the ISL command.

Defining Security Parameters for the IIOP Listener/Handler

For the IIOP Listener/Handler to participate in SSL connections, the IIOP Listener/Handler authenticates itself to the peer that initiated the SSL connection. This authentication requires a digital certificate. The private key associated with the digital certificate is used as part of establishing an SSL connection that results in an agreement between the principal and the peer (in this case a client application and the IIOP Listener/Handler) on the session key. The session key is a symmetric key (as opposed to the private-public keys) that is used to encrypt data during an SSL session. You define the following information for the IIOP Listener/Handler so that it can be authenticated by peers:
Example of Setting Parameters on the ISL System Process

- **SEC_PRINCIPAL_NAME**
  Specifies the identity of the IIOP Listener/Handler.

- **SEC_PRINCIPAL_LOCATION**
  Specifies the location of the private key file. For example, $TUXDIR/udataobj/security/keys/milozzi.pem.

- **SEC_PRINCIPAL_PASSVAR**
  Specifies an environment variable that holds the pass phrase for the private key of the IIOP Listener/Handler. If this parameter is not specified, you will be prompted for it when you enter the `tmloadcf` command.

These parameters are included in the part of the `SERVERS` section of the `UBBCONFIG` file that defines the ISL system process.

Example of Setting Parameters on the ISL System Process

You set parameters for the SSL protocol in the portion of the `SERVERS` section of the `UBBCONFIG` that defines information for the ISL server process. Listing 3-1 includes code from a `UBBCONFIG` file that set parameters to configure the IIOP Listener/Handler for the SSL protocol and certificate-based authentication.

**Listing 3-1  Using the ISL Command in the UBBCONFIG File**

```bash
... ISL SRVGRP = SYS_GRP SRVID = 5 CLOPT = "-A -- -a -z40 -z128 -S3579 -n //ICEPICK:2569 SEC_PRINCIPAL_NAME="BLOTTO" SEC_PRINCIPAL_LOCATION="BLOTTO.pem" SEC_PRINCIPAL_VAR="AUDIT_PASS"
```

Using Security  3-9
Example of Setting Command Line Options on the CORBA C++ ORB

Listing 3-2 contains sample code that illustrates using the command line options on the CORBA C++ ORB to configure the ORB for the SSL protocol.

**Listing 3-2 Example of Setting the Command Line Options on the CORBA C++ ORB**

ChatClient  -ORBid BEA_IIO
-ORBsecurePort 2100
-ORBminCrypto 40
-ORBMaxCrypto 128
TechTopics

Example of Setting System Properties on the CORBA Java ORB

Listing 3-3 contains sample code that illustrates using the system properties of the CORBA Java ORB to configure the ORB for the SSL protocol.

**Listing 3-3 Example of Setting the System Properties on the CORBA Java ORB**

ChatClient  -DTOBJADDR=corbalocs://piglet:1900
-Dorg.omg.CORBA=ORBPort=1948
-classpath=%CLASSPATH% client
-ORBMaxCrypto 128
4 Defining Security for a WLE CORBA Application

This topic includes the following sections:

- Setting Parameters for Security in the UBBCONFIG File
- Defining Authorized Users
Setting Parameters for Security in the UBBCONFIG File

To configure security for your WLE application, you need to set parameters in the UBBCONFIG file that define the following:

- The server process being used as the authentication server in the WLE application. (This parameter is required for Username/Password authentication only).
- The security level of the WLE application.
- The level of encryption to be used when using link-level encryption.
- The identity of the IIOP Listener/Handler, which is the location of and the password phrase for the private key for the IIOP Listener/Handler. (These parameters are required for certificate-based authentication only).

Note: For information about setting security parameters for the IIOP Listener/Handler in the UBBCONFIG file, see “Defining Security Parameters for the IIOP Listener/Handler” on page 3-8.

To set parameters in the UBBCONFIG file, open the file in any text editor. The parameters for security take effect when you use the tmloadcf command to update the configuration parameters for your WLE application. The following sections describe setting the parameters for security in the UBBCONFIG file.

Configuring the Authentication Server

Note: You only need to configure the Authentication Server, if you have specified a value of USER_AUTH or higher for the SECURITY parameter.

Username/Password authentication requires that an authentication server be configured for the purpose of authenticating users by checking their individual passwords against a file of legal users. The WLE system uses a default authentication server called AUTHSRV to perform authentication.
Setting Parameters for Security in the UBBCONFIG File

For a WLE application to authenticate users, the value of the \texttt{AUTHSVC} parameter in the \texttt{RESOURCES} section of the \texttt{UBBCONFIG} file needs to specify the name of the process to be used as the authentication server for the WLE application. The service must be called \texttt{AUTHSVC}. If the \texttt{AUTHSVC} parameter is specified in the \texttt{RESOURCES} section of the \texttt{UBBCONFIG} file, the \texttt{SECURITY} parameter must also be specified with a value of at least \texttt{USER_AUTH}. If the value is not specified, an error will occur when the system executes the \texttt{tmloadcf} command.

In addition, you need to define \texttt{AUTHSVR} in the \texttt{SERVERS} section of the \texttt{UBBCONFIG} file. The \texttt{SERVERS} section contains information about the server processes to be booted in the WLE application. For more information about the parameters in the \texttt{SERVERS} section of the \texttt{UBBCONFIG} file, see the \textit{Administration Guide} in the WebLogic Enterprise online documentation.

Listing 4-1 contains the portion of the \texttt{UBBCONFIG} file that defines the authentication server.

\textbf{Listing 4-1  Parameters for the Authentication Server}

\begin{verbatim}
*RESOURCES
  SECURITY  USER_AUTH
  AUTHSVC   "AUTHSVC"
*SERVERS
  AUTHSVR
    SRVGRP="SYS_GRP"
    SRVID=1
    RESTART=Y
    GRACE=60
    MAXGEN=2
\end{verbatim}

Defining a Security Level

As part of defining security for a WLE application, you need to define the \texttt{SECURITY} parameter in the \texttt{RESOURCES} section of the \texttt{UBBCONFIG} file. The \texttt{SECURITY} parameter has the following format:

\begin{verbatim}
*RESOURCES
  SECURITY {NONE|APP_PW|USER_AUTH|ACL|MANDATORY_ACL}
\end{verbatim}

Table 4-1 describes the values for the \texttt{SECURITY} parameter.
Defining Security for a WLE CORBA Application

When using Username/Password authentication, the value of the `SECURITY` parameter must be `APP_PW` or greater.

### Table 4-1  Values for the `SECURITY` Parameter

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NONE</td>
<td>Indicates that no password or access checking is performed in the WLE application. Tobj::PrincipalAuthenticator::get_auth_type() returns a value of TOBJ_NOAUTH.</td>
</tr>
<tr>
<td>APP_PW</td>
<td>Indicates that client applications are required to supply an application password to access the WLE domain. The tmloadcf command prompts for an application password. Tobj::PrincipalAuthenticator::get_auth_type() returns a value of TOBJ_SYSAUTH.</td>
</tr>
<tr>
<td>USER_AUTH</td>
<td>Indicates that client applications are required to authenticate themselves to the WLE domain using a password. The value USER_AUTH is similar to APP_PW but, in addition, indicates that user authentication will be done during client initialization. The tmloadcf command prompts for an application password. Tobj::PrincipalAuthenticator::get_auth_type() returns a value of TOBJ_APPAUTH.</td>
</tr>
<tr>
<td>ACL</td>
<td>Indicates that authentication is used in the WLE application and access control checks are performed on interfaces, services, queue names, and event names. If an associated ACL is not found for a name, it is assumed that permission is granted. The tmloadcf command prompts for an application password. Tobj::PrincipalAuthenticator::get_auth_type() returns a value of TOBJ_APPAUTH.</td>
</tr>
<tr>
<td>MANDATORY_ACL</td>
<td>Indicates that authentication is used in the WLE application and access control checks are performed on interfaces, services, queue names, and event names. The value MANDATORY_ACL is similar to ACL, but permission is denied if an associated ACL is not found for the name. The tmloadcf command prompts for an application password. Tobj::PrincipalAuthenticator::get_auth_type() returns a value of TOBJ_APPAUTH.</td>
</tr>
</tbody>
</table>
If the IIOP Listener/Handler is configured for using certificate-based authentication, the value of the `SECURITY` parameter must be `USER_AUTH` or greater.

Setting the Level of Encryption

You can encrypt the messages between WLE applications on different machines in the same WLE domain using link-level encryption. In the `UBBCONFIG` file for each WLE application, you need to set the `MINENCRYPTBITS` and `MAXENCRYPTBITS` parameters for the machines that establish the network connection, as follows.

- The `MINENCRYPTBITS` parameter specifies that at least the defined number of bits are meaningful.
- The `MAXENCRYPTBITS` parameter specifies that encryption should be negotiated up to the defined level.

The possible values for the `MINENCRYPTBITS` and `MAXENCRYPTBITS` parameters are 0, 40, and 128. A value of zero means no encryption is used, while 40 and 128 specify the number of significant bits in the encryption key.

Sample UBBCONFIG File for Username/Password Authentication

Listing 4-3 includes a `UBBCONFIG` file for an application which uses Username/Password authentication. The key sections of the `UBBCONFIG` file are noted in bold face text.

Listing 4-2 Sample UBBCONFIG File for Username/Password Authentication

```
*RESOURCES
IPKEY 55432
DOMAINID securapp
MASTER SITE1
MODEL SHM
LDBAL N
SECURITY USER_AUTH
AUTHSVR "AUTHSVC"
```
*MACHINES
"ICEAXE"
LMID = SITE1
APPDIR = "D:\M3\samples\corba\SECURAPP"
TUXCONFIG = "D:\M3\samples\corba\SECURAPP\results\tuxconfig"
TUXDIR = "D:\WLE5"
MAXWSCLIENTS = 10

*GROUPS
SYS_GRP
  LMID = SITE1
  GRPNO = 1
APP_GRP
  LMID = SITE1
  GRPNO = 2

*SERVERS
DEFAULT:
  RESTART = Y
  MAXGEN = 5

AUTHSVR
  SRVGRP = SYS_GRP
  SRVID = 1
  RESTART = Y
  GRACE = 60
  MAXGEN = 2

TMSYSEVT
  SRVGRP = SYS_GRP
  SRVID = 1

TMFFNAME
  SRVGRP = SYS_GRP
  SRVID = 2
  CLOPT = "-A -- -N -M"

TMFFNAME
  SRVGRP = SYS_GRP
  SRVID = 3
  CLOPT = "-A -- -N"

TMFFNAME
  SRVGRP = SYS_GRP
  SRVID = 4
  CLOPT = "-A -- -F"

simple_server
  SRVGRP = APP_GRP
Setting Parameters for Security in the UBBCONFIG File

SRVID   = 1
RESTART = N

ISL
SRVGRP  = SYS_GRP
SRVID   = 5
CLOPT    = "-A --n //PCWIZ::2500"

Sample UBBCONFIG File for Certificate-Based Authentication

Listing 4-3 includes a UBBCONFIG file for an application which uses certificate-based authentication. The key sections of the UBBCONFIG file are noted in bold face text.

Listing 4-3  Sample UBBCONFIG File for Certificate-Based Authentication

*RESOURCES
IPKEY    55432
DOMAINID simpapp
MASTER    SITE1
MODEL     SHM
LDBAL     N
SECURITY  USER_AUTH

*MACHINES
"ICEAXE"
LMID        = SITE1
APPDIR      = "D:\M3\samples\corba\SIMPAP~1"
TUXCONFIG   = "D:\M3\samples\corba\SIMPAP~1\results\tuxconfig"
TUXDIR      = "D:\WLE5"
MAXWSCLIENTS = 10

*GROUPS
SYS_GRP
LMID    = SITE1
GRPNO   = 1
APP_GRP
LMID    = SITE1
GRPNO   = 2

*SERVERS
DEFAULT:
Defining Authorized Users

As part of configuring security for a WLE application, you need to define the principals and groups of principals who have access to the WLE application. The WLE system uses the email address of a principal to map the external identity of a principal represented by a digital certificate to an identity used by a WLE application to authenticate a principal.
Defining Authorized Users

You use the `tpusradd` command to create files containing lists of authorized principals. The `tpusradd` command adds a new principal entry to the WLE security data files. This information is used by the `AUTHSRV` to authenticate principals. The file that contains the principals is called `tpusr`.

The file is a colon-delimited, flat ASCII file, readable only by the administrator of the WLE application. The system file entries have a limit of 512 characters per line. The file is kept in the application directory, specified by the environment variable `$APPDIR`. The environment variable `$APPDIR` must be set to the path name of the WLE application.

The `tpusradd` file should be owned by the administrator account. BEA recommends that the file be protected so that only the owner has read and write privileges for the file and all other users have only read privileges for the file.

When defining names of authorized users for a WLE EJB, there is a one-to-one association between the users defined with the `tpusradd` command and the security roles defined in the deployment descriptor of the WLE EJB.

The `tpusradd` command has the following options:

- `-u uid`

  The user identification number. `uid` must be a positive decimal integer below 128K. `uid` must be unique within the list of existing identifiers for the application. `uid` defaults to the next available (unique) identifier greater than 0.

- `-c client_name`

  A string of printable characters that specifies the name of the principal. The name may not contain a colon (:), pound sign (#), or a newline (n). The principal name must be unique within the list of existing principals for the WLE application. The name of the principal can be either the name of a WLE client application or a WLE EJB.

Listing 4-4 includes a sample `tpusradd` file.

Listing 4-4 Sample `tpusradd` File

<table>
<thead>
<tr>
<th>Cltname</th>
<th>Uid</th>
</tr>
</thead>
<tbody>
<tr>
<td>milozzi</td>
<td>122</td>
</tr>
<tr>
<td>smart</td>
<td>555</td>
</tr>
<tr>
<td>patt</td>
<td>1234</td>
</tr>
<tr>
<td>butler</td>
<td>15555</td>
</tr>
</tbody>
</table>
Defining Security for a WLE CORBA Application

Note: Use the `tpgrpadd` command to add groups of principals to the WLE security data files.

In addition to the `tpusradd` and `tpgrpadd` commands, the WLE product provides the following commands to modify the `tpusr` and `tpgrp` files:

- `tpusrdel`
- `tpusrmod`
- `tpgrpdel`
- `tpgrpmod`

For a complete description of the commands, see *WLE Reference* in the WebLogic Enterprise online documentation.
CHAPTER

5 Writing a WLE CORBA Application That Implements Security

This topic contains the following sections:

- Understanding the Address Formats of the Bootstrap Object
- Using Username/Password Authentication
- Using Certificate-based Authentication
- Using the Invocations_Options_Required() Method
Understanding the Address Formats of the Bootstrap Object

The Bootstrap object in the WLE product has been enhanced so that users can specify that all communication to a given IIOP Listener/Handler be protected. The Bootstrap object supports corbaloc and corbalocs Uniform Resource Locator (URL) address formats to be used when specifying the location of the IIOP Listener/Handler. The type of security provided depends on the format of URL used to specify the location of the IIOP Listener/Handler.

As with the Host and Port address format, you use the URL address formats to specify the location of the IIOP Listener/Handler, but the bootstrapping process behaves differently. When using the corbaloc or corbalocs URL address format, the initial connection to the IIOP Listener/Handler is deferred until either:

- The principal uses Username/Password authentication with either the
  Tobj::PrincipalAuthenticator::logon or the
  SecurityLevel2::PrincipalAuthenticator::authenticate methods.

- The principal calls the Tobj_Bootstrap::resolve_initial_references method using an object ID value other than SecurityCurrent.

Using the corbalocs URL address format indicates that the SSL protocol is used to protect at least the integrity of the connection between the principal and the IIOP Listener/Handler.

Table 5-1 highlights the differences between the two URL address formats.

<table>
<thead>
<tr>
<th>URL Address Formats</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>corbaloc</td>
<td>By default, invocations on the IIOP Listener/Handler are unprotected. Configuring the IIOP Listener/Handler for the SSL protocol is optional. A principal can secure the bootstrapping process by using the SecurityLevel2::Current::authenticate() and SecurityLevel2::Credentials::invocation_options_required() methods to specify that certificate-based authentication is to be used.</td>
</tr>
<tr>
<td>corbalocs</td>
<td></td>
</tr>
</tbody>
</table>
Understanding the Address Formats of the Bootstrap Object

Both the corbaloc and corbalocs URL address formats provide stringified object references that are easily manipulated in both TCP/IP and Domain Name System (DNS) environments. The corbaloc and corbalocs URL address formats contain a DNS-style host name or an IP address and port.

The URL address formats follow and extend the definition of object URLs adopted by the Object Management Group (OMG) as part of the Interoperable Naming Service submission. The WLE software also extends the URL format described in the OMG Interoperable Naming Service submission to support a secure form that is modeled after the URL for secure HTTP, as well as to support functionality in previous releases of the WLE product.

Listing 5-1 contains examples of the new URL address formats.

**Listing 5-1   Examples of the corbaloc and corbalocs URL Address Formats**

```plaintext
corbaloc://555xyz.com:1111
corbalocs://24.128.122.32:1011, corbalocs://24.128.122.34
```

As an enhancement to the URL syntax described in the OMG Interoperable Naming Service submission, the WLE product extends the syntax to support a list of multiple URLs, each with a different scheme. Listing 5-2 contains examples of specifying multiple URLs.

**Listing 5-2   Examples of Specifying Multiple URL Address Formats**

```plaintext
corbalocs://555xyz.com:1024, corbalocs://555xyz.com:1111
```
In the examples in Listing 5-2, if the parser reaches the URL `corbaloc://force.com:1111`, it resets its internal state as if it had never attempted secure connections, and then begins attempting unprotected connections. This situation occurs if the client application has not set any SSL parameters on the Credentials object.

The following sections describe the behavior when using the different address formats of the Bootstrap object.

### Using the Host and Port Address Format

If a WLE client application uses the Host and Port address format of the Bootstrap object, the constructor method of the Bootstrap object constructs an object reference using the specified host name and port number. The invocation to the IIOP Listener/Handler is made without the protections offered by the SSL protocol.

The client application can still authenticate using Username/Password authentication. However, since the bootstrapping process is performed over an unprotected and unverified link, all communications are vulnerable to the following security attacks:

- The Man-in-the-Middle attack, because there was no verification that the principal to which the connection was made was the desired principal.
- The Denial of Service attack, because no object references were returned, the object references returned were invalid, or the security token was invalid.
- The Sniffer attack, because the information was sent in the clear so that anyone with a packet sniffer can see the content of a message that was not encrypted (for example, only the username/password information is encrypted).
- The Tamper attack, because the integrity of the information is not protected. The contents of the message could be changed and the change would not be detected.
- The Replay attack, because the same request can be sent repeatedly without detection.

**Note:** If the IIOP Listener/Handler is configured for the SSL protocol and the Host and Port address format of the Bootstrap object is used, the invocation on the specified WLE object results in a `INVALID_DOMAIN` exception.
Using the corbaloc URL Address Format

By default, the invocation on the IIOP Listener/Handler is unprotected when using the corbaloc URL address format and Username/Password authentication. Therefore, all communications are vulnerable to the following security attacks:

- The Man-in-the-Middle attack, because there was no verification that the principal to which the connection was made was the desired principal.
- The Denial of Service attack, because no object references were returned, the object references returned were invalid, or the security token was invalid.
- The Sniffer attack, because the information was sent in the clear so that anyone with a packet sniffer can see the content of a message that was not encrypted (for example, only the username/password information is encrypted).
- The Tamper attack, because the integrity of the information is not protected. The content of the message could be changed and the change would not be detected.
- The Replay attack, because the same request can be sent repeatedly without detection.

You can protect the bootstrapping process when using the corbaloc URL address format by using the SecurityLevel2::Current::authenticate() method, specifying that certificate-based authentication is to be used, and setting the invocation_methods_required method on the Credentials object.

Note: If the IIOP Listener/Handler is configured for the SSL protocol and not for certificate-based authentication and the corbaloc URL address format is used, the invocation on the specified WLE object results in an INVALID_DOMAIN exception.

BEA recommends that existing WLE applications migrate to the corbaloc URL address format instead of using the Host and Port Address format.

Using the corbalocs URL Address Format

The corbalocs URL address format is the recommended format to use to ensure that communications between principals and the IIOP Listener/Handler are protected. The corbalocs URL address format functions in the same way as the corbaloc URL.
address format, except the SSL protocol is used to protect all communications with the IIOP Listener/Handler, the CORBA C++ ORB, or the CORBA Java ORB regardless of the type of authentication used.

When the defaults are used with the corbaloc URL address format, communications are vulnerable only to Denial of Service security attacks. Using the SSL protocol and certificate-based authentication guards against Sniffer, Tamper, and Replay attacks. In addition, the validation check of the host specified in the digital certificate guards against Man-in-the-Middle attacks.

To use the corbaloc URL address format, the IIOP Listener/Handler, the CORBA C++ ORB, or the CORBA Java ORB must be configured to enable the use of the SSL protocol. For more information about configuring the IIOP Listener/Handler, the CORBA C++ ORB, or the CORBA Java ORB for the SSL protocol, see “Configuring the WLE Environment for the SSL Protocol” on page 3-1.

Using Username/Password Authentication

This section describes implementing Username/Password authentication in WLE applications.

The Security Sample Application

The Security sample application demonstrates Username/Password authentication. The Security sample application requires each student using the application to have an ID and a password. The Security sample application works in the following manner:

1. The client application has a logon method. This method invokes operations on the PrincipalAuthenticator object, which is obtained as part of the process of logging on to access the domain.

2. The server application implements a get_student_details() method on the Registrar object to return information about a student. After the user is authenticated and the logon is complete, the get_student_details() method accesses the student information in the database to obtain the student information needed by the client logon method.
3. The database in the Security sample application contains course and student information.

