
SilverStream eXtend Application Server
Facilities Guide

Version 4.0

July 2002

SilverStream[®]

Copyright ©2002 SilverStream Software, Inc. All rights reserved.

SilverStream software products are copyrighted and all rights are reserved by SilverStream Software, Inc.

SilverStream and jBroker are registered trademarks and SilverStream eXtend is a trademark of SilverStream Software, Inc.

Title to the Software and its documentation, and patents, copyrights and all other property rights applicable thereto, shall at all times remain solely and exclusively with SilverStream and its licensors, and you shall not take any action inconsistent with such title. The Software is protected by copyright laws and international treaty provisions. You shall not remove any copyright notices or other proprietary notices from the Software or its documentation, and you must reproduce such notices on all copies or extracts of the Software or its documentation. You do not acquire any rights of ownership in the Software.

Jakarta-Regexp Copyright ©1999 The Apache Software Foundation. All rights reserved. Ant Copyright ©1999 The Apache Software Foundation. All rights reserved. Xalan Copyright ©1999 The Apache Software Foundation. All rights reserved. Xerces Copyright ©1999-2000 The Apache Software Foundation. All rights reserved. Jakarta-Regexp, Ant, Xalan and Xerces software is licensed by The Apache Software Foundation and redistribution and use of Jakarta-Regexp, Ant, Xalan and Xerces in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notices, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear. 4. The names "The Jakarta Project", "Jakarta-Regexp", "Xerces", "Xalan", "Ant" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org <<mailto:apache@apache.org>>. 5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of The Apache Software Foundation. THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright ©1996-2000 Autonomy, Inc.

Copyright ©2000 Brett McLaughlin & Jason Hunter. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the disclaimer that follows these conditions in the documentation and/or other materials provided with the distribution. 3. The name "JDOM" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact license@jdom.org <<mailto:license@jdom.org>>. 4. Products derived from this software may not be called "JDOM", nor may "JDOM" appear in their name, without prior written permission from the JDOM Project Management (pm@jdom.org <<mailto:pm@jdom.org>>). THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Sun Microsystems, Inc. Sun, Sun Microsystems, the Sun Logo Sun, the Sun logo, Sun Microsystems, JavaBeans, Enterprise JavaBeans, JavaServer Pages, Java Naming and Directory Interface, JDK, JDBC, Java, HotJava, HotJava Views, Visual Java, Solaris, NEO, Joe, Netra, NFS, ONC, ONC+, OpenWindows, PC-NFS, SNM, SunNet Manager, Solaris sunburst design, Solstice, SunCore, SolarNet, SunWeb, Sun Workstation, The Network Is The Computer, ToolTalk, Ultra, Ultracomputing, Ultraserver, Where The Network Is Going, SunWorkShop, XView, Java WorkShop, the Java Coffee Cup logo, Visual Java, and NetBeans are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

The software licensed hereby contains modified Sun NetBeans software which is available, together with the Sun Public License pursuant to which such NetBeans software may be used, at <http://www.silverstream.com/workbenchdownload>. Terms of this Agreement that differ from the terms of the Sun Public License are offered solely by SilverStream.

SilverStream eXtend Workbench software contains Sun NetBeans software that has been modified by SilverStream. The source code for such software may be found at <http://www.silverstream.com/workbenchdownload> together with the Sun Public License that governs the use of such modified software. The Original Code is NetBeans. The Initial Developer of the Original Code is Sun Microsystems, Inc. Portions Copyright 1997-2000 Sun Microsystems, Inc. All Rights Reserved. The Contributor to Covered Code is SilverStream Software, Inc.

IBM Jikes™ and Bean Scripting Framework (BSF) Copyright ©2001, International Business Machines Corporation and others. All Rights Reserved. This software contains code in executable form obtained pursuant to, and the use of which is subject to, the IBM Public License, a copy of which may be obtained at <http://oss.software.ibm.com/developerworks/opensource/license10.html>. Source code for Jikes™ is available at <http://oss.software.ibm.com/developerworks/opensource/jikes/>. Source code for BSF is available at <http://oss.software.ibm.com/developerworks/projects/bsf>.

Copyright ©2001 Extreme! Lab, Indiana University License. <http://www.extreme.indiana.edu>. Permission is hereby granted, free of charge, to any person obtaining a copy of the Indiana University software and associated Indiana University documentation files (the "IU Software"), to deal in the IU Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the IU Software, and to permit persons to whom the IU Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the IU Software. THE IU SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE IU SOFTWARE OR THE USE OR OTHER DEALINGS IN THE IU SOFTWARE.

Graph Layout Toolkit and Graph Editor Toolkit ©1992 - 2001 Tom Sawyer Software, Oakland, California, All Rights Reserved.

This Software is derived in part from the SSLava™ Toolkit, which is Copyright ©1996-1998 by Phaos Technology Corporation. All Rights Reserved.

SearchServer © 2000 Hummingbird Communications, Inc.

Copyright © 1994-2002 W3C® (Massachusetts Institute of Technology, Institut National de Recherche Informatique et en Automatique, Keio University), all Rights Reserved. <http://www.w3.org/consortium/legal>. This W3C work (including software, documents, or other related items) is being provided by the copyright holders under the following license. By obtaining, using and/or copying this work, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions: Permission to use, copy, modify, and distribute this software and its documentation, with or without modification, for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the software and documentation or portions thereof, including modifications, that you make: 1. The full text of this NOTICE in a location viewable to users of the redistributed or derivative work. 2. Any pre-existing intellectual property disclaimers, notices, or terms and conditions. If none exist, a short notice of the following form (hypertext is preferred, text is permitted) should be used within the body of any redistributed or derivative code: "Copyright © [date-of-software] World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved. <http://www.w3.org/Consortium/Legal/>" 3. Notice of any changes or modifications to the W3C files, including the date changes were made. (We recommend you provide URIs to the location from which the code is derived.) THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS. COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SOFTWARE OR DOCUMENTATION. The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to the software without specific, written prior permission. Title to copyright in this software and any associated documentation will at all times remain with copyright holders.

Contents

About This Book ix

- Purpose ix
- Audience ix
- Prerequisites ix
- Organization ix
- Related resources x

Chapter 1 JSP Deployment to the File System 1

- Using JSP pages 1
 - About JSP pages and Web applications 1
 - About file system deployment 2
 - Using eXtend Workbench 2
- Setting up your application 3
- Deploying your application to the file system 3
 - Creating the deployment plan 4
 - Deploying the application 4
- Updating your application 5
- Putting your application into production 7

Chapter 2 J2EE Archive Deployment 9

- Deploying J2EE application client archives 9
 - Packaging application clients 10
 - Writing an application client deployment plan 10
 - Running SilverCmd DeployCAR 11
- Deploying EJB archives 12
 - Packaging EJB components 12
 - Writing an EJB deployment plan 12
 - Tips for completing the EJB deployment plan 15
 - Deploying an EJB JAR 37
 - Restructuring EJB JAR files after deployment 39
- Deploying Web archives (WARs) 40
 - Packaging a Web application 40
 - Writing a WAR deployment plan 40
 - Deploying a WAR file 42
- Deploying resource adapter archives (RARs) 42
 - Packaging a RAR 42
 - Writing a RAR deployment plan 43
 - Deploying a RAR file 44

Deploying enterprise archives (EARs)	44
Packaging an enterprise application	45
Writing an EAR deployment plan	45
Running SilverCmd DeployEAR	46
Specifying classpath JARs on the server	47
Accessing user-supplied JARs	47
Overriding server-supplied JARs	47

Chapter 3 Deployment Plan DTDs 49

About the deployment plan DTDs	49
Client JAR deployment plan DTD	50
EJB JAR deployment plan DTD	55
WAR deployment plan DTD	78
RAR deployment plan DTD	85
EAR deployment plan DTD	88

Chapter 4 SilverCmd Reference 95

Command locator	95
About SilverCmd	98
Running SilverCmd	98
Specifying values in input files and deployment plans	101
Alphabetical list of commands	102
AddCP	102
AddDatabase	105
Build	109
BuildWAR	111
ClearDefaultURL	112
ClearLog	113
ComGen	114
ConvertEJB	115
CreatePackage	116
Delete	117
DeployCAR	119
DeployEAR	120
DeployEAR12	123
DeployEJB	125
DeployEJB11	127
DeployRAR	129
DeployWAR	130
ExportSource	132
GetConsole	134
GetDefaultURL	135

ImportClass	136
ImportMedia	138
ImportPage	140
ImportSource	141
ListCP	143
ModifyCP	144
ModifyTableList	145
Prefs	147
PrintLog	149
Publish	150
PublishFromFile	151
PublishToFile	153
QueryCP	155
RebuildJAR	156
RemoveCP	157
RemoveDatabase	158
ServerState	159
SetDefaultURL	161
SetSecurity	162
SetUserGroupInfo	163
SourceControl	172
Undeploy	176
ValidateEAR	177
ValidateEJB	178
ValidateEJB11	179

Chapter 5 SilverJ2EEClient and SilverJRunner 181

About SilverJ2EEClient	181
SilverJ2EEClient features	181
About SilverJRunner	182
Installing SilverJRunner and SilverJ2EEClient	183
Providing the SilverJRunner install page	183
Going to the SilverJRunner install page	183
Installing on Windows	184
Installing on UNIX or Linux	185
Installing from your SilverStream product CD	185
Starting SilverJRunner and SilverJ2EEClient	186
Running on Windows	186
Running on UNIX or Linux	189
Displaying a console window	192
Displaying your own splash screen	194

How SilverJRunner updates itself	194
SilverJ2EEClient in the development environment	194
Using startup options	195
Using - options	195
Using + options	199
Passing application arguments	200
Specifying the arguments to pass	200
Accessing the arguments from a form	201
Accessing the arguments from a client	201
Supporting access to secured EJBs	202

Chapter 6 Server Implementation Notes 203

J2EE containers	203
Web container	203
EJB containers	206
Client container	221
Session-level failover	222
EJB support for session-level failover	222
Web application support for session-level failover	223
Application client support for session-level failover	224
CORBA support	225
XML support	225
SilverStream XML support	225
Resources for learning about XML	228
Internationalization support	228
Database support	228
Client-side support	229

About This Book

Purpose

This book describes core facilities (tools, utilities, services) provided with the SilverStream eXtend Application Server.

Audience

This book is for anyone who manages the SilverStream server or develops applications for it.

Prerequisites

This book assumes some familiarity with J2EE (Java 2 Enterprise Edition).

Organization

Here's a summary of the topics you'll find in this book:

Chapter	Description
Chapter 1, "JSP Deployment to the File System"	How to deploy a JSP application to the file system for a quick develop/test/refine cycle
Chapter 2, "J2EE Archive Deployment"	How to deploy J2EE-compatible archive files to a SilverStream server
Chapter 3, "Deployment Plan DTDs"	Reference documentation about the DTDs (XML document type definitions) used when deploying J2EE archives to a SilverStream server
Chapter 4, "SilverCmd Reference"	Reference documentation about the SilverStream command-line tool SilverCmd and its utilities

Chapter	Description
Chapter 5, “SilverJ2EEClient and SilverJRunner”	All about the facilities provided with the SilverStream server to host Java-based clients: SilverJ2EEClient (for J2EE applications) and SilverJRunner (for classic SilverStream applications)
Chapter 6, “Server Implementation Notes”	Details about the SilverStream server’s implementation of various features, including its J2EE containers and its support for CORBA, XML, and internationalization

Related resources

The SilverStream eXtend Application Server also provides several facilities that come with their own documentation. These include:

- jBroker ORB
- jBroker MQ
- jBroker TM
- IBM XML4J

1 JSP Deployment to the File System

This chapter describes how to use SilverStream's JSP File System (JSP/FS) deployment to speed your JSP development. With JSP/FS you can deploy your Web application to the file system and instantly see the result of changes you make in the file system without redeployment. The chapter contains these sections:

- Using JSP pages
- Setting up your application
- Deploying your application to the file system
- Updating your application
- Putting your application into production

Using JSP pages

This section provides a quick introduction to JSP pages and their deployment.

About JSP pages and Web applications

The JavaServer Pages technology is an important part of Sun's J2EE platform, which recommends using JSP pages (with supporting servlets) to provide the core of the user interface of your application. JSP pages are typically used in Web-based J2EE applications (**Web applications**). A Web application includes JSP pages, servlets, JavaBeans, utility classes, images, and so on that are packaged in an archive called a Web application archive (WAR) file. These applications are accessed by browser clients.

The SilverStream eXtend Application Server provides full support for JSP pages.



For more information on JSP pages and how to write them, see the eXtend Workbench help.

About file system deployment

To make your Web application available to users, you **deploy** it to an application server. Users access your application by specifying appropriate URLs in their browser.

But when developing, testing, and refining your application, you want fast turnaround—you want to make a change to your JSP pages and immediately see the result in a browser without having to redeploy the application. Using JSP/FS, you can.

NOTE JSP/FS is meant only to enhance development. Don't use it with your production applications. For more information, see “Putting your application into production” on page 7.

After deploying your application to the file system, you can create or change a JSP page, refresh the browser, and immediately see the change. There is no need to redeploy.

Similarly, you can change any static resource in your application and see the change immediately. (A **static resource** is any file that the server serves as is. Static resources include HTML files, images files, and style sheets.)

The rest of this chapter describes how to deploy a Web application to the file system for an accelerated development/test/refine loop.

NOTE Deploying JSP pages to the file system in a SilverStream cluster is not supported.

Using eXtend Workbench

If you are using SilverStream eXtend Workbench to do your development and deployment, you don't have to perform the procedures described in the rest of this chapter. You only need to specify Enable Rapid Deployment in your project's deployment settings when you deploy your WAR (or EAR containing a WAR). Workbench will manage everything for you, including:

- Updating the deployment plan with the proper specification for JSP/FS
- Deploying the project to the file system
- Managing changes you make in your project's files (including creating and removing a RELOAD file as necessary, as described in “Making changes to Java source files” on page 6)

You continue to work with your project's files as usual. Changes will be reflected immediately in your deployed application.

When you are ready to do your production deployment, simply deselect Enable Rapid Deployment in your project's deployment settings and redeploy.



For more information, see the Workbench help.

Setting up your application

You begin development of your Web application as usual. First you set up your development area using a directory structure that conforms to the format required for Web applications.

When you are ready to test and refine your application, you package your application in a WAR file.

What must be in the WAR file You needn't have created your JSP pages yet, though you can have. To create your WAR file for file-system deployment, all you need is:

- Any **supporting files** needed by your Web application, such as
 - Compiled servlet and utility classes, either as CLASS files (in WEB-INF/classes) or as JAR files (in WEB-INF/lib)
 - Tag libraries (typically in WEB-INF/tlds)
- A **deployment descriptor** for the application. The file must be named **web.xml** and must be in the WEB-INF directory

The web.xml file must follow the format specified by the Sun J2EE Web application DTD called **web-app_2_2.dtd** located in the Resources/DTDCatalog subdirectory of your SilverStream installation. The Java Servlet Specification Version 2.2 provides complete documentation on each tag. You can find this document on the Sun Java Web site at <http://java.sun.com/j2ee/docs.html>.

Creating the WAR file You can create the WAR file using the archive tool of your choice.

For example, you can use SilverStream eXtend Workbench, Sun's jar utility (located in the bin directory of your SilverStream installation), or the SilverStream BuildWAR SilverCmd (but don't use the -d option to deploy the WAR).

Deploying your application to the file system

Once you have the WAR file, you are ready to deploy it to the file system. To deploy a J2EE archive (such as a WAR file) for use with the SilverStream server, you create a **deployment plan**, which is an XML file that specifies SilverStream-specific information about how to manage the Web application and how it should be deployed.

Creating the deployment plan

A deployment plan for a WAR file must follow the format specified by **deploy_war.dtd** (or the section on WAR files in **deploy_ear.dtd**), located in the Resources/DTDCatalog directory of your SilverStream installation.



For more information, see J2EE Archive Deployment.