Figure 5-1 illustrates the Security sample application.

**Figure 5-1 Security Sample Application**

The source files for the Security sample application are located in the `\samples\corba\university` directory in the WLE software. For information about building and running the Security sample application, see “Building and Running the CORBA Sample Applications” on page 6-1.

**Writing the Client Application**

When using Username/Password authentication, write client application code that does the following:

1. Uses the Bootstrap object to obtain a reference to the SecurityCurrent object for the specific WLE domain. You can use the Host and Port Address format, the `corbaloc` URL address format, or the `corbalocs` URL address format.
2. Gets the PrincipalAuthenticator object from the SecurityCurrent object.

3. Uses one of the following methods to authenticate the principal:
   - C++—SecurityLevel2::PrincipalAuthenticator::authenticate() using Tobj::TuxedoSecurity
   - Java—SecurityLevel2.PrincipalAuthenticator.authenticate() using Tobj::TuxedoSecurity
   - C++—Tobj::PrincipalAuthenticator::logon()
   - Java—Tobj.PrincipalAuthenticator.logon()

The SecurityLevel2::PrincipalAuthenticator interface is defined in the CORBAservices Security Service specification. This interface contains two methods that are used to accomplish the authentication of the principal. There are two methods because authentication of principals may require more than one step. The authenticate() method allows the caller to authenticate and optionally select attributes for the principal of this session.

The WLE product extends the PrincipalAuthenticator object with functionality to support similar security to that found in BEA TUXEDO. The enhanced functionality is provided by the Tobj::PrincipalAuthenticator interface.

The methods defined for the Tobj::PrincipalAuthenticator interface provide a focused, simplified form of the equivalent CORBA-defined interface. You can use either the CORBA-defined or the WLE extensions when developing a WLE application.

The Tobj::PrincipalAuthenticator interface provides the same functionality as the SecurityLevel2::PrincipalAuthenticator interface. However, unlike the SecurityLevel2::PrincipalAuthenticator::authenticate() method, the logon() method of the Tobj::PrincipalAuthenticator interface does not return a Credentials object. As a result, WLE applications that need to use more than one principal identity are required to call the Current::get_credentials() method immediately after the logon() method to retrieve the Credentials object as a result of the logon. Retrieval of the Credentials object directly after a logon method should be protected with serialized access.

The following sections contain C++ and Java code examples that illustrate implementing Username/Password authentication. For a Visual Basic code example, see “Automation Security Reference” on page 13-1.
C++ Code Example of Using the SecurityLevel2::PrincipalAuthenticator::authenticate() Method

Listing 5-3 contains C++ code that performs Username/Password authentication using the SecurityLevel2::PrincipalAuthenticator::authenticate() method.

Listing 5-3   C++ Client Application that uses the SecurityLevel2::PrincipalAuthenticator::authenticate() Method

```cpp
//Create Bootstrap object
Tobj_Bootstrap* bootstrap = new Tobj_Bootstrap(orb,
corbalocs://sling.com:2143);

//Get SecurityCurrent object
CORBA::Object_var var_security_current_oref =
    bootstrap.resolve_initial_references("SecurityCurrent");
SecurityLevel2::Current_var var_security_current_ref =
    SecurityLevel2::Current::_narrow(var_security_current_oref.in());

//Get the PrincipalAuthenticator
SecurityLevel2::PrincipalAuthenticator_var var_principal_authenticator_oref =
    var_security_current_oref->principal_authenticator();

const char * user_name = "john"
const char * client_name = "university";
char system_password[31] = {'\0'};
char user_password[31] = {'\0'};

//Determine the security level
Tobj::AuthType auth_type = var_bea_principal_authenticator->get_auth_type();
switch (auth_type)
{
    case Tobj::TOBJ_NOAUTH:
        break;
    case Tobj::TOBJ_SYSAUTH
        strcpy(system_password, "sys_pw");
        break;
    case Tobj::TOBJ_APPAUTH
        strcpy(system_password, "sys_pw");
        strcpy(user_password, "john_pw");
        break;
}
if (auth_type != Tobj::TOBJ_NOAUTH)
```


Java Code Example of Using the
SecurityLevel2.PrincipalAuthenticator.authenticate() Method

Listing 5-4 contains Java code that performs Username/Password authentication using the SecurityLevel2.PrincipalAuthenticator.authenticate() method.

Listing 5-4  Java Client Application that uses the

5-10 Using Security
Using Username/Password Authentication

**SecurityLevel2.PrincipalAuthenticator.authenticate() Method**

...  
// Create Bootstrap object  
Tobj_Bootstrap bs =  
    new Tobj_Bootstrap(orb, corbalocs://sling.com:2143);  
// Get SecurityCurrent object  
org.omg.CORBA.Object secCurObj =  
    bs.resolve_initial_references( "SecurityCurrent" );  
    org.omg.SecurityLevel2.CurrentHelper.narrow(secCurObj);  
// Get Principal Authenticator  
org.omg.Security.PrincipalAuthenticator princAuth =  
    secCur2Obj.principal_authenticator();  
com.beasys.Tobj.PrincipalAuthenticator auth =  
    Tobj.PrincipalAuthenticatorHelper.narrow(princAuth);  
// Get Authentication type  
com.beasys.Tobj.AuthType authType = auth.get_auth_type();  
// Initialize arguments  
String userName = "John";
String clientName = "Teller";
String systemPassword = null;
String userPassword = null;
byte[] userData = new byte[0];  
// Prepare arguments according to security level requested  
switch(authType.value())  
{  
    case com.beasys.Tobj.AuthType._TPNOAUTH:  
        break;  
    case com.beasys.Tobj.AuthType._TPSYSAUTH:  
        systemPassword = "sys_pw";
        break;  
    case com.beasys.Tobj.AuthType._TPAPPAUTH:  
        systemPassword = "sys_pw";
        userPassword = "john_pw";
        break;  
}  
// Build security data
org.omg.Security.OpaqueHolder auth_data =  
    new org.omg.Security.OpaqueHolder();  
org.omg.Security.AttributeListHolder privs =
Writing a WLE CORBA Application That Implements Security

```java
new Security.AttributeListHolder();
auth.build_auth_data(userName, clientName, systemPassword,
userPassword, userData, authData, privs);

// Authenticate user
org.omg.SecurityLevel2.CredentialsHolder creds =
    new org.omg.SecurityLevel2.CredentialHolder();
org.omg.Security.OpaqueHolder cont_data =
    new org.omg.Security.OpaqueHolder();
org.omg.Security.OpaqueHolder auth_spec_data =
    new org.omg.Security.OpaqueHolder();
    auth.authenticate(com.beasys.Tobj.TuxedoSecurity.value,
0, userName, auth_data.value(),
    privs.value(), creds, cont_data, auth_spec_data);
if (status != AuthenticatoinStatus.SecAuthSuccess)
    System.exit(1);
```

C++ Code Example of Using the Tobj::PrincipalAuthenticator::logon() Method

Listing 5-5 contains C++ code that performs Username/Password authentication using the Tobj::PrincipalAuthenticator::logon() method.

Listing 5-5   C++ Client Application that uses the Tobj::PrincipalAuthenticator::logon() Method

```c++
...  
CORBA::Object_var var_security_current_oref =
    bootstrap.resolve_initial_references("SecurityCurrent");
SecurityLevel2::Current_var var_security_current_ref =
    SecurityLevel2::Current::_narrow(var_security_current_oref.in());

//Get the PrincipalAuthenticator
SecurityLevel2::PrincipalAuthenticator_var var_principal_authenticator_oref =
    var_security_current_oref->principal_authenticator();

//Narrow the PrincipalAuthenticator
Tobj::PrincipalAuthenticator_var var_bea_principal_authenticator =
    Tobj::PrincipalAuthenticator::narrow
    var_principal_authenticator_oref.in());
```

5-12    Using Security
Using Username/Password Authentication

const char * user_name = "john";
const char * client_name = "university";
char system_password[31] = {'\0'};
char user_password[31] = {'\0'};

//Determine the security level
Tobj::AuthType auth_type = var_bea_principal_authenticator->get_auth_type();
switch (auth_type)
{
    case Tobj::TOBJ_NOAUTH:
        break;
    case Tobj::TOBJ_SYSAUTH:
        strcpy(system_password, "sys_pw");
        break;
    case Tobj::TOBJ_APPAUTH:
        strcpy(system_password, "sys_pw");
        strcpy(user_password, "john_pw");
        break;
}
if (auth_type != Tobj::TOBJ_NOAUTH)
{
    SecurityLevel2::Credentials_var creds;
    Security::Opaque_var auth_data;
    Security::AttributeList_var privileges;
    Security::Opaque_var cont_data;
    Security::Opaque_var auth_spec_data;
    var_bea_principalauthenticator->build_auth_data(user_name,
                                                    client_name,
                                                    system_password,
                                                    user_password,
                                                    NULL,
                                                    auth_data,
                                                    privileges);
    //Determine the security level
    Tobj::AuthType auth_type = var_bea_principal_authenticator->get_auth_type();
    Security::AuthenticationStatus status = var_bea_principal_authenticator->logon(
        user_name,
        client_name,
        system_password,
        user_password,
        0);

    if (status != Security::SecAuthSuccess)
    {
        //Failed authentication
    }
}
return;
}

// Proceed with application
...

// Log off
try
{
    logoff();
}
...

Java Code Example of Using the Tobj.PrincipalAuthenticator.logon() Method

Listing 5-6 contains Java code that performs Username/Password authentication using the Tobj.PrincipalAuthenticator.logon() method.

Listing 5-6  Java Client Application That Uses the Tobj.PrincipalAuthenticator.logon() Method

... // Create bootstrap object
Tobj_Bootstrap bs =
    new Tobj_Bootstrap(orb, corbaloc://sling.com:2143);

// Get security current
org.omg.CORBA.Object secCurObj =
    bs.resolve_initial_references( "SecurityCurrent" );
    org.omg.SecurityLevel2.CurrentHelper.narrow(secCurObj);

// Get Principal Authenticator
org.omg.Security.PrincipalAuthenticator princAuth =
    secCur2Obj.principal_authenticator();
com.beasys.Tobj.PrincipalAuthenticator auth =
    Tobj.PrincipalAuthenticatorHelper.narrow(princAuth);

// Get Authentication type
com.beasys.Tobj.AuthType authType = auth.get_auth_type();

// Initialize arguments
String userName = "John";
String clientName = "Teller";
String systemPassword = null;
String userPassword = null;
byte[] userData = new byte[0];
Using Certificate-based Authentication

// Prepare arguments according to security level requested
switch(authType.value())
{
    case com.beasys.Tobj.AuthType._TPNOAUTH:
        break;
    case com.beasys.Tobj.AuthType._TPSYSAUTH:
        systemPassword = "sys_pw";
        break;
    case com.beasys.Tobj.AuthType._TPAPPAUTH:
        systemPassword = "sys_pw";
        userPassword = "john_pw";
        break;
}

// TUXEDO-style Authentication
    auth.logon(userName, clientName, systemPassword,
               userPassword, userData);

...  

// Proceed with application

// Log off
try
{
    auth.logoff();
}

...
The Secure Simpapp Sample Application

The Secure Simpapp sample application uses the existing Simpapp sample application and modifies the code and configuration files to support secure communications through the SSL protocol and certificate-based authentication.

The server application in the secure Simpapp sample application provides an implementation of a CORBA object that has the following two methods:

- The `upper` method accepts a string from the client application and converts the string to uppercase letters.
- The `lower` method accepts a string from the client application and converts the string to lowercase letters.

Figure 5-2 illustrates how the Secure Simpapp sample application works.
The Simpapp sample application was modified in the following ways to support certificate-based authentication and the SSL protocol:

- In the **ISL** section of the **UBBCONFIG** file, the `-a`, `-S`, `-z`, and `-Z` options of the ISL command are specified to configure the IIOP Listener/Handler for the SSL protocol.

- In the **ISL** section of the **UBBCONFIG** file, the **SEC_PRINCIPAL_NAME**, the **SEC_PRINCIPAL_LOCATION**, and the **SEC_PRINCIPAL_PASSVAR** parameters are defined to specify proof material for the IIOP Listener/Handler.

- The code for the client application uses the **corbalocs** URL address format.
The code for the client application uses the authenticate() method of the SecurityLevel2:Current interface to authenticate the principal and obtain credentials for the principals.

The source files for the C++ and Java versions of the Secure Simpapp sample application are located in the \samples\corba\simpappSSL and \samples\corba\simpappSSL_java directories of the WLE software. For instructions for building and running the Secure Simpapp sample application, see “Building and Running the CORBA Sample Applications” on page 6-1.

Writing the Client Application

When using certificate-based authentication, write client application code that does the following:

1. Uses the Bootstrap object to obtain a reference to the SecurityCurrent object for the specific WLE domain. Use either the corbalocs URL address format.

2. Gets the PrincipalAuthenticator object from the SecurityCurrent object.

3. Uses the authenticate() method of the SecurityLevel2:Current interface to authenticate the principals and obtain credentials for the principals. When using certificate-based authentication, specify Tobj::CertificateBased for the method argument and the pass phrase for the private key as the auth_data argument for Security::Opaque.

The following sections contain C++ and Java code examples that illustrate implementing certificate-based authentication.

C++ Code Example of Certificate-based Authentication

Listing 5-7 illustrates using certificate-based authentication in a C++ client application.

Listing 5-7   C++ Client Application That Uses Certificate-Based Authentication

....

5-18   Using Security
Using Certificate-based Authentication

// Initialize the ORB
CORBA::ORB_var v_orb = CORBA::ORB_init(argc, argv, "");

// Create the bootstrap object
Tobj_Bootstrap bootstrap(v_orb.in(), corbalocs://sling.com:2143);

// Resolve SecurityCurrent
CORBA::Object_ptr seccurobj = 
    bootstrap.resolve_initial_references("SecurityCurrent");
SecurityLevel2::Current_ptr seccur = 
    SecurityLevel2::Current::_narrow(seccurobj);

// Perform certificate-based authentication
SecurityLevel2::Credentials_ptr the_creds;
Security::AttributeList_varprivileges;
Security::Opaque_var continuation_data;
Security::Opaque_var auth_specific_data;
Security::Opaque_var response_data;

// Principal email address
char emailAddress[] = "milozzi@bigcompany.com;"

// Pass phrase for principal's digital certificate
char password[] = "asdawrewe98infldi7;"

// Convert the certificate private key password to opaque
unsigned long password_len = strlen(password);
Security::Opaque ssl_auth_data(password_len);

// Authenticate principal certificate with principal authenticator
for(int i = 0; (unsigned long) i < password_len; i++)
    ssl_auth_data[i] = password[i];
Security::AuthenticationStatus auth_status;
SecurityLevel2::PrincipalAuthenticator_var PA =
    seccur->principal_authenticator();

auth_status = PA->authenticate(Tobj::CertificateBased,
    emailAddress,
    ssl_auth_data,
    privileges,
    the_creds,
    continuation_data,
    auth_specific_data);

while(auth_status == Security::SecAuthContinue) {
    auth_status = PA->continue_authentication(
        response_data,
        the_creds,
        continuation_data,
        auth_specific_data);
}
Java Code Example of Certificate-based Authentication

Listing 5-8 illustrates using certificate-based authentication in a C++ client application.

Listing 5-8  Java Client Application That Uses Certificate-based Authentication

```java
// Initialize the ORB.
Properties Prop;
Prop = new Properties(System.getProperties());
Prop.put("org.omg.CORBA.ORBClass","com.beasys.CORBA.iiop.ORB");
Prop.put("org.omg.CORBA.ORBSingletonClass",
"com.beasys.CORBA.idl.ORBSingleton");
ORB orb = ORB.init(args, Prop);

// Create the Bootstrap object
Tobj_Bootstrap bs = new Tobj_Bootstrap(orb,
corbalocs://foo:2501);

//Resolve SecurityCurrent
org.omg.CORBA.object ocurr =
  bs.resolve_initial_references("SecurityCurrent");
  org.omg.SecurityLevel2.CurrentHelper.narrow(occur);

// Get Principal Authenticator
com.beasys.Tobj.PrincipalAuthenticator pa =
  (com.beasys.Tobj.PrincipalAuthenticator)
curr.principal_authenticator();

OpaqueHolder auth_data = new OpaqueHolder();
AttributeListHolder privileges = new AttributeListHolder();
org.omg.SecurityLevel2.CredentialsHolder creds =
  new org.omg.SecurityLevel2.CredentialsHolder();
OpaqueHolder continuation_data = new OpaqueHolder();
```
Using the Invocations_Options_Required() Method

When using certificate-based authentication, it may be necessary for a principal to explicitly define the security attributes it requires. For example, a bank application may have specific security requirements it needs met before the bank application can transfer data to a database. The invocation_options_required() method of the SecurityLevel2::Credentials interface allows the principal to explicitly control the security characteristics of the SSL connection. When using the corbaloc URL address format, you can secure the bootstrapping process by using the authenticate() and invocation_options_required() methods of the SecurityLevel2::Credentials interface.

Perform the following steps to use the invocation_options_required() method:

1. Write application code that uses the authenticate() method of the SecurityLevel2::Current object to specify certificate-based authentication is being used
2. Use the `invocation_options_required()` method to specify the security attributes the principal requires. See the description of the `invocation_options_required()` method in the “C++ Security Reference” on page 11-1 and “Java Security Reference” on page 12-1 for a complete list of security options.

Listing 5-9 provides a C++ example of using the `invocation_options_required()` method.

**Listing 5-9  C++ Example of Using the `invocation_options_required()` Method**

```cpp
// Initialize the ORB
CORBA::ORB_var v_orb = CORBA::ORB_init(argc, argv, "")

// Create the bootstrap object
Tobj_Bootstrap bootstrap(v_orb.in(), corbalocs://sling.com:2143);

// Resolve SecurityCurrent
CORBA::Object_ptr seccurobj =
   bootstrap.resolve_initial_references("SecurityCurrent");
SecurityLevel2::Current_ptr seccur =
   SecurityLevel2::Current::_narrow(seccurobj);

// Perform certificate-based authentication
SecurityLevel2::Credentials_ptr the_creds;
Security::AttributeList_var privileges;
Security::Opaque_var continuation_data;
Security::Opaque_var auth_specific_data;
Security::Opaque_var response_data;

//Principal email address
char emailAddress[] = "milozzi@bigcompany.com;"

// Pass phrase for principal’s digital certificate
char password[] = "asdawrew98infldi7;"

// Convert the certificate private key password to opaque
unsigned long password_len = strlen(password);
Security::Opaque ssl_auth_data(password_len);

// Authenticate principal certificate with principal authenticator
unsigned long password_len = strlen(password);
for(int i = 0; (unsigned long) i < password_len; i++)
   ssl_auth_data[i] = password[i];

Security::AuthenticationStatus auth_status;

SecurityLevel2::PrincipalAuthenticator_var PA =
   seccur->principal_authenticator();
```
auth_status = PA->authenticate(Tobj::CertificateBased, emailAddress, ssl_auth_data, privileges, the_creds, continuation_data, auth_specific_data);
the_creds->invocation_options_required(
  Security::Integrity|
  Security::DetectReplay|
  Security::DetectMisordering|
  Security::EstablishTrustInTarget|
  Security::EstablishTrustInClient|
  Security::SimpleDelegation);

while(auth_status == Security::SecAuthContinue) {
  auth_status = PA->continue_authentication(
    response_data, the_creds, continuation_data, auth_specific_data);
}
...

Listing 5-10 provides a Java example of using the invocation_options_required() method

Listing 5-10  Java Example of Using the invocation_options_required() Method

// Initialize the ORB.

Properties Prop;
Prop = new Properties(System.getProperties());
Prop.put("org.omg.CORBA.ORBClass","com.beasys.CORBA.iior.ORB");
Prop.put("org.omg.CORBA.ORBSingletonClass",
  "com.beasys.CORBA.idl.ORBSingleton");

ORB orb = ORB.init(args, Prop);

// Create the Bootstrap object

Tobj_Bootstrap bs = new Tobj_Bootstrap(orb, corbalocs://foo:2501);
Writing a WLE CORBA Application That Implements Security

//Resolve SecurityCurrent
org.omg.CORBA.object ocurr =
    bs.resolve_initial_references("SecurityCurrent");
    org.omg.SecurityLevel2.CurrentHelper.narrow(occur);

// Get Principal Authenticator
com.beasys.Tobj.PrincipalAuthenticator pa =
    (com.beasys.Tobj.PrincipalAuthenticator)
    curr.principal_authenticator();
OpaqueHolder auth_data = new OpaqueHolder();
AttributeListHolder privileges = new AttributeListHolder();
org.omg.SecurityLevel2.CredentialsHolder creds =
    new org.omg.SecurityLevel2.CredentialsHolder();
OpaqueHolder continuation_data = new OpaqueHolder();
OpaqueHolder auth_specific_data = new OpaqueHolder();
auth_data.value=new String ("deathstar").getBytes("UTF8");
if(pa.authenticate(com.beasys.Tobj.CertificateBased.value,
    "vader@largecompany.com",
    auth_data.value,
    privileges.value,
    the_creds,
    continuation_data,
    auth_specific_data)
org.omg.SecurityLevel2.Credentials credentials = curr.get_credentials(
    org.omg.Security.CredentialType.SecInvocationCredentials);

credentials.invocation_options_required(
    (short) (org.omg.Security.Integrity.value |
    org.omg.Security.DetectMisordering.value| 
    org.omg.Security.EstablishTrustInTarget.value| 
    org.omg.Security.EstablishTrustInClient.value| 
);
!AuthenticationStatus.SecAuthSuccess) {
    System.err.println("logon failed");
    System.exit(1);
}

...
6 Building and Running the CORBA Sample Applications

The topic contains the following sections:

- Building and Running the Security Sample Application
- Building and Running the Secure Simpapp Sample Application
Building and Running the Security Sample Application

The Security sample application demonstrates using Username/Password authentication. The sample application has both C++ and Java client applications. For a description of the Security sample application, see “Writing a WLE CORBA Application That Implements Security” on page 5-1. This section describes how to build the Security sample application and how to use the client applications in the Security sample application.