The deployment plan for file-system deployment is the same as for standard database deployment, with one exception: it includes a line within the <warJar> section that specifies that you want the server to deploy the application to the file system. To deploy to the file system, include this line in your deployment plan:

```
<deployToFilesystem type="Boolean">true</deployToFilesystem>
```

Example Here is JSPSampleDeplPlan.xml, a deployment plan that specifies that the application is to be deployed to the file system:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE warJarOptions PUBLIC "-//SilverStream Software, Inc.//DTD J2EE
WAR Deployment Plan//EN" "deploy_war.dtd">
<?AgMetaXML 1.0?><warJarOptions isObject="true">
  <warJar isObject="true">
    <warJarName>C:\WorkbenchProjects\JSPSample\JSPSample.war</warJarName>
    <isEnabled type="Boolean">True</isEnabled>
    <deployToFilesystem type="Boolean">true</deployToFilesystem>
    <sessionTimeout type="String">5</sessionTimeout>
    <urls type="StringArray">
      <el>JSPSample</el>
    </urls>
    <deployedObject type="String">JSPSample</deployedObject>
  </warJar>
</warJarOptions>
```

Deploying the application

Now run the DeployWAR SilverCmd to deploy your application (you can also use DeployEAR, as long as it includes your WAR file).

What happens The application is deployed to the file system as follows: the server expands the WAR file in the directory /webapps/DBname/URL in the SilverStream installation directory, where:

- *DBname* is the name of the database containing the application deployed to the file system
- *URL* is the URL specified in the deployment plan for the application (if you have specified more than one, the first one is used)

Example Here is the structure of the sample application whose deployment plan is shown above (entries in bold are directories):

```

JSPSample
  jsps (directory containing the JSP pages)
  WEB-INF
    web.xml (deployment descriptor)
    classes (directory containing the supporting CLASS files)
    tlds (directory containing the TLD files)

```

Here is the deployment command line:

```
SilverCmd DeployWAR localhost JSPSampleDB JSPSample.war -f JSPSampleDeplPlan.xml -o
```

After deploying this application to the file system, the WAR file is expanded in /webapps in the SilverStream installation directory as follows:

```

SilverStreamInstallDir
  webapps
    JSPSampleDB (directory whose name is the deployment database)
    JSPSample (directory whose name is the URL in the deployment plan)
      com (directory where JSPs will be compiled -- see below)
        jsps
        WEB-INF
          web.xml
          classes
          tlds

```

The subdirectories under webapps/JSPSampleDB/JSPSample make up the deployment area, and you can work with the application there.

Updating your application

Once you have deployed your application to the file system, you can make changes in the deployment area and immediately see the result of the changes.

What you can change You can:

- Change an existing JSP page
- Add a new JSP page
- Delete an existing JSP page
- Change, add, or delete a static resource

To see the result To see the result of your change, simply save the file(s) in the deployment area, go to your browser, and specify the appropriate URL.

For example, to see the result of changing `sample.jsp`, you would specify an URL similar to this:

```
http://localhost/JSPSampleDB/JSPSample/jsp/sample.jsp
```

What the server does The SilverStream server checks to see whether the JSP page has ever been accessed. If not, it compiles it and displays the result in the browser. The server stores the resulting JAVA and CLASS files in the `com/sssw/gen/jsp` directory in the deployment area.

Similarly, the server checks to see whether the JSP page (the JSP source file) has been updated since it was last accessed. If so, the file is recompiled and redisplayed.

Also, the server serves updated static resources as needed.

If there is an error If there are any JSP compilation errors for the requested page, the server generates an HTML page describing the error and returns it with a 500 status code to the client.

If there is an error compiling a JSP page other than the one requested, the server cannot intercept the failure. Depending on where in the compilation process the error occurred, you might see an error message in the generated JSP page with a link that describes the error. You can prevent the server from trying to compile JSP pages that are known to fail by using the `<excludedJSPs>` tag in the deployment plan.

Making changes to Java source files Sometimes in the course of development you will need to update utility classes, servlets, JAR files, and so on in the application and will want to refresh your deployment area. Here is how to do that:

1. Make the changes in your development area.
2. Compile the classes and refresh any JARs that need updating.
3. Copy the updated CLASS and JAR files to the appropriate locations in the deployment area.
4. Create a file named RELOAD (all uppercase) in the root of the deployment area.

In the sample deployment, you would create RELOAD in `webapps/JSPSampleDB/JSPSample`.

TIP To easily create the file, open a DOS command prompt, `cd` to the directory, type **copy con RELOAD**, then press **Return, Ctrl+Z, Return**.

5. Access your application in the browser.

The server will automatically reload the application, getting all the updated files, and delete the RELOAD file.

You can now continue to make changes to JSP pages in the updated deployment area.

Redeploying the application to the file system After deploying to the file system, you shouldn't redeploy your application to the file system with DeployWAR. Work with your application as described above. When you are ready to put your application into production, follow the procedure in "Putting your application into production" on page 7.

If you do redeploy your application to the file system with DeployWAR, the server notices that the deployment directory already exists in webapps. It renames that directory to *DeploymentDir.1*, then redeploys the application to the file system. So the current application continues to be *DeploymentDir*, with the previous version archived as *DeploymentDir.1*. If you do the deployment again, *DeploymentDir* is renamed *DeploymentDir.2*, and so on. The current application is always *DeploymentDir*, and the version before that is archived as the highest numbered *DeploymentDir.n*. The server is responsive only to changes in *DeploymentDir*.

In the sample shown, if the application is deployed to the file system a second time, webapps/JSPSampleDB/JSPSample is renamed to JSPSample.1, and the current application is deployed to JSPSample.

Putting your application into production

JSP/FS is meant only for use in the development phase of your application. When you have completed the application and are ready to put it into production, do the following:

1. Bring your development area up to date with the changes you made in the deployment area.
2. Rebuild your WAR or EAR file.
3. Delete or comment out the **<deployToFilesystem>** line in the deployment plan.
4. Do a full deployment using DeployWAR or DeployEAR.

2

J2EE Archive Deployment

This chapter describes how to deploy J2EE-compatible archive files to a SilverStream server. It covers these topics:

- Deploying J2EE application client archives
- Deploying EJB archives
- Deploying Web archives (WARs)
- Deploying resource adapter archives (RARs)
- Deploying enterprise archives (EARs)
- Specifying classpath JARs on the server

Using eXtend Workbench This chapter focuses on deployment using the basic server facilities. To learn about developing, packaging, and deploying with eXtend Workbench, see the Workbench help.

Deploying J2EE application client archives

You use application client archive files to contain J2EE application clients (client classes and supporting files) for deployment to a J2EE server.



To learn about J2EE application clients and how to write them, see the eXtend Workbench help.

➤ **To deploy an application client archive file to your SilverStream server:**

1. Package the Java classes that implement the application client in a J2EE-compatible **JAR file**.
2. Write a SilverStream **application client deployment plan**. (Exception: you don't need to do this if your application client has no external references.)
3. Run **SilverCmd DeployCAR**.
(Deploying from eXtend Workbench runs this command for you.)

Packaging application clients

The application client archive must comply with Sun’s J2EE specification. For more information, see:

- *Java 2 Platform Enterprise Edition Specification, v1.2*, Chapter 9, “Application Clients”
- *J2EE Blueprints*



These publications are available from the Sun Java Web site at:

<http://java.sun.com/j2ee/docs.html>

Writing an application client deployment plan

The application client deployment plan includes information the SilverStream server needs to provide the appropriate runtime environment for an application client. You supply this deployment plan as an option to the SilverCmd DeployCAR command.

Application clients that reference EJBs, environment entries, or other external resource references (such as databases) must have a deployment plan. Application clients without external references do not require one.

Requirements of the plan

Your application client deployment plan must be an XML file based on the client JAR deployment plan DTD that SilverStream provides:

```
deploy_car.dtd
```

Your plan must:

- Include a corresponding DOCTYPE statement
- Comply with the structure documented by this DTD



For details on writing a deployment plan XML file that meets these requirements, see Chapter 3, “Deployment Plan DTDs”:

To learn about	See
The DOCTYPE statement and structure for your application client deployment plan	“Client JAR deployment plan DTD” on page 50

To learn about	See
A sample application client deployment plan you can follow	“About the deployment plan DTDs” on page 49
Where to find the DTD file	“About the deployment plan DTDs” on page 49

Editing the plan

To create and edit your application client deployment plan, you can use either of the following:

- An XML editor or text editor of your choice
- The Deployment Plan Editor of eXtend Workbench

Overview of the plan

The application client deployment plan can specify:

- Environment entries
- Bean references
- Resource references

Running SilverCmd DeployCAR

Once you have the archive file and (if appropriate) the deployment plan, you can deploy the client application to a SilverStream server using SilverCmd DeployCAR. This command deploys the archive file to the Jars subdirectory of the EJB Jars & Media directory on the SilverStream server. Once the deployed object is on the server, any application client can access it using SilverJ2EEClient.

All application components are automatically available for client requests; you do not need to restart the server.



For more information on:

- Using SilverCmd DeployCAR, see “DeployCAR” on page 119
- Using SilverJ2EEClient, see Chapter 5, “SilverJ2EEClient and SilverJRunner”

Deploying EJB archives

You use EJB archive files to contain Enterprise JavaBeans (session beans, entity beans, message-driven beans, and supporting files) for deployment to a J2EE server.



To learn about Enterprise JavaBeans and how to write them, see the eXtend Workbench help.

➤ To deploy an EJB archive file to your SilverStream server:

1. Package the Java classes that implement the beans, the appropriate interfaces, the primary key classes (if necessary), and any other utility classes in a J2EE-compatible **JAR file**.
2. Write a SilverStream **EJB deployment plan**.
3. Run **SilverCmd DeployEJB** (to deploy to the 2.0 container) or **SilverCmd DeployEJB11** (to deploy to the 1.1 container).

(Deploying from eXtend Workbench runs the appropriate command for you.)

Packaging EJB components

The EJB archive must comply with Sun's *Enterprise JavaBeans Specification, Version 1.1*.



This publication is available from the Sun Java Web site at:

<http://java.sun.com/j2ee/docs.html>

Writing an EJB deployment plan

The EJB deployment plan includes information that the SilverStream server's EJB container needs to provide the appropriate runtime environment for the beans. The deployment plan:


- Resolves any environment entry, resource, or bean reference dependencies
- Provides information about role mapping for the target operational environment
- Provides the container with other SilverStream-specific runtime information

Requirements of the plan

Your EJB deployment plan must be an XML file based on the deployment plan DTD that is appropriate for the version of your archive. Use `deploy-ejb_2_0.dtd` for JARs that contain EJB2.0 beans; use `deploy-ejb_1_1.dtd` for JARs that contain EJB1.1 beans.

The deployment plan must:

- Include a corresponding DOCTYPE statement
- Comply with the structure documented by this DTD

 For details on writing a deployment plan XML file that meets these requirements, see Chapter 3, “Deployment Plan DTDs”:

To learn about	See
The DOCTYPE statement and structure for your EJB deployment plan	“EJB JAR deployment plan DTD” on page 55
A sample EJB deployment plan you can follow	“About the deployment plan DTDs” on page 49
Where to find the DTD file	“About the deployment plan DTDs” on page 49

Editing the plan

To create and edit your EJB deployment plan, you can use either of the following:

- An XML editor or text editor of your choice
- The Deployment Plan Editor of eXtend Workbench

Overview of the plan

The following table summarizes the information you can include in the EJB deployment plan:

For this component	The plan specifies
EJB JAR	<ul style="list-style-type: none">• Enabled or disabled• Mapping of security roles to principals in the target environment (when security roles are specified in the deployment descriptor)• Confidentiality and integrity cipher suites
Each EJB (regardless of type)	<ul style="list-style-type: none">• JNDI name• Resource references• EJB references• Environment entries• Instance pooling specifications• Security configuration
Container-managed entity beans	<ul style="list-style-type: none">• The data source for each bean and for persistent fields• How finder methods are mapped• Data loading type (lazy or eager)• Relationships (for 2.0 beans only)

Tips for completing the EJB deployment plan

This section describes some tips and examples for completing the EJB deployment plan. It includes these sections:

- Supporting autoincrement
- Mapping CMP entity beans to a table
- Mapping persistent fields
- Using TRANSACTION_READ_COMMITTED isolation levels
- Mapping relationships
- Mapping a primary key
- Mapping for message-driven beans
- IOR configurations for EJB security

Supporting autoincrement

SilverStream supports autoincrement through the deployment plan's **autoInc** element. When autoInc is marked for a cmp-field, the database column that it maps to must support autoincrement, as follows:

Database	Requirement
Oracle	You must provide a sequence name through autoIncSequenceName element
Sybase, Microsoft, and IBM DB2	The column has to be an identity column
Informix	The column has to be a SERIAL type
IBM Cloudscape	The column type has to be autoincrement type

Mapping CMP entity beans to a table

You map a CMP entity bean to one (and only one) table. You use the beanPersistenceInfo element of the deployment plan to map a bean to a table.

Suppose that you had a deployment descriptor with an entry for an entity bean, like this:

```
<entity>
  <ejb-name>BEAN_NAME</ejb-name>
  . . .
</entity>
```

The corresponding entry in the SilverStream deployment plan would be in the BeanPersistenceInfo node. It would look something like this:

```
<BeanPersistenceInfo>
  <beanName>BEAN_NAME</beanName>
  <dataSourceName>DATABASE_OR_POOL_NAME</dataSourceName>
  <sqlHandler>SQL_HANDLER</sqlHandler>
  <isolationLevel>ISOLATION_LEVEL</isolationLevel>
  <table>
    <name>DB_TABLE</name>
    . . .
  </table>
</BeanPersistenceInfo>
```

Notice that:

- The beanName element of the BeanPersistenceInfo node must match the ejb-name element in the deployment descriptor (in this example, BEAN_NAME).
- The dataSourceName element is the name of the database or connection pool where the table (in this example, DB_TABLE) resides.
- The sqlHandler element of the deployment plan can be one of the following:

For this database or class	Valid sqlHandler element values
Oracle	Oracle
IBM Cloudscape	Cloudscape
Sybase Adaptive Server Anywhere	AdaptiveServerAnywhere
Sybase Adaptive Server Enterprise	AdaptiveServerEnterprise
Informix	Informix
Microsoft SQL Server	MicrosoftSQLServer
IBM DB2	DB2
Class that implements com.sssw.shr.ejb2.api.AgiEJBSqlHandler	The fully qualified name of the class

NOTE The `sqlHandler` element values are not case sensitive.

- The `isolationLevel` element identifies the isolation level to be used. SilverStream supports `TRANSACTION_READ_COMMITTED` or `TRANSACTION_SERIALIZABLE`. When `TRANSACTION_READ_COMMITTED` isolation level is used, the container performs checks to make sure no values are changed by other threads during a commit. If they are changed, the container throws a concurrency violation exception.

If you have multiple beans (mapped to different tables) using different isolation levels you need multiple pools—except for Oracle databases (see just below).

Use `TRANSACTION_SERIALIZABLE` with caution, as the likelihood of deadlock increases.

Exceptions for Oracle For Oracle databases, all `READ SQL` statements for `TRANSACTION_SERIALIZABLE` are appended with `FOR UPDATE` to explicitly place a write lock on the row.

Mapping persistent fields

The persistent fields (the `cmp-field` elements) listed in the deployment descriptor must be mapped to a database column in the database table in the deployment plan. This enables the container to persist the fields appropriately. Suppose your deployment descriptor looked like this:

```
<entity>
  <ejb-name>BEAN_NAME</ejb-name>
  <cmp-field>field1</cmp-field>
  <cmp-field>field2</cmp-field>
  <cmp-field>field3</cmp-field>
  . . .
</entity>
```

The `cmp-field` elements in the deployment descriptor would have a corresponding elements in the table node of the deployment plan—for example:

```
<table>
  <name>DB_TABLE</name>
  <field>
    <cmpFieldName>field1</cmpFieldName>
    <columnName>COLUMN1</columnName>
  </field>
  <field>
    <cmpFieldName>field2</cmpFieldName>
    <columnName>COLUMN2</columnName>
  </field>
  . . .
</table>
```

Notes about the deployment plan:

- The `cmpFieldNames` in the deployment plan must match the `cmp-field` elements in the deployment descriptor.
- The `columnName` is an actual column name in the database. This field is case-sensitive and must exactly match the database column name.

Using TRANSACTION_READ_COMMITTED isolation levels

When you specify `TRANSACTION_READ_COMMITTED` for the isolation level, you may be able to specify some columns as `deltaType` columns in the deployment plan to improve performance.

The `deltaType` element specifies whether the column data is used to maintain a count or a total, such as total count or total sales. If you use this element, you should set up a database constraint to guard against overflow/underflow. For example, suppose you have a column that contains the quantity left for a specific product; many transactions may try to increase or decrease this quantity. Instead of generating SQL like this:

```
UPDATE table SET quantity = ? WHERE col1=old_col1_value AND col2 =
old_col2_value
```

and

```
quantity = old_quantity
```

SilverStream will generate SQL like this:

```
UPDATE table SET quantity = quantity + ? WHERE col1=old_col1_value
AND col2 = old_col2_value
```

For this example, you would apply a constraint that does not allow the quantity to drop below zero, and the column should not allow `NULL`.

Using the `deltaType` element appropriately can greatly reduce the possibility of a `CONCURRENCY_VIOLATION` exception and improve performance. To use the `deltaType` element:

- The column must represent a quantity—because the data type can only be a number type like an `int`, a `short`, a `float`, a `double`, a `BigDecimal`, and so on.
- The field cannot be null.

Mapping relationships

A relationship can exist between two entity beans with CMP. This relationship is expressed via the **relationships** node of the deployment descriptor. The relationship is mapped to real tables via the **relationsList** node of the deployment plan.

Based on the contents of the deployment descriptor and the deployment plan, the container will be able to generate the appropriate SQL code to traverse the tables and get and set the appropriate values. To generate the SQL, the container needs to know the answers to these questions:

- Does a relationship exist?
- What beans are involved in the relationship?
- What is the multiplicity of the relationship?
- What is the direction of the relationship (bidirectional or unidirectional)? If it is unidirectional, which bean is the one that can access the other bean?
- How do you navigate the relationship?
- What are the foreign key/primary key relationships?
- Does the relationship support cascade delete?

The container gets the all of this information (except the foreign key mapping) from the deployment descriptor relationships node. The relationships node contains the elements shown here:

```
<relationships>
  <ejb-relation>
    <ejb-relationship-role>
      <multiplicity></multiplicity>
      <relationship-role-source>
        <ejb-name></ejb-name>
      </relationship-role-source>
      <cmr-field>
        <cmr-field-name></cmr-field-name>
        <cmr-field-type></cmr-field-type>
      </cmr-field>
    </ejb-relationship-role>
  <ejb-relationship-role>
    <multiplicity></multiplicity>
    <relationship-role-source>
      <ejb-name></ejb-name>
    </relationship-role-source>
    <cmr-field>
      <cmr-field-name></cmr-field-name>
      <cmr-field-type></cmr-field-type>
    </cmr-field>
  </ejb-relation>
</relationships>
```

```
        </ejb-relationship-role>
    </ejb-relation>
    . . .
</relationships>
```

Note that:

- For each relationship there are exactly two `ejb-relationship-role` nodes.
- The `multiplicity` element determines whether the relationship is one-to-one, one-to-Many, many-to-one, or many-to-many.
- For each `cmr-field` element in the deployment descriptor, there will be a corresponding set or get method in the bean class. So the `cmr-field` element (or lack of) determines the direction (unidirectional or bidirectional).
- The `<ejb-relation-name>` element is optional (according to the EJB specification).
- For a many-to-many relationship, a `linkTable` element (not shown here) is required. SilverStream allows the `linkTable` element to be used in a one-to-many relationship but does not recommend it.

Once you have the deployment descriptor, you'll be able to create a new deployment plan or complete an existing one.

How to express a one-to-one bidirectional relationship

This example illustrates how you would express a one-to-one bidirectional relationship for the `CustomerEJB` (which maps to the `CUSTOMER` table) and the `AddressEJB` (which maps to the `ADDRESS` table). The primary keys are `CustomerID` and `AddressID` respectively.

The deployment descriptor would look like this:

```
<ejb-relation>
  <ejb-relationship-role>
    <multiplicity>One</multiplicity>
    <relationship-role-source>
      <ejb-name>CustomerEJB</ejb-name>
    </relationship-role-source>
    <cmr-field>
      <cmr-field-name>address</cmr-field-name>
    </cmr-field>
  </ejb-relationship-role>
  <ejb-relationship-role>
    <multiplicity>One</multiplicity>
    <relationship-role-source>
      <ejb-name>AddressEJB</ejb-name>
    </relationship-role-source>
    <cmr-field>
      <cmr-field-name>customer</cmr-field-name>
```

```

        </cmr-field>
    </ejb-relationship-role>
</ejb-relation>

```

Some things to note about the deployment descriptor:

- The multiplicity element in both `ejb-relationship-roles` is `One` to indicate the one-to-one relationship.
- Both `ejb-relationship-role` elements list `cmr-field-names`. This indicates that the relationship is bidirectional. The `cmr-field-names` do not actually represent columns or foreign keys in the related tables. You don't know the actual names of the target columns until deployment; the `cmr-field-names` are just entries that represent the direction of the relationship.

Suppose you are ready to deploy the EJB JAR and now need to create the deployment plan that would represent the `CustomerEJB` and `AddressEJB` EJBs. You might be mapping the `CustomerEJB` and `AddressEJB` to a target database where the `CUSTOMER` table contained a foreign key to the `ADDRESS` table; the foreign key field name is `ADDRESS_ID`. The relation node of your deployment plan would look like this:

```

<relation>
    <relationRole>
        <beanName>CustomerEJB</beanName>
        <cmrFieldName>address</cmrFieldName>
        <columnNames>
            <el>ADDRESS_ID</el>
        </columnNames>
    </relationRole>
    <relationRole>
        <beanName>AddressEJB</beanName>
        <cmrFieldName>customer</cmrFieldName>
    </relationRole>
</relation>

```

Some things to note about the deployment plan:

- The `beanName` in the deployment plan must exactly match the `ejb-name` element of the deployment descriptor.
- The `cmrFieldName` of the deployment plan must exactly match the `cmr-field-name` of the deployment descriptor.
- The deployment plan's `columnName` element is the column name of the foreign key in the table (`CUSTOMER`) to which it is mapped. This means that the container will construct SQL so that it can navigate from the customer to the associated address and vice versa.
- The `cmrFieldName` of the `AddressEJB` is not mapped to a column name. This is because the `ADDRESS` table does not (and should not) contain a foreign key to the `CUSTOMER` table.

If your database should be set up so that the ADDRESS table contained a foreign key CUSTOMER_ID, the deployment plan would look like this:

```
<relation>
  <relationRole>
    <beanName>CustomerEJB</beanName>
    <cmrFieldName>address</cmrFieldName>
  </relationRole>
  <relationRole>
    <beanName>AddressEJB</beanName>
    <cmrFieldName>customer</cmrFieldName>
    <columnNames>
      <el>CUSTOMER_ID</el>
    </columnNames>
  </relationRole>
</relation>
```

How to express a one-to-one unidirectional relationship

This example illustrates how you would express a one-to-one unidirectional relationship for CustomerEJB (which maps to the CUSTOMER table) and AddressEJB (which maps to the ADDRESS table). The primary keys are CustomerID and AddressID respectively.

The deployment descriptor would look like this:

```
<ejb-relation>
  <ejb-relationship-role>
    <multiplicity>One</multiplicity>
    <relationship-role-source>
      <ejb-name>CustomerEJB</ejb-name>
    </relationship-role-source>
    <cmr-field>
      <cmr-field-name>address</cmr-field-name>
    </cmr-field>
  </ejb-relationship-role>
  <ejb-relationship-role>
    <multiplicity>One</multiplicity>
    <relationship-role-source>
      <ejb-name>AddressEJB</ejb-name>
    </relationship-role-source>
  </ejb-relationship-role>
</ejb-relation>
```


Some things to note about the deployment descriptor:

- The multiplicity element in both `ejb-relationship-roles` is `One` to indicate the one-to-one relationship.
- The `CustomerEJB` includes a `cmr-field-name` element (in this case, `address`), but the `AddressEJB` does not. This indicates that the relationship is unidirectional—you can get to the `AddressEJB` from the `CustomerEJB` but not vice versa. (The `CustomerEJB` will include `get` and `set` methods for the `ADDRESS` table.)
- The `cmr-field-name` `address` does not actually represent a column in the `AddressEJB`. When you create the deployment plan, you'll have to map the `address` `cmr-field` to the actual foreign key column.

This deployment plan shows how to map the `cmr-field-name` when the `CUSTOMER` table includes the foreign key `ADDRESS_ID`:

```
<relation>
  <relationRole>
    <beanName>CustomerEJB</beanName>
    <cmrFieldName>address</cmrFieldName>
    <columnNames>
      <el>ADDRESS_ID</el>
    </columnNames>
  </relationRole>
  <relationRole>
    <beanName>AddressEJB</beanName>
  </relationRole>
</relation>
```

Some things to note about the deployment plan:

- The `beanName` in the deployment plan must exactly match the `ejb-name` element of the deployment descriptor.
- The `cmrFieldName` of the deployment plan must exactly match the `cmr-field-name` of the deployment descriptor (this also means that the deployment plan should not have a `cmrFieldName` when the deployment descriptor does not have a `cmr-field-name`).
- The deployment plan's `columnName` element is the column name of the foreign key in the table (`CUSTOMER`) to which it is mapped. This means that the container will construct SQL so that it can navigate from the customer to the associated address.
- The `AddressEJB` does not have a `cmrFieldName` or `columnNames` element, because the `ADDRESS` table does not (and should not) contain a foreign key to the `CUSTOMER` table.

Suppose that your existing database used a different structure and the ADDRESS table had the foreign key CUSTOMER_ID. The deployment plan would look like this:

```
<relation>
  <relationRole>
    <beanName>CustomerEJB</beanName>
    <cmrFieldName>address</cmrFieldName>
  </relationRole>
  <relationRole>
    <beanName>AddressEJB</beanName>
    <columnNames>
      <el>CUSTOMER_ID</el>
    </columnNames>
  </relationRole>
</relation>
```

Notes about this deployment plan:

- The relationRole and cmrFieldNames exactly match the entries in the deployment descriptor as noted above—but **this time** the columnName is not included in this node of the relationRole element. This is because the foreign key is not in the CUSTOMER table. (It's in the ADDRESS table.)
- The relationship is still unidirectional, because only one cmrFieldName element is present.
- You specify the foreign key in the table in which it belongs. The foreign key has nothing to do with the direction of the relationship.

Expressing a one-to-many bidirectional relationship

This example uses the OrderEJB (which maps to the ORDER table) and the LineItemEJB (which maps to the LINEITEM table). For each OrderEJB there can be many LineItems. You can navigate from the ORDER table to the LINEITEM table and back.

The deployment descriptor looks like this:

```
<ejb-relation>
  <ejb-relationship-role>
    <multiplicity>One</multiplicity>
    <relationship-role-source>
      <ejb-name>OrderEJB</ejb-name>
    </relationship-role-source>
    <cmr-field>
      <cmr-field-name>lineItems</cmr-field-name>
      <cmr-field-type>java.util.Collection</cmr-field-type>
    </cmr-field>
  </ejb-relationship-role>
</ejb-relationship-role>
```

```

    <multiplicity>Many</multiplicity>
    <relationship-role-source>
      <ejb-name>LineItemEJB</ejb-name>
    </relationship-role-source>
    <cmr-field>
      <cmr-field-name>order</cmr-field-name>
    </cmr-field>
  </ejb-relationship-role>
</ejb-relation>

```

Notes about the deployment descriptor:

- The multiplicity element for the OrderEJB is One and the multiplicity element for the LineItemEJB is Many—one OrderEJB can have many related LineItemEJBs.
- The OrderEJB includes a cmr-field-name element (in this case, lineitems), and the LineItemEJB also has a cmr-field-name element (in this case order). This indicates that the relationship is bidirectional.
- The data type of the lineitems cmr-field-name element includes a data type specification (a java.util.Collection) because more than one item can be returned.

Suppose that you are writing a deployment descriptor for the situation where:

- The OrderEJB bean maps to the ORDERS table (whose primary key is ORDERID)
- The LineItemEJB bean maps to the LINEITEM table (whose primary key is LINEITEMID)
- The LINEITEM table has a foreign key ORDER_ID

In this case the deployment plan would look like this:

```

<relation>
  <relationRole>
    <beanName>OrderEJB</beanName>
    <cmrFieldName>lineItems</cmrFieldName>
  </relationRole>
  <relationRole>
    <beanName>LineItemEJB</beanName>
    <cmrFieldName>order</cmrFieldName>
    <columnNames>
      <el>ORDER_ID</el>
    </columnNames>
  </relationRole>
</relation>

```

Note the following:

- As always, the beanName has to be the same as the ejb-name of the deployment descriptor, and the cmrFieldName has to be the same as cmr-field-name of the deployment descriptor.
- The LineItemEJB contains a columnName element that maps to the foreign key.
- In a one-to-one relationship, the location of the foreign key may be in either table. In a one-to-many relationship, the foreign key always resides on the Many side. The location of the foreign key does not determine the direction of the relationship.

Expressing a one-to-many unidirectional relationship

This example illustrates how to express a One-to-Many unidirectional relationship. This example uses the ProductEJB (which maps to the PRODUCT table) and the LineItemEJB (which maps to the LINEITEM table). For each ProductEJB there can be many LineItemEJBs. You can navigate from the LINEITEM table to the PRODUCT table but not back.

The deployment descriptor looks like this:

```
<ejb-relation>
  <ejb-relationship-role>
    <multiplicity>One</multiplicity>
    <relationship-role-source>
      <ejb-name>ProductEJB</ejb-name>
    </relationship-role-source>
  </ejb-relationship-role>
  <ejb-relationship-role>
    <multiplicity>Many</multiplicity>
    <relationship-role-source>
      <ejb-name>LineItemEJB</ejb-name>
    </relationship-role-source>
    <cmr-field>
      <cmr-field-name>product</cmr-field-name>
    </cmr-field>
  </ejb-relationship-role>
</ejb-relation>
```

- The multiplicity element for the ProductEJB is One and for the LineItemEJB is Many— one ProductEJB can have many related LineItemEJBs.
- The LineItemEJB includes a cmr-field-name element (in this case, products), but the ProductEJB does not have one. This indicates that the relationship goes in a single direction from LineItem to Product.

Suppose that you are writing a deployment plan for the deployment descriptor above where:

- The ProductEJB bean maps to the PRODUCT table (whose primary key is PRODUCTID)
- The LineItemEJB bean maps to the LINEITEM table (whose primary key is LINEITEMID)
- The LINEITEM table has a foreign key PRODUCT_ID

In this case the deployment plan would look like this:

```
<relation>
  <relationRole>
    <beanName>ProductEJB</beanName>
  </relationRole>
  <relationRole>
    <beanName>LineItemEJB</beanName>
    <cmrFieldName>product</cmrFieldName>
    <columnNames>
      <el>PRODUCT_ID</el>
    </columnNames>
  </relationRole>
</relation>
```

Note the following:

- As always, the beanName must be the same as the ejb-name of the deployment descriptor, and the cmrFieldName must be the same as the cmr-field-name of the deployment descriptor.
- The LineItemEJB contains a columnNames element that maps to the foreign key.

Using a link table

It is possible to use a link table. For example, the link table might be PROD_LINEITEM, which has the PRODUCT_ID and LINEITEM_ID, mapped to PRODUCTID and LINEITEMID respectively. In that case the deployment plan would look like this:

```
<relation>
  <linkTable>PROD_LINEITEM</linkTable>
  <relationRole>
    <beanName>ProductEJB</beanName>
    <columnNames>
      <el>PRODUCT_ID</el>
    </columnNames>
  </relationRole>
  <relationRole>
    <beanName>LineItemEJB</beanName>
    <cmrFieldName>product</cmrFieldName>
    <columnNames>
```

```
        <el>LINEITEM_ID</el>
    </columnNames>
</relationRole>
</relation>
```

How to express a many-to-many unidirectional relationship

This example illustrates how you would express a many-to-many unidirectional relationship for the OrderEJB (which maps to the ORDER table) and the ProductEJB (which maps to the PRODUCT table). The primary keys are OrderID and ProductID respectively. Many-to-many relationships always use a linkTable.

The deployment descriptor would look like this:

```
<ejb-relation>
  <ejb-relationship-role>
    <multiplicity>Many</multiplicity>
    <relationship-role-source>
      <ejb-name>OrderEJB</ejb-name>
    </relationship-role-source>
    <cmr-field>
      <cmr-field-name>products</cmr-field-name>
      <cmr-field-type>java.util.Collection</cmr-field-type>
    </cmr-field>
  </ejb-relationship-role>
  <ejb-relationship-role>
    <multiplicity>Many</multiplicity>
    <relationship-role-source>
      <ejb-name>ProductEJB</ejb-name>
    </relationship-role-source>
  </ejb-relationship-role>
</ejb-relation>
```

Note on the deployment descriptor:

- The multiplicity element for both ejb-relationship-roles is Many.

Suppose that you had to write a deployment plan where:

- The ProductEJB bean maps to the PRODUCT table (whose primary key is PRODUCTID)
- The OrderEJB bean maps to the ORDERS table (whose primary key is ORDERID)
- The link table is PROD_ORDER. The PROD_ORDER table's primary key is PRODUCT_ID and ORDER_ID (which are also foreign keys to the PRODUCT and ORDER tables). Your deployment plan would look like this:

```
<relation>
  <linkTable>PROD_ORDER</linkTable>
  <relationRole>
    <beanName>OrderEJB</beanName>
```

```

        <cmrFieldName>products</cmrFieldName>
        <columnNames>
            <el>ORDER_ID</el>
        </columnNames>
    </relationRole>
    <relationRole>
        <beanName>ProductEJB</beanName>
        <columnNames>
            <el>PRODUCT_ID</el>
        </columnNames>
    </relationRole>
</relation>

```

Notes on the deployment plan:

- As always, the beanName must be the same as the ejb-name of the deployment descriptor, and the cmrFieldName must be the same as the cmr-field-name of the deployment descriptor.
- The relation node would include the linkTable element that specifies the actual name of the database table.
- The relation roles for both OrderEJB and ProductEJB include foreign key mappings (via the columnNames element) to the linkTable.

General restrictions on relationship mapping

Here are some things to keep in mind:

- You can use a single column as both a cmp-field and a cmr-field; but when the single column is both a cmp-field and a cmr-field, the cmp-field should be Read-only. Calling a set method on the cmp-field results in an Exception in this case.
- You cannot use the same database column for more than one cmp-field or cmr-field.
- The way to change a relationship is through its cmr-field (not the cmp-field).

Mapping a primary key

You specify the primary key for the entity bean using the primkey-field and/or the prim-key-class elements. (The prim-key-class element is required, but the primkey-field is optional.) For single-field primary keys, the primkey-field has to be one of the cmp-field.

If <primkey-field> is missing, it may be due to multifold primary keys or unknown primary keys. If it is multifold key, <prim-key-class> specifies the class (which should have all fields public), and the name must correspond to the field names of the entity bean class.

If the `primkey-field` element is missing and the `prim-key-class` element is `java.lang.Object`, this is the unknown primary key class (see the EJB 2.0 specification section 10.8.3). In this case the EJB container generates a unique key. The deployment plan supports this through the `primaryKey` element.

First you must specify a `primaryKeyClass` element for the `primaryKey`. SilverStream supports `java.lang.String`, `java.lang.Integer`, or `java.lang.Long` for `primaryKeyClass` in this case. If `primaryKeyClass` is `java.lang.String`, the `autoInc` element is ignored, and the column has to be able to store a 32-byte length of `String` data. If `primaryKeyClass` is `java.lang.Integer` or `java.lang.Long`, `autoInc` is required—so that an autoincrement column can be used to generate unique primary keys. In general, performance is better with autoincrement columns. The following shows what a primary key class entry in the deployment plan might look like:

```
<primaryKey>
  <primaryKeyClass>java.lang.Integer</primaryKeyClass>
  <columnName>AUTO_INC_COL</columnName>
  </autoInc>
</primaryKey>
```

Mapping for message-driven beans

Message-driven beans use a JMS server to transmit and consume messages. For the container to locate the JMS server, you must specify at least the `destinationName` element and the `ConnectionFactoryName` elements—for example:

```
<message>
  <destinationName>
    corbaname:iiop:JMSServer:3506#queue/queueName
  </destinationName>
  <connectionFactoryName>
    corbaname:iiop:JMSServer:3506#queue/xaConnectionFactory
  </connectionFactoryName>
</message>
```

NOTE The specified `ConnectionFactory` must support global transactions.

IOR configurations for EJB security

This section describes the information you must supply in the deployment plan to support secure invocations of the EJB. You supply the information via the `iorSecurityConfig` element of the deployment plan. The `iorSecurityConfig` node is part of the entity and session elements.

Interoperable Object References

Every object that is remotely accessible using CORBA is referred to via an Interoperable Object Reference or IOR. The IOR is a remote reference to an object; it can be stored (in the CORBA naming service, for example) and subsequently converted to a stub that can be used to call the remote object. So the IOR must contain **all** information the client ORB needs to construct a stub for the remote object and to issue remote method calls on the stub. The information includes:

- **One or more addresses** (host IP address and TCP port number) at which the remote object can be called. The server ORB will be listening at this address.
- **The object identifier** assigned to the remote object by the server ORB when the object was created. The ORB uses this to find the object.
- **The object's type** (that is, the list of remote interfaces the object implements). The client uses this to determine what kind of stub to create.
- **Security information** for the object. The client uses this information to determine whether a secure (encrypted) connection should be used for calls to the object, and whether a client certificate, user name and password, or other caller ID and credential information should be passed to the object on each call. This is the information specified by the <iorSecurityConfig> element at deployment time (as described in detail just below).
- **Transactional information** for the object. The client uses this to decide whether to propagate two-phase commit transaction information to the object.

When the client looks up a remote CORBA object (such as an EJB) in the naming service (such as via JNDI), what's returned is the IOR for the object. The client calls `PortableRemoteObject.narrow()` to convert the IOR into a stub. When the client attempts to call a method on the stub, the client's ORB uses the information obtained from the IOR to decide what type of connection (encrypted or plain) to create; what server address and port number to connect to, and whether or not to encode and send client identity, credentials, and transaction information on the call. The server in turn verifies that the information supplied matches what the IOR demands and then dispatches the call to the remote object.

Contents of the IOR security configuration

The IOR security configuration as supplied in the `iorSecurityConfig` element has three sections:

Section	What it defines
Transport configuration	Whether an encrypted communications channel is to be used, and if so defines the encryption parameters and certificate information required
Authentication context configuration	What kind of authentication mechanism (such as user name and password) should be supplied on calls to this object, and whether anonymous calls to the object are allowed.
Security attributes context configuration	Whether or not caller identity propagation is supported for this object

Transport configuration

The transport configuration, as supplied in the `transportConfig` element, tells the client whether to use an encrypted communication channel (that is, SSL or TLS) for calls to the object—and if so, which encryption algorithms should be used and whether the client must supply a client certificate for authentication purposes. (This replaces the old `usesSSL` and `cipherSuites` elements in SilverStream's EJB 1.1 deployment plans.)

The contents of the `<transportConfig>` element are four attributes:

Attribute	Description
Integrity	<p>Specifies whether the object supports or requires encryption using an encryption algorithm that at least guarantees message integrity—that can detect message corruption.</p> <ul style="list-style-type: none"> • If set to <code>REQUIRED</code>, the caller must use SSL (or TLS) and will choose an appropriate cipher suite • If set to <code>SUPPORTED</code>, the caller may use SSL (see below for information on how the client makes the decision)

Attribute	Description
Confidentiality	<p>Specifies whether the object supports or requires encryption using an encryption algorithm that at least guarantees message confidentiality—that can prevent eavesdroppers from reading the message.</p> <ul style="list-style-type: none"> • If set to REQUIRED, the caller must use SSL (or TLS) and will choose an appropriate cipher suite • If set to SUPPORTED, the caller may use SSL
establishTrustInClient	<p>Specifies whether the object requires that the client authenticate using a client certificate (x.509).</p> <ul style="list-style-type: none"> • If set to REQUIRED, the client must supply a client certificate when setting up the SSL connection to this object; this also requires use of SSL (or TLS) • If set to SUPPORTED, the client may supply a client certificate
establishTrustInServer	<p>Specifies whether the object's server must be able to authenticate itself to the client. At present the only mechanism for a server to authenticate itself to the client is via the use of SSL (or TLS); so this flag is equivalent to controlling use of SSL.</p> <ul style="list-style-type: none"> • If set to REQUIRED, the client must use SSL (or TLS) on calls to this object • If set to SUPPORTED, the client may use SSL

Note that these flags interact; for example, the client must use SSL (or TLS) if any of the flags are set to **REQUIRED**. Similarly, it is legal to specify the use of SSL (by setting `establishTrustInClient` to **REQUIRED**) but not specify any particular cipher suites (by setting both integrity and confidentiality to none).

Client algorithm for choosing SSL If any of the four `transportConfig` flags is set to **REQUIRED**, the client must use SSL for calls to the object. If none is **REQUIRED** but at least one is **SUPPORTED**, the client must choose whether or not to use SSL. The client makes this decision based on the `-SS_USE_SSL` flag to `SilverJ2EEClient`. If the flag is set to true, the calls will use SSL; otherwise, they will not.

Cipher suites The sets of cipher suites that are used for message integrity and message protection may be explicitly specified in the deployment plan, using the `integrityCipherSuites` and `confidentialityCipherSuites` elements respectively. If supplied, each element is a String array of cipher suite names.

In this situation	This happens
If the <code>integrityCipherSuites</code> element is supplied	A cipher suite from the supplied list will be used for calls to any object that supports or requires integrity but does not support or require confidentiality
If <code>confidentialityCipherSuites</code> element is supplied	A cipher suite from the supplied list will be used for calls to any object that supports or requires confidentiality
If either element is not supplied	The server will supply a default list of cipher suites for that category

Authentication context configuration The `asContext` element describes the authentication information that will be passed from the client to the server as part of each method call on the object. This information is separate from any client certificate that may be passed when the SSL connection is established. The `asContext` element has three subelements.

Subelement	Description
<code>authMethod</code>	The supported authentication method for callers. Possible values are: <ul style="list-style-type: none"> <code>NONE</code>—No caller authentication can be supplied on calls to this object <code>USERNAME_PASSWORD</code>—A user name and password string can be supplied on each call

Subelement	Description
realm	<p>Supports a well-known realm named default. The default realm matches any SilverStream-supported realm.</p> <p>When the realm is specified as default, the user name and password are passed as qualified names, so that the server can distinguish LDAP user names from NT user names.</p>
asContextRequired	<p>A boolean indicating whether or not the caller must supply a user name and password. If true, the authMethod element must not be NONE, and the caller must supply a user name and password on each call to the object.</p>

Security attribute context configuration

The security attribute context configuration is specified in the <sasContext> element via callerPropagation, an attribute indicating whether or not this object supports caller identity propagation. Caller identity propagation is the ability for the client to propagate the caller's identity to the server without supplying any credentials, such as a password. This is only useful if the server trusts the client and the client has already verified the caller's identity—for example, if the caller is another application server owned by the organization that has already verified the client's password. Possible values are:

Value	Means
NONE	Caller propagation is not supported for this object
SUPPORTED	The caller may propagate an identity (subject to the server verifying that the caller is trusted)
REQUIRED	The caller must propagate an identity

The server determines whether the caller is trusted by the use of a list of trusted clients. The list of trusted clients can be set using the SMC.

Suppose that multiple identities are supplied on the call (for example, a client certificate and identity asserted via identity propagation). The identities are determined as follows:

1. If a client certificate was supplied, use it to obtain the caller's identity.
2. If an identity was asserted via caller propagation, use it as the caller's identity.

3. If a user name and password were passed on the call, use them to obtain the caller's identity.
4. Otherwise, treat the caller as Anonymous.

IOR configuration examples

This section includes examples three IOR configurations.

Example 1 This example shows the IOR configuration for an object that:

- Is to be called using SSL with message confidentiality preserved (encryption)
- Supports passing caller identity via either client certificate or user name and password

```
<iorConfig>
  <transportConfig>
    <integrity>NONE</integrity>
    <confidentiality>REQUIRED</confidentiality>
    <establishTrustInClient>SUPPORTED</establishTrustInClient>
    <establishTrustInServer>SUPPORTED</establishTrustInServer>
  </transportConfig>
  <asConfig>
    <authMethod>USERNAME_PASSWORD</authMethod>
    <realm>default</realm>
    <asContextRequired>FALSE</asContextRequired>
  </asConfig>
  <sasConfig>
    <callerPropagation>NONE</callerPropagation>
  </sasConfig>
</iorConfig>
```

Example 2 This example shows the IOR configuration for an object that:

- Is to be called without SSL
- But requires caller authentication using user name and password

```
<iorConfig>
  <transportConfig>
    <integrity>NONE</integrity>
    <confidentiality>NONE</confidentiality>
    <establishTrustInClient>NONE</establishTrustInClient>
    <establishTrustInServer>NONE</establishTrustInServer>
  </transportConfig>
  <asConfig>
    <authMethod>USERNAME_PASSWORD</authMethod>
    <realm>default</realm>
    <asContextRequired>TRUE</asContextRequired>
  </asConfig>
  <sasConfig>
```

```

        <callerPropagation>NONE</callerPropagation>
    </sasConfig>
</iorConfig>

```

Example 3 This example shows the IOR configuration for an object that:

- Is to be called using SSL, with message confidentiality protection
- Passes a client certificate but supports caller propagation for server-to-server calls:

```

<iorConfig>
  <transportConfig>
    <integrity>NONE</integrity>
    <confidentiality>REQUIRED</confidentiality>
    <establishTrustInClient>REQUIRED</establishTrustInClient>
    <establishTrustInServer>NONE</establishTrustInServer>
  </transportConfig>
  <asConfig>
    <authMethod>NONE</authMethod>
    <realm>default</realm>
    <asContextRequired>FALSE</asContextRequired>
  </asConfig>
  <sasConfig>
    <callerPropagation>SUPPORTED</callerPropagation>
  </sasConfig>
</iorConfig>

```

Deploying an EJB JAR

After you package your EJB components in the archive file and create the deployment plan, you are ready to deploy the EJB JAR using the SilverCmd DeployEJB command.

What happens when you deploy an EJB JAR

When you deploy the EJB JAR, the SilverStream server constructs a deployed object and a remote EJB JAR in the SilverStream database. The original EJB JAR is not actually deployed; it just provides the raw materials for the construction of the deployed object and remote JAR.

The deployed object includes the implementation classes for the bean's remote and home interfaces. It is used only by the SilverStream server.

The remote JAR file includes stub or reference classes that you can use to call the server-side implementation classes. For convenience, the remote JAR also includes each bean's home and remote interface and any classes directly referenced by them. For example, it includes any classes referenced as parameters or return values. In general, the remote JAR includes any of the classes a caller needs to use the bean.

All clients should use the remote JAR. This includes SilverStream clients within the same server, forms served to SilverJRunner by the same server, SilverStream clients within another SilverStream server, and standalone Java programs. You need to include this remote JAR in SilverStream forms, pages, and business objects. You need to add this JAR to your path if you are accessing EJBs from an external client.

Restructuring EJB JAR files after deployment

If you deploy an EJB JAR and then later restructure the EJB JAR file (or restructure a utility JAR file referenced by the EJB JAR), note the following:

If you	Then you must
<p>Restructure your EJB JAR by making any of the following changes:</p> <ul style="list-style-type: none"> • By creating a utility JAR and changing the EJB JAR's manifest class-path attribute to refer to the utility JAR • By removing a utility JAR and removing the reference to it in the EJB JAR's manifest class-path attribute • By creating an EJB client JAR and adding an <ejb-client-jar> tag to the EJB JAR's deployment descriptor • By removing an EJB client JAR and removing the <ejb-client-jar> tag from the EJB JAR's deployment descriptor 	<p>Do the following:</p> <ol style="list-style-type: none"> 1. Resave and redeploy the EJB JAR. 2. Resave any objects (such as pages, servlets, or forms) that reference the EJB remote JAR. 3. Resave and redeploy any EARs or WARs that reference the EJB remote JAR in their deployment plans. <p>(Rebuilding the application database will not accomplish this.)</p>
<p>Restructure a utility JAR referenced by the EJB by making any of the following changes:</p> <ul style="list-style-type: none"> • By creating a subutility JAR and changing the utility JAR's manifest class-path attribute to refer to the subutility JAR • By removing a subutility JAR and removing the reference to the subutility JAR in the utility JAR's manifest class-path attribute 	

Deploying Web archives (WARs)

A Web archive file is a logical grouping of static HTML pages, servlets, JSP pages, and other resources (images, sounds, and so on) that comprise a complete application.



To learn about servlets or JSP pages and how to write them, see the eXtend Workbench help.

➤ To deploy a WAR file to your SilverStream server:

1. Package the JSP source and other files for the application (including compiled servlet classes and other supporting Java components, HTML documents, images, and so on) in a J2EE-compatible **WAR file**.
2. Write a SilverStream **WAR deployment plan**.
3. Run **SilverCmd DeployWAR**.

(Deploying from eXtend Workbench runs this command for you.)

Packaging a Web application

The WAR must comply with Sun's *Java Servlet Specification, Version 2.2*.



This publication is available from the Sun Java Web site at:

<http://java.sun.com/j2ee/docs.html>

Writing a WAR deployment plan

The WAR deployment plan includes information about the contents of the WAR file and how it should be managed in the SilverStream server environment. You supply this deployment plan as an option to the SilverCmd DeployWAR command.


Requirements of the plan

Your WAR deployment plan must be an XML file based on the WAR deployment plan DTD that SilverStream provides:

```
deploy_war.dtd
```

Your plan must:

- Include a corresponding DOCTYPE statement
- Comply with the structure documented by this DTD

 For details on writing a deployment plan XML file that meets these requirements, see Chapter 3, “Deployment Plan DTDs”:

To learn about	See
The DOCTYPE statement and structure for your WAR deployment plan	“WAR deployment plan DTD” on page 78
A sample WAR deployment plan you can follow	“About the deployment plan DTDs” on page 49
Where to find the DTD file	“About the deployment plan DTDs” on page 49

Editing the plan

To create and edit your WAR deployment plan, you can use either of the following:

- An XML editor or text editor of your choice
- The Deployment Plan Editor of eXtend Workbench

Overview of the plan

The WAR deployment plan can specify:

- One or more URLs for the WAR file
- Whether the WAR file is enabled or disabled
- Whether you want to deploy the application to the file system for a quick develop/test/refine cycle (see Chapter 1, “JSP Deployment to the File System”)
- Mappings for deployment descriptor values such as environment variables, EJB references, resource references, and role references

Deploying a WAR file

After you package your Web application components in the archive file and create the deployment plan, you are ready to deploy the WAR using the SilverCmd DeployWAR command.


What happens when you deploy a WAR

DeployWAR deploys a WAR file to a SilverStream server. As part of the deployment process, DeployWAR compiles all of the JSP pages in the WAR into Java source files and then compiles these Java sources. It adds the result to the WAR file and uploads the WAR file to the server. (It does **not** include the Java source files in the WAR.)

 For more information on using SilverCmd DeployWAR, see “DeployWAR” on page 130.

Deploying resource adapter archives (RARs)

A resource adapter archive (RAR) contains the software that allows J2EE components to interact with enterprise information systems that reside outside the J2EE server.

 To learn about resource adapters and how to write them, see the eXtend Workbench help.

➤ To deploy a RAR file to your SilverStream server:

1. Obtain the RAR from a vendor.

OR

If you have written your own RAR, package the files in a J2EE-compatible **RAR file**.


2. Write a SilverStream **RAR deployment plan**.

3. Run **SilverCmd DeployRAR**.

(Deploying from eXtend Workbench runs this command for you.)

Packaging a RAR

For information about writing and packaging a RAR, see the J2EE Connector architecture specification.

 This publication is available from the Sun Java Web site at:

<http://jcp.org/jsr/detail/016.jsp>

Writing a RAR deployment plan

The RAR deployment plan includes information about the contents of the RAR file and how it should be managed in the SilverStream server environment. You supply this deployment plan as an option to the SilverCmd DeployRAR command.


Requirements of the plan

Your RAR deployment plan must be an XML file based on the SilverStream RAR deployment plan:

```
deploy-rar_1_1.dtd
```

Your plan must:

- Include a corresponding DOCTYPE statement
- Comply with the structure documented by this DTD

 For details on writing a deployment plan XML file that meets these requirements, see Chapter 3, “Deployment Plan DTDs”:

To learn about	See
The DOCTYPE statement and structure for your RAR deployment plan	“RAR deployment plan DTD” on page 85
Where to find the DTD file	“About the deployment plan DTDs” on page 49

Editing the plan

To create and edit your RAR deployment plan, you can use either of the following:

- An XML editor or text editor of your choice
- The Deployment Plan Editor of eXtend Workbench

Overview of the plan

The RAR deployment plan can specify:

- The name of the resource adapter
- Information about the connection pool for the RAR including initial and maximum pool sizes, user names and passwords, and wait and idle timeout values.

Deploying a RAR file

DeployRAR deploys a RAR file to a SilverStream server. As part of the deployment process, DeployRAR:

- Uploads the RAR to the server
- Creates and configures any connection pools specified in the deployment plan



For more information on using SilverCmd DeployRAR, see “DeployRAR” on page 129.

Deploying enterprise archives (EARs)

An enterprise archive (EAR) represents a J2EE application rather than a single J2EE module. It is a collection of one or more of the J2EE deployable modules described earlier in this chapter (application clients, EJBs, and WARs), packaged in a JAR file with the .EAR file extension.

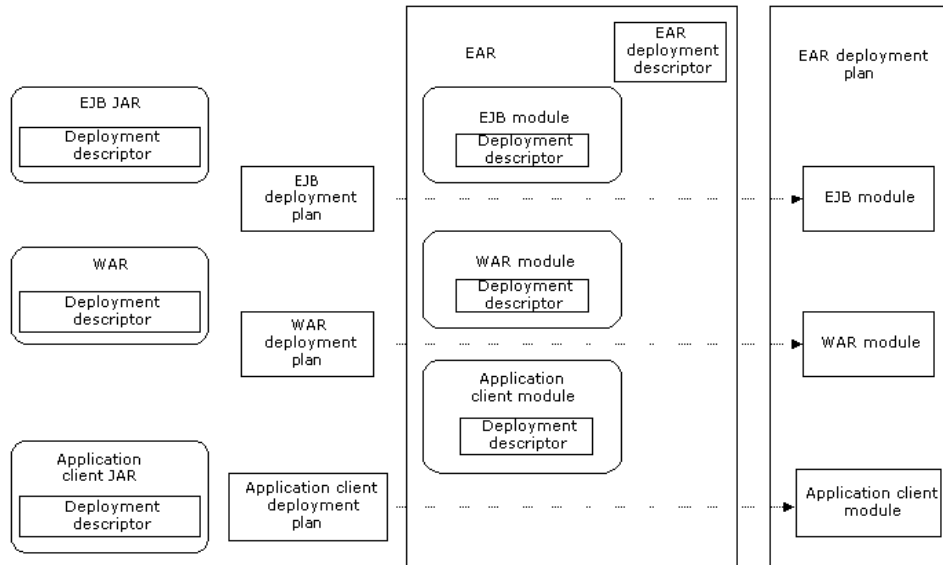
➤ To deploy an EAR file to your SilverStream server:

1. Package the application components (application client JAR files, EJB JAR files, WAR files) in a J2EE-compatible **EAR file**. Note that each component (and the EAR itself) must include a deployment descriptor.
2. Write a SilverStream **EAR deployment plan**.
3. Run **SilverCmd DeployEAR** (to deploy J2EE 1.? EARs) or **SilverCmd DeployEAR12** (to deploy J2EE1.2 EARs)

(Deploying from eXtend Workbench runs the appropriate command for you.)

Packaging an enterprise application

The following diagram shows how the components are combined to create an EAR before deployment:



The EAR must comply with Sun's J2EE specification. For more information, see *Java 2 Platform Enterprise Edition Specification, v1.2*, Chapter 8, "Application Assembly and Deployment".

 This publication is available from the Sun Java Web site at:

<http://java.sun.com/j2ee/docs.html>

Writing an EAR deployment plan

An EAR deployment plan provides a top-level view of the application's contents as well as the details for each J2EE deployable module (application client, EJB, WAR) that will be contained in the EAR. It's really a concatenation of the deployment plans for each of the J2EE modules it contains.


You create the EAR deployment plan by copying the elements from these module deployment plans into it.

Requirements of the plan

Your EAR deployment plan must be an XML file based on the EAR deployment plan that is appropriate for the version of your archive. Use `deploy-ear_1_2.dtd` or `deploy-ear_1_3.dtd`.

Your plan must:

- Include a corresponding DOCTYPE statement
- Comply with the structure documented by this DTD

 For details on writing a deployment plan XML file that meets these requirements, see Chapter 3, “Deployment Plan DTDs”:

To learn about	See
The DOCTYPE statement and structure for your EAR deployment plan	“EAR deployment plan DTD” on page 88
A sample EAR deployment plan you can follow	“About the deployment plan DTDs” on page 49
Where to find the DTD file	“About the deployment plan DTDs” on page 49

Editing the plan


To create and edit your EAR deployment plan, you can use either of the following:

- An XML editor or text editor of your choice
- The Deployment Plan Editor of eXtend Workbench

Running SilverCmd DeployEAR

DeployEAR performs these tasks:

1. Opens the EAR file and extracts all the files to a local temporary directory.
2. For each application client archive, EJB JAR, or WAR, DeployEAR performs the corresponding SilverCmd `DeployXXX` command to deploy that archive to the server.

 For more information on using SilverCmd DeployEAR, see “DeployEAR” on page 120.

Specifying classpath JARs on the server

The SilverStream deployment plans for EJB 2.0, WAR 2.3, and EAR 1.3 support a **classpathJars** element that you can use to adjust the list of JAR files on your application's classpath. This lets you do either of the following:

- **Access user-supplied JARs** that you've copied to the server
- **Override server-supplied JARs** used for XML parsing and transformation



To learn about the syntax for classpathJars, see Chapter 3, "Deployment Plan DTDs".

Accessing user-supplied JARs

You may have one or more commonly used JARs that you'd rather locate directly on the server than deploy in multiple applications. You can copy such JARs to your SilverStream server's **userlib** directory and then list them as needed in an archive's deployment plan via classpathJars and its **userlibJars** subelement.

For example:

```
<classpathJars>
  <userlibJars>
    <el>MyJarA.jar</el>
    <el>MyJarB.jar</el>
  </userlibJars>
</classpathJars>
```

This will enable the deployed archive to find classes in the listed userlib JARs at runtime.

Overriding server-supplied JARs

By default, archives deployed to the SilverStream server have access to the standard J2EE 1.3 XML JARs for parsing and transformation. These are the **crimson** and **xalan** JARs:

- crimson-java_xml_pack_fall_2001.jar
- xalan-java_xml_pack_fall_2001.jar

In some circumstances, you might want to override the default XML JARs and specify others. This involves:

1. Using classpathJars in an archive's deployment plan to specify the **excludeJ2EEXMLJars** subelement with a value of `true`.
This removes the default XML JARs from the application's classpath.

2. Doing one of the following:

- Deploying your substitute XML JARs in the archive.
- Copying those JARs to your SilverStream server's **userlib** directory and listing them in the archive's deployment plan via the **userlibJars** subelement of **classpathJars**.

Note that the userlib directory already contains several versions of common XML JARs you might want to use.

For example:

```
<classpathJars>
  <excludeJ2EEXMLJars>true</excludeJ2EEXMLJars>
  <userlibJars>
    <el>xerces-1_4_4.jar</el>
    <el>xalan-2_3_1.jar</el>
  </userlibJars>
</classpathJars>
```

This will enable the deployed archive to find classes in the listed userlib XML JARs (instead of the default XML JARs) at runtime.

3 Deployment Plan DTDs

This chapter provides reference documentation for the SilverStream deployment plan DTDs. Topics include:

- About the deployment plan DTDs
- Client JAR deployment plan DTD
- EJB JAR deployment plan DTD
- WAR deployment plan DTD
- RAR deployment plan DTD
- EAR deployment plan DTD

Before you begin A basic understanding of XML is recommended for this chapter.

About the deployment plan DTDs


The SilverStream deployment plan DTD files are used when you deploy J2EE archives to the SilverStream eXtend Application Server. These DTDs (XML document type definitions) describe the structure you must follow when writing deployment plans (XML files) for particular kinds of archives (client JARs, EJB JARs, WARs, and EARs).

DTD and sample XML files Here's a summary of the names and locations of these files:

To find	Look here in your SilverStream install
The SilverStream deployment plan DTD files	Resources\DTDCatalog directory
Sample deployment plan XML files that use the DTDs	samples\SilverCmd directory

DTD documentation You can learn about the DTDs by looking at:

- The DTD files themselves (for comments about the elements they define)
- This chapter (for more detailed reference documentation about each element and how to structure them in your XML files)

 For more information on writing and using SilverStream deployment plans, see Chapter 2, “J2EE Archive Deployment”.

Client JAR deployment plan DTD

This section provides reference information about the SilverStream client JAR deployment plan DTD:

- DTD file
- DOCTYPE statement
- Elements

DTD file There are two client JAR deployment plan DTDs:

Server version	DTD file name
3.7.2 and lower	deploy_car.dtd
3.7.3 and higher	deploy-car_1_2.dtd
4.0 and higher	deploy-car_1_3.dtd

DOCTYPE statement Specify the appropriate DOCTYPE statement based on your version of the SilverStream eXtend Application Server:

Server version	DOCTYPE statement
3.7.2 and lower	<code><!DOCTYPE carJarOptions PUBLIC "-//SilverStream Software, Inc.//DTD J2EE CAR Deployment Plan//EN" "deploy_car.dtd"></code>
3.7.3 and higher	<code><!DOCTYPE carJarOptions PUBLIC "-//SilverStream Software, Inc.//DTD J2EE CAR Deployment Plan 1.2//EN" "deploy-car_1_2.dtd"></code>
4.0 and higher	<code><!DOCTYPE carJarOptions PUBLIC "-//SilverStream Software, Inc.//DTD J2EE CAR Deployment Plan 1.3//EN" "deploy-car_1_3.dtd"></code>

Elements The elements defined by the client JAR deployment plan DTD are described below (in alphabetical order):

- **beanLink element**

```
<!ELEMENT beanLink (#PCDATA)>
```

Specifies the JNDI name of an EJB referenced by the application client. The referenced EJB can be located on a server other than the one on which the application client is deployed.

Specifying the server host for a beanLink Whether you deploy your client JAR by itself or in an EAR, you must be careful about how you specify the SilverStream server host name for any EJB references you map—because the host name you type when invoking SilverJ2EEClient must **exactly match** the host name that appears in the beanLink elements of your deployment plan:

- **Guidelines** For instance, if the host name portion of the beanLink specifies myserver:

```
<beanLink>sssw://myserver/RMI/sbOrderSummary</beanLink>
```

then the SilverJ2EEClient command line must also specify that name:

```
SilverJ2EEClient myserver snuckerby appclient Sam
```

Alternatively, they may both specify the IP address of that server or they may both specify localhost (although it's generally best to avoid using localhost). If you specify the server's port number (such as myserver:80) in one, you must specify it in both.

- **Troubleshooting** If there isn't an exact match (such as if one specifies the server's host name while the other specifies its IP address), then a runtime error will occur when your client tries to access an EJB:

```
java.rmi.RemoteException: Unable to get a valid session for a request.
```

- **For EARs** If you decide to deploy your client JAR within an EAR, this matching requirement still applies, whether you specify the host name in your EAR deployment plan or on the SilverCmd DeployEAR command line.

- **beanReference element**

```
<!ELEMENT beanReference (name, beanLink)>
```

Container element for EJBs referenced by the application client.

- **beanReferenceList element**

```
<!ELEMENT beanReferenceList (beanReference*)>
```

Container element for any EJB references used by the application client.

- **carJar element**

```
<!ELEMENT carJar (version?, carJarName?, environmentList?,  
beanReferenceList?, resourceReferenceList?, usesJars?)>
```

Container element.

- **carJarName element**

```
<!ELEMENT carJarName (#PCDATA)>
```

The name of the application client JAR.

- **carJarOptions element**

```
<!ELEMENT carJarOptions (carJar)>
```

The root element for the application client deployment plan.

- **clientDataSource element**

```
<!ELEMENT clientDataSource (jdbcURL?, jdbcDriver?,  
jdbcUsername?, jdbcPassword?)>
```

Container element that describes access to a `javax.sql.DataSource` accessed by the application client. Clients access data source resources directly, so the client must have the appropriate JDBC drivers installed for the data sources it needs to access. These values are passed as Strings. See the documentation for your particular data source for the correct syntax for `jdbcURL`, `jdbcDriver`, `jdbcUsername`, and `jdbcPassword`.

- **el element**

```
<!ELEMENT el (#PCDATA)>
```

Element of a string array.

- **environmentEntry element**

```
<!ELEMENT environmentEntry (name, value)>
```

Container element that allows the deployer to provide or override deployment descriptor environment entries.

- **environmentList element**

```
<!ELEMENT environmentList (environmentEntry*)>
```

Container element for zero or more `environmentEntry` elements.

- **jdbcDriver element**

```
<!ELEMENT jdbcDriver (#PCDATA)>
```

The name of the JDBC driver used to create a JDBC Connection to a specific database at runtime.

- **jdbcPassword element**

```
<!ELEMENT jdbcPassword (#PCDATA)>
```

The password used when creating the JDBC connection.

- **jdbcURL element**

```
<!ELEMENT jdbcURL (#PCDATA)>
```

The JDBC URL used to create a connection to a specific database at runtime.

- **jdbcUsername element**

```
<!ELEMENT jdbcUsername (#PCDATA)>
```

The JDBC user name used when creating the JDBC connection.

- **mailRefProperties element**

```
<!ELEMENT mailRefProperties (el+)>
```

Specifies the set of name/value pairs to associate with a resource reference of type `javax.mail.Session`, as determined by the JavaMail specification.

Some examples include:

```
mail.host
mail.from
mail.user
mail.store.protocol
mail.transport.protocol
mail.debug
mail.[protocol].host (for example, mail.http.host)
mail.[protocol].user (for example, mail.http.user)
```

- **name element**

```
<!ELEMENT name (#PCDATA)>
```

Specifies the name of an environment entry, a resource reference, or an EJB reference.

The name must match the name of an entry of the same type in the deployment descriptor, (if it does not, you'll get a warning message during deployment).

- **resourceEnvReference**

```
<!ELEMENT resourceEnvReference (name, resourceJNDIName)>
```

Maps a resource environment reference name to a JNDI name. Each occurrence of `resource-env-ref` in the deployment descriptor should have one corresponding `resourceEnvReference` in the deployment plan to specify the JNDI name. The name element must be the same as the corresponding `resource-env-ref-name` in the deployment descriptor.

- **resourceEnvReferenceList**

```
<!ELEMENT resourceEnvReferenceList (resourceEnvReference*)>
```

Container element for a list of resource environment references.

- **resourceJNDIName element**

```
<!ELEMENT resourceJNDIName (#PCDATA)>
```

The JNDI name for a resource reference. All resource references must contain a JNDI name except `javax.sql.DataSource` and `java.net.URL` resource references.

The JNDI name can be located on a server other than the application client is on. To find a `resourceJNDIName`, use this syntax:

```
<resourceJNDIName>sssw://server:port/RMI/JNDINameOfResource
</resourceJNDIName>
```

- **resourceReference element**

```
<!ELEMENT resourceReference (name, (clientDataSource|resourceURL|
resourceJNDIName|mailRefProperties))>
```

Container element for resource references. You must specify the resource reference name (from the deployment descriptor) and **one** of the following:

- A **data source** for references of type `javax.sql.DataSource`
- An **URL** for references of type `java.net.URL`
- A **JNDI name** for references of type `javax.jms.QueueConnectionFactory` or `javax.jms.TopicConnectionFactory`
- A **String array** (to be treated as an array of name/value pairs) for references of type `javax.mail.Session`

- **resourceReferenceList element**

```
<!ELEMENT resourceReferenceList (resourceReference*)>
```

Container element for the set of resource references used by this application client.

- **resourceURL element**

```
<!ELEMENT resourceURL (#PCDATA)>
```

Specifies the URL that will be looked up at runtime for a specific JNDI name.

- **usesJars element**

```
<!ELEMENT usesJars (el+)>
```

A String array that lists other JARs that this JAR uses.

- **value element**

```
<!ELEMENT value (#PCDATA)>
```

An environment entry value. A value set in the deployment plan overrides a value set in the deployment descriptor.

- **version element**

```
<!ELEMENT version (#PCDATA)>
```

Specifies the version number of this deployment plan. If it is not present, the version number is assumed to be 0.0. Deployment plans with version 0.0 are associated with SilverStream eXtend Application Server Version 3.7.2 and lower.

EJB JAR deployment plan DTD

This section provides reference information about the SilverStream EJB JAR deployment plan DTD:

- DTD file
- DOCTYPE statement
- Elements

DTD file There are several EJB JAR deployment plan DTDs. Use the appropriate one for your version of the SilverStream eXtend Application Server:

Server version	DTD file name
3.7.2 and lower	deploy_ejb.dtd
3.7.3 and higher	deploy-ejb_1_1.dtd
4.0	To deploy to the J2EE1.3 container use <code>deploy-ejb_2_0.dtd</code> To deploy to the J2EE1.2 container use <code>deploy-ejb_1_1.dtd</code>

DOCTYPE statement Specify the DOCTYPE statement appropriate for your version of the SilverStream eXtend Application Server:

Server version	DOCTYPE statement
3.7.2 and lower	<code><!DOCTYPE ejbJarOptions PUBLIC "-//SilverStream Software, Inc.//DTD J2EE EJB Deployment Plan//EN" "deploy_ejb.dtd"></code>
3.7.3 and higher	<code><!DOCTYPE ejbJarOptions PUBLIC "-//SilverStream Software, Inc.//DTD J2EE EJB Deployment Plan 1.1//EN" "deploy-ejb_1_1.dtd"></code>
4.0	<p>To deploy to the J2EE1.2 container:</p> <pre><!DOCTYPE ejbJarOptions PUBLIC "-//SilverStream Software, Inc.//DTD J2EE EJB Deployment Plan 1.1//EN" "deploy-ejb_1_1.dtd"></pre> <p>To deploy to the J2EE1.3 container:</p> <pre><!DOCTYPE ejbJarOptions PUBLIC "-//SilverStream Software, Inc.//DTD J2EE EJB Deployment Plan 2.0//EN" "deploy-ejb_2_0.dtd"></pre>

Elements The elements defined by the EJB JAR deployment plan DTD are described below (in alphabetical order):

- **alternate column element**

```
<!ELEMENT alternateColumn (#PCDATA)>
```

Specifies an alternate column with a SQL type of BIGINT to manage the handling of BLOB type of data, when isolationLevel is TRANSACTION_READ_COMMITTED.

- **asContext element**

```
<!ELEMENT asContext (authMethod, realm, asContextRequired)>
```

Describes the authentication mechanism used to authenticate the client.

- **asContextRequired element**

```
<!ELEMENT asContextRequired (#PCDATA)>
```

Specifies whether the authentication method specified is required for client authentication.

- **authMethod element**

```
<!ELEMENT authMethod (#PCDATA)>
```

Specifies the authentication method. It can be one of these:

```
<authMethod>NONE</authMethod>  
<authMethod>USERNAME_PASSWORD</authMethod>
```

- **autoInc element**

```
<!ELEMENT autoInc (autoIncSequenceName?, (schemaName?, autoIncTableName,  
columnName)?)>
```

Specifies the support for an autoincrement column.

- For databases that support autoincrement (most databases), an empty autoInc element is all that is required.
- For Oracle databases, the autoIncSequenceName element specifies the sequence name used.
- For databases that do not support autoincrement, use the schemaName element, autoIncTableName element, and columnName element to generate a unique number.

- **autoIncSequenceName element**

```
<!ELEMENT autoIncSequenceName (#PCDATA)>
```

Specifies the sequence name for an autoincrement column in an Oracle database.

- **autoIncTableName element**

```
<!ELEMENT autoIncTableName (#PCDATA)>
```

Specifies the name of the table containing the column columnName element, which is used for maintaining the current values for an autoincrement column. Use this only for databases that **do not** support autoincrement.

- **bean element**

```
<!ELEMENT bean (beanName, beanJNDIName, usesSSL?, cipherSuites?,
  resourceReferenceList?, environmentList?,
  beanReferenceList?)>
```

Container element for common deployment information for session and entity beans.

- **beanJNDIName element**

```
<!ELEMENT beanJNDIName (#PCDATA)>
```

JNDI name of an EJB. The JNDI name is used by the container to register the bean within the JNDI namespace. Any code that calls the bean must find it first using the JNDI name. JNDI names must be unique within a server and across servers in a cluster. You should consider using a hierarchical structure for naming your beans. You might even want to include the company name (or initials) within the hierarchy to ensure that EJBs are unique. Some examples include:

```
abccorp/samples/SalesDemo/Customers
com/sssw/samples/BankDemo/SessionBeans/AddCustomer
com/sssw/samples/BankDemo/EntityBeans/Customer
```

The JNDI lookup for the previous examples would be (respectively):

```
contextEnv.lookup("RMI/abccorp/samples/SalesDemo/Customers");
contextEnv.lookup("RMI/com/sssw/samples/BankDemo/SessionBeans/
  AddCustomer");
contextEnv.lookup("RMI/com/sssw/samples/BankDemo/EntityBeans/
  Customer");
```

- **beanLink element**

```
<!ELEMENT beanLink (#PCDATA)>
```

Specifies the JNDI name of a referenced EJB. That EJB can be located on another server.

Examples of specifying the JNDI name of an EJB:

- On the same server:

```
<beanLink>SBBankATM</beanLink>
```

- On a different server:

```
<beanLink>sssw://myServer:80/RMI/SBBankATM</beanLink>
```

When you specify a link to an EJB that resides on a different server, you have to copy the remote JAR for the referenced bean to the server from which it is being referenced.

- **beanLocalJNDIName element**