Perform the following steps to build the Security sample application:

1. Copy the files for the Security sample application into a work directory.
2. Verify the settings of the environment variables.
3. Change the protection on the files for the Security sample application.
4. Set the environment variables.
5. Initialize the database.
7. Compile the client and server sample applications.
8. Start the server application in the Security sample application.
10. Start the Java client application in the Security sample application.

The following sections describe these steps.

Refer to Readme.txt in the \WLEdir\samples\corba\university\security directory for troubleshooting information and the latest information about using the Security sample application.
Building and Running the Security Sample Application

Step 1: Copy the files for the Security sample application into a work directory.

You need to copy the files for the Security sample application into a work directory on your local machine. The files for the Security sample application are located in the following directories:

**Windows NT**

drive:\WLEdir\samples\corba\university\security

**UNIX**

/usr/WLEdir/samples/corba/university/security

In addition, you need to copy the `utils` directory into your work directory. The `utils` directory contains files that set up logging, tracing, and access to the database used with the Security sample application.

You will use the files listed in Table 6-1 to create the Security sample application.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>univs.idl</td>
<td>The OMG IDL code that declares the CourseSynopsisEnumerator, Registrar, and RegistrarFactory interfaces.</td>
</tr>
<tr>
<td>univss.cpp</td>
<td>The C++ source code for the server application in the Security sample application.</td>
</tr>
<tr>
<td>univs_i.h</td>
<td>The C++ source code for method implementations of the CourseSynopsisEnumerator, Registrar, and RegistrarFactory interfaces.</td>
</tr>
<tr>
<td>univs_i.cpp</td>
<td></td>
</tr>
<tr>
<td>univsc.cpp</td>
<td>The C++ source code for the C++ client application in the Security sample application.</td>
</tr>
<tr>
<td>UnivSApplet.java</td>
<td>The Java source code for the Java client application in the Security sample application.</td>
</tr>
</tbody>
</table>
### Table 6-1  Files Included in the Security Sample Application

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>univs_utils.h</code></td>
<td>The files that define database access functions for the CORBA C++ client application.</td>
</tr>
<tr>
<td><code>univs_utils.cpp</code></td>
<td></td>
</tr>
<tr>
<td><code>univs.icf</code></td>
<td>The Implementation Configuration File (ICF) for the Security sample application.</td>
</tr>
<tr>
<td><code>setenvs.sh</code></td>
<td>A UNIX script that sets the environment variables needed to build and run the Security sample application.</td>
</tr>
<tr>
<td><code>setenvs.cmd</code></td>
<td>An MS-DOS command procedure that sets the environment variables needed to build and run the Security sample application.</td>
</tr>
<tr>
<td><code>ubb_s.mk</code></td>
<td>The UBBCONFIG file for the UNIX operating system.</td>
</tr>
<tr>
<td><code>ubb_s.nt</code></td>
<td>The UBBCONFIG file for the Windows NT operating system.</td>
</tr>
<tr>
<td><code>makefiles.mk</code></td>
<td>The makefile for the Security sample application on the UNIX operating system.</td>
</tr>
<tr>
<td><code>makefiles.nt</code></td>
<td>The makefile for the Security sample application on the Windows NT operating system.</td>
</tr>
<tr>
<td><code>log.cpp, log.h, log_client.cpp, and log_server.cpp</code></td>
<td>The client and server applications that provide logging and tracing functions for the sample applications. These files are located in the <code>\utils</code> directory.</td>
</tr>
<tr>
<td><code>oradbconn.cpp</code> and <code>oranoconn.cpp</code></td>
<td>The files that provide access to an Oracle SQL database instance. These files are located in the <code>\utils</code> directory.</td>
</tr>
<tr>
<td><code>samplesdb.cpp</code> and <code>samplesdb.h</code></td>
<td>The files that provide print functions for the database exceptions in the sample applications. These files are located in the <code>\utils</code> directory.</td>
</tr>
<tr>
<td><code>unique_id.cpp</code> and <code>unique_id.h</code></td>
<td>C++ Unique ID class routines for the sample applications. These files are located in the <code>\utils</code> directory.</td>
</tr>
</tbody>
</table>
Building and Running the Security Sample Application

### Table 6-1  Files Included in the Security Sample Application

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>samplesdbsql.h and</td>
<td>C++ class methods that implement access to the SQL database. These files are located in the <code>\utils</code> directory.</td>
</tr>
<tr>
<td>samplesdbsql.pc</td>
<td></td>
</tr>
<tr>
<td>university.sql</td>
<td>The SQL for the University database. This file is located in the <code>\utils</code> directory.</td>
</tr>
<tr>
<td>Readme.txt</td>
<td>The file that provide the latest information about building and running the Security sample application.</td>
</tr>
</tbody>
</table>

### Step 2: Verify the settings of the environment variables.

Before building and running the Security sample application, you need to ensure that certain environment variables are set on your system. In most cases, these environment variables are set as part of the installation procedure. However, you need to check the environment variables to ensure they reflect correct information.

Table 6-5 lists the environment variables required to run the Security sample application.

### Table 6-2  Required Environment Variables for the Security Sample Application

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPDIR</td>
<td>The directory path where you copied the sample application files. For example:</td>
</tr>
<tr>
<td></td>
<td>Windows NT</td>
</tr>
<tr>
<td></td>
<td>APPDIR=c:\work\securityapp</td>
</tr>
<tr>
<td></td>
<td>UNIX</td>
</tr>
<tr>
<td></td>
<td>APPDIR=/usr/work/securityapp</td>
</tr>
</tbody>
</table>
To verify that the information for the environment variables defined during installation is correct, perform the following steps:

**Windows NT**

1. From the Start menu, select Settings.
2. From the Settings menu, select the Control Panel.
   The Control Panel appears.
3. Click the System icon.
   The System Properties window appears.
4. Click the Environment tab.
   The Environment page appears.

**UNIX**

1. From the Start menu, select Settings.
2. From the Settings menu, select the Control Panel.
   The Control Panel appears.
3. Click the System icon.
   The System Properties window appears.
4. Click the Environment tab.
   The Environment page appears.
5. Check the settings of the environment variables.

UNIX

ksh prompt>printenv TUXDIR
ksh prompt>printenv JAVA_HOME

To change the settings, perform the following steps:

Windows NT

1. On the Environment page in the System Properties window, click the environment variable you want to change or enter the name of the environment variable in the Variable field.
2. Enter the correct information for the environment variable in the Value field.
3. Click OK to save the changes.

UNIX

ksh prompt>export TUXDIR=directorypath
ksh prompt>export JAVA_HOME=directorypath

Step 3: Change the Protection on the Files for the Security Sample Application.

During the installation of the WLE software, the sample application files are marked read-only. Before you can edit the files or build the files in the Security sample application, you need to change the protection of the files you copied into your work directory, as follows:

Windows NT

prompt>attrib -r drive:\workdirectory\*.*

UNIX

prompt>chmod u+rw /workdirectory/\*./*
Step 4: Set the Environment Variables

Use the following command to set the environment variables used to build the client and server applications in the Security sample application:

**Windows NT**

```bash
prompt>setenvs
```

**UNIX**

```bash
prompt>/bin/ksh
prompt>./setenvs.sh
```

Step 5: Initialize the Database

Use the following command to initialize the database used with the Security sample application:

**Windows NT**

```bash
prompt>nmake -f makefiles.nt initdb
```

**UNIX**

```bash
prompt>make -f makefiles.mk initdb
```

Step 6: Load the UBBCONFIG File

Use the following command to load the UBBCONFIG file:

**Windows NT**

```bash
prompt>tmloadcf -y ubb_s.nt
```

**UNIX**

```bash
prompt>tmloadcf -y ubb_s.mk
```
Building and Running the Security Sample Application

The build process for the \texttt{UBBCONFIG} file prompts you for an application password. This password will be used to log on to the client applications. Enter the password and press Enter. You are then prompted to verify the password by entering it again.

**Step 7: Compile the Security Sample Application**

The directory for the Security sample application contains a \texttt{makefile} that builds the client and server sample applications in the Security sample application.

Use the following command to build the C++ client and server applications in the Security sample application:

\textbf{Windows NT}

\begin{verbatim}
prompt>nmake -f makefiles.nt
\end{verbatim}

\textbf{UNIX}

\begin{verbatim}
prompt>make -f makefiles.mk
\end{verbatim}

To build the Java client application in the Security sample application:

\textbf{Windows NT}

\begin{verbatim}
prompt>nmake -f makefiles.nt javaclient
\end{verbatim}

\textbf{UNIX}

\begin{verbatim}
prompt>make -f makefiles.mk javaclient
\end{verbatim}

**Step 8: Start the server application**

Start the system processes and the server application in the Security sample application by entering the following command:

\begin{verbatim}
prompt>tmboot -y
\end{verbatim}

Before using another sample application, enter the following command to stop the system processes and the server application in the Security sample application.

\begin{verbatim}
prompt>tmshutdown
\end{verbatim}
Step 8: Start the C++ client application

Start the C++ client application in the Security sample application by performing the following steps:

1. At the MS-DOS prompt, enter the following command:
   
   ```
   prompt>univs_client
   ```

2. At the Enter student id: prompt, enter any number between 100001 and 100010.

3. Press Enter.

4. At the Enter domain password: prompt, enter the password you defined when you loaded the UBBCONFIG file.

5. Press Enter.

Step 9: Start the Java client application.

To run the Java client application in the Security sample application, perform the following steps:

1. Modify the following lines in the UnivSApplet.html file:
   
   ```
   code="UnivSApplet.class"
   codebase=.
   ```

   to read as follows:

   ```
   code="UnivSApplet"
   archive="UnivSApplet.jar,m3envobj.jar"
   ```

2. Copy the modified UnivSApplet.html file to the source directory for the Web server (the directory varies by Web server product).

3. Create a UnivSApplet.jar file, as follows:

   a. Create a tmp directory under the directory where you built the sample application and copy the UniversityS subdirectory and the class files it contains into the tmp directory.
Copy the class files in the Security sample application directory that were generated by the makefile into the tmp directory, set the directory (cd) to the tmp directory, and issue one of the following commands to create a jar file that contains all the Security sample application classes:

```
jar -cf ..\UnivSApplet.jar *.* (Microsoft Windows NT systems)
jar -cf ../UnivSApplet.jar * (UNIX systems)
```

4. Copy the UnivSApplet.jar file you just created to the source directory for the Web server (the directory name varies by Web server product).

5. Copy the m3envobj.jar file from the appropriate subdirectory (%TUXDIR%\udataobj\java Microsoft Windows NT systems or ${TUXDIR}/udataobj/java UNIX systems) to the Web server source directory.

6. Make sure the Security server application is running, start up your Web browser, and point it to the node where the Web server is running.

**Note:** On Microsoft Windows NT systems, the node name needs to be in all uppercase characters. For example, if the node is specified as SERVER in the UBBCONFIG file and in the UnivSApplet.html file, set your browser to http://SERVER/UnivSApplet.html.

1. Modify the following lines in the UnivSApplet.html file:
   ```
   code="UnivSApplet.class"
   codebase=.
   ```
   to read as follows:
   ```
   code="UnivSApplet"
   archive="UnivSApplet.jar,m3envobj.jar"
   ```

2. Copy the modified UnivSApplet.html file to the source directory for the Web server (the directory varies by Web server product).

3. After executing the makefile to build the Security sample application, create a UnivSApplet.jar file, as follows:
   a. Create a tmp directory under the directory where you built the sample application and copy the UniversityS subdirectory and the class files it contains into the tmp directory.

   Copy the class files in the Security sample application directory that were generated by the makefile into the tmp directory, set the directory (cd) to
Building and Running the CORBA Sample Applications

the tmp directory, and issue one of the following commands to create a jar file that contains all the Production sample application classes:

```
jar -cf ..\UnivSApplet.jar *.* (Microsoft Windows NT systems)
jar -cf ../UnivSApplet.jar * (UNIX systems)
```

4. Copy the UnivSApplet.jar file you just created to the source directory for the Web server (the directory name varies by Web server product).

5. Copy the m3envobj.jar file from the appropriate subdirectory (%TUXDIR%\udataobj\java Microsoft Windows NT systems or ${TUXDIR}/udataobj/java UNIX systems) to the Web server source directory.

6. Make sure the Security server application is running, start up your Web browser, and point it to the node where the Web server is running.

**Note:** On Microsoft Windows NT systems, the node name needs to be in all uppercase characters. For example, if the node is specified as SERVER in the UBBCONFIG file and in the UnivSApplet.html file, set your browser to http://SERVER/UnivPApplet.html.

A logon window appears.

7. Enter a number between 100001 and 100010 in the student ID field.

8. Enter the password you defined when you loaded the UBBCONFIG file in the Domain Password field.

9. Click the Logon button.

10. Enter a search string to find a course.
Building and Running the Secure Simpapp Sample Application

The Secure Simpapp sample application demonstrates using the SSL protocol and certificate-based authentication to protect communications between client applications and the WLE domain. There are C++ and Java versions of the Secure Simpapp sample application.

Perform the following steps to build and run the Secure Simpapp sample application:

1. Copy the files for the Secure Simpapp sample application into a work directory.
2. Change the protection attribute on the files for the Secure Simpapp sample application.
3. Verify the environment variables.
4. Execute the `runme` command.

Before you can use the Secure Simpapp sample application, obtain a certificate and private key (`IIOPListener.pem`) for the IIOP Listener/Handler from the certificate authority in your enterprise and load the certificate in a Lightweight Directory Access Protocol (LDAP)-enabled directory service. The `runme` command prompts you for the pass phrase for the private key for the IIOP Listener/Handler.

Step 1: Copy the Files for the Secure Simpapp Sample Application into a Work Directory

You need to copy the files for the Secure Simpapp sample application into a work directory on your local machine. The following sections detail the directory location and sources files for the C++ and Java versions of the Secure Simpapp sample application.
C++ Version of the Secure Simpapp Sample Application

The files for the C++ version of the Secure Simpapp sample application are located in the following directories:

**Windows NT**

drive:\WLEdir\samples\corba\simpappSSL

**UNIX**

/usr/local/WLEdir/samples/corba/simpappSSL

You will use the files listed in Table 6-3 to build and run the C++ version of the Secure Simpapp sample application.

Table 6-3  Files Included in the C++ Version of the Secure Simpapp Sample Application

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple.idl</td>
<td>The OMG IDL code that declares the Simple and SimpleFactory interfaces. This file is copied from the WLE simpappSSL_java directory by the runme command file.</td>
</tr>
<tr>
<td>Simples.cpp</td>
<td>The C++ source code that overrides the default Server::initialize and Server::release methods.</td>
</tr>
<tr>
<td>Simplec.cpp</td>
<td>The C++ source code for the client application in the Secure Simpapp sample application.</td>
</tr>
<tr>
<td>Simple_i.cpp</td>
<td>The C++ source code that implements the Simple and SimpleFactory methods.</td>
</tr>
<tr>
<td>Simple_i.h</td>
<td>The C++ header file that defines the implementation of the Simple and SimpleFactory methods.</td>
</tr>
<tr>
<td>Readme.html</td>
<td>This file provides the latest information about building and running the C++ version of the Secure Simpapp sample application.</td>
</tr>
<tr>
<td>runme.cmd</td>
<td>The Windows NT batch file that builds and runs the C++ version of the Secure Simpapp sample application.</td>
</tr>
</tbody>
</table>
Java Version of the Secure Simpapp Sample Application

The files for the Java version of the Secure Simpapp sample application are located in the following directories:

Windows NT

`drive:\WLEdir\samples\corba\simpappSSL_java`

UNIX

`/usr/local/WLdir/samples/corba/simpappSSL_java`

You will use the files listed in Table 6-4 to build and run the Java Secure Simpapp sample application.
### Table 6-4  Files Included in the Java Version of the Secure Simpapp Sample Application

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple.idl</td>
<td>The OMG IDL code that declares the Simple and SimpleFactory interfaces. This file is copied from the WLE simpappSSL java directory by the runme command file.</td>
</tr>
<tr>
<td>ServerImpl.java</td>
<td>The Java source code that overrides the Server.initialize and Server.release methods.</td>
</tr>
<tr>
<td>SimpleClient.java</td>
<td>The Java source code for the client application in the Secure Simpapp sample application.</td>
</tr>
<tr>
<td>SimpleFactoryImpl.java</td>
<td>The Java source code that implements the SimpleFactory methods.</td>
</tr>
<tr>
<td>SimpleImpl.java</td>
<td>The Java source code that implements the Simple methods.</td>
</tr>
<tr>
<td>Simple.xml</td>
<td>The Server Description File used to associate activation and transaction policy values with CORBA interfaces. For the Java version of the Secure Simpapp sample application, the Simple and SimpleFactory interfaces have an activation policy of method and a transaction policy of optional.</td>
</tr>
<tr>
<td>Readme.html</td>
<td>The file that provides the latest information about building and running the Java version of the Secure Simpapp sample application.</td>
</tr>
<tr>
<td>runme.cmd</td>
<td>The Windows NT batch file that builds and runs the Java version of the Secure Simpapp sample application.</td>
</tr>
<tr>
<td>runme.ksh</td>
<td>The UNIX Korn shell script that builds and executes the Java version of the Secure Simpapp sample application.</td>
</tr>
</tbody>
</table>
Step 2: Change the protection attribute on the files for the Secure Simpapp sample application.

During the installation of the WLE software, the sample application files are marked read-only. Before you can edit or build the files in the Secure Simpapp sample application, you need to change the protection attribute of the files you copied into your work directory, as follows:

**Windows NT**

`prompt>attrib -r drive:\workdirectory\*. *

**UNIX**

`prompt>/bin/ksh

`ksh prompt>chmod u+w /workdirectory//*. *
On the UNIX operating system platform, you also need to change the permission of runme.ksh to give execute permission to the file, as follows:

```
sh prompt>chmod +x runme.ksh
```

### Step 3: Verify the settings of the environment variables.

Before building and running the Secure Simpapp sample application, you need to ensure that certain environment variables are set on your system. In most cases, these environment variables are set as part of the installation procedure. However, you need to check the environment variables to ensure they reflect correct information.

Table 6-5 lists the environment variables required to run the Secure Simpapp sample application.

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPDIR</td>
<td>The directory path where you copied the sample application files. For example:</td>
</tr>
<tr>
<td></td>
<td><strong>Windows NT</strong></td>
</tr>
<tr>
<td></td>
<td>APPDIR=c:\work\simpappSSL</td>
</tr>
<tr>
<td></td>
<td><strong>UNIX</strong></td>
</tr>
<tr>
<td></td>
<td>APPDIR=/usr/work/simpappSSL</td>
</tr>
<tr>
<td>TUXCONFIG</td>
<td>The directory path and name of the configuration file. For example:</td>
</tr>
<tr>
<td></td>
<td><strong>Windows NT</strong></td>
</tr>
<tr>
<td></td>
<td>TUXCONFIG=c:\work\simpappSSL\tuxconfig</td>
</tr>
<tr>
<td></td>
<td><strong>UNIX</strong></td>
</tr>
<tr>
<td></td>
<td>TUXCONFIG=/usr/work/simpappSSL/tuxconfig</td>
</tr>
</tbody>
</table>
To verify that the information for the environment variables defined during installation is correct, perform the following steps:

### Table 6-5  Required Environment Variables for the Secure Simpapp Sample Application

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| JDKDIR               | The directory path where you installed the JDK software. For example:  
  **Windows NT**  
  JDKDIR=c:\jdk1.2.2  
  **UNIX**  
  JDKDIR=/usr/local/jdk1.2.1  
  You need to specify this parameter only if you plan to use the Java version of the Secure Simpapp sample application. |
| TOBJADDR             | The host name and port number of the IIOP Listener/Handler. The port number must be defined as a port for SSL communications. For example:  
  **Windows NT**  
  TOBJADDR=trixie::1111  
  **UNIX**  
  TOBJADDR=trixie::1111 |
| JAVA_HOME            | The directory path where you installed the JDK software. For example:  
  **Windows NT**  
  JAVA_HOME=c:\JDK1.2  
  **UNIX**  
  JAVA_HOME=/usr/local/JDK1.2  
  You need to define this environment variable only when you use the Java version of the Secure Simpapp sample application. |
| RESULTSDIR or JRESULTSDIR | A subdirectory of APPDIR where files that are created as a result of executing the runme command are stored. For example:  
  **Windows NT**  
  RESULTSDIR=c:\workdirectory\  
  **UNIX**  
  RESULTSDIR=/usr/local/workdirectory/  
  When using the Java version of the Secure Simpapp sample application, specify the JRESULTSDIR environment variable. |
Windows NT
1. From the Start menu, select Settings.
2. From the Settings menu, select the Control Panel.
   The Control Panel appears.
3. Click the System icon.
   The System Properties window appears.
4. Click the Environment tab.
   The Environment page appears.
5. Check the settings of the environment variables.

UNIX
ksh prompt>printenv TUXDIR
ksh prompt>printenv JAVA_HOME
To change the settings, perform the following steps:

Windows NT
1. On the Environment page in the System Properties window, click the environment variable you want to change or enter the name of the environment variable in the Variable field.
2. Enter the correct information for the environment variable in the Value field.
3. Click OK to save the changes.

UNIX
ksh prompt>export TUXDIR=directorypath
ksh prompt>export JAVA_HOME=directorypath

Step 4: Execute the runme command.

The runme command automates the following steps:
1. Setting the system environment variables
2. Loading the UBBCONFIG file

3. Compiling the code for the client application

4. Compiling the code for the server application

5. Starting the server application using the tmboot command

6. Starting the client application

7. Stopping the server application using the tmshutdown command

**Note:** You can also run the Secure Simpapp sample application manually. The steps for manually running the Secure Simpapp sample application are described in the Readme.html file.

To build and run the Secure Simpapp sample application, enter the `runme` command, as follows:

**Windows NT**

prompt>cd workdirectory
prompt>runme

**UNIX**

ksh prompt>cd workdirectory
ksh prompt>./runme.ksh

The Secure Simpapp sample application runs and prints the following messages:

Testing simpapp
  cleaned up
  prepared
  built
  loaded ubb
  booted
  ran
  shutdown
  saved results
  PASSED

During execution of the runme command, you are prompted for a password. Enter the pass phrase of the private key of the IIOP Listener/Handler.

Table 6-6 lists the C++ files in the work directory generated by the `runme` command.
Table 6-6 C++ Files Generated by the *runme* Command

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple_c.cpp</td>
<td>Generated by the idl command, this file contains the client stubs for the SimpleFactory and Simple interfaces.</td>
</tr>
<tr>
<td>Simple_c.h</td>
<td>Generated by the idl command, this file contains the client definitions of the SimpleFactory and Simple interfaces.</td>
</tr>
<tr>
<td>Simple_s.cpp</td>
<td>Generated by the idl command, this file contains the server skeletons for the SimpleFactory and Simple interfaces.</td>
</tr>
<tr>
<td>Simple_s.h</td>
<td>Generated by the idl command, this file contains the server definition for the SimpleFactory and Simple interfaces.</td>
</tr>
<tr>
<td>.adm/.keybd</td>
<td>A file that contains the security encryption key database. The subdirectory is created by the tmloadcf command in the runme command.</td>
</tr>
<tr>
<td>results</td>
<td>A directory generated by the runme command.</td>
</tr>
</tbody>
</table>

Table 6-7 lists the Java files in the work directory generated by the *runme* command.

Table 6-7 Java Files Generated by the *runme* Command

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpleFactory.java</td>
<td>Generated by the m3idltojava command for the SimpleFactory interface. The SimpleFactory interface contains the Java version of the OMG IDL interface. It extends org.omg.CORBA.Object.</td>
</tr>
<tr>
<td>SimpleFactoryHolder.java</td>
<td>Generated by the m3idltojava command for the SimpleFactory interface. This class holds a public instance member of type SimpleFactory. The class provides operations for out and inout arguments that are included in CORBA, but that do not map exactly to Java.</td>
</tr>
</tbody>
</table>
Table 6-7  Java Files Generated by the runme Command

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpleFactoryHelper.java</td>
<td>Generated by the m3idltojava command for the SimpleFactory interface. This class provides auxiliary functionality, notably the narrow method.</td>
</tr>
<tr>
<td>_SimpleFactoryStub.java</td>
<td>Generated by the m3idltojava command for the SimpleFactory interface. This class is the client stub that implements the SimpleFactory.java interface.</td>
</tr>
<tr>
<td>_SimpleFactoryImplBase.java</td>
<td>Generated by the m3idltojava command for the SimpleFactory interface. This abstract class is the server skeleton. It implements the SimpleFactory.java interface. The user-written server class SimpleFactoryImpl extends _SimpleFactoryImplBase.</td>
</tr>
<tr>
<td>Simple.java</td>
<td>Generated by the m3idltojava command for the Simple interface. The Simple interface contains the Java version of the OMG IDL interface. It extends org.omg.CORBA.Object.</td>
</tr>
<tr>
<td>SimpleHolder.java</td>
<td>Generated by the m3idltojava command for the Simple interface. This class holds a public instance member of type Simple. The class provides operations for out and inout arguments that CORBA has but that do not match exactly to Java.</td>
</tr>
<tr>
<td>SimpleHelper.java</td>
<td>Generated by the m3idltojava command for the Simple interface. This class provides auxiliary functionality, notably the narrow method.</td>
</tr>
<tr>
<td>_SimpleStub.java</td>
<td>Generated by the m3idltojava command for the Simple interface. This class is the client stub that implements the Simple.java interface.</td>
</tr>
</tbody>
</table>
Table 6-7  Java Files Generated by the runme Command

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_SimpleImplBase.java</td>
<td>Generated by the m3idltojava command for the Simple interface. This abstract class is the server skeleton. It implements the Simple.java interface. The user-written server class SimpleImpl extends _SimpleImplBase.</td>
</tr>
<tr>
<td>Simple.ser</td>
<td>The Server Descriptor File generated by the buildjobjserver command in the runme command.</td>
</tr>
<tr>
<td>Simple.jar</td>
<td>The server Java Archive file generated by the buildjavaserver command in the runme command.</td>
</tr>
<tr>
<td>.adm/.keybd</td>
<td>A file that contains the security encryption key database. The subdirectory is created by the tmloadcf command in the runme command.</td>
</tr>
<tr>
<td>results</td>
<td>A directory generated by the runme command.</td>
</tr>
</tbody>
</table>

Table 6-8 lists files in the RESULTS or JRESULTS directory generated by the runme command.

Table 6-8  Files in the results Directory Generated by the runme Command

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>Contains the input that the runme command provides to the Java client application.</td>
</tr>
<tr>
<td>output</td>
<td>Contains the output produced when the runme command executes the Java client application.</td>
</tr>
<tr>
<td>expected_output</td>
<td>Contains the output that is expected when the Java client application is executed by the runme command. The data in the output file is compared to the data in the expected_output file to determine whether or not the test passed or failed.</td>
</tr>
</tbody>
</table>
Table 6-8 Files in the results Directory Generated by the runme Command

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log</td>
<td>Contains the output generated by the runme command. If the runme command fails, check this file for errors.</td>
</tr>
<tr>
<td>setenv.cmd</td>
<td>Contains the commands to set the environment variables needed to build and run the Java Secure Simpapp sample application on the Windows NT operating system platform.</td>
</tr>
<tr>
<td>setenv.ksh</td>
<td>Contains the commands to set the environment variables needed to build and run the Java Secure Simpapp sample application on the UNIX operating system platform.</td>
</tr>
<tr>
<td>stderr</td>
<td>Generated by the tmboot command, which is executed by the runme command. If the -noredirect JavaServer option is specified in the UBBCONFIG file, the System.err.println method sends the output to the stderr file instead of to the ULOG file.</td>
</tr>
<tr>
<td>stdout</td>
<td>Generated by the tmboot command, which is executed by the runme command. If the -noredirect JavaServer option is specified in the UBBCONFIG file, the System.out.println method sends the output to the stdout file instead of to the ULOG file.</td>
</tr>
<tr>
<td>tmsysevt.dat</td>
<td>Contains filtering and notification rules used by the TMSYSEVT (system event reporting) process. This file is generated by the tmboot command in the runme command.</td>
</tr>
<tr>
<td>tuxconfig</td>
<td>A binary version of the UBBCONFIG file.</td>
</tr>
<tr>
<td>ubb</td>
<td>The UBBCONFIG file for the Java Secure Simpapp sample application.</td>
</tr>
<tr>
<td>ULOG.&lt;date&gt;</td>
<td>A log file that contains messages generated by the tmboot command.</td>
</tr>
</tbody>
</table>
6 Using the Secure Simpapp Sample Application

Run the server application in the Secure Simpapp sample application, as follows:

**Windows NT**

```
prompt>tmboot
```

**UNIX**

```
ksh prompt>tmboot
```

Run the client application in the Secure Simpapp sample application, as follows:

**Windows NT**

```
prompt>java -classpath .;%TUXDIR%\udataobj\java\jdk\m3envobj.jar
-DTOBJADDR=%TOBJADDR% SimpleClient
String?
Hello World
HELLO WORLD
hello world
```

**UNIX**

```
ksh prompt>java -classpath .:$TUXDIR/udataobj/java/jdk\n/m3envobj.jar -DTOBJADDR=$TOBJADDR SimpleClient
String?
Hello World
HELLO WORLD
hello world
```

**Note:** The Secure Simpapp sample client application uses the client-only JAR file `m3envobj.jar`. However, you can also use the `m3.jar` file to run the client application.

Before using another sample application, enter the following commands to stop the Secure Simpapp sample application and to remove unnecessary files from the work directory:

**Windows NT**

```
prompt>tmshutdown -y
```

```
prompt>nmake -f makefile.nt clean
```

**UNIX**
Building and Running the Secure Simpapp Sample Application

ksh prompt>tmshutdown -y
ksh prompt>make -f makefile.mk clean
CHAPTER

7 Writing a WLE Enterprise JavaBean that Implements Security

This topic includes the following sections:

- Before You Begin
- How Authentication Works with WLE EJBs
- Development Steps
  - Step 1: Define security roles for the methods of the WLE EJB.
  - Step 2: Specify security roles in the Deployment Descriptor of the EJB.
  - Step 3: Define the JNDI environment properties.
  - Step 4: Establish the InitialContext.
  - Step 5: Use Home to get a WLE EJB.
  - Step 6: Use the getCallerPrincipal Method to authenticate a WLE EJB.
Before You Begin

This document describes the BEA implementation of the Security feature. The information in this document supplements the Sun Microsystems, Inc. evolving Enterprise JavaBeans 1.1 Specification (Public Release, October 18, 1999).

**Note:** Before proceeding with the remainder of this topic, you should be familiar with the entire content of Sun’s specification, particularly Chapter 15, “Security Management.”

This topic describes only the integrating security into WLE EJBs. For a complete description of developing an EJB using the WLE product, see *Getting Started* in the WebLogic Enterprise online documentation.

**Note:** An EJB in the WLE domain that issues a callback to a remote J2EE client application cannot propagate the security context of that client application in the callback.

How Authentication Works with WLE EJBs

From the perspective of an EJB container, EJBs are nontrusted entities that require authentication. The WLE product uses a JNDI implementation that runs within the EJB container’s trusted environment. Using the `WLEInitialContextFactory` JNDI factory with security environment properties establishes the security context for the WLE client application. The WLE client application authenticates itself with the WLE domain when establishing the JNDI Initial context.

Development Steps

Table 7-1 lists the development steps required to implement security in a WLE EJB.
Step 1: Define security roles for the methods of the WLE EJB.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Define security roles for the methods of the WLE EJB.</td>
</tr>
<tr>
<td>2</td>
<td>Specify security roles in the Deployment Descriptor of the EJB.</td>
</tr>
<tr>
<td>3</td>
<td>Define the JNDI environment properties.</td>
</tr>
<tr>
<td>4</td>
<td>Establish the InitialContext.</td>
</tr>
<tr>
<td>5</td>
<td>Use Home to get the WLE EJB.</td>
</tr>
<tr>
<td>6</td>
<td>Use the getCallerPrincipal method to authenticate the WLE EJB.</td>
</tr>
</tbody>
</table>

Step 1: Define security roles for the methods of the WLE EJB.

During the assembly and deployment of an EJB package, you define security roles and associate roles with methods in the deployment descriptor. Security roles are mapped to groups of users in the WLE security environment. You can use any of the techniques described in the Security Management chapter of the Enterprise JavaBeans 1.1 Specification to define security roles for the methods of a WLE EJB.

It is possible that two methods with the same name or name/signature appear in both the bean’s home and remote interfaces. To handle this case, the optional `<method-intf>` element may further restrict the selection to either Home or Remote interface methods.

In a mandatory access control environment, any method invocation not specifically authorized is denied. Sometimes a method does not have a defined `<method-permission>` element. If the `SECURITY` parameter in the `RESOURCES` section of the `UBBCONFIG` file is set to `MANDATORY_ACL` access on a method without an associated `<method-permission>` element, access is denied. This is the
recommended setting for production environments. For all other settings of the SECURITY parameter, access to a method without an associated <method-permission> element is allowed.

There may be methods that should be available to everyone, even in a mandatory access control environment. The WLE system defines a special role name * which means everyone has access to the method.

### Step 2: Specify security roles in the Deployment Descriptor of the EJB.

You specify security roles for the methods of an EJB in the deployment descriptor of the bean. In the WLE product, there is a one-to-one association between the security roles defined in the deployment descriptor of the EJB and the groups defined with the tpgrpadd commands. Role names may be referenced in deployment descriptors before the corresponding group exists. A run time, if a bean’s deployment descriptor references a role that does not have a corresponding group, the role is ignored.

Role names are restricted to any alphanumeric characters, a dash (-), an underscore(_), the at-sign(@), and a period(.). The maximum length of a role name is 30 characters. If the name of a security role does not conform to these limitations, it will not be possible for users to have the defined security role.

Listing 7-1 includes code that defines a security role.

**Listing 7-1  Defining a Security Role for a Method in an EJB**

```xml
...<assembly-descriptor>
  <security-role>
    <description>
      "teller" is a role name
    </description>
  </security-role>
  <method-permission>
    <role-name>teller</role-name>
  </method-permission>
</assembly-descriptor>
```
Step 3: Define the JNDI environment properties.

The following sections describe the JNDI environment properties that must be set to enable either Username/Password or certificate-based authentication.

WLEContext.INITIAL_CONTEXT_FACTORY Property

The class com.beasys.jndi.WLEInitialContextFactory is the JNDI Service Provider Interface (SPI). This initial context provides an entry point into the WLE domain. Set WLEContext.INITIAL_CONTEXT_FACTORY to com.beasys.jndi.WLEInitialContextFactory to access the WLE domain.

Listing 7-2 includes code that defines the WLEContext.INITIAL_CONTEXT_FACTORY property for the WLE environment.

Listing 7-2    WLEContext.INITIAL_CONTEXT_FACTORY Property

```java
Hashtable env = new Hashtable();
/*
 *Specify the initial context implementation to use.
 *The service provider supplies the factory class.
 */
env.put(WLEContext.INITIAL_CONTEXT_FACTORY,

Using Security    7-5
7 Writing a WLE Enterprise JavaBean that Implements Security

“com.beasys.jndi.WLEInitialContextFactory”);
...

WLEContext.PROVIDER_URL Property

Specifies the entry point into the WLE domain. The value should reflect the host and port of the IIOP Listener/Handler of the target WLE domain. Use one of the following URL address formats when specifying the location of the IIOP Listener/Handler:

- corbaloc://hostname:portnumber

  Indicates that the IIOP/RMI protocol is to be used to communicate with the WLE domain. This URL address format only supports Username/Password authentication.

- corbalocs://hostname:portnumber

  Indicates that the SSL protocol is to be used to communicate with the WLE domain. This URL address format supports both Username/password and certificate-based authentication.

The host and port combination in the URL must match the ISL parameter in the WLE application’s UBBCONFIG file. The format of the host and port combination as well as the capitalization must match. If the addresses do not match, communication with the WLE domain fails.

Listing 7-3 includes code that defines the WLEContext.PROVIDER_URL property for the WLE environment.

Listing 7-3  WLEContext.PROVIDER_URL Property

...  
env.put(WLEContext.PROVIDER_URL,
   "corbaloc://"myhost:1000");
...

A WLE server application that acts as a client application (referred to as a joint client/server application) must set the WLEContext.PROPERTY_URL as an empty or null string. The joint client/server application connects to the current application in which it was booted.
Step 3: Define the JNDI environment properties.

**WLEContext.SECURITY_AUTHENTICATION Property**

Set this property to indicate the type of authentication to be used. The valid values for this property are as follows:

- **None**—Indicates that no authentication is performed
- **Simple**—Indicates that Username/Password authentication is performed
- **Strong**—Indicates that certificate-based authentication is performed

See Table 7-2 for additional keys that need to be specified to use Username/Password or certificate-based authentication.

Listing 7-4 includes code that defines the `WLEContext.SECURITY_AUTHENTICATION` property for the WLE environment.

```
Listing 7-4  WLEContext.SECURITY_AUTHENTICATION Property
...
env.put(WLEContext.SECURITY_AUTHENTICATION, "strong");
...
```

<table>
<thead>
<tr>
<th>Property Key</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLEContext.SECURITY_PRINCIPAL</td>
<td>Specifies the identity of the principal used when authenticating the caller to the WLE domain.</td>
</tr>
</tbody>
</table>
Writing a WLE Enterprise JavaBean that Implements Security

Listing 7-5 includes the WLE keys used to define Username/Password authentication.

**Listing 7-5  WLE Keys for Username/Password Authentication**

```java
... 
Hashtable env = new Hashtable();
env.put(Context.PROVIDER_URL, “corbalocs://myhost:1000”)
env.put(Context.INITIAL_CONTEXT_FACTORY,
        “com.beasys.jndi.WLEInitialContextFactory”);

//Password-Based Authentication
env.put(WLEContext.SECURITY_PRINCIPAL, “milozzi”);
env.put(WLEContext.SYSTEM_CREDENTIALS, “mypassword”);
env.put(WLEContext.CLIENT_NAME, “writers”);
env.put(WLEContext.SECURITY_AUTHENTICATION, “simple”);
env.put(WLEContext.SYSTEM_PASSWORD, “password”);
... 
```

Listing 7-6 includes the WLE keys used to define certificate-based authentication.

Table 7-2  WLE Property Keys for Security

<table>
<thead>
<tr>
<th>Property Key</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLEContext.SECURITY_CREDENTIALS</td>
<td>Specifies the credentials of the principal when authenticating the caller to the WLE domain.</td>
</tr>
<tr>
<td></td>
<td>- For certificate-based authentication enabled via SECURITY_AUTHENTICATION=”strong”, it specifies the pass phrase used to access the private key and certificate for the EJB.</td>
</tr>
<tr>
<td></td>
<td>- For password-based authentication enabled via SECURITY_AUTHENTICATION=”simple”, it specifies a string that is the user’s password or an arbitrary object user_data used by the authentication server (AUTHSVR) to verify the credentials of the EJB.</td>
</tr>
<tr>
<td>WLEContext.CLIENT_NAME</td>
<td>Specifies the name of the EJB defined by the -c option of the tpusradd command. For more information, see “Defining Authorized Users” on page 4-8</td>
</tr>
<tr>
<td>WLEContext.SYSTEM_PASSWORD</td>
<td>The system password. Required only when using Username/Password authentication.</td>
</tr>
</tbody>
</table>
Step 4: Establish the InitialContext.

Listing 7-6  WLE Keys for Certificate-Based Authentication

```java
//Certificate-Based Authentication
env.put(WLEContext.SECURITY_AUTHENTICATION, "strong");
env.put(WLEContext.SYSTEM_PASSWORD, "SSL");
env.put(WLEContext.SECURITY_PRINCIPAL, "milozzi");
env.put(WLEContext.SECURITY_CREDENTIALS, "credentials");
```

Step 4: Establish the InitialContext.

To access a WLE EJB using JNDI, you establish an InitialContext using the following code:

```java
Context ctx = new InitialContext(env);
```

Specifying `env` as `com.beasys.com.jndi.WLEInitialContextFactory`. After the context is created, the client application has access to bean homes in the WLE domain using WLE as the name service provider.

A WLE EJB is implicitly associated with the security context specified when the `WLEContext` object is created. To specify a new security context, the EJB needs to close the current security context and establish a new security context with new security attributes. Use the following code to close the current security context:

```java
ctx.close();
```

Step 5: Use Home to get a WLE EJB.

Client applications use the bean’s home interface to create or find beans. The bean’s home is obtained by using the `lookup` method on the InitialContext.
Step 6: Use the getCallerPrincipal Method to authenticate a WLE EJB.

Use the getCallerPrincipal method on the javax.ejb.EJBContext associated with a WLE EJB to authenticate the principal. You can also use the isCallerInRole method to determine the role of the client application invoking methods on the EJB. The default principal is IIOP Client.

Limitations and Restrictions

It is possible to deploy the same EJB more than once with different deployment descriptors that set different access control policies. In this case access control is based on the deployment descriptor from which a particular bean is loaded. Security policies are not considered when the WLE system has a choice of how to route a request to any particular bean or container.

Example of Using Security in a WLE EJB

Listing 7-7 illustrates using Username/Password authentication in a WLE EJB.

Note: The code example in Listing 7-7 uses the corbalocs URL address format so that the SSL protocol is used to protect the integrity of the communications.

Listing 7-7  Username/Password Authentication in a WLE EJB

```java
static public Context getInitialContext() throws Exception {
    Hashtable env = new Hashtable ();
    env.put(WLEContext.INITIAL_CONTEXT_FACTORY,
              "com.beasys.jndi.WLEInitialContextFactory");
```
Example of Using Security in a WLE EJB

```
env.put(WLEContext.PROVIDER_URL, corbalocs://myhost:7002);
return new InitialContext(env);

//Password-Based Authentication
env.put(WLEContext.SECURITY_AUTHENTICATION, "simple");
env.put(WLEContext.SYSTEM_PASSWORD, "RMI");
env.put(WLEContext.SECURITY_PRINCIPAL, "milozzi");
env.put(WLEContext.CLIENT_NAME, "writers");
env.put(WLEContext.SECURITY_CREDENTIALS, "password");
```

Listing 7-8 illustrates using certificate-based authentication in a WLE EJB.

**Listing 7-8  Certificate-based Authentication in a WLE EJB**

```
Listing 7-8  Certificate-based Authentication in a WLE EJB

```
```
Writing a WLE Enterprise JavaBean that Implements Security
8 Troubleshooting

This topic includes the following sections:

- Using ULOGS and ORB Tracing
- CORBA::ORB_init Problems
- Username/Password Authentication Problems
- Certificate-Based Authentication Problems
- Tobj::Bootstrap:: resolve_initial_references Problems
- IIOP Listener/Handler Startup Problems
- Configuration Problems
- Problems with Using Callbacks Objects with the SSL Protocol
- Troubleshooting Tips for Digital Certificates

Note: The problems in this topic pertain to using the SSL protocol and certificate-based authentication with WLE CORBA applications.