```
<!ELEMENT beanLocalJNDIName (#PCDATA)>
```

Specifies the JNDI name of a local bean.

- **beanLocalReferenceList element**

```
<!ELEMENT beanLocalReferenceList (beanReference*)>
```

Specifies the mapping of EJB local references.

- **beanName element**

```
<!ELEMENT beanName (#PCDATA)>
```

The name of an EJB. It corresponds to ejb-name from the deployment descriptor.

- **beanPersistenceInfo element**

```
<!ELEMENT beanPersistenceInfo (beanName, dataSourceName?, sqlHandler?,  
delayInstantiation?, table)>
```

The beanPersistenceInfo element is the persistence info for a CMP entity bean. It contains:

- The name of the bean
- An optional data source name to override the one specified in beanPersistenceInfoList
- An optional SQL handler class name to override the one specified in beanPersistenceInfoList
- An optional flag to delay the instantiation of the bean
- The description of a table this bean is deployed to

- **beanPersistenceInfoList element**

```
<!ELEMENT beanPersistenceInfoList (dataSourceName?, sqlHandler?,  
isolationLevel?, beanPersistenceInfo+)>
```

Persistence info for all CMP entity beans in this JAR. It contains:

- An optional data source used by most (if not all) of the CMP entity beans
- An optional SQL handler class name to handle SQL-related works for most (if not all) of the CMP entity beans
- An optional Isolation Level (default is Connection.TRANSACTION_READ_COMMITTED)
- A list of persistence info for each bean

- **beanReference element**

```
<!ELEMENT beanReference (name, beanLink?)>
```

Container element for mapping an EJB reference to an EJB.

- **beanReferenceList element**

```
<!ELEMENT beanReferenceList (beanReference+)>
```

Container element for specifying a list of references to other EJBs.

- **beansList element**

```
<!ELEMENT beansList ((entity|session)+)>
```

Container element for one or more entity or session beans.

- **callerPropagation element**

```
<!ELEMENT callerPropagation (#PCDATA)>
```

Specifies whether the target will accept propagated caller identities. Must be one of the following:

```
<callerPropagation>NONE</callerPropagation>
<callerPropagation>SUPPORTED</callerPropagation>
<callerPropagation>REQUIRED</callerPropagation>
```

- **classpathJars element**

```
<!ELEMENT classpathJars (excludeJ2EEXMLJars?, userlibJars?)>
```

Allows a list of JAR files to be used in an application without deploying them to the server. It also allows the user to disable the inclusion of the default XML JARs so that different versions of XML parsers and XML transformation engines can be used.

- **clientID element**

```
<!ELEMENT clientID (#PCDATA)>
```

Specifies the JMS Client ID used for JMS Topic Connection.

- **cmpFieldName element**

```
<!ELEMENT cmpFieldName (#PCDATA)>
```

Specifies the CMP field name.

- **cmrFieldName**

```
<!ELEMENT cmrFieldName (#PCDATA)>
```

Specifies the CMR field name in a relationship.

- **columnName element**

```
<!ELEMENT columnName (#PCDATA)>
```

Specifies the database column name to which a CMP entity bean's persistent field is mapped. This name must match a name in the database table to which the entity bean is bound, and must be the name by which the SilverStream server knows the column.



See dataSrcTable element.

- **columnNames element**

```
<!ELEMENT columnNames (el+)>
```

Specifies a list of column names in the table used to define a relationship.

- **confidentiality element**

```
<!ELEMENT confidentiality (#PCDATA)>
```

Specifies whether the target supports privacy protected messages. Must be one of the following:

```
<confidentiality>NONE</confidentiality>
<confidentiality>SUPPORTED</confidentiality>
<confidentiality>REQUIRED</confidentiality>
```

- **confidentialityCipherSuites element**

<!ELEMENT confidentialityCipherSuites (el+)>

Specifies the names of the cipher suites to be used when confidentiality (in the TransportConfig element) is specified for a session bean or an entity bean in the JAR.

- **connectionFactoryName element**

<!ELEMENT connectionFactoryName (#PCDATA)>

Specifies the JNDI name used to look up the JMS Connection Factory.

- **dataSource element**

<!ELEMENT dataSource (#PCDATA)>

Specifies the name of the database to associate with an EJB's resource reference of type javax.sql.DataSource.

- **dataSourceName element**

<!ELEMENT dataSourceName (#PCDATA)>

Specifies the name of the data source (database) to associate with a bean's resource reference of type javax.sql.DataSource.

- **dataSrcTable element**

<!ELEMENT dataSrcTable (#PCDATA)>

Specifies the name of a database table to associate with a CMP entity bean. If any persistent fields are specified, this element **must** be specified. This is the primary table for the bean.

- **delayInstantiation element**

<!ELEMENT delayInstantiation (#PCDATA)>

Specifies whether a CMP entity bean is instantiated immediately (FALSE—the default) or only when required (TRUE).

This element determines how the server will retrieve and cache data during finder method execution. Beans whose instantiation is delayed are called *lazy beans*.

Here are the possible scenarios:

Scenario	Description
You set delay instantiation to TRUE , but the finder method is not part of a transaction context (an unspecified context)	The server ignores the setting. When you call a finder method, the server retrieves the primary key and does not cache any of the data. If your bean requires subsequent use of this data, the server must retrieve the data at that point.

Scenario	Description
<p>You set delay instantiation to TRUE, and the finder method is part of a transaction context</p>	<p>The server retrieves only the primary key for each of the finder methods.</p> <p>Any subsequent method calls on that bean cause the server to access the database again to retrieve the data and populate the fields for the bean.</p> <p>Although this causes an additional trip to the database, it can be more efficient in cases where the data is large or the application is only interested in a single record. It allows the application to do some processing to determine which record it wants.</p> <p>For applications that will only access a single bean, the initial load time for all of the beans might not be worth it.</p>
<p>You set delay instantiation to FALSE, and the finder method is part of a transaction context</p>	<p>The server retrieves the primary keys, and also retrieves all of the values for all of the persistent mapped fields for each bean. The server caches this data.</p> <p>If the client makes subsequent method calls for any of the fields, the server will not need to retrieve any additional data from the database (because it will have cached all of the records).</p> <p>Depending on your data, this might improve or degrade your performance. It might degrade performance if the bean has a lot of fields or the fields contain large amounts of data. For example, if the beans you are retrieving contain a BLOB field and you do not need access to all of the records, this might not be the right strategy for your application.</p>

- **deployedObject element**

```
<!ELEMENT deployedObject (#PCDATA)>
```

Name of the EJB deployed object to create on the SilverStream server. It is optional and can also be specified on the command line (when deploying with SilverCmd DeployEJB).

When this element is not specified, deployment generates a name by appending the word **Deployed** to the value of the ejbJarName element.

- **destinationName element**

```
<!ELEMENT destinationName (#PCDATA)>
```

Specifies the Queue or Topic Name. It must be specified like this:

```
corbaname:iiop:JMSServer:3506#queue/queueName
```

- **ejbCreate element**

```
<!ELEMENT.ejbCreate (#PCDATA)>
```

Specifies the SQL statement to be used for `ejbCreate` (insert), substituting the one generated by the container.

- **ejbJar element**

```
<!ELEMENT.ejbJar (version?, isEnabled?, ejbJarName?, remoteAccessJar?,  
deployedObject?, usesJars?, lenientSecurity?, beansList, roleMap?)>
```

Container element that includes the JAR-level information.

- **ejbJarName element**

```
<!ELEMENT.ejbJarName (#PCDATA)>
```

Name of the EJB JAR to deploy. You may specify this in the deployment plan or on the command line (when deploying with `SilverCmd DeployEJB`). Values specified on the command line take precedence.

You can specify a full file path (if the JAR is on disk) or just the name. When you specify just the name and the EJB JAR is not in the current directory, the EJB JAR is assumed to be on the same database and server to which it is being deployed.

- **ejbJarOptions element**

```
<!ELEMENT.ejbJarOptions (ejbJar)>
```

The root element of the EJB deployment plan.

- **ejbLoad element**

```
<!ELEMENT.ejbLoad (#PCDATA)>
```

Specifies the SQL statement to be used for `ejbLoad` (select), substituting the one generated by the container.

- **ejbQL element**

```
<!ELEMENT.ejbQL (#PCDATA)>
```

The EJB QL for this finder.

- **ejbRemove element**

```
<!ELEMENT.ejbRemove (#PCDATA)>
```

Specifies the SQL statement to be used for `ejbRemove` (delete), substituting the one generated by the container.

- **ejbStore element**

```
<!ELEMENT ejbStore (#PCDATA)>
```

Specifies the SQL statement to be used for ejbStore (update), substituting the one generated by the container.

- **el element**

```
<!ELEMENT el (#PCDATA)>
```

Element of a String array.

- **entity element**

```
<!ELEMENT entity (dataSrcTable?, primaryKeyClass?, delayInstantiation?,
bean, nonModifyingMethodsList?,fieldMap?, finderMethodsList?)>
```

Container element for the description of entity bean-specific deployment information.

- **environmentEntry element**

```
<!ELEMENT environmentEntry (name, value)>
```

Container element that allows the deployer to provide or override deployment descriptor environment entries.

- **environmentList element**

```
<!ELEMENT environmentList (environmentEntry+)>
```

Container element for any environmentEntry elements.

- **establishTrustInClient element**

```
<!ELEMENT establishTrustInClient (#PCDATA)>
```

Specifies whether the target (server) is capable of authenticating a client. If this is **REQUIRED**, the server wants to see a client certificate. If it's **SUPPORTED**, the server will accept a client certificate if provided but allow the call anyway if not. It must be one of the following:

```
<establishTrustInClient>NONE</establishTrustInClient>
<establishTrustInClient>SUPPORTED</establishTrustInClient>
<establishTrustInClient>REQUIRED</establishTrustInClient>
```

- **establishTrustInServer element**

```
<!ELEMENT establishTrustInServer (#PCDATA)>
```

Specifies whether the target (server) is capable of authenticating itself to a client. If this value is **SUPPORTED**, the target supports one or more cipher suites that include a way to securely identify the server to the client; the client will use SSL. This value can be:

```
<establishTrustInServer>NONE</establishTrustInServer>
<establishTrustInServer>SUPPORTED</establishTrustInServer>
```

- **excludeJ2EEXMLJars element**

`<!ELEMENT excludeJ2EEXMLJars (#PCDATA)>`

Tells the EJB if it should include the default J2EE 1.3 XML JARs in the classloader for this application. The default JAR files are the specific versions of crimson.jar and xalan.jar that are used in the J2EE 1.3 CTS tests. If this element is set to TRUE, XML parsing and XML transformation classes from crimson.jar and xalan.jar will not automatically be made available to the application. This permits application developers to package (or include via userlibJars) other versions of XML parsers and transformation engines in their applications. If this element is absent or is FALSE, the default JARs will automatically be made available to the application.

- **field element**

`<!ELEMENT field (cmpFieldName, columnName, autoInc?, alternateColumn?)>`

Specifies the mapping of a CMP field to a database column.

- **fieldMap element**

```
<!ELEMENT fieldMap (fieldMapping+)>
```

The container element for the mapping of persistent fields to database columns. CMP entity bean persistent fields can be mapped as follows:

Persistent field type	Description
Columns in a primary table	<p>You can map the persistent fields of each entity bean to columns in the primary table. (The primary table is the table to which the entity bean is mapped.)</p> <p>You can map entity beans where a persistent field represents another object. For example, suppose an entity bean contains a persistent field called <code>m_address</code> that is an <code>EmployeeAddress</code> object and has the following fields:</p> <ul style="list-style-type: none"> • <code>m_street</code> • <code>m_city</code> • <code>m_state</code> <p>According to the EJB specification, the field's class must have a public constructor with public member variables that are primitive, well-known, <code>Serializable</code>, or compound.</p> <p>When you map <code>m_address</code>, you actually map each of its constituent parts to a specific database column. The <code>m_street</code> field might map to the database column called <code>street</code>, the <code>m_city</code> field might map to the database column called <code>city</code>, and so on.</p> <p>Persistent fields mapped to one or more fields in the primary table are read-write.</p>
Columns in related tables	<p>You can also map the persistent fields to columns in a different table as long as the column is related to the primary table. Persistent fields mapped this way are read-only.</p>

Persistent field type	Description
Member variables in other beans	<p>You can map the persistent fields of one entity bean to the persistent fields of a different entity bean. To accomplish this:</p> <ul style="list-style-type: none"> • The bean developer must declare the persistent field's type as the referenced bean's remote interface. • The referenced bean must be deployed in the same EJB JAR file. • At deployment time you map the persistent field(s) to the referenced bean instead of a database table. • At deployment time you map the persistent fields of the referenced bean's primary key to database columns in the primary table. <p>Bean references are read-write.</p> <p>The SilverStream server supports mapping only to an individual bean reference. It directly supports many-to-one and one-to-one relationships. A one-to-many relationship would look like a Collection in the referring bean. You can implement a one-to-many relationship programmatically.</p>

- **fieldMapping element**

```
<!ELEMENT fieldMapping (name, columnName?, foreignBeanName?,
    primaryKeyClassSimple?, primaryKeyClass?,
    homeInterfaceClass?, fieldMapping*)>
```

Container element for mapping a persistent field of a CMP entity bean to a database column.

- **finderMethod element**

```
<!ELEMENT finderMethod (method, finderMethodType?,
    whereClause?)>
```

Container element for defining a finder method.

- **finderMethodsList element**

```
<!ELEMENT finderMethodsList (finderMethod+)>
```

Container element for a list of finder methods defined for an entity bean.

- **finderMethodType element**

```
<!ELEMENT finderMethodType (#PCDATA)>
```

Specifies one of these finder method types:

Type	Use it when
expression	You want to create a finder method that is a classic SilverStream expression (such as those used for Link clauses on SilverStream forms and pages)
method	<ul style="list-style-type: none"> • Arguments are compound or require processing • You want to perform additional validation or error checking • You want to include Order by, Distinct, and result count limits

If the finderMethodType is not specified, it is assumed to be expression.

- **foreignBeanName element**

```
<!ELEMENT foreignBeanName (#PCDATA)>
```

Specifies the ejb-name of an EJB when a persistent field from one bean is bound to a field in a different bean.

If your deployment descriptor includes a persistent field that has another bean's remote interface as its data type, you can map the persistent field to another bean. The other bean is called the *foreign bean*. You can map to a single bean reference supporting many-to-one and one-to-one relationships (but not one-to-many relationships).

The foreign bean **must** be in the same EJB JAR.

- **homeInterfaceClass element**

```
<!ELEMENT homeInterfaceClass (#PCDATA)>
```

Specifies the foreign bean's fully qualified home interface class.

- **initialPoolSize element**

```
<!ELEMENT initialPoolSize (#PCDATA)>
```

Specifies the initial pool size for session or entity beans. The default is 0.

- **integrity element**

```
<!ELEMENT integrity (#PCDATA)>
```

Specifies whether the target supports integrity protected messages. It can be:

```
<integrity>NONE</integrity>
<integrity>SUPPORTED</integrity>
<integrity>REQUIRED</integrity>
```

- **integrityCipherSuites element**

```
<!ELEMENT integrityCipherSuites (el+)>
```

Specifies the names of the cipher suites to be used, when integrity (in transportConfig) is specified for a session bean or entity bean in the JAR.

- **iorSecurityConfig element**

```
<!ELEMENT iorSecurityConfig (transportConfig?, asContext?, sasContext?)>
```

Specifies the IOR security information for an EJB.

- **isEnabled element**

```
<!ELEMENT isEnabled (#PCDATA)>
```

The state of the EJB JAR.

EJB JARs created and deployed using SilverStream graphical tools are enabled by default. If the deployment information is not complete, a deployed JAR cannot be enabled.

- **isolationLevel element**

```
<!ELEMENT isolationLevel (#PCDATA)>
```

Specifies the isolation level to be used. It can be:

```
<isolationLevel>TRANSACTION_READ_COMMITTED</isolationLevel>  
<isolationLevel>TRANSACTION_SERIALIZABLE</isolationLevel>
```

The default value is TRANSACTION_READ_COMMITTED.

- **jarFileName element**

```
<!ELEMENT jarFileName (#PCDATA)>
```

Specifies the name of the JAR file that contains the classes.

- **lenientSecurity element**

```
<!ELEMENT lenientSecurity (#PCDATA)>
```

A boolean that specifies whether the EJB requires a *lenient* interpretation of the method permission entries—that is, whether a method not listed in any method permission entry is callable by all users (lenient) or no users (strict). The default is strict.

- **linkTable column element**

```
<!ELEMENT linkTable (schemaName?, name)>
```

Specifies the name of the link table used for a many-to-many relationship.

- **mailRefProperties element**

```
<!ELEMENT mailRefProperties (el+)>
```

Specifies the set of name/value pairs to associate with a resource reference of type javax.mail.Session, as determined by the JavaMail specification.

Some examples include:

```
mail.host  
mail.from  
mail.user
```

```

mail.store.protocol
mail.transport.protocol
mail.debug
mail.[protocol].host (for example, mail.http.host)
mail.[protocol].user (for example, mail.http.user)

```

- **maxPoolSize element**

```
<!ELEMENT maxPoolSize (#PCDATA)>
```

Specifies the maximum pool size for session, entity, or message driven beans. If omitted, the default value for session beans is 500, for entity beans it is 0, and for message driven beans it is 5.

- **message element**

```
<!ELEMENT message (destinationName, connectionFactoryName, userName?,
password?, serverSessionMaxMsgs?, maxPoolSize?, clientID?,
subscriptionName?, selectorAddon?, bean)>
```

Description of deployment information for a message-driven bean. It contains:

- The destination for this Message Driven Bean
 - The JNDI name used to look up its JMS connection factory
 - An optional user name and password for JMS connection
 - An optional integer that specifies the maximum number of messages to be processed by the ServerSession at a time
 - An optional integer that specifies the maximum ServerSession pool size
 - An optional Client ID for a topic
 - An optional Subscription name for a topic
 - An optional additional selector to be appended to the message selector in the deployment descriptor
 - A bean element for more bean info, including resource and environment entries
- **maxSessionPoolSize element**