Using ULOGS and ORB Tracing

In general, Object Request Brokers (ORBs) write important failures to the ULOG file. When using the CORBA C++ ORB, you can also enable ORB internal tracing which may provide information in addition to the information that appears in the ULOG file.

When looking the ULOG file, note that remote ORB processes by default do not write data to the ULOG file in APPDIR.
On UNIX, the remote ORB writes information to a `ULOG` file in the current directory.

On Windows NT, the remote ORB writes information to a `ULOG` file in the `c:\ulog` directory.

You can set the `ULOGPFX` environment variable to control the location of the `ULOG` file for remote ORBs (for example, you can set the location of the `ULOG` file to `APPDIR` so that all information is put in the same `ULOG` file). Set the `ULOGPFX` environment variable as follows:

**UNIX**

```
setenv ULOGPFX $APPDIR/ULOG
```

**Windows NT**

```
set ULOGPFX=%APPDIR%\ULOG
```

To enable ORB tracing, perform the following steps:

1. Create a file named `trace.dat` in `APPDIR`. The contents of `trace.dat` should have `all=on`.

2. Use the following command to set the `OBB_TRACE_INPUT` environment variable to point to the `trace.dat` file before running the application:

```
set OBB_TRACE_INPUT=%APPDIR%\trace.dat
```

If you want ORB tracing sent to separate files, add the following line to the `trace.dat` file:

```
output=obbtrace%p.log
```

This command sends the trace output to files that are named after each running process. You may want to do this if you are using ORB tracing on UNIX to an NFS mounted drive. In this case, trace performance is slow due to the user log opening, writing, and closing the file for each trace statement.

The CORBA Java ORB logs error messages to the `ULOG` file in all error situations as well as puts minor codes to all system exceptions thrown by the ORB. Therefore, tracing is not necessary.
CORBA: ORB_init Problems

Note: This section applies to the CORBA C++ ORB only.

The ORB_init routine does not perform internal ORB tracing so you will not see any trace output for invalid argument processing. Therefore, you need to double check the arguments that were passed to the ORB_init routine.

If a CORBA::BAD_PARAM exception occurs when executing the ORB_init routine, verify that all required arguments have values. Also, check that arguments which expect a value from a specific set of valid values have the correct value. Note that values for the arguments of the ORB_init routine are case sensitive.

If a CORBA::NO_PERMISSION exception occurs and an SSL argument was specified to the ORB_init routine, make sure the WLE Security Pack is installed. Also, verify that the specified level of encryption does not exceed the encryption level supported by the WLE Security Pack.

If a CORBA::IMP_LIMIT exception occurs when executing the ORB_init routine, verify that the ORBport and ORBSecurePort system properties have the same value.

If a CORBA::Initialize exception occurs when executing the ORB_init routine, verify that the values for OrbId or configset are valid.

Note: The OrbId and configset values apply to the CORBA C++ ORB only.

If Secure Socket Layer (SSL) arguments are passed to the ORB_init routine, the ORB attempts to load and initialize the SSL protocol. If no SSL arguments are passed, the ORB does not attempt to initialize the SSL protocol.

The ORB is not aware of the new URL address formats for the Bootstrap object so if you specify a corbaloc or corbalocs URL address format, the ORB does not try to load the SSL protocol during the ORB_init routine.

If SSL arguments were specified to the ORB_init routine, check the following:

- The specified values for the SSL arguments do not conflict with each other or other ORB arguments.

- Whether or not the ORB is a native process. If the ORB is a native process, SSL arguments are not supported.
Troubleshooting

- That the value specified for the maxCrypto system property is less than the value specified for the minCrypto system property.

- Application controlled SSL configuration parameters that are not correct. The ORB_init routine does not perform digital certificate lookups check so look for missing or corrupted files that would case the dynamic libraries not the loaded. Also, verify the dynamic libraries are loaded. The ORB trace function will provide information about whether or not the dynamic libraries are loaded.

If the problem persists, turn on ORB tracing. ORB tracing will log SSL failures that occur when the liborbssl dynamic library is loaded and initialized.

Username/Password Authentication Problems

If the client application fails when using the corbaloc URL address format with Username/Password authentication, check the following:

- The proper configuration steps were performed. See “Configuring the WLE Environment for the SSL Protocol” and “Defining Security for a WLE CORBA Application” for the list of the required configuration steps.

- An initialization error occurred. Specify a valid SSL system property to the ORB_init routine, an error occurs if:
  - The IIOP Listener/Handler is not available. The ORB trace log will show failed connection attempts.
  - The IIOP Listener/Handler is available but it does not support the SSL protocol. The ULOG file will show that a non-GIOP message was received.
  - The IIOP Listener/Handler was available and configured for the SSL protocol but the SSL connection could not be established. This error can occur when the range of encryption strengths supported by the IIOP Listener/Handler and the range of encryption strengths required by the client application don't match.
Certificate-Based Authentication Problems

If the client application fails when using the corbalocs URL address format with certificate-based authentication, check the following:

- The proper configuration steps were performed. See “Configuring the WLE Environment for the SSL Protocol” and “Defining Security for a WLE CORBA Application” for the list of the required configuration steps.

- Determine whether or not an initialization error occurred.

- Specify a valid SSL system property to the ORB_init routine, an error occurs if:
  - The IIOP Listener/Handler is not available. The ORB trace log will show failed connection attempts.
  - The IIOP Listener/Handler is available but it does not support the SSL protocol. The ULOG file will show that a non-GIOP message was received.
  - The IIOP Listener/Handler was available and configured for the SSL protocol but the SSL connection could not be established. This error can occur when the range of encryption strengths supported by the IIOP Listener/Handler and the range of encryption strengths required by the client application don’t match. The error can also occur when the client application does not trust the certificate chain of the IIOP Listener/Handler or the client application did not receive a certificate from the IIOP Listener/Handler. The error will be written to the ULOG file and the error will also show up in the ORB trace output.

If an error does not occur, the problem is in the authentication process and the ULOG file will contain one of the following error statements indicating the problem:

- Couldn’t connect to an LDAP server
- Couldn’t find a filter that matched the client certificate
- The client certificate was not found in LDAP
- The private key file could not be found
- The passphrase used to open the private key is not correct
The public key from the client certificate did not match the private key

Additional certificate problems can also occur. See “Tobj::Bootstrap::resolve_initial_references Problems” for more information about the types of certificate errors that can occur.

Note: At this point of the initialization process, the failure is not due to a problem in the IIOP Listener/Handler.

**Tobj::Bootstrap::resolve_initial_references Problems**

If a failure occurs when performing a Tobj::Bootstrap::resolve_initial_references with the corbaloc or corbalocs URL address format, a CORBA::InvalidDomain exception is raised. This exception may mask CORBA::NO_PERMISSION or CORBA::COMM_FAILURE exceptions that are raised internally. Look at the ULOG file and turn on ORB tracing to get more details on the error. The following errors may occur:

- If the IIOP Listener/Handler is not available, the ORB trace log will show failed connection attempts.

- If the IIOP Listener/Handler is available but it does not support the SSL protocol, the ULOG file will show that a non-GIOP message was received.

- If the IIOP Listener/Handler is available and configured for the SSL protocol but the SSL connection could not be established. An error can occur if the range of encryption strengths supported by the IIOP Listener/Handler and required by the client application don't match.

- The IIOP Listener/Handler couldn't map a certificate to a Username/Password user name. Verify that the security level for the WLE application is set to USER_AUTH and that Username/Password user name matches the principal name passed into the authenticate call. Also, check that the user name doesn't exceed the 30 character limit.
Additional certificate problems can occur. See “Troubleshooting Tips for Digital Certificates” on page 8-9 for more information about the types of certificate errors that can occur.

**Note:** The Java implementation of the
Tobj_Bootstrap::resolve_initial_references() method does not throw an InvalidDomain exception. When the corbaloc or corbalocs URL address formats are used, the
Tobj_Bootstrap::resolve_initial_references() method internally catches the InvalidDomain exception and throws the exception as a COMM_FAILURE. The method functions this way in order to provide backward compatibility.

## IIOP Listener/Handler Startup Problems

This section describes problems that can occur during the startup of the IIOP Listener/Handler.

If a failure occurs when starting the IIOP Listener/Handler, check the ULOG file for a description of the error. The IIOP Listener/Handler verifies that the values for the SSL arguments specified in the CLOPT parameters are valid. If any of the values are invalid, the appropriate error is recorded in the ULOG file. This check is similar to the argument checking done by the ORB.

The IIOP Listener/Handler will not start its processes unless the `-m` option is specified. The ISH is the process that actually loads and initializes the SSL libraries. If there is a problem loading and initializing the SSL libraries in the ISH process, the error will not be recorded in the ULOG file until the ISH process starts to handle incoming requests from client application.

If you suspect a problem with the startup of the IIOP Listener/Handler processes, check the ULOG file.
Troubleshooting

Configuration Problems

The following are miscellaneous tips to resolve the common configuration problems which may occur when using the WLE Security Pack:

- The ORB -ORBpeerValidate command line option or system property and the -v option of the ISL command do not control the peer validation rules checking. This system property and option only control the checking of the host name specified in the peer certificate against the host name of the machine to which the principal was connected.

- The only way to disable the peer validation rules on an installed kit is to create an empty file for %TUXDIR%\udataobj\security\certs\peer_val.rul. If you are writing a script that builds your WLE application, you can also not register the peer_val.rul file in the script.

- When enabling renegotiation intervals in the IIOP Listener/Handler, check that the option on the ISL command is -R not -r. If you use an -r, the IIOP Listener/Handler will use the SSL protocol but the renegotiation interval will not be used. In addition, the ULOG file will note that an unknown option was specified on the IIOP Listener/Handler.

Another way to determine if the IIOP Listener/Handler is performing renegotiations is to enable ORB tracing on the client side and check whether the cipher suite negotiation callback is being called the configured renegotiation interval. Note that the client application must be sending requests for in order for renegotiations to occur.

- If you have defined the SECURITY parameter in the WLE application’s UBB_CONFIG file to be APP_PW or greater and you have configured the IIOP Listener/Handler to use the SSL protocol but not mutual authentication, you must use Username/Password authentication with the corbalocs URL address format to communicate with the IIOP Listener/Handler. If you try to use certificate-based authentication, the IIOP Listener/Handler will not ask the principal for a certificate when establishing an SSL connection and the IIOP Listener/Handler is not able to map the identity of the principal to a TUXEDO identity.
Problems with Using Callbacks Objects with the SSL Protocol

If you have a joint client/server application and the client portion of the joint client/server application specifies security requirements using either the corbalocs URL address format or by requiring credentials, you must use the -ORBsecurePort system property with the ORB_init routine to specify that a secure port be used.

If you do not specify the -ORBsecurePort system property, the server registration will fail with a CORBA::NO_PERMISSION exception. To verify this is the problem, enable ORB tracing and look for the following trace output:

TCPTransport::Listen: FAILURE: Attempt to listen on clear port while Credentials require SSL be used

If you want to use the SSL protocol with callback objects, the joint client/server application must use the SecurityLevel2::PrincipalAuthenticator::authenticate() method with certificate-based authentication. Otherwise, the joint client/server application does not have a certificate with which to identify itself to the IIOP Listener/Handler which in this case is the initiator of the SSL connection.

Troubleshooting Tips for Digital Certificates

In general, problems with digital certificates occur when:

- One of the digital certificates in the certificate chain of the IIOP Listener/Handler is not from a certificate authority defined in the trust_ca.cer file.

- The name the IIOP Listener/Handler connected to the client application does not match the host name specified in digital certificates of the IIOP Listener/Handler when a host match is performed. The name of the IIOP Listener/Handler is specified in the CommonName attribute of the distinguish name of the IIOP
Troubleshooting

Listener/Handler. The host name and the CommonName attribute must match exactly.

You can verify this error by setting the -ORBpeerValidate system property to none and executing the ORB_init routine again.

- One of the digital certificates in the certificate chain of the IIOP Listener/Handler does not match the specified peer validation rules.

- The digital certificate of the IIOP Listener/Handler is invalid. The digital certificate of the IIOP Listener/Handler becomes invalid when the digital certificate is tampered with, it expires, or the certificate authority that issued the digital certificate expires.

If a digital certificate is rejected for no explainable reason, perform the following steps:

1. Open the digital certificate in a viewer, for example, Windows Explorer.

2. Look at the KeyUsage and BasicConstraints properties of the digital certificate. A small yellow triangle with an exclamation mark indicates the property is critical. Any digital certificate with a property marked critical is rejected by the WLE software.

3. If none of the properties of the digital certificate are critical, check the properties of the next digital certificate in the certificate chain. Perform this step until all the properties of all the digital certificates in the certificate chain have been verified.
CHAPTER

9 WLE Security Service APIs

This topic includes the following sections:

- The WLE Security Model
- Functional Components of the WLE Security Service
- The Principal Authenticator Object
- The Credentials Object
- The SecurityCurrent Object

For the C++, Java, and Automation method descriptions for the WLE Security Service, see the following topics:

- “C++ Security Reference” on page 11-1
- “Java Security Reference” on page 12-1
- “Automation Security Reference” on page 13-1
The WLE Security Model

The security model in the WLE product defines only a framework for security. The WLE product provides the flexibility to support different security mechanisms and policies that can be used to achieve the appropriate level of functionality and assurance for a particular WLE application.

The security model in the WLE product defines:

- Under what conditions client applications may access objects in a WLE domain
- What type of proof material principals are required to authenticate themselves to the WLE domain

The security model in the base WLE product is a combination of the security model defined in the CORBA services Security Service specification and the value-added extensions that provide a focused, simplified form of the security model found in BEA TUXEDO.

The following sections describe the general characteristics of the WLE security model.

Authentication of Principals

Authentication of principals (for example, an individual user, a client application, a server application, a joint client/server application, or an IIOP Listener/Handler) provides security officers with the ability to ensure that only registered principals have access to the objects in the system. An authenticated principal is used as the primary mechanism to control access to objects. The act of authenticating principals allows the security mechanisms to:

- Make principals accountable for their actions
- Control access to protected objects

1. All references to the CORBA services Security Service specification in this document are to the Revision 1.5, December 1998 edition, published by the Object Management Group.
The WLE Security Model

- Identify the originator of a request
- Identify the target of request

Controlling Access to Objects

The WLE security model provides a simple framework through which a security officer can limit access to the WLE domain to authorized users only. Limiting access to objects allows security officers to prohibit access to objects by unauthorized principals. The access control framework consists of two parts:

- The object invocation policy that is enforced automatically on object invocation
- An application access policy that the user-written application can enforce

Administrative Control

The system administrator is responsible for setting security policies for the WLE application. The WLE product provides a set of configuration parameters and utilities. Using the configuration parameters and utilities, a system administrator can configure the WLE application to force the principals to be authenticated to access a system on which WLE software is installed. To enforce the configuration parameters, the system administrator uses the tmloadcf command to update the configuration file for a particular WLE application.

For more information about configuring security for your WLE application, see “Configuring the WLE Environment for the SSL Protocol” on page 3-1 and “Defining Security for a WLE CORBA Application” on page 4-1.
The WLE security model is based on the process of authenticating principals to the WLE domain. The objects of the WLE Security Service are used to authenticate a principal. The principal provides identity and authentication data, such as a password, to the client application. The client application uses the Principal Authenticator object to make the calls necessary to authenticate the principal. The credentials for the authenticated principal are associated with the security system’s implementation of the SecurityCurrent object and are represented by a Credentials object.

Figure 9-1 illustrates the authentication process used in the WLE security model.

Figure 9-1   Authentication Process in the WLE Product

The following sections describe the objects in the WLE security model.
The Principal Authenticator Object

The Principal Authenticator object is used by a principal that requires authentication but has not been authenticated prior to calling the object system. The act of authenticating a principal results in the creation of a Credentials object that is made available as the default credentials for the application.

The Principal Authenticator object is a singleton object; there is only a single instance allowed in a process address space. The Principal Authenticator object is also stateless. A Credentials object is not associated with the Principal Authenticator object that created it.

All Principal Authenticator objects support the SecurityLevel2::PrincipalAuthenticator interface defined in the CORBA Services Security Service specification. This interface contains two methods that are used to accomplish the authentication of the principal. This is because authentication of principals may require more than one step. The authenticate method allows the caller to authenticate, and optionally select, attributes for the principal of this session.

Any invocation that fails because the security infrastructure does not permit the invocation will raise the standard exception CORBA::NO_PERMISSION. A method that fails because the feature requested is not supported by the security infrastructure implementation will raise the CORBA::NO_IMPLEMENT standard exception. Any parameter that has inappropriate values will raise the CORBA::BAD_PARAM standard exception. If a timing-related problem raises a CORBA::COMM_FAILURE. The Bootstrap object maps most system exceptions to CORBA::Invalid_Domain.

The Principal Authenticator object is a locality-constrained object. Therefore, a Principal Authenticator object may not be used through the DII/DSI facilities of CORBA. Any attempt to pass a reference to this object outside of the current process, or any attempt to externalize it using CORBA::ORB::object_to_string, will result in the raising of the CORBA::MARSHAL exception.
Using the Principal Authenticator Object with Certificate-based Authentication

The Principal Authenticator object has been enhanced to support certificate-based authentication. The use of certificate-based authentication is controlled by specifying the `Security::AuthenticationMethod` value of `Tobj::CertificateBased` as a parameter to the `PrincipalAuthenticator::authenticate` operation. When certificate-based authentication is used, the implementation of the `PrincipalAuthenticator::authenticate` operation must retrieve the credentials for the principal by obtaining the private key and digital certificates for the principal and registering them for use with the SSL protocol.

The values of the `security_name` and `auth_data` parameters of the `PrincipalAuthenticator::authenticate` operation are used to open the private key for the principal. If the user does not specify the proper values for both of these parameters, the private key cannot be opened and the user fails to be authenticated. As a result of successfully opening the private key, a chain of digital certificates that represent the local identity of the principal is built. Both the private key and the chain of digital certificates must be registered to be used with the SSL protocol.

WLE Extensions to the Principal Authenticator Object

The WLE product extends the Principal Authenticator object to support a security mechanism similar to the security in BEA TUXEDO. The enhanced functionality is provided by defining the `Tobj::PrincipalAuthenticator` interface. This interface contains methods to provide similar capability to that available from BEA TUXEDO through the `tpinit` function. The interface `Tobj::PrincipalAuthenticator` is derived from the `CORBA SecurityLevel2::PrincipalAuthenticator` interface.

The extended Principal Authenticator object adheres to all the same rules as the Principal Authenticator object defined in the CORBA Services Security Service specification.

The implementation of the extended Principal Authenticator object requires users to supply a user name, client name, and additional authentication data (for example, passwords) used for authentication. Because the information needs to be transmitted over the network to the IIOP Listener/Handler, it is protected to ensure confidentiality. The protection must include encryption of any information provided by the user.
The Credentials Object

An extended Principal Authenticator object that supports the `Tobj::PrincipalAuthenticator` interface provides the same functionality as if the `SecurityLevel2::PrincipalAuthenticator` interface were used to perform the authentication of the principal. However, unlike the `SecurityLevel2::PrincipalAuthenticator::authenticate` method, the logon method defined on the `Tobj::PrincipalAuthenticator` interface does not return a Credentials object.

The Credentials Object

A Credentials object (as shown in Figure 9-2) holds the security attributes of a principal. The Credentials object provides methods to obtain and set the security attributes of the principals it represents. These security attributes include its authenticated or unauthenticated identities and privileges. It also contains information for establishing security associations.

Credentials objects are created as the result of:

- Authentication
- Copying an existing Credentials object
- Asking for a Credentials object via the SecurityCurrent object
Multiple references to a Credentials object are supported. A Credentials object is stateful. It maintains state on behalf of the principal for which it was created. This state includes any information necessary to determine the identity and privileges of the principal it represents. Credentials objects are not associated with the Principal Authenticator object that created it, but must contain some indication of the authentication authority that certified the principal’s identity.

The Credentials object is a locality-constrained object; therefore, a Credentials object may not be used through the DII/DSI facilities. Any attempt to pass a reference to this object outside of the current process, or any attempt to externalize it using CORBA::ORB::object_to_string, will result in the raising of the CORBA::MARSHAL exception.

The Credentials object has been enhanced to allow application developers to indicate the security attributes for establishing secure connections. These attributes allow developers to indicate whether a secure connection requires integrity, confidentiality, or both. To support this capability, two new attributes were added to the SecurityLevel2::Credentials interface.

- The invocation_options_supported attribute indicates which security options are allowed when establishing a secure connection.
- The invocation_options_required attribute allows the application developer to specify the minimum set of security options that must be used in establishing a secure connection.
The SecurityCurrent Object

The SecurityCurrent object (see Figure 9-3) represents the current execution context at both the principal and target objects. The SecurityCurrent object represents service-specific state information associated with the current execution context. Both client and server applications have SecurityCurrent objects that represent state associated with the thread of execution and the process in which the thread is executing.

Figure 9-3   The SecurityCurrent Object

The SecurityCurrent object is a singleton object; there is only a single instance allowed in a process address space. Multiple references to the SecurityCurrent object are supported.

The CORBA services Security Service specification defines two interfaces for the SecurityCurrent object associated with security:

- SecurityLevel1::Current, which derives from CORBA::Current
- SecurityLevel2::Current, which derives from the SecurityLevel1::Current interface

Both interfaces give access to security information associated with the execution context.

At any stage, a client application can determine the default credentials for subsequent invocations by calling the Current::get_credentials method and asking for the invocation credentials. These default credentials are used in all invocations that use object references.
When the `Current::get_attributes` method is invoked by a client application, the attributes returned from the Credentials object are those of the principal.

The SecurityCurrent object is a locality-constrained object; therefore, a SecurityCurrent object may not be used through the DII/DSI facilities. Any attempt to pass a reference to this object outside of the current process, or any attempt to externalize it using `CORBA::ORB::object_to_string`, results in a `CORBA::MARSHAL` exception.
CHAPTER

10 Security Modules

This topic contains the Object Management Group (OMG) Interface Definition Language (IDL) definitions for the following modules that are used in the WLE Security Service:

- CORBA
- TimeBase
- Security
- Security Level 1
- Security Level 2
- Tobj
CORBA Module

The OMG added the `CORBA::Current` interface to the CORBA module to support the Current pseudo-object. The change enables the CORBA module to support Security Replaceability and Security Level 2.

Listing 10-1 shows the `CORBA::Current` interface OMG IDL statements.

Listing 10-1  CORBA::Current Interface OMG IDL Statements

```
module CORBA {
    // Extensions to CORBA
    interface Current {
    };
}

```

TimeBase Module

All data structures pertaining to the basic Time Service, Universal Time Object, and Time Interval Object are defined in the TimeBase module. This allows other services to use these data structures without requiring the interface definitions. The interface definitions and associated enums and exceptions are encapsulated in the TimeBase module.

Listing 10-2 shows the TimeBase module OMG IDL statements.

Listing 10-2  TimeBase Module OMG IDL Statements

```
// From time service
module TimeBase {
    // interim definition of type ulonglong pending the
    // adoption of the type extension by all client ORBs.
```
struct ulonglong {
    unsigned long low;
    unsigned long high;
};
typedef ulonglong TimeT;
typedef short TdfT;
struct UtcT {
    TimeT time;            // 8 octets
    unsigned long inacclo; // 4 octets
    unsigned short inacchi; // 2 octets
    TdfT tdf;              // 2 octets
    // total 16 octets
};

// This information is taken from CORBA services: Common Object Services

Table 10-1 defines the TimeBase module data types.


### Table 10-1 TimeBase Module Data Type Definitions

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>TimeT represents a single time value, which is 64 bit in size, and holds the number of 100 nanoseconds that have passed since the base time. For absolute time, the base is 15 October 1582 00:00.</td>
</tr>
<tr>
<td>ulonglong</td>
<td>OMG IDL does not at present have a native type representing an unsigned 64-bit integer. The adoption of technology submitted against that RFP will provide a means for defining a native type representing unsigned 64-bit integers in OMG IDL. Pending the adoption of that technology, you can use this structure to represent unsigned 64-bit integers, understanding that when a native type becomes available, it may not be interoperable with this declaration on all platforms. This definition is for the interim, and is meant to be removed when the native unsigned 64-bit integer type becomes available in OMG IDL.</td>
</tr>
</tbody>
</table>
The Security module defines the OMG IDL for security data types common to the other security modules. This module depends on the TimeBase module and must be available with any ORB that claims to be security ready.

Listing 10-3 shows the data types supported by the Security module.

Listing 10-3  Security Module OMG IDL Statements

```
module Security {
    typedef sequence<octet> Opaque;

    // Extensible families for standard data types
    struct ExtensibleFamily {
        unsigned short family_definer;
        unsigned short family;
    };

    //security attributes
    typedef unsigned long SecurityAttributeType;
```
// identity attributes; family = 0
const SecurityAttributeType AuditId = 1;
const SecurityAttributeType AccountingId = 2;
const SecurityAttributeType NonRepudiationId = 3;

// privilege attributes; family = 1
const SecurityAttributeType Public = 1;
const SecurityAttributeType AccessId = 2;
const SecurityAttributeType PrimaryGroupId = 3;
const SecurityAttributeType GroupId = 4;
const SecurityAttributeType Role = 5;
const SecurityAttributeType AttributeSet = 6;
const SecurityAttributeType Clearance = 7;
const SecurityAttributeType Capability = 8;

struct AttributeType {
    ExtensibleFamily attribute_family;
    SecurityAttributeType attribute_type;
};

typedef sequence <AttributeType> AttributeTypeLists;
struct SecAttribute {
    AttributeType attribute_type;
    Opaque defining_authority;
    Opaque value;
    // The value of this attribute can be
    // interpreted only with knowledge of type
};

typedef sequence<SecAttribute> AttributeList;

// Authentication return status
enum AuthenticationStatus {
    SecAuthSuccess,
    SecAuthFailure,
    SecAuthContinue,
    SecAuthExpired
};

// Authentication method
typedef unsigned long AuthenticationMethod;

enum CredentialType {
    SecInvocationCredentials,
    SecOwnCredentials,
    SecNRCredentials
}
10 Security Modules

// Pick up from TimeBase
typedef TimeBase::UtcT UtcT;
};


Table 10-2 describes the Security module data type.

Table 10-2 Security Module Data Type Definition

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequence&lt;octet&gt;</td>
<td>Data whose representation is known only to the Security Service implementation.</td>
</tr>
</tbody>
</table>

Security Level 1 Module

This section defines those interfaces available to client application objects that use only Level 1 Security functionality. This module depends on the CORBA module and the Security and TimeBase modules. The Current interface is implemented by the ORB.

Listing 10-4 shows the Security Level 1 module OMG IDL statements.

Listing 10-4 Security Level 1 Module OMG IDL Statements

module SecurityLevel1 {
    interface Current : CORBA::Current {// PIDL
        Security::AttributeList get_attributes(
            in Security::AttributeTypeList attributes
        );
    };
};

// This information is taken from CORBA services: Common Object Services Specification, p. 15-198. Revised Edition:
Security Level 2 Module

This section defines the additional interfaces available to client application objects that use Level 2 Security functionality. This module depends on the CORBA and Security modules.

Listing 10-5 shows the Security Level 2 module OMG IDL statements.

Listing 10-5  Security Level 2 Module OMG IDL Statements

```idl
module SecurityLevel2 {
    // Forward declaration of interfaces
    interface PrincipalAuthenticator;
    interface Credentials;
    interface Current;

    // Interface Principal Authenticator
    interface PrincipalAuthenticator {
        Security::AuthenticationStatus authenticate(
            in Security::AuthenticationMethod method,
            in string security_name,
            in Security::Opaque auth_data,
            in Security::AttributeList privileges,
            out Credentials creds,
            out Security::Opaque continuation_data,
            out Security::Opaque auth_specific_data
        );

        Security::AuthenticationStatus continue_authentication(
            in Security::Opaque response_data,
            inout Credentials creds,
            out Security::Opaque continuation_data,
            out Security::Opaque auth_specific_data
        );
    }

    // Interface Credentials
    interface Credentials {

```
attribute Security::AssociationOptions
    invocation_options_supported;
attribute Security::AssociationOptions
    invocation_options_required;
Security::AttributeList get_attributes(
    in Security::AttributeTypeList attributes);
boolean is_valid(
    out Security::UtcT expiry_time);
);

// Interface Current derived from SecurityLevel1::Current
// providing additional operations on Current at this
// security level. This is implemented by the ORB.
interface Current : SecurityLevel1::Current {
    // PIDL
    void set_credentials(
        in Security::CredentialType cred_type,
        in Credentials cred);
    Credentials get_credentials(
        in Security::CredentialType cred_type);
    readonly attribute PrincipalAuthenticator
        principal_authenticator;
};

// This information is taken from CORBA services: Common Object
// Services Specification, pp. 15-198 to 200. Revised Edition:
// OMG.

## Tobj Module

This section defines the Tobj module interfaces.

This module provides the interfaces you use to program the BEA TUXEDO style of authentication.

Listing 10-6 shows the Tobj module OMG IDL statements.
Listing 10-6  Tobj Module OMG IDL Statements

//Tobj Specific definitions

enum AuthType {
    TOBJ_NOAUTH,
    TOBJ_SYSAUTH,
    TOBJ_APPAUTH
};

typedef sequence<octet>    UserAuthData;

interface PrincipalAuthenticator :
    SecurityLevel2::PrincipalAuthenticator { // PIDL
    AuthType get_auth_type();
    Security::AuthenticationStatus logon(
        in string            user_name,
        in string            client_name,
        in string            system_password,
        in string            user_password,
        in UserAuthData      user_data
    );
    void logoff();
    void build_auth_data(
        in string                    user_name,
        in string                    client_name,
        in string                    system_password,
        in string                    user_password,
        in UserAuthData              user_data,
        out Security::Opaque         auth_data,
        out Security::AttributeList  privileges
    );
};
CHAPTER

11 C++ Security Reference

This topic contains the C++ method descriptions for the WLE Security Service.
SecurityLevel1::Current::get_attributes

Synopsis
Returns attributes for the Current interface.

OMG IDL
Security::AttributeList get_attributes(
    in Security::AttributeTypeList   attributes
);

Argument
attributes
The set of security attributes (privilege attribute types) whose values are desired. If this list is empty, all attributes are returned.

Description
This method gets privilege (and other) attributes from the principal’s credentials for the Current interface.

Return Values
The following table describes valid return values.

Table 11-1

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security::Public</td>
<td>Empty (Public is returned when no authentication was performed)</td>
</tr>
<tr>
<td>Security::AccessId</td>
<td>Null terminated ASCII string containing the WLE user name</td>
</tr>
<tr>
<td>Security::PrimaryGroupId</td>
<td>Null terminated ASCII string containing the WLE name of the principal</td>
</tr>
</tbody>
</table>

Note: The other attribute types are never returned. The defining_authority field is always empty.

Note: This information is taken from CORBA services: Common Object Services Specification, pp. 15-103, 104. Revised Edition: March 31, 1995. Updated: November 1997. Used with permission by OMG.
SecurityLevel2::Current::authenticate

Synopsis
Authenticates the principal and optionally obtains credentials for the principal.

OMG IDL
Security::AuthenticationStatus
authenticate(
    in Security::AuthenticationMethod method,
    in Security::SecurityName security_name,
    in Security::Opaque auth_data,
    in Security::AttributeList privileges,
    out Credentials creds,
    out Security::Opaque continuation_data,
    out Security::Opaque auth_specific_data);

Arguments
method
The security mechanism to be used. Valid values are Tobj::TuxedoSecurity and Tobj::CertificateBased.

security_name
The principal's identification information (for example, logon information). The value must be a pointer to a NULL-terminated string containing the user name of the principal. The string is limited to 30 characters, excluding the NULL character.

When using certificate-based authentication, this name is used to look up a certificate in the LDAP-enabled directory service. It is also used as the basis for the name of the file in which the private key is stored. For example: milozzi@company.com is email address used to look up a certificate in the LDAP-enabled directory service and milozzi_company.pem is the name of the private key file.

auth_data
The principals' authentication, such as their password or private key. If the Tobj::TuxedoSecurity security mechanism is specified, the value of this argument is dependent on the configured level of authentication. If the Tobj::CertificateBased argument is specified, the value of this argument is the pass phrase used to decrypt the private key of the principal.

privileges
The privilege attributes requested.

creds
The object reference of the newly created Credentials object. The object reference is not fully initialized; therefore, the object reference cannot be used until the return value of the SecurityLevel2::Current::authenticate method is SecAuthSuccess.

Using Security  11-3
continuation_data
If the return value of the SecurityLevel2::Current::authenticate method is SecAuthContinue, this argument contains the challenge information for the authentication to continue. The value returned will always be empty.

auth_specific_data
Information specific to the authentication service being used. The value returned will always be empty.

Description
The SecurityLevel2::Current::authenticate method is used by the client application to authenticate the principal and optionally request privilege attributes that the principal requires during its session with the WLE domain.

If the Tobj::TuxedoSecurity security mechanism is to be specified, the same functionality can be obtained by calling the Tobj::PrincipalAuthenticator::logon operation, which provides the same functionality but is specifically tailored for use with the TUXEDO-style authentication security mechanism.

Return Values
The following table describes the valid return values.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SecAuthSuccess</td>
<td>The object reference of the newly created Credentials object returned as the value of the creds argument is initialized and ready to use.</td>
</tr>
<tr>
<td>SecAuthFailure</td>
<td>The authentication process was inconsistent or an error occurred during the process. Therefore, the creds argument does not contain an object reference to a Credentials object. If the Tobj::TuxedoSecurity security mechanism is used, this return value indicates that authentication failed or that the client application was already authenticated and did not call either the Tobj::PrincipalAuthenticator::logoff or Tobj_Bootstrap::destroy_current operation.</td>
</tr>
</tbody>
</table>

11 Using Security
SecAuthContinue
Indicates that the authentication procedure uses a challenge/response mechanism. The creds argument contains the object reference of a partially initialized Credentials object. The continuation_data indicates the details of the challenge.

SecAuthExpired
Indicates that the authentication data contained some information, the validity of which had expired; therefore, the creds argument does not contain an object reference to a Credentials object.
If the Tobj::TuxedoSecurity security mechanism is used, this return value is never returned.

CORBA::BAD_PARAM
The CORBA::BAD_PARAM exception occurs if:
- Values for the security_name, auth_data, or privileges arguments are not specified.
- The length of an input argument exceeds the maximum length of the argument.
- The value of the method argument is Tobj::TuxedoSecurity and the content of the auth_data argument contains a username or a clientname as an empty or a NULL string.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SecAuthContinue</td>
<td>Indicates that the authentication procedure uses a challenge/response mechanism. The creds argument contains the object reference of a partially initialized Credentials object. The continuation_data indicates the details of the challenge.</td>
</tr>
<tr>
<td>SecAuthExpired</td>
<td>Indicates that the authentication data contained some information, the validity of which had expired; therefore, the creds argument does not contain an object reference to a Credentials object. If the Tobj::TuxedoSecurity security mechanism is used, this return value is never returned.</td>
</tr>
<tr>
<td>CORBA::BAD_PARAM</td>
<td>The CORBA::BAD_PARAM exception occurs if:</td>
</tr>
<tr>
<td></td>
<td>- Values for the security_name, auth_data, or privileges arguments are not specified.</td>
</tr>
<tr>
<td></td>
<td>- The length of an input argument exceeds the maximum length of the argument.</td>
</tr>
<tr>
<td></td>
<td>- The value of the method argument is Tobj::TuxedoSecurity and the content of the auth_data argument contains a username or a clientname as an empty or a NULL string.</td>
</tr>
</tbody>
</table>
SecurityLevel2::Current::set_credentials

Synopsis
Sets credentials type.

OMG IDL
void set_credentials(
    in Security::CredentialType   cred_type,
    in Credentials                creds
);

Arguments
cred_type
The type of credentials to be set; that is, invocation, own, or non-repudiation.

creds
The object reference to the Credentials object, which is to become the default.

Description
This method can be used only to set SecInvocationCredentials; otherwise, set_credentials raises CORBA::BAD_PARAM. The credentials must have been obtained from a previous call to SecurityLevel2::Current::get_credentials or SecurityLevel2::PrincipalAuthenticator::authenticate.

Return Values
None.

Note: This information is taken from CORBAservices: Common Object Services Specification, p. 15-104. Revised Edition: March 31, 1995. Updated: November 1997. Used with permission by OMG.
**SecurityLevel2::Current::get_credentials**

**Synopsis**
Gets credentials type.

**OMG IDL Definition**
```plaintext
Credentials get_credentials(
    in Security::CredentialType   cred_type
);
```

**Argument**
cred_type
The type of credentials to get.

**Description**
This call can be used only to get SecInvocationCredentials; otherwise, get_credentials raises CORBA::BAD_PARAM. If no credentials are available, get_credentials raises CORBA::BAD_INV_ORDER.

**Return Values**
Returns the active credentials in the client application only.

**Note:** This information is taken from *CORBA services: Common Object Services Specification*, p. 15-105. Revised Edition: March 31, 1995. Updated: November 1997. Used with permission by OMG.
SecurityLevel2::Current::principal_authenticator

Synopsis  Returns the PrincipalAuthenticator.

OMG IDL Definition

readonly attribute PrincipalAuthenticator
principal_authenticator;

Description  The PrincipalAuthenticator returned by the principal_authenticator attribute is of actual type Tobj::PrincipalAuthenticator. Therefore, it can be used both as a Tobj::PrincipalAuthenticator and as a SecurityLevel2::PrincipalAuthenticator.

Note:  This method raises CORBA::BAD_INV_ORDER if it is called on an invalid SecurityCurrent object.

Return Values  Returns the PrincipalAuthenticator.
SecurityLevel2::Credentials

Synopsis
Represents a particular principal’s credential information that is specific to a process. A Credentials object that supports the SecurityLevel2::Credentials interface is a locality-constrained object. Any attempt to pass a reference to the object outside its locality, or any attempt to externalize the object using the CORBA::ORB::object_to_string() operation, results in a CORBA::Marshall exception.

OMG IDL Definition
#ifndef _SECURITY_LEVEL_2_IDL
#define _SECURITY_LEVEL_2_IDL

#include <SecurityLevel1.idl>

#pragma prefix "omg.org"

module SecurityLevel2
{
  interface Credentials
  {
    attribute Security::AssociationOptions
      invocation_options_supported;
    attribute Security::AssociationOptions
      invocation_options_required;
    Security::AttributeList
      get_attributes(
        in   Security::AttributeTypeList     attributes );

    boolean
    is_valid(
      out Security::UtcT                  expiry_time );
  }
};
#endif /* _SECURITY_LEVEL_2_IDL */

C++ Declaration
class SecurityLevel2
{
  public:
    class Credentials;
    typedef Credentials *Credentials_ptr;

    class Credentials : public virtual CORBA::Object
    {
      public:

static Credentials_ptr _duplicate(Credentials_ptr obj);
static Credentials_ptr _narrow(CORBA::Object_ptr obj);
static Credentials_ptr _nil();

virtual Security::AssociationOptions
  invocation_options_supported() = 0;
virtual void
  invocation_options_supported(
    const Security::AssociationOptions options ) = 0;
virtual Security::AssociationOptions
  invocation_options_required() = 0;
virtual void
  invocation_options_required(
    const Security::AssociationOptions options ) = 0;

virtual Security::AttributeList *
  get_attributes(
    const Security::AttributeTypeList & attributes) = 0;

virtual CORBA::Boolean
  is_valid( Security::UtcT_out expiry_time) = 0;

protected:
  Credentials(CORBA::Object_ptr obj = 0);
  virtual ~Credentials() { }

private:
  Credentials( const Credentials&) { }
  void operator=(const Credentials&) { }
}; // class Credentials
}; // class SecurityLevel2
SecurityLevel2::Credentials::get_attributes

Synopsis  Gets the attribute list attached to the credentials.

OMG IDL  Security::AttributeList get_attributes(
          in AttributeTypeList   attributes
       );

Argument  attributes
          The set of security attributes (privilege attribute types) whose values are
          desired. If this list is empty, all attributes are returned.

Description  This method returns the attribute list attached to the credentials of the principal. In the
          list of attribute types, you are required to include only the type value(s) for the
          attributes you want returned in the AttributeList. Attributes are not currently
          returned based on attribute family or identities. In most cases, this is the same result
          you would get if you called SecurityLevel1::Current::get_attributes(),
          since there is only one valid set of credentials in the principal at any instance in time.
          The results could be different if the credentials are not currently in use.

Return Values  Returns attribute list.

Note:  This is information taken from CORBA services: Common Object Services Specification, p. 15-97. Revised Edition: March 31, 1995. Updated: November 1997. Used with permission by OMG.
11 C++ Security Reference

SecurityLevel2::Credentials::invocation_options_supported

Synopsis Indicates the maximum number of security options that can be used when establishing an SSL connection to make an invocation on an object in the WLE domain.

OMG IDL Definition

```
attribute Security::AssociationOptions(
  invocation_options_supported;
```

Argument None.

Description This method should be used in conjunction with the SecurityLevel2::Credentials::invocation_options_required method.

The following security options can be specified:

<table>
<thead>
<tr>
<th>Security Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoProtection</td>
<td>The SSL protocol does not provide message protection.</td>
</tr>
<tr>
<td>Integrity</td>
<td>The SSL protocol provides an integrity check of messages. Digital signatures are used to protect the integrity of messages.</td>
</tr>
<tr>
<td>Confidentiality</td>
<td>The SSL connection protects the confidentiality of messages. Cryptography is used to protect the confidentiality of messages.</td>
</tr>
<tr>
<td>DetectReplay</td>
<td>The SSL protocol provides replay detection. Replay occurs when a message is sent repeatedly with no detection.</td>
</tr>
<tr>
<td>DetectMisordering</td>
<td>The SSL protocol provides sequence error detection for requests and request fragments.</td>
</tr>
<tr>
<td>EstablishTrustInTarget</td>
<td>Indicates that the target of a request authenticates itself to the initiating principal.</td>
</tr>
<tr>
<td>EstablishTrustInTarget</td>
<td>Indicates that the initiating principal authenticates itself to the target of the request.</td>
</tr>
<tr>
<td>NoDelegation</td>
<td>Indicates that the principal permits an intermediate object to use its privileges for the purpose of access control decisions. However, the principal’s privileges are not delegated so the intermediate object cannot use the privileges when invoking the next object in the chain.</td>
</tr>
</tbody>
</table>

11-12 Using Security
Return Values

The list of defined security options.

<table>
<thead>
<tr>
<th>Security Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpleDelegation</td>
<td>Indicates that the principal permits an intermediate object to use its privileges for the purpose of access control decisions, and delegates the privileges to the intermediate object. The target object receives only the privileges of the client application and does not know the identity of the intermediate object. When this invocation option is used without restrictions on the target object, the behavior is known as impersonation.</td>
</tr>
<tr>
<td>CompositeDelegation</td>
<td>Indicates that the principal permits the intermediate object to use its credentials and delegate them. The privileges of both the principal and the intermediate object can be checked.</td>
</tr>
</tbody>
</table>

Note: A CORBA::BAD_PARAM exception is returned if the security options specified are not supported by the security mechanism defined for the WLE application. This exception can also occur if the security options specified have less capabilities than the security options specified by the SecurityLevel2::Credentials::invocation_options_required method.