```
<!ELEMENT maxSessionPoolSize (#PCDATA)>
```

Specifies the maximum pool size (number of instances) for a specific session bean. When omitted, the default is 500.

- **method element**

```
<!ELEMENT method (name, methodParams?)>
```

Container element for specifying method descriptions for the nonModifyingMethodsList element and the finderMethod element.

- **methodParams element**

```
<!ELEMENT methodParams (el+)>
```

Specifies a list of method parameters for methods in the `nonModifyingMethodsList` and `finderMethod` elements. The parameters list the data type. If the data type is not a native Java type (`int`, `char`, and so on) or is not in the `java.lang` package, it must be a fully qualified name.

- **name element**

```
<!ELEMENT name (#PCDATA)>
```

Specifies the name of one of the following:

- An environment entry, a resource reference, or an EJB reference. The name must match the name of an entry of the same type in the deployment descriptor, or you'll get a warning message during deployment. (See `environmentEntry` element.)
- A persistent field as referred to in the bean code. (See `fieldMapping` element.)
- A method, see `method` element.

- **nonSSSWPersistenceInfo element**

```
<!ELEMENT nonSSSWPersistenceInfo (persistenceInfoFileName,  
jarFileName,persistenceManagerClass)>
```

Specifies the use of a non-SilverStream Persistence Manager.

- **nonModifyingMethodsList element**

```
<!ELEMENT nonModifyingMethodsList (method+)>
```

Container element for the list of methods that do **not** modify fields in the database associated with the bean.

Specifying these methods might improve performance—because each time a method is called on an entity bean, the server checks all of the bean's persistent fields to see if any of them have been modified. If any have been modified, the server writes any changes to the database. If you specify that a particular method does not modify a field, the server does not perform this check.

- **password element**

```
<!ELEMENT password (#PCDATA)>
```

Specifies the password to be used to create a JMS Connection.

- **persistenceInfo element**

```
<!ELEMENT persistenceInfo (beanPersistenceInfoList, relationsList?)>
```

Specifies the information required by the SilverStream Persistence Manager. It contains:

- A list of persistence info for each CMP entity bean
- An optional list of relation information

- **persistenceInfoFileName element**

```
<!ELEMENT persistenceInfoFileName (#PCDATA)>
```

Specifies the name of the file that contains the contents of non-SilverStream persistence information. It will be read in as an array of bytes that will be passed to the non-SilverStream Persistence Manager at runtime.

- **persistenceManagerClass element**

```
<!ELEMENT persistenceManagerClass (#PCDATA)>
```

Specifies the class name of the non-SilverStream Persistence Manager.

- **poolingPolicy element**

```
<!ELEMENT poolingPolicy (#PCDATA)>
```

Specifies the pooling policy to use for session or entity bean pooling. If the container requests a bean from the pool when all the instances from the pool are active and the pool has reached its `maxPoolSize`, the specified policy will take effect:

- If the CREATE policy is specified, a new bean instance will be created.
- If the FAIL policy is specified, the container will throw an exception.

The poolingPolicy element must be one of the following:

```
<poolingPolicy>CREATE</poolingPolicy>
<poolingPolicy>FAIL</poolingPolicy>
```

The default policy is CREATE.

- **primaryKey element**

```
<!ELEMENT primaryKey (primaryKeyClass, columnName?, autoInc?)>
```

Specifies the mapping of primary keys when the deployment descriptor's `prim-key-class` is `java.lang.Object`. SilverStream supports `String`, `Integer`, or `Long` for the `primaryKeyClass` element in the deployment plan. If the `primaryKeyClass` element is `java.lang.String`, the `autoInc` element is ignored and the column has to be able to store a 32-byte length of `String` data. If `primaryKeyClass` is `java.lang.Integer` or `java.lang.Long`, `autoInc` is required—so that an autoincrement column can be used to generate unique primary keys.

- **primaryKeyClass element**

```
<!ELEMENT primaryKeyClass (#PCDATA)>
```

Specifies the fully qualified primary key class for a CMP entity bean. This can be a user-defined class or a standard data type like `java.lang.Integer`.

- **primaryKeyClassSimple element**

```
<!ELEMENT primaryKeyClassSimple (#PCDATA)>
```

Specifies whether the primary key class is a Java data type (`TRUE`) such as `String` or `Integer` or a user-defined class (`FALSE`).

- **principalList element**

```
<!ELEMENT principalList (el+)>
```

Specifies the name of the principals to be mapped to a role.

- **realm element**

```
<!ELEMENT realm (#PCDATA)>
```

Specifies the realm in which the user is authenticated. This can be the well-known realm name (default), which matches any SilverStream-supported realm.

- **recoverable element**

```
<!ELEMENT recoverable (#PCDATA)>
```

Specifies whether a stateful session bean is recoverable or not. It can be:

```
<recoverable>TRUE</recoverable>  
<recoverable>FALSE</recoverable>
```

The default value is FALSE.

- **relation element**

```
<!ELEMENT relation (linkTable?, relationRole, relationRole)>
```

Specifies the following relation information:

- An optional link table when the relationship is many-to-many
- The name of the relation
- The two roles that comprise the relation

- **relationRole element**

```
<!ELEMENT relationRole (beanName, cmrFieldName?, columnNames?)>
```

Defines a role within a relation and contains:

- The bean name (from the deployment descriptor) for the source of a role that participates in the relationship. It needs to match that of the relationship-role-source in ejb-relationship-role.
- The CMR field name for the role (from the deployment descriptor). It needs to match that of cmr-field-name (or lack of). The deployer uses beanName and cmrFieldName to match the relationship.
- A list of column names (that most frequently contains only one element) that the container uses to manage the relationships. This is typically the foreign keys. For many-to-many relationships, this is the foreign key column name in the link table that corresponds to the primary key for the bean.

- **relationsList element**

```
<!ELEMENT relationsList (relation+)>
```

Contains the list of relations for which the user has specified deployment information.

- **remoteAccessJar element**

```
<!ELEMENT remoteAccessJar (#PCDATA)>
```

Name to use for the remote EJB JAR created at deployment. The remote JAR file includes stub or reference classes that you can use to call the server-side implementation classes. All clients use the remote JAR.

This element is optional and can also be specified on the command line (when deploying with SilverCmd DeployEJB). When this element is not specified, deployment generates a name by appending the word **Remote** to the value of the `ejbJarName` element.

- **resourceEnvReference element**

```
<!ELEMENT resourceEnvReference (name, resourceJNDIName)>
```

Specifies the mapping of a resource environment reference name to a JNDI name. Each occurrence of `resource-env-ref` in the deployment descriptor should have one corresponding `resourceEnvReference` in the deployment plan, to specify the JNDI name. The name element must be the same as the corresponding `resource-env-ref-name` in the deployment descriptor.

- **resourceEnvReferenceList element**

```
<!ELEMENT resourceEnvReferenceList (resourceEnvReference*)>
```

Specifies a list of resource environment references.

- **resourceJNDIName element**

```
<!ELEMENT resourceJNDIName (#PCDATA)>
```

Specifies the JNDI name for an EJB's resource reference of type `javax.jms.QueueConnectionFactory` or `javax.jms.TopicConnectionFactory`.

To find a `resourceJNDIName` on a different server, use this syntax:

```
<resourceJNDIName>sssw://server:port/RMI/JNDINameOfResource
</resourceJNDIName>
```

- **resourceReference element**

```
<!ELEMENT resourceReference (name, (dataSource | resourceURL |
resourceJNDIName | mailRefProperties))>
```

Container element for resource references. You must specify the resource reference name (from the deployment descriptor) and one of the following:

- A **data source** for references of type `javax.sql.DataSource`
- An **URL** for references of type `java.net.URL`
- A **JNDI name** for references of type `javax.jms.QueueConnectionFactory` or `javax.jms.TopicConnectionFactory`
- A **String array** (to be treated as an array of name/value pairs) for references of type `javax.mail.Session`

- **resourceReferenceList element**

`<!ELEMENT resourceReferenceList (resourceReference+)>`

Container element for the list of resource references for an EJB.

- **resourceURL**

`<!ELEMENT resourceURL (#PCDATA)>`

Specifies the URL to associate with a bean's resource reference of type `java.net.URL`.

- **roleLink element**

`<!ELEMENT roleLink (#PCDATA)>`

A reference to a defined security role. The `roleLink` element must contain one of the role names defined in the `roleMap` element.

- **roleMap element**

`<!ELEMENT roleMap (roleMapping+)>`

Container element for EJB-level role mappings.

If the deployment descriptor defines security roles, the deployment plan must map those roles to actual security groups or users on the target server.

- **roleMapping element**

`<!ELEMENT roleMapping (name, userOrGroupName)>`

Maps a single role to a user or group name.

- **sasContext element**

`<!ELEMENT sasContext (callerPropagation)>`

Describes the `SasContextInfo` security information.

- **schemaName element**

`<!ELEMENT schemaName (#PCDATA)>`

Specifies the schema name for a table. Some databases use the schema name to qualify a table within a catalog.

- **securityRoleRef element**

`<!ELEMENT securityRoleRef (name, roleLink)>`

Specifies the mapping of a role reference to a security role. For every security-role-ref that has no `roleLink` element specified in the deployment descriptor, there needs to be a corresponding `securityRoleRef` element in the deployment plan, to specify the role itself. The name element must be the same as the corresponding role-name in the deployment descriptor.

- **securityRoleReferenceList element**

`<!ELEMENT securityRoleReferenceList (securityRoleRef*)>`

Specifies a list of security roles reference mappings for the bean.

- **selectorAddon element**

```
<!ELEMENT selectorAddon (#PCDATA)>
```

Specifies the selector string to be appended to the message-selector specified in the deployment descriptor.

- **serverSessionMaxMsgs element**

```
<!ELEMENT serverSessionMaxMsgs (#PCDATA)>
```

Specifies the maximum number of messages a JMS server session will handle at a time. The default value is 1.

- **session element**

```
<!ELEMENT session (maxSessionPoolSize?, bean)>
```

Container for session bean description.

- **sqlHandler element**

```
<!ELEMENT sqlHandler (sqlType | (jarFileName, sqlHandlerClassName))>
```

Specifies the SQL handler used to handle all SQL-related operations.

- For a SilverStream-provided SQL handler, you need specify only the sqlType element.
- For a non-SilverStream-provided SQL handler, you must specify both the jarFileName element (which specifies the name of JAR file that contains the class to be used) and the sqlHandlerClassName element.

- **sqlHandlerClassName element**

```
<!ELEMENT sqlHandlerClassName (#PCDATA)>
```

Specifies the class name of a non-SilverStream-provided SQL handler is used to handle all SQL-related operations.

- **sqlStatement element**

```
<!ELEMENT sqlStatement (#PCDATA)>
```

Specifies a SQL statement to be used instead of the SilverStream-generated SQL statement.

- **sqlSubstitution element**

```
<!ELEMENT sqlSubstitution ((method | ejbCreate | ejbLoad | ejbStore | ejbRemove), sqlStatement)>
```

Specifies the SQL statement to replace the SilverStream-generated SQL statement.

- **sqlSubstitutionList element**

```
<!ELEMENT sqlSubstitutionList (sqlSubstitution+)>
```

Specifies the list of SQL statements to replace SilverStream-generated statements.

- **sqlType element**

```
<!ELEMENT sqlType (#PCDATA)>
```

Specifies the type of a SilverStream-provided SQL handler used to handle all SQL -related operations. It must be one of the following values: AdaptiveServerAnywhere, AdaptiveServerEnterprise, CLOUDSCAPE, DB2, INFORMIX, MicrosoftSQLServer, ORACLE, or the name of a user-defined class.

- **sqlWhereClause element**

```
<!ELEMENT sqlWhereClause (#PCDATA)>
```

Specifies the SQL WHERE clause for a (EJB 1.1) finder. The parameter index is embedded in the WHERE clause. To specify a parameter index, use *?n*, where *n* is the index, starting from 1.

- **subscriptionName element**

```
<!ELEMENT subscriptionName (#PCDATA)>
```

Specifies the JMS Subscription name used for JMS Topic Connection.

- **table element**

```
<!ELEMENT table (schemaName?, name, field+)>
```

Specifies a table the CMP bean maps to.

- **timeout element**

```
<!ELEMENT timeout (#PCDATA)>
```

Specifies the timeout value, in minutes, for a stateful session bean. The default value is 5 minutes. If the bean has not been accessed in this amount of time, the container removes the bean.

- **TransportConfig element**

```
<!ELEMENT transportConfig (integrity, confidentiality,  
establishTrustInClient, establishTrustInServer)>
```

Specifies the properties of security mechanism.

- **userlibJars element**

```
<!ELEMENT userlibJars (el+)>
```

Lists any additional JAR files (found in the userlib directory on the server) that should be made available to the application. The value is a StringArray of JAR names, relative to (and contained within) the userlib directory.

- **userName element**

```
<!ELEMENT userName (#PCDATA)>
```

Specifies the user name to be used to create a JMS Connection.

- **userOrGroupName element**

```
<!ELEMENT userOrGroupName (#PCDATA)>
```

The name of a principal in a security policy domain or a user group in the operational environment. This is mapped to a role reference name from the deployment descriptor. You can map a role reference to any of the security domains supported by the SilverStream server.

- **usesJars element**

```
<!ELEMENT usesJars (el+)>
```

A list of JARs used by the deployed object.

- **usesSSL element**

```
<!ELEMENT usesSSL (#PCDATA)>
```

Specifies whether the deployed EJB uses SSL security. The default is FALSE.

If you want an EJB's conversation with the server to be secure, you must specify that it use SSL and one or more RSA cipher suites. Each EJB in a JAR can support a different cipher suite. The JAR can also contain both beans that use SSL security and beans that do not.

For a bean that specifies SSL and one or more cipher suites to contact the server securely, you do not need to be running HTTPS—but there must be an RSA certificate installed on the SilverStream server.

- **value element**

```
<!ELEMENT value (#PCDATA)>
```

Specifies the actual value of an environment entry.

- **version element**


```
<!ELEMENT version (#PCDATA)>
```

Specifies the version number of this deployment plan. If it is not present, the version number will be assumed to be 0.0. Deployment plans with version 0.0 are associated with SilverStream eXtend Application Server Version 3.7.2 and lower.

- **whereClause element**

```
<!ELEMENT whereClause (#PCDATA)>
```

Specifies a SilverStream WHERE clause for when finderMethodType is expression.

 For information on writing the WHERE clause, see the chapter on using SilverStream expressions in the *Programmer's Guide* of the SilverStream eXtend Application Server's Classic Development Help.

WAR deployment plan DTD

This section provides reference information about the SilverStream WAR deployment plan DTD:

- DTD file
- DOCTYPE statement
- Elements

DTD file There are multiple WAR deployment plan DTDs. Use the appropriate one for your version of the SilverStream eXtend Application Server:

Server version	DTD file name
3.7.2 and lower	deploy_war.dtd
3.7.3 and higher	deploy-war_2_2.dtd
4.0 and higher	deploy-war_2_3.dtd

DOCTYPE statement Specify the DOCTYPE statement appropriate for your version of the SilverStream eXtend Application Server:

Server version	DOCTYPE statement
3.7.2 and lower	<pre><!DOCTYPE warJarOptions PUBLIC "-//SilverStream Software, Inc. //DTD J2EE WAR Deployment Plan//EN" "deploy_war.dtd"></pre>
3.7.3 and higher	<pre><!DOCTYPE warJarOptions PUBLIC "-//SilverStream Software, Inc. //DTD J2EE WAR Deployment Plan 2.2//EN" "deploy- war_2_2.dtd"></pre>
4.0 and higher	<pre><!DOCTYPE warJarOptions PUBLIC "-//SilverStream Software, Inc. //DTD J2EE WAR Deployment Plan 2.3//EN" "deploy- war_2_3.dtd"></pre>

Elements The elements defined by the WAR deployment plan DTD are described below (in alphabetical order):

- **beanLink element**

```
<!ELEMENT beanLink (#PCDATA)>
```

Specifies a link to a referenced EJB. This may be an internal name if the EJB is in the same JAR, or a JNDI name if it's in another JAR. The EJB can be located on another server.

Examples of specifying the JNDI name of an EJB:

- On the same server:

```
<beanLink>SBBankATM</beanLink>
```

- On a different server:

```
<beanLink>sssw://myServer:80/RMI/SBBankATM</beanLink>
```

- **beanLocalReferenceList element**

```
<!ELEMENT beanLocalReferenceList (beanReference*)>
```

Container element for specifying a list of local references to EJBs.

- **beanReference element**

```
<!ELEMENT beanReference (name, beanLink)>
```

Container element for mapping an EJB reference to an EJB.

- **beanReferenceList element**

```
<!ELEMENT beanReferenceList (beanReference*)>
```

Container element for specifying a list of references to EJBs.

- **classpathJars element**

```
<!ELEMENT classpathJars (excludeJ2EEXMLJars?, userlibJars?)>
```

Lists the JAR files used in the application that should not be deployed to the server. Use this to specify different versions of XML parsers and XML transformation engines other than the defaults.

- **contextParamEntry**

```
<!ELEMENT contextParamEntry (name, value)>
```

Context parameter entry. This value overrides any value specified in the deployment descriptor.

- **contextParamsList element**

```
<!ELEMENT contextParamsList (contextParamEntry+)>
```

List of context parameters that will be bound at runtime.

- **dataSource element**

```
<!ELEMENT dataSource (#PCDATA)>
```

Name of the database to associate with a resource reference of type `javax.sql.DataSource`.

- **deployedObject element**

```
<!ELEMENT deployedObject (#PCDATA)>
```

Name of the deployed object to create on the SilverStream server. When this element is not specified, deployment generates a name by appending the word **Deployed** to the value of the warJarName element.

- **deployToFilesystem element**

```
<!ELEMENT deployToFilesystem (#PCDATA)>
```

Whether or not the application should be deployed to the file system on the server. The default is FALSE (no file system deployment).

- **el element**

```
<!ELEMENT el (#PCDATA)>
```

Element of a string array.

- **environmentEntry element**

```
<!ELEMENT environmentEntry (name, value)>
```

Container element that allows the deployer to provide environment entry deployment values or to override values specified in the deployment descriptor.

- **environmentList element**