A Credentials object with a security mechanism of Tobj::TuxedoSecurity always returns the CORBA::BAD_PARAM exception.
SecurityLevel2::Credentials::invocation_options_required

Synopsis
Specifies the minimum number of security options to be used when establishing an
SSL connection to make an invocation on a target object in the WLE domain.

OMG IDL
Definition
attribute Security::AssociationOptions(
  invocation_options_required;

Argument
None.

Description
Use this method to specify that communication between principals and the WLE
domain should be protected. After using this method, a Credentials object makes an
invocation on a target object using the SSL protocol with the defined level of security
options. This method should be used in conjunction with the
SecurityLevel2::Credentials::invocation_options_supported method.

The following security options can be specified:

Table 11-4

<table>
<thead>
<tr>
<th>Security Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoProtection</td>
<td>The SSL protocol does not provide message protection.</td>
</tr>
<tr>
<td>Integrity</td>
<td>The SSL protocol provides an integrity check of messages. Digital signatures are used to protect the integrity of messages.</td>
</tr>
<tr>
<td>Confidentiality</td>
<td>The SSL connection protects the confidentiality of messages. Cryptography is used to protect the confidentiality of messages.</td>
</tr>
<tr>
<td>DetectReplay</td>
<td>The SSL protocol provides replay detection. Replay occurs when a message is sent repeatedly with no detection.</td>
</tr>
<tr>
<td>DetectMisordering</td>
<td>The SSL protocol provides sequence error detection for requests and request fragments.</td>
</tr>
<tr>
<td>EstablishTrustInTarget</td>
<td>Indicates that the target of a request authenticates itself to the initiating principal.</td>
</tr>
<tr>
<td>EstablishTrustInTarget</td>
<td>Indicates that the initiating principal authenticates itself to the target of the request.</td>
</tr>
</tbody>
</table>
Return Values

The list of defined security options.

If the Tobj::TuxedoSecurity security mechanism is used to create the security association, only the NoProtection, EstablishTrustInClient, and SimpleDelegation security options are returned. The EstablishTrustInClient security option appears only if the security level of the WLE application is defined to require passwords to access the WLE domain.

Note: A CORBA::BAD_PARAM exception is returned if the security options specified are not supported by the security mechanism defined for the WLE application. This exception can also occur if the security options specified have more capabilities than the security options specified by the SecurityLevel2::Credentials::invocation_options_supported method.

A Credentials object with a parameter of Tobj::TuxedoSecurity always returns the CORBA::BAD_PARAM exception.
SecurityLevel2::Credentials::is_valid

Synopsis    Checks status of credentials.
OMG IDL     boolean is_valid(
Definition   out Security::UtcT      expiry_time
              )

Description This method returns TRUE if the credentials used are active at the time; that is, you did not call Tobj::PrincipalAuthenticator::logoff or Tobj_Bootstrap::destroy_current. If this method is called after Tobj::PrincipalAuthenticator::logoff(), FALSE is returned. If this method is called after Tobj_Bootstrap::destroy_current(), the CORBA::BAD_INV_ORDER exception is raised.

Return Values The expiration date returned contains the maximum unsigned long long value in C++ and maximum long in Java. Until the unsigned long long datatype is adopted, the ulonglong datatype is substituted. The ulonglong datatype is defined as follows:

```c
// interim definition of type ulonglong pending the
// adoption of the type extension by all client ORBs.
struct ulonglong {    
    unsigned long      low;
    unsigned long      high;
};
```

SecurityLevel2::PrincipalAuthenticator

Synopsis

Allows a principal to be authenticated. A Principal Authenticator object that supports the SecurityLevel2::PrincipalAuthenticator interface is a locality-constrained object. Any attempt to pass a reference to the object outside its locality, or any attempt to externalize the object using the CORBA::ORB::object_to_string() operation, results in a CORBA::Marshall exception.

OMG IDL Definition

#ifndef _SECURITY_LEVEL_2_IDL
#define _SECURITY_LEVEL_2_IDL

#include <SecurityLevel1.idl>

#pragma prefix "omg.org"

module SecurityLevel2
{

    interface PrincipalAuthenticator
    {
        // Locality Constrained
        Security::AuthenticationStatus authenticate ( 
            in Security::AuthenticationMethod method,
            in Security::SecurityName security_name,
            in Security::Opaque auth_data,
            in Security::AttributeList privileges,
            out Credentials creds,
            out Security::Opaque continuation_data,
            out Security::Opaque auth_specific_data
        );

        Security::AuthenticationStatus continue_authentication ( 
            in Security::Opaque response_data,
            in Credentials creds,
            out Security::Opaque continuation_data,
            out Security::Opaque auth_specific_data
        );
    };
};

#pragma prefix "beasys.com"
module Tobj
{
    const Security::AuthenticationMethod TuxedoSecurity = 0x54555800;
};

#endif // SECURITY_LEVEL_2_IDL
CertificateBased = 0x43455254;
};

C++ Declaration

class SecurityLevel2
{
public:
    class PrincipalAuthenticator;
    typedef PrincipalAuthenticator * PrincipalAuthenticator_ptr;

class PrincipalAuthenticator : public virtual CORBA::Object
{
public:
    static PrincipalAuthenticator_ptr _duplicate(PrincipalAuthenticator_ptr obj);
    static PrincipalAuthenticator_ptr _narrow(CORBA::Object_ptr obj);
    static PrincipalAuthenticator_ptr _nil();

    virtual Security::AuthenticationStatus authenticate (  
        Security::AuthenticationMethod method,  
        const char * security_name,  
        const Security::Opaque & auth_data,  
        const Security::AttributeList & privileges,  
        Credentials_out creds,  
        Security::Opaque_out continuation_data,  
        Security::Opaque_out auth_specific_data) = 0;

    virtual Security::AuthenticationStatus continue_authentication (  
        const Security::Opaque & response_data,  
        Credentials_ptr & creds,  
        Security::Opaque_out continuation_data,  
        Security::Opaque_out auth_specific_data) = 0;

protected:
    PrincipalAuthenticator(CORBA::Object_ptr obj = 0);
    virtual ~PrincipalAuthenticator() { }

private:
    PrincipalAuthenticator( const PrincipalAuthenticator&) { }  
    void operator=(const PrincipalAuthenticator&) { }  
};  // class PrincipalAuthenticator
SecurityLevel2::PrincipalAuthenticator::continue_authentication

**Synopsis**  
Always fails.

**OMG IDL Definition**  
```idl  
Security::AuthenticationStatus continue_authentication(  
in Security::Opaque response_data,  
inout Credentials creds,  
out Security::Opaque continuation_data,  
out Security::Opaque auth_specific_data  
);  
```

**Description**  
Because the WLE software does authentication in one step, this method always fails and returns `Security::AuthenticationStatus::SecAuthFailure`.

**Return Values**  
Always returns `Security::AuthenticationStatus::SecAuthFailure`.

**Note:**  
11 C++ Security Reference

Tobj::PrincipalAuthenticator::get_auth_type

Synopsis
Gets the type of authentication expected by the WLE domain.

OMG IDL Definition
AuthType get_auth_type();

Description
This method returns the type of authentication expected by the WLE domain.

Note: This method raises CORBA::BAD_INV_ORDER if it is called with an invalid SecurityCurrent object.

Return Values
A reference to the Tobj_AuthType enumeration. The following table describes the valid return values.

Table 11-5

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOBJ_NOAUTH</td>
<td>No authentication is needed; however, the client application can still authenticate itself by specifying a user name and a client application name. No password is required. To specify this level of security, specify the NONE value for the SECURITY parameter in the RESOURCES section of the UBBCONFIG file.</td>
</tr>
<tr>
<td>TOBJ_SYSAUTH</td>
<td>The client application must authenticate itself to the WLE domain, and must specify a user name, a name, and a password for the client application. To specify this level of security, specify the APP_PW value for the SECURITY parameter in the RESOURCES section of the UBBCONFIG file.</td>
</tr>
<tr>
<td>TOBJ_APPAUTH</td>
<td>The client application must provide proof material that authenticates the client application to the WLE domain. The proof material may be a password or a digital certificate. To specify this level of security, specify the USER_AUTH value for the SECURITY parameter in the RESOURCES section of the UBBCONFIG file.</td>
</tr>
</tbody>
</table>

Returns the type of authentication required to access the WLE domain.
**Synopsis**

Authenticates the principal.

**OMG IDL Definition**

```java
Security::AuthenticationStatus logon(
    in string user_name,
    in string client_name,
    in string system_password,
    in string user_password,
    in UserAuthData user_data
);
```

**Arguments**

- **user_name**
  
  The WLE user name. The authentication level is `TOBJ_NOAUTH`. If `user_name` is NULL or empty, or exceeds 30 characters, `logon` raises the `CORBA::BAD_PARAM` exception.

- **client_name**
  
  The WLE name of the client application. The authentication level is `TOBJ_NOAUTH`. If the `client_name` is NULL or empty, or exceeds 30 characters, `logon` raises the `CORBA::BAD_PARAM` exception.

- **system_password**
  
  The WLE client application password. The authentication level is `TOBJ_SYSAUTH`. If the `client_name` is NULL or empty, or exceeds 30 characters, `logon` raises the `CORBA::BAD_PARAM` exception.

  **Note:** The `system_password` must not exceed 30 characters.

- **user_password**
  
  The user password (needed for use by the default WLE authentication service). The authentication level is `TOBJ_APPAUTH`.

- **user_data**
  
  Data that is specific to the client application (needed for use by a custom WLE authentication service). The authentication level is `TOBJ_APPAUTH`.

  **Note:** `TOBJ_SYSAUTH` includes the requirements of `TOBJ_NOAUTH`, plus a client application password. `TOBJ_APPAUTH` includes the requirements of `TOBJ_SYSAUTH`, plus additional information, such as a user password or user data.

  **Note:** The `user_password` and `user_data` arguments are mutually exclusive, depending on the requirements of the authentication service used in the configuration of the WLE domain. The WLE default authentication service expects a user password. A customized authentication service may
require user data. The logon call raises the CORBA::BAD_PARAM exception if both user_password and user_data are specified.

**Description**
This method authenticates the principal via the IIOP Listener/Handler so that the principal can access a WLE domain. This method is functionally equivalent to SecurityLevel2::PrincipalAuthenticator::authenticate, but the arguments are oriented to TUXEDO-style authentication.

**Note:** This method raises CORBA::BAD_INV_ORDER if it is called with an invalid SecurityCurrent object.

**Return Values**
The following table describes the valid return values.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security::AuthenticationStatus::SecAuthSuccess</td>
<td>The authentication succeeded.</td>
</tr>
<tr>
<td>Security::AuthenticationStatus::SecAuthFailure</td>
<td>The authentication failed, or the client application was already authenticated and did not call one of the following methods: Tobj::PrincipalAuthenticator::logoff Tobj_Bootstrap::destroy_current</td>
</tr>
<tr>
<td>CORBA::INVALID_DOMAIN</td>
<td>The method was used with the corbaloc or corbalocs URL address format.</td>
</tr>
</tbody>
</table>
**Tobj::PrincipalAuthenticator::logoff**

<table>
<thead>
<tr>
<th><strong>Synopsis</strong></th>
<th>Discards the security context associated with the principal.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OMG IDL</strong></td>
<td><strong>void logoff();</strong></td>
</tr>
<tr>
<td><strong>Definition</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This call discards the security context, but does not close the network connections to the WLE domain. Logoff also invalidates the current credentials. After logging off, invocations using existing object references fail if the authentication type is not TOBJ_NOAUTH.</td>
</tr>
<tr>
<td></td>
<td>If the principal is currently authenticated to a WLE domain, calling Tobj_Bootstrap::destroy_current() calls logoff implicitly.</td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td>This method raises CORBA::BAD_INV_ORDER if it is called with an invalid SecurityCurrent object.</td>
</tr>
<tr>
<td><strong>Return Values</strong></td>
<td><strong>None.</strong></td>
</tr>
</tbody>
</table>
Tobj::PrincipalAuthenticator::build_auth_data

Synopsis
Creates authentication data and attributes for use by
SecurityLevel2::PrincipalAuthenticator::authenticate.

OMG IDL
Definition
void build_auth_data(
    in string                     user_name,
    in string                     client_name,
    in string                     system_password,
    in string                     user_password,
    in UserAuthData               user_data,
    out Security::Opaque          auth_data,
    out Security::AttributeList   privileges
);

Arguments
user_name
The WLE user name.

client_name
The WLE client name.

system_password
The WLE client application password.

user_password
The user password (default WLE authentication service).

user_data
Client application-specific data (custom WLE authentication service).

auth_data
For use by authenticate.

privileges
For use by authenticate.

Note: If user_name, client_name, or system_password is NULL or empty, or
exceeds 30 characters, the subsequent authenticate method invocation
raises the CORBA::BAD_PARAM exception.
**Note:** The `user_password` and `user_data` parameters are mutually exclusive, depending on the requirements of the authentication service used in the configuration of the WLE domain. The WLE default authentication service expects a user password. A customized authentication service may require user data. If both `user_password` and `user_data` are specified, the subsequent authentication call raises the `CORBA::BAD_PARAM` exception.

**Description**
This method is a helper function that creates authentication data and attributes to be used by `SecurityLevel2::PrincipalAuthenticator::authenticate`.

**Note:** This method raises `CORBA::BAD_INV_ORDER` if it is called with an invalid `SecurityCurrent` object.

**Return Values**
None.
For information about the security package application programming interface (API), see the WLE Javadoc.
This topic contains the Automation method descriptions for the WLE Security service. In addition, the topic contains programming examples that illustrate using the Automation methods to implement security in an ActiveX client application.

**Note:** The Automation security methods do not support certificate-based authentication or the use of the SSL protocol.
Method Descriptions

This section describes the Automation Security Service methods.

**DISecurityLevel2_Current**

The DISecurityLevel2_Current object is a BEA implementation of the CORBA Security model. In this release of the WLE software, the get_attributes(), set_credentials(), get_credentials(), and Principal_Authenticator() methods are supported.
**DISecurityLevel2_Current.get_attributes**

**Synopsis**  
Returns attributes for the Current interface.

**MIDL Mapping**  
```c
HRESULT get_attributes(
    [in] VARIANT attributes,
    [in,out,optional] VARIANT* exceptionInfo,
    [out,retval] VARIANT* returnValue);
```

**Automation Mapping**  
Function `get_attributes(attributes, [exceptionInfo])`

**Parameters**

- **attributes**  
The set of security attributes (privilege attribute types) whose values are desired. If this list is empty, all attributes are returned.

- **exceptionInfo**  
An optional input argument that allows the client application to get additional exception data if an error occurs. For the ActiveX client applications, all exception data is returned in the OLE Automation Error Object.

**Description**  
This method gets privilege (and other) attributes from the credentials for the client application from the Current interface.

**Return Values**  
A variant containing an array of `DISecurity_SecAttribute` objects. The following table describes the valid return values.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security::Public</td>
<td>Empty (Public is returned when no authentication was performed.)</td>
</tr>
<tr>
<td>Security::AccessId</td>
<td>Null-terminated ASCII string containing the WLE user name</td>
</tr>
<tr>
<td>Security::PrimaryGroupId</td>
<td>Null-terminated ASCII string containing the WLE name of the client application</td>
</tr>
</tbody>
</table>
**DISecurityLevel2_Current.set_credentials**

*Synopsis*  
Sets credentials type.

*MIDL Mapping*  
```c
HRESULT set_credentials(
    [in] Security_CredentialType cred_type,
    [in] DISecurityLevel2_Credentials* cred,
    [in,out,optional] VARIANT* exceptionInfo);
```

*Automation Mapping*  
```vbnet
Sub set_credentials(cred_type As Security_CredentialType,
    cred As DISecurityLevel2_Credentials,
    [exceptionInfo])
```

*Description*  
This method can be used only to set `SecInvocationCredentials`; otherwise, `set_credentials` raises `CORBA::BAD_PARAM`. The credentials must have been obtained from a previous call to `DISecurityLevel2_Current.get_credentials`.

*Arguments*  
- `cred_type`  
The type of credentials to be set; that is, invocation, own, or nonrepudiation.
- `cred`  
The object reference to the Credentials object, which is to become the default.
- `exceptionInfo`  
An optional input argument that allows the client application to get additional exception data if an error occurs. For the ActiveX client applications, all exception data is returned in the OLE Automation Error Object.

*Return Values*  
None.
**DISecurityLevel2_Current.get_credentials**

**Synopsis**
Gets credentials type.

**MIDL Mapping**
```c
HRESULT get_credentials(
    [in] Security_CredentialType cred_type,
    [in,out,optional] VARIANT* exceptionInfo,
    [out,retval] DISecurityLevel2_Credentials** returnValue);
```

**Automation Mapping**
```c
Function get_credentials(cred_type As Security_CredentialType, 
    [exceptionInfo]) As DISecurityLevel2_Credentials
```

**Description**
This call can be used only to get SecInvocationCredentials; otherwise, 
get_credentials raises CORBA::BAD_PARAM. If no credentials are available, 
get_credentials raises CORBA::BAD_INV_ORDER.

**Arguments**
- **cred_type**
  The type of credentials to get.
- **exceptionInfo**
  An optional input argument that allows the client application to get additional 
  exception data if an error occurs. For the ActiveX client application, all 
  exception data is returned in the OLE Automation Error Object.

**Return Values**
A DISecurityLevel2_Credentials object for the active credentials in the client 
application only.
**Synopsis**

Returns the `PrincipalAuthenticator`.

**MIDL Mapping**

```c
HRESULT principal_authenticator([out, retval]
    DITobj_PrincipalAuthenticator** returnValue);
```

**Automation Mapping**

Property `principal_authenticator` As `DITobj_PrincipalAuthenticator`

**Description**

The `PrincipalAuthenticator` returned by the `principal_authenticator` property is of actual type `DITobj_PrincipalAuthenticator`. Therefore, it can be used as a `DISecurityLevel2_PrincipalAuthenticator`.

**Note:** This method raises `CORBA::BAD_INV_ORDER` if it is called on an invalid `SecurityCurrent` object.

**Return Values**

A `DITobj_PrincipalAuthenticator` object.
DITobj_PrincipalAuthenticator

The DITobj_PrincipalAuthenticator object is used to log in to and log out of the WLE domain. In this release of the WLE software, the authenticate, build_auth_data(), continue_authentication(), get_auth_type(), logon(), and logoff() methods are implemented.
**DIObj_PrincipalAuthenticator.authenticate**

**Synopsis**
Authenticates the client application.

**MIDL Mapping**
```c
HRESULT authenticate(
    [in] long method,
    [in] BSTR security_name,
    [in] VARIANT auth_data,
    [in] VARIANT privileges,
    [out] DISecurityLevel2_Credentials** creds,
    [out] VARIANT* continuation_data,
    [out] VARIANT* auth_specific_data,
    [in,out,optional] VARIANT* exceptionInfo,
    [out,retval] Security_AuthenticationStatus* returnValue);
```

**Automation Mapping**
```c
Function authenticate(method As Long, security_name As String,
    auth_data, privileges, creds As DISecurityLevel2_Credentials,
    continuation_data, auth_specific_data, [exceptionInfo]) As Security_AuthenticationStatus
```

**Arguments**
- **method**
  Must be Tobj::TuxedoSecurity. If method is invalid, authenticate raises CORBA::BAD_PARAM.
- **security_name**
  The WLE user name.
- **auth_data**
  As returned by DIObj_PrincipalAuthenticator.build_auth_data. If auth_data is invalid, authenticate raises CORBA::BAD_PARAM.
- **privileges**
  As returned by DIObj_PrincipalAuthenticator.build_auth_data. If privileges is invalid, authenticate raises CORBA::BAD_PARAM.
- **creds**
  Placed into the SecurityCurrent object.
- **continuation_data**
  Always empty.
auth_specific_data
Always empty.

exceptioninfo
An optional input argument that allows the client application to get additional exception data if an error occurs. For the ActiveX client application, all exception data is returned in the OLE Automation Error Object.

Description
This method authenticates the client application via the IIOP Listener/Handler so that it can access a WLE domain.

Return Values
A Security::AuthenticationStatus Enum value. The following table describes the valid return values.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security::AuthenticationStatus::SecAuthSuccess</td>
<td>The authentication succeeded.</td>
</tr>
<tr>
<td>Security::AuthenticationStatus::SecAuthFailure</td>
<td>The authentication failed, or the client application was already authenticated and did not invoke Tobj::PrincipalAuthenticator::logoff or Tobj_Bootstrap::destroy_current.</td>
</tr>
</tbody>
</table>
**DITobj_PrincipalAuthenticator.build_auth_data**

**Synopsis**
Creates authentication data and attributes for use by
DITobj_PrincipalAuthenticator.authenticate.

**MIDL Mapping**

```c
HRESULT build_auth_data(
    [in] BSTR user_name,
    [in] BSTR client_name,
    [in] BSTR system_password,
    [in] BSTR user_password,
    [in] VARIANT user_data,
    [out] VARIANT* auth_data,
    [out] VARIANT* privileges,
    [in,out,optional] VARIANT* exceptionInfo);
```

**Automation Mapping**

```c
 Sub build_auth_data(user_name As String, client_name As String,
    system_password As String, user_password As String, user_data,
    auth_data, privileges, [exceptionInfo])
```

**Arguments**

- **user_name**
  The WLE user name.

- **client_name**
  A name of the WLE client application.

- **system_password**
  The password for the WLE client application.

- **user_password**
  The user password (default WLE authentication service).

- **user_data**
  Client application-specific data (custom WLE authentication service).

- **auth_data**
  For use by authenticate.

- **privileges**
  For use by authenticate.

- **exceptioninfo**
  An optional input argument that allows the client application to get additional exception data if an error occurs. For the ActiveX client application, all exception data is returned in the OLE Automation Error Object.
**Method Descriptions**

**Note:** If `user_name`, `client_name`, or `system_password` is NULL or empty, or exceeds 30 characters, the subsequent authenticate method invocation raises the `CORBA::BAD_PARAM` exception.

**Note:** The `user_password` and `user_data` parameters are mutually exclusive, depending on the requirements of the authentication service used in the configuration of the WLE domain. The WLE default authentication service expects a user password. A customized authentication service may require user data. If both `user_password` and `user_data` are specified, the subsequent authentication call raises the `CORBA::BAD_PARAM` exception.

**Description**

This method is a helper function that creates authentication data and attributes to be used by `DITobj_PrincipalAuthenticator.authenticate`.

**Note:** This method raises `CORBA::BAD_INV_ORDER` if it is called with an invalid `SecurityCurrent` object.

**Return Values**

None.
**DIObj_PrincipalAuthenticator.continue_authentication**

**Synopsis**
Always returns `Security::AuthenticationStatus::SecAuthFailure`.

**MIDL Mapping**
```c
HRESULT continue_authentication(
    [in] VARIANT response_data,
    [in,out] DISecurityLevel2_Credentials** creds,
    [out] VARIANT* continuation_data,
    [out] VARIANT* auth_specific_data,
    [in,out,optional] VARIANT* exceptionInfo,
    [out,retval] Security_AuthenticationStatus* returnValue);
```

**Automation Mapping**
```c
Function continue_authentication(response_data,
    creds As DISecurityLevel2_Credentials, continuation_data,
    auth_specific_data, [exceptionInfo]) As Security_AuthenticationStatus
```

**Description**
Because the WLE software does authentication in one step, this method always fails and returns `Security::AuthenticationStatus::SecAuthFailure`.

**Return Values**
Always returns `SecAuthFailure`. 
**DITobj_PrincipalAuthenticator.get_auth_type**

**Synopsis**
Gets the type of authentication expected by the WLE domain.

**MIDL Mapping**
```c
HRESULT get_auth_type(
    [in, out, optional] VARIANT* exceptionInfo,
    [out, retval] Tobj_AuthType* returnValue);
```

**Automation Mapping**
```vbnet
Function get_auth_type([exceptionInfo]) As Tobj_AuthType
```

**Argument**
exceptioninfo
An optional input argument that allows the client application to get additional exception data if an error occurs. For the ActiveX client application, all exception data is returned in the OLE Automation Error Object.

**Description**
This method returns the type of authentication expected by the WLE domain.

**Note:** This method raises CORBA::BAD_INV_ORDER if it is called with an invalid SecurityCurrent object.

**Returned Values**
A reference to the Tobj_AuthType enumeration. The following table describes the valid return values.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOBJ_NOAUTH</td>
<td>No authentication is needed; however, the client application can still authenticate itself by specifying a user name and a client application name. No password is required. To specify this level of security, specify the NONE value for the SECURITY parameter in the RESOURCES section of the UBBCONFIG file.</td>
</tr>
</tbody>
</table>
The client application must authenticate itself to the WLE domain, and must specify a user name, a name, and a password for the client application.

To specify this level of security, specify the APP_PW value for the SECURITY parameter in the RESOURCES section of the UBBCONFIG file.

The client application must provide proof material that authenticates the client application to the WLE domain. The proof material may be a password or a digital certificate.

To specify this level of security, specify the USER_AUTH value for the SECURITY parameter in the RESOURCES section of the UBBCONFIG file.
DIObject_PrincipalAuthenticator.logon

**Synopsis**
Logs in to the WLE domain. The correct input parameters depend on the authentication level.

**MIDL Mapping**

```c
HRESULT logon(
    [in] BSTR user_name,
    [in] BSTR client_name,
    [in] BSTR system_password,
    [in] BSTR user_password,
    [in] VARIANT user_data,
    [in,out,optional] VARIANT* exceptionInfo,
    [out,retval] Security_AuthenticationStatus* returnValue);
```

**Automation Mapping**

```c
Function logon(user_name As String, client_name As String,
               system_password As String, user_password As String,
               user_data, [exceptionInfo]) As Security_AuthenticationStatus
```

**Description**
For remote WLE client applications, this method authenticates the client application via the IIOP Listener/Handler so that the remote client application can access a WLE domain. This method is functionally equivalent to DIObject_PrincipalAuthenticator.authenticate, but the parameters are oriented to WLE security.

**Arguments**

- **user_name**
  The WLE user name. This parameter is required for TOBJ_NOAUTH, TOBJ_SYSAUTH, and TOBJ_APPAUTH authentication levels.

- **client_name**
  The name of the WLE client application. This parameter is required for TOBJ_NOAUTH, TOBJ_SYSAUTH, and TOBJ_APPAUTH authentication levels.

- **system_password**
  A password for the WLE client application. This parameter is required for TOBJ_SYSAUTH and TOBJ_APPAUTH authentication levels.

- **user_password**
  The user password (default WLE authentication service). This parameter is required for the TOBJ_APPAUTH authentication level.
### user_data
Application-specific data (custom authentication service). This parameter is required for the TOBJ_APPAUTH authentication level.

**Note:** If `user_name`, `client_name`, or `system_password` is NULL or empty, or exceeds 30 characters, the subsequent `authenticate` method invocation raises the `CORBA::BAD_PARAM` exception.

**Note:** If the authorization level is `TOBJ_APPAUTH`, only one of `user_password` or `user_data` may be supplied.

### exceptioninfo
An optional input argument that allows the client application to get additional exception data if an error occurs. For the ActiveX client application, all exception data is returned in the OLE Automation Error Object.

## Return Values
The following table describes the valid return values.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security::AuthenticationStatus::SecAuthSuccess</td>
<td>The authentication succeeded.</td>
</tr>
<tr>
<td>Security::AuthenticationStatus::SecAuthFailure</td>
<td>The authentication failed, or the client application was already authenticated and did not call one of the following methods:</td>
</tr>
<tr>
<td></td>
<td><code>Tobj::PrincipalAuthenticator::logoff</code></td>
</tr>
<tr>
<td></td>
<td><code>Tobj_Bootstrap::destroy_current</code></td>
</tr>
</tbody>
</table>
**DI_Tobj_PrincipalAuthenticator.logoff**

**Synopsis**
Discards the current security context associated with the WLE client application.

**MIDL Mapping**
HRESULT logoff([in, out, optional] VARIANT* exceptionInfo);

**Automation Mapping**
Sub logoff([exceptionInfo])

**Description**
This call discards the context associated with the WLE client application, but does not close the network connections to the WLE domain. `logoff` also invalidates the current credentials. After logging off, calls using existing object references fail if the authentication type is not `TOBJ_NOAUTH`.

If the client application is currently authenticated to a WLE domain, calling `Tobj_Bootstrap.destroy_current()` calls `logoff` implicitly.

**Argument**
exceptioninfo
An optional input argument that allows the client application to get additional exception data if an error occurs. For the ActiveX client applications, all exception data is returned in the OLE Automation Error Object.

**Return Values**
None.

**DISecurityLevel2_Credentials**

The `DISecurityLevel2_Credentials` object is a BEA implementation of the CORBA Security model. In this release of the WLE software, the `get_attributes()` and `is_valid()` methods are supported.
**DISecurityLevel2_Credentials.get_attributes**

**Synopsis**

Gets the attribute list attached to the credentials.

**MIDL Mapping**

```c
HRESULT get_attributes(
    [in] VARIANT attributes,
    [in,out,optional] VARIANT* exceptionInfo,
    [out,retval] VARIANT* returnValue);
```

**Automation Mapping**

Function `get_attributes(attributes, [exceptionInfo])`

**Arguments**

- `attributes`  
  The set of security attributes (privilege attribute types) whose values are desired. If this list is empty, all attributes are returned.

- `exceptionInfo`  
  An optional input argument that allows the client application to get additional exception data if an error occurs. For the ActiveX client application, all exception data is returned in the OLE Automation Error Object.

**Description**

This method returns the attribute list attached to the credentials of the client application. In the list of attribute types, you are required to include only the type value(s) for the attributes you want returned in the `AttributeList`. Attributes are not currently returned based on attribute family or identities. In most cases, this is the same result you would get if you called `DISecurityLevel2.Current::get_attributes()`, since there is only one valid set of credentials in the client application at any instance in time. The results could be different if the credentials are not currently in use.

**Return Values**

A variant containing an array of `DISecurity_SecAttribute` objects.
DISecurityLevel2_Credentials.is_valid

**Synopsis**
Checks the status of credentials.

**MIDL Mapping**
```c
HRESULT is_valid(
    [out] IDispatch** expiry_time,
    [in,out,optional] VARIANT* exceptionInfo,
    [out,retval] VARIANT_BOOL* returnValue
)
```

**Automation Mapping**
```c
Function is_valid(expiry_time As Object,
                  [exceptionInfo]) As Boolean
```

**Description**
This method returns **TRUE** if the credentials used are active at the time; that is, you did not call `DITobj_PrincipalAuthenticator.logoff` or `destroy_current`. If this method is called after `DITobj_PrincipalAuthenticator.logoff()`, **FALSE** is returned. If this method is called after `destroy_current()`, the `CORBA::BAD_INV_ORDER` exception is raised.

**Return Values**
The output `expiry_time` as a `DITimeBase.UtcT` object set to max.
Programming Example

This section contains the portions of an ActiveX client application that implement the following:

♦ Using the Bootstrap object to obtain the SecurityCurrent object
♦ Getting the Principal Authenticator object from the SecurityCurrent object
♦ Using TUXEDO-style authentication
♦ Logging off the WLE domain

Listing 13-1  ActiveX Client Application That Uses TUXEDO-Style Authentication

```vbscript
Set objSecurityCurrent = objBootstrap.CreateObject("TObj.SecurityCurrent")
Set objPrincipalAuthenticator = objSecurityCurrent.principal_authenticator

AuthorityType = objPrincipalAuthenticator.get_auth_type
If AuthorityType = TOBJ_APPAUTH Then logonStatus =
    objPrincipalAuthenticator.Logon(
        UserName,_,
        ClientName,_,
        SystemPassword,_,
        UserPassword
        User Data)
End If

objPrincipalAuthenticator.logoff()
```
Index

A
administration steps
  certificate-based authentication 1-17
  link-level encryption 1-4
  the SSL protocol 1-12
  username/password authentication 1-8
authentication
  certificate-based 1-14
  username/password 1-4
authorized users
  defining 4-8
AUTHSRV
  code example 4-3
  configuring 4-2
  described 1-5
  use with username/password
    authentication 1-8

B
building
  Secure Simpapp sample application 6-11
  Security sample application 6-2

C
certificate authorities
  defined 2-8
  obtaining a digital certificate for 2-8
  certificate-based authentication
    administration steps 1-17
  C++ code example 5-18
  configuration illustrated 1-19
  configuring 3-3
  defining JNDI environment
    properties 7-5
  described 1-14
  development process 1-17
  how it works 1-15
  illustrated 1-14
  Java code example 5-20
  programming steps 1-17
  requirements 1-16
  sample UBBCONFIG file 4-7
  writing the client application 5-18
cipher suites
  supported by the WLE product 3-7
compiling
  client applications
    Secure Simpapp sample
      application 6-19
    Security sample
      application 6-9
  server applications
    Secure Simpapp sample
      application 6-19
    Security sample
      application 6-9
concepts
  AUTHSRV 1-5
  certificate-based authentication 1-14
  digital certificates 1-9

Using Security I-1
Using Security

link-level encryption 1-3
SSL protocol 1-9
username/password authentication 1-4
configuring
a port for SSL communications 3-2
certificate-based authentication 3-3
host matching 3-3
setting session renegotiation 3-8
setting the encryption strength 3-5
the SSL protocol
  CORBA C++ ORB 3-2
  CORBA Java ORB 3-2
  IIOP Listener/Handler 3-2
CORBA C++ client applications
starting
  Secure Simpapp sample
  application 6-19
  Security sample
  application 6-10
CORBA C++ ORB
defining a port for SSL communications 3-2
enabling certificate-based authentication 3-3
enabling host matching 3-4
setting the encryption strength 3-5
CORBA Java client applications
starting
  Secure Simpapp sample
  application 6-19
  Security sample
  application 6-10
CORBA Java ORB
defining a port for SSL communications 3-2
enabling certificate-based authentication 3-3
enabling host matching 3-4
example of configuring the SSL protocol 3-10
CORBA module
described 10-2
CORBA Module IDL 10-2
corbaloc URL Address format
described 5-5
corbalocs URL Address format
described 5-5
Credentials object
described 9-7

D
Data types
security module 10-4
deployment descriptor
  specifying security roles 7-4
development process
certificate-based authentication 1-17
for security in EJBs 7-2
the SSL protocol 1-12
username/password authentication 1-7
digital certificates
certificate-based authentication 1-14
for principals 2-6
obtaining 2-5
publishing in LDAP 2-5
SSL protocol 1-9
troubleshooting 8-9
directory location of source files
  Secure Simpapp sample
  application 6-12, 6-14
  Security sample application 6-3

E
EJBs
assigning security roles to methods 7-3
code example
certificate-based authentication 7-10
username/password
Using Security

I

IOP Listener/Handler
- configuring session renegotiation 3-8
- defining a port for SSL communications 3-2
- enabling certificate-based authentication 3-3
- enabling host matching 3-4
- SEC_PRINCIPAL_LOCATION parameter 3-9
- SEC_PRINCIPAL_NAME parameter 3-9
- SEC_PRINCIPAL_PASSVAR parameter 3-9
- setting security parameters 3-8
- setting the encryption strength 3-5
- use with certificate-based authentication 1-14
- use with the SSL protocol 1-9
- invocation_options_required method
  - C++ code example 5-22
  - described 5-21
  - Java code example 5-23

ISL command
- configuring session renegotiation 3-8
- enabling certificate-based authentication 3-3
- enabling host matching 3-4
- example 3-9
- setting the encryption strength 3-5
- specifying a port for SSL communications 3-2

ISL parameter
- Security sample application 6-10

J

JAVA_HOME parameter
- Secure Simpapp sample application 6-5, 6-16

F

file protections
- Secure Simpapp sample application 6-16
- Security sample application 6-7

H

host matching
- enabling 3-3
- values 3-4

I

IIOP Listener/Handler
- configuring session renegotiation 3-8
- defining a port for SSL communications 3-2
- enabling certificate-based authentication 3-3
- enabling host matching 3-4
- SEC_PRINCIPAL_LOCATION parameter 3-9
- SEC_PRINCIPAL_NAME parameter 3-9
- SEC_PRINCIPAL_PASSVAR parameter 3-9
- setting security parameters 3-8
- setting the encryption strength 3-5
- use with certificate-based authentication 1-14
- use with the SSL protocol 1-9
- invocation_options_required method
  - C++ code example 5-22
  - described 5-21
  - Java code example 5-23

ISL command
- configuring session renegotiation 3-8
- enabling certificate-based authentication 3-3
- enabling host matching 3-4
- example 3-9
- setting the encryption strength 3-5
- specifying a port for SSL communications 3-2

ISL parameter
- Security sample application 6-10

J

JAVA_HOME parameter
- Secure Simpapp sample application 6-5, 6-16
Using Security

JDKDIR parameter
  setenv file 6-6, 6-17
JNDI environment properties
  for certificate-based
    authentication 7-5
  for username/password
    authentication 7-5
WLEContext.INITIAL_CONTEXT_FACTORY 7-5
WLEContext.PROVIDER_URL 7-6
WLEContext.SECURITY_AUTHENTICATION 7-7
JNDI factory
  use in authentication 7-2
  joint client/server applications
  using the SSL protocol 3-2

LDAP directory service
  directory structure 2-3
  search filter file 2-3
  use with certificate-based
    authentication 1-16
  use with the SSL protocol 1-11
  use with WLE security 2-2
LDAP Search Filter file
  modifying 2-3
  stanzas used by SSL protocol 2-4
  stanzas used for certificate-based
    authentication 2-4
  tags 2-4
link-level encryption
  administration steps 1-4
  described 1-3
  development process 1-4
  how it works 1-3
  illustrated 1-3
loading the UBBCONFIG file
  Security sample application 6-8

makefile
  Secure Simpapp sample application 6-15
  Security sample application 6-9

OMG IDL
  CORBA module 10-2
  Security Level 2 module 10-7
  Security module 10-4
  SecurityLevel 1 module 10-6
  TimeBase module 10-2
  Tobj module 10-7

Peer Rules file
  described 2-10
  elements 2-10
  example 2-10
  syntax 2-11
PrincipalAuthenticator object
  certificate-based authentication 9-6
  described 9-5
  using in client applications 5-6
  WLE extensions 9-6
private keys
  example 2-7
  for principals 2-6
  format 2-6
  location 2-6
protocols
  link-level encryption 1-3
  SSL 1-9

runme command
  description 6-19
  files generated by 6-20, 6-21
SEC_PRINCIPAL_LOCATION parameter defined 3-9
SEC_PRINCIPAL_NAME parameter defined 3-9
SEC_PRINCIPAL_PASSVAR parameter defined 3-9
Secure Simpapp sample application
building 6-11
changing protection on files 6-16
compiling the Java client application 6-19
compiling the Java server application 6-19
description 5-16
development process 5-17
illustrated 5-16
loading the UBBCONFIG file 6-19
locations of files 6-12
required environment variables 6-5, 6-16
runme command 6-19
setting up the work directory 6-12
source files 6-12, 6-14
starting the Java client application 6-24
starting the Java server application 6-24
using the client applications 6-24
Security Level 2 module described 10-7
Security module described 10-4
SECURITY parameter defining in UBBCONFIG file 4-4
setting for username/password authentication 1-8
values for 4-4
security roles assigning to EJB methods 7-3
defining in deployment descriptor 7-4
syntax rules 7-4

Security sample application
building 6-9
changing protection on files 6-7
compiling client applications 6-9
compiling server application 6-9
description 5-6
illustrated 5-7
initializing the database 6-8
ISL parameter 6-10
loading the UBBCONFIG file 6-8
location of files 5-7
makefile 6-9
PrincipalAuthenticator object 5-6
SecurityCurrent object 5-6
setenv file 6-8
setting up the work directory 6-3
source files 6-3
tmloadcf command 6-8
SecurityCurrent object described 9-9
using in client applications 5-6
SecurityLevel 1 module described 10-6
source files
Secure Simpapp sample application 6-14
Security sample application 6-3
SSL parameters
SEC_PRINCIPAL_LOCATION 1-12
SEC_PRINCIPAL_NAME 1-12
SEC_PRINCIPAL_PASSVAR 1-12
SSL protocol administration steps 1-12
configuration illustrated 1-13
described 1-9
development process 1-12
how it works 1-9
illustrated 1-9
requirements 1-11
support documentation xiv
technical xiv

T
TimeBase module
   described 10-2
TimeBase Module IDL 10-2
tmboot command
   Secure Simpapp sample application 6-24
   Security sample application 6-9
tmloadcf command
   Secure Simpapp sample application 6-19
   Security sample application 6-8
Tobj module
   described 10-7
tpgpadd command
   defining security groups 1-8, 4-9
tpusradd command
   defining users for security 1-8, 4-9
troubleshooting
   bootstrapping problems 8-6
   callback objects 8-9
   certificate-based authentication
   problems 8-5
   configuration problems 8-8
   digital certificates 8-9
   IIOP Listener/Handler startup problems
   8-7
   ORB initialization problems 8-3
   tracing 8-1
   Ulog file 8-1
   username/password
   authentication problems 8-4
Trusted Certificate Authority file
   described 2-8
   example 2-8
TUXCONFIG parameter
   setenv file 6-6, 6-17
TUXDIR parameter
   Secure Simpapp sample
   application 6-5, 6-16

U
UBBCONFIG file
   configuring the authentication
   server 4-2
   defining a security level 4-3
   defining link-level encryption 1-4
   defining security parameters for
   the IIOP Listener/Handler 3-9
   example of certificate-based
   authentication 4-7
   example of username/password
   authentication 4-5
   link-level encryption 1-4
   Secure Simpapp sample application 6-19
   Security sample application 6-4
   setting parameters for security 4-2
   setting the encryption 4-5
   username/password authentication 1-8
URL Address formats
   certificate-based authentication 1-15
   corbaloc 5-2, 5-5
   corbalocs 5-2, 5-5
   described 5-2
   Host and Port 5-4
   syntax 5-3
   the SSL protocol 1-10
   username/password authentication 1-22
   using with EJBs 7-6
username/password authentication
   administration steps 1-7
   application password 1-5
   C++ example
      SecurityLevel2
       PrincipalAuthenticator 5-9
      Tobj PrincipalAuthenticator 5-12
      defining JNDI environment
      properties 7-5
      defining users and groups 1-8
      described 1-4
      development process 1-7
how it works 1-5
illustrated 1-5
interfaces explained 5-8
Java example
  SecurityLevel2
    PrincipalAuthenticator
      5-10
  Tobj PrincipalAuthenticator 5-14
programming steps 1-7
sample UBBCONFIG file 4-5
system authentication 1-5
writing the client application 5-7

W
WLE domain
  adding security to 5-6
WLE Security model
  accessing objects 9-3
  administrative control 9-3
  authenticating principals 9-2
  components 9-4
    Credentials object 9-7
    PrincipalAuthenticator object 9-5
    SecurityCurrent object 9-9
described 9-2
WLE Security Pack
  described 2-2
  use with certificate-based authentication
    1-16
  use with SSL protocol 1-11
WLEContext.
  INITIAL_CONTEXT_FACTORY
    property 7-5
WLEContext.
  PROVIDER_URL property 7-6
WLEContext.
  SECURITY_AUTHENTICATION
    property 7-7