```
<!ELEMENT environmentList (environmentEntry*)>
```

Container element for any environmentEntry elements.

- **excludedJSPs element**

```
<!ELEMENT excludedJSPs (el+)>
```

List of JSP resources that should not be compiled. (Typically, JSP pages intended to be included in other JSP pages should not be compiled on their own.)

- **excludeJ2EEXMLJars**

```
<!ELEMENT excludeJ2EEXMLJars (#PCDATA)>
```

Specifies whether the WAR should include the default J2EE 1.3 XML JARs in the classloader for this application. The default JAR files are the specific versions of crimson.jar and xalan.jar that are used in the J2EE 1.3 CTS tests.

- When this element is absent or is set to FALSE, the default JARs are automatically made available to the application.
- When this element is set to TRUE, XML parsing and XML transformation classes from crimson.jar and xalan.jar are not automatically available to the application; the application developer must package (or include via the userlibJars element) other versions of XML parsers and transformation engines in their application.

- **filter element**

```
<!ELEMENT filter (filterName, initParamsList)>
```

Specifies initialization parameters in the deployment descriptor.

- **filterName element**

```
<!ELEMENT filterName (#PCDATA)>
```

The name of the filter to which the `initParamsList` of the filter element is to be bound.

- **filtersList element**

```
<!ELEMENT filtersList (filter+)>
```

A list of filters that specify init params in the deployment descriptor.

- **initParamEntry element**

```
<!ELEMENT initParamEntry (name, value)>
```

A specific servlet initialization parameter entry.

- **initParamsList element**

```
<!ELEMENT initParamsList (initParamEntry+)>
```

A list of servlet initialization parameters that will be bound at runtime.

- **isEnabled element**

```
<!ELEMENT isEnabled (#PCDATA)>
```

The state of the WAR. When you disable a WAR, you make it inactive (means clients cannot access it). It remains inactive until specifically enabled (server restarts do not change the enabled/disabled state).

- **mailRefProperties element**

```
<!ELEMENT mailRefProperties (el+)>
```

Specifies the set of name/value pairs to associate with a resource reference of type `javax.mail.Session`, as determined by the JavaMail specification.

Some examples include:

```
mail.host
mail.from
mail.user
mail.store.protocol
mail.transport.protocol
mail.debug
mail.[protocol].host (for example, mail.http.host)
mail.[protocol].user (for example, mail.http.user)
```

- **name element**

```
<!ELEMENT name (#PCDATA)>
```

Name of an environment entry, a resource reference, an EJB reference, a role mapping, or a context parameter entry. The name must match the name of an entry of the same type in the deployment descriptor (if it does not, you'll get a warning message during deployment).

- **principalList element**

```
<!ELEMENT principalList (el+)>
```

Specifies the name of the principals to be mapped to the role.

- **recoverable**

```
<!ELEMENT recoverable (#PCDATA)>
```

Specifies whether a component is recoverable. Values are:

```
<recoverable>TRUE</recoverable>  
<recoverable>FALSE</recoverable>
```

Default value is FALSE.

- **resourceEnvReference**

```
<!ELEMENT resourceEnvReference (name, resourceJNDIName)>
```

Specifies the mapping of a resource environment reference to a JNDI name for `javax.jms.Queue` or `javax.jms.Topic` references.

- **resourceEnvReferenceList**

```
<!ELEMENT resourceEnvReferenceList (resourceEnvReference*)>
```

Container element for a list of resource environment references.

- **resourceJNDIName element**

```
<!ELEMENT resourceJNDIName (#PCDATA)>
```

JNDI name for a resource reference. All resource references must contain a JNDI name except for `javax.sql.DataSource` or `java.net.URL` resource references.

The JNDI name can be located on a server other than the one containing the WAR. When accessing a different server, use this syntax:

```
<resourceJNDIName>sssw://server:port/RMI/JNDINameOfResource  
</resourceJNDIName>
```

- **resourceReference element**

```
<!ELEMENT resourceReference (name, (dataSource | resourceURL |  
resourceJNDIName | mailRefProperties))>
```

Container element for resource references. You must specify the resource reference name (from the deployment descriptor) and one of the following:

- A **data source** for references of type `javax.sql.DataSource`
- An **URL** for references of type `java.net.URL`
- A **JNDI name** for references of type `javax.jms.QueueConnectionFactory` or `javax.jms.TopicConnectionFactory`
- A **String array** (to be treated as an array of name/value pairs) for references of type `javax.mail.Session`

- **resourceReferenceList element**

```
<!ELEMENT resourceReferenceList (resourceReference*)>
```

Container element for the set of resource references used by the WAR.
- **resourceURL**

```
<!ELEMENT resourceURL (#PCDATA)>
```

Specifies the URL that will be looked up at runtime for a specific JNDI name.
- **roleLink element**

```
<!ELEMENT roleLink (#PCDATA)>
```

A reference to a defined security role. It must contain the name of one of the role name defined in the roleMap element.
- **roleMap element**

```
<!ELEMENT roleMap (roleMapping+)>
```

Container element for WAR-level role mappings. If the deployment descriptor defines security roles, the deployment plan must map those roles to actual security groups or users on the target server.
- **roleMapping element**

```
<!ELEMENT roleMapping (name, userOrGroupName)>
```

Maps a single role to a user or group name.
- **securityRoleRef**

```
<!ELEMENT securityRoleRef (name, roleLink)>
```

Container element that specifies the mapping of a role reference to a security role.
For every security-role-ref that has no role-link specified in the deployment descriptor, there must be a corresponding securityRoleRef element in the deployment plan. The name element must be the same as the corresponding role-name in the deployment descriptor.
- **securityRoleReferenceList**

```
<!ELEMENT securityRoleReferenceList (securityRoleRef*)>
```

Container element that provides a list of security role mappings.
- **servlet element**

```
<!ELEMENT servlet (servletName, initParamsList)>
```

A servlet that specifies initialization parameters in the deployment descriptor.
- **servletName element**

```
<!ELEMENT servletName (#PCDATA)>
```

The name of the servlet to which the initialization parameters list is to be bound.

- **servletsList element**

`<!ELEMENT servletsList (servlet+)>`

A list of servlets that specify initialization parameters in the deployment descriptor.

- **sessionTimeout element**

`<!ELEMENT sessionTimeout (#PCDATA)>`

Session timeout in minutes. Here is the hierarchy of session timeout precedence, from highest to lowest:

- The timeout set programmatically with `HttpSession.setMaxInactiveInterval()` (in seconds)
- The deployment plan (in minutes)
- The deployment descriptor (in minutes)
- The server's default session timeout

- **urls element**

`<!ELEMENT urls (el+)>`

Location(s) at which the WAR is deployed on the server. The URL is database-relative.

- **userlibJars**

`<!ELEMENT userlibJars (el+)>`

Allows the deployer to list any additional JAR files (found in the `userlib` directory on the server) so they will be made available to the application. The value is a `StringArray` of JAR names, relative to (and contained within) the `userlib` directory.

- **userOrGroupName element**

`<!ELEMENT userOrGroupName (#PCDATA)>`

Name of a principal in a security policy domain or a user group in the operational environment. This is mapped to a role reference name from the deployment descriptor. You can map a role reference to any of the security domains supported by the SilverStream server.

- **usesJars element**

`<!ELEMENT usesJars (el+)>`

List of JARs this WAR uses.

- **value element**

`<!ELEMENT value (#PCDATA)>`

An environment entry value. A value set in the deployment plan overrides a value set in the deployment descriptor.

- **version element**

`<!ELEMENT version (#PCDATA)>`

Specifies the version number of this deployment plan. If it is not present, the version number is assumed to be 0.0. Deployment plans with version 0.0 are associated with SilverStream eXtend Application Server Version 3.7.2 and lower.

- **warJar element**

```
<!ELEMENT warJar (version?, warJarName?, isEnabled?,
  deployToFilesystem?, sessionTimeout?, urls, deployedObject?, usesJars?,
  excludedJSPs?, contextParamsList?, servletsList?,environmentList?,
  beanReferenceList?, resourceReferenceList?, roleMap?)>
```

Container element for the WAR.

- **warJarName element**

```
<!ELEMENT warJarName (#PCDATA)>
```

Name of the WAR file to deploy. You may specify this in the deployment plan or on the command line (when deploying with SilverCmd DeployWAR). Values specified on the command line take precedence.

You can specify a full file path (if the WAR is on disk) or just the name. When you specify just the name and the WAR is not in the current directory, the WAR is assumed to be on the same database and server to which it is being deployed.

- **warJarOptions element**

```
<!ELEMENT warJarOptions (warJar)>
```

Root element of the WAR deployment plan.

RAR deployment plan DTD

This section provides reference information about the SilverStream resource adapter archive deployment plan DTD:

- DTD file
- DOCTYPE statement
- Elements

DTD file The RAR deployment plan DTD is `deploy-rar_1_0.dtd`.

DOCTYPE statement The DOCTYPE statement for the RAR deployment plan is

```
<!DOCTYPE rarJarOptions PUBLIC
  "-//SilverStream Software, Inc.//DTD J2EE RAR Deployment Plan 1.0//EN"
  "deploy-rar_1_0.dtd">
```

Elements The elements defined by the client JAR deployment plan DTD are described below (in alphabetical order):

- **configProperty**

```
<!ELEMENT configProperty (name, value)>
```

Container element for the ManagedConnectionFactory configuration properties.

- **configPropertyList**

```
<!ELEMENT configPropertyList (configProperty+)>
```

Container element for the list of connection pool configuration properties.

- **connectionPool**

```
<!ELEMENT connectionPool (poolName, user, password, xa, minPoolSize?,  
maxPoolSize?, waitTimeout?, idleTimeout?, debug?, configPropertyList?)>
```

Container element for connection pool configuration properties.

- **connectionPoolList**

```
<!ELEMENT connectionPoolList (connectionPool+)>
```

Container element for connection pools associated with the resource adapter.

- **debug**

```
<!ELEMENT debug (#PCDATA)>
```

Specifies whether the connection pool should run in verbose mode. If it is in verbose mode, informational and warning messages are written to the server console. The default is FALSE (not verbose).

- **extractDirectory**

```
<!ELEMENT extractDirectory (#PCDATA)>
```

Specifies the directory (on the deployer's machine). When this element is specified, the contents of the RAR are extracted to this directory.

- **idleTimeout**

```
<!ELEMENT idleTimeout (#PCDATA)>
```

Specifies the number of seconds an unused connection remains in the pool before the server destroys it. The default is 60 seconds. The value you specify will depend on the type of applications that will rely on the connection pool. If you specify shorter timeout periods, then the server might be forced to create connections more often—and creating connections is an expensive operation. But if you specify a timeout that is too long, other applications might be forced to wait for an available connection.

- **isEnabled**

```
<!ELEMENT isEnabled (#PCDATA)>
```

Specifies whether the RAR is enabled (the default) or not. You disable a deployed RAR by setting this flag to FALSE and redeploying the RAR.

- **maxPoolSize**

```
<!ELEMENT maxPoolSize (#PCDATA)>
```

Specifies the maximum number of connections that can be created in the pool.
- **minPoolSize**

```
<!ELEMENT minPoolSize (#PCDATA)>
```

Specifies the minimum number of connections maintained in the pool.
- **name**

```
<!ELEMENT name (#PCDATA)>
```

Specifies the name of the ManagedConnectionFactory.
- **password**

```
<!ELEMENT password (#PCDATA)>
```

Specifies the password for the default connections created in the pool. A default connection is created when users invoke `ConnectionFactory.getConnection()`. This is the preferred method for obtaining a connection, because it allows the server to efficiently pool connections.
- **poolName**

```
<!ELEMENT poolName (#PCDATA)>
```

Specifies the name of the connection pool added to the server. Use this name to administer the connection pool.
- **rarJar**

```
<!ELEMENT rarJar (version?, isEnabled?, rarJarName?,  
resourceAdapterName, extractDirectory?, connectionPoolList?)>
```

Specifies the main RAR element.
- **rarJarName**

```
<!ELEMENT rarJarName (#PCDATA)>
```

Specifies the name of the RAR file to be deployed. If the name is not specified in the deployment plan, it must be entered on the command line.
- **rarJarOptions**

```
<!ELEMENT rarJarOptions (rarJar)>
```

The root element of the RAR deployment plan.
- **resourceAdapterName**

```
<!ELEMENT resourceAdapterName (#PCDATA)>
```

Specifies the name under which the resource adapter will be deployed in the server. This is a logical name.

- **user**
`<!ELEMENT user (#PCDATA)>`
Specifies the user name for the default connections created in the pool. A default connection is created when users invoke `ConnectionFactory.getConnection()`. This is the preferred method for obtaining a connection, because it allows the server to efficiently pool connections.
- **value**
`<!ELEMENT value (#PCDATA)>`
Specifies the value of the property of `ManagedConnectionFactory` identified by the name.
- **version**
`<!ELEMENT version (#PCDATA)>`
Specifies the version number of this deployment plan. If it is not present, the version number is assumed to be 0.0. This value is automatically added when the deployment plan is built using Workbench's Deployment Plan Editor.
- **waitTimeout**
`<!ELEMENT waitTimeout (#PCDATA)>`
Specifies the maximum number of seconds the pool manager attempts to service a connection request. If the connection can not be obtained in the specified number of seconds, an exception is thrown.
- **xa**
`<!ELEMENT xa (#PCDATA)>`
Specifies whether the connections returned by the pool participate in the XA transactions. The default is TRUE.

EAR deployment plan DTD

This section provides reference information about the SilverStream EAR deployment plan DTD:

- DTD file
- DOCTYPE statement
- Elements

DTD file There are two EAR deployment plan DTDs. Use the appropriate one for your version of the SilverStream eXtend Application Server:

Server version	DTD file name
3.7.2 and lower	deploy_ear.dtd
3.7.3 and higher	deploy-ear_1_2.dtd
4.0 and higher	deploy-ear_1_3.dtd

DOCTYPE statement Specify the DOCTYPE statement appropriate for your version of the SilverStream eXtend Application Server:

Server version	DOCTYPE statement
3.7.2 and lower	<code><!DOCTYPE earJarOptions PUBLIC "-//SilverStream Software, Inc.//DTD J2EE EAR Deployment Plan//EN" "deploy_ear.dtd"></code>
3.7.3 and higher	<code><!DOCTYPE earJarOptions PUBLIC "-//SilverStream Software, Inc.//DTD J2EE EAR Deployment Plan 1.2//EN" "deploy-ear_1_2.dtd"></code>
4.0 and higher	<code><!DOCTYPE earJarOptions PUBLIC "-//SilverStream Software, Inc.//DTD J2EE EAR Deployment Plan 1.3//EN" "deploy-ear_1_3.dtd"></code>

Elements The top-level elements defined by the EAR deployment plan DTD are described below (in alphabetical order):

- **alternateDeplDesc element**

```
<!ELEMENT alternateDeplDesc (#PCDATA)>
```

Identifies a deployment descriptor to use for a specified module. Use this element when you do not want to use the deployment descriptor in the specified module's archive. The alternate deployment descriptor that you specify must be in the EAR.

There are two situations when you might want to use this element:

- If you want to use the same module twice but want to configure the two uses differently
- If there are two instances of the module in the EAR (for example, two EJB JARs) and you want to configure the modules differently

- **carJar element**

```
<!ELEMENT carJar (version?, carJarName?, deployAs?, environmentList?,
beanReferenceList?, resourceReferenceList?, usesJars?)>
```

The main application client element.

- **classpathJars element**

```
<!ELEMENT classpathJars (excludeJ2EEXMLJars?, userlibJars?)>
```

This element allows a list of JAR files to be used in an application without deploying them to the server. It also allows the user to disable the inclusion of the default XML JARs that are used to pass the CTS tests.

- **deployAs element**

```
<!ELEMENT deployAs (#PCDATA)>
```

Overrides the name under which the module would normally be deployed. By default, the EAR deployment process creates a deployment name that is a combination of the EAR name and the module name. For EJB modules, this name (if specified) is also used as the base name for the remote JAR. Use with caution.

- **earJar element**

```
<!ELEMENT earJar (version?, earJarName?, moduleList, usesJars?,
roleMap?)>
```

Container element for the list of J2EE modules and a top-level role map.

- **earJarName element**

```
<!ELEMENT earJarName (#PCDATA)>
```

Name of the EAR to deploy. You may specify this in the deployment plan or on the command line (when deploying with SilverCmd DeployEAR). Values specified on the command line take precedence. You can specify a full path (if the EAR is on disk) or just the name. When you specify just the name and the EAR is not in the current directory, the EAR is assumed to already exist in the same database and server to which it is being deployed.

- **earJarOptions element**

```
<!ELEMENT earJarOptions (earJar)>
```

The root element for the EAR.

- **ejbJar element**

```
<!ELEMENT ejbJar (version?, isEnabled?, ejbJarName?,
deployAs?, remoteAccessJar?, deployedObject?, usesJars?,
lenientSecurity?, beansList, roleMap?)>
```

The main EJB element.

- **el element**

```
<!ELEMENT el (#PCDATA)>
```

Element of a string array.

- **excludeJ2EEXMLJars element**

```
<!ELEMENT excludeJ2EEXMLJars (#PCDATA)>
```

Specifies whether the EAR should include the default J2EE 1.3 XML JARs in the classloader for the application. The default JAR files are the specific versions of crimson.jar and xalan.jar that are used in the J2EE 1.3 CTS tests. If this element is set to TRUE, XML parsing and XML transformation classes from crimson.jar and xalan.jar will not automatically be made available to the application. This permits application developers to package (or include via userlibJars) other versions of XML parsers and transformation engines in their applications. If this element is absent or set to FALSE, the default JARs will automatically be made available to the application.

- **module element**

```
<!ELEMENT module (ejbJar | warJar | carJar)>
```

Describes a specific J2EE module in the EAR.

The information contained in this element is the deployment plan for the specific module. You should cut and paste the deployment plan from the application client JAR, EJB JAR, or WAR to complete this section.

For EJBs, the beanName components of the bean element must be unique within the EAR or you will encounter errors when deploying the EAR (because the bean names are used to resolve bean references within an EAR).

- **moduleList element**

```
<!ELEMENT moduleList (module+)>
```

Container element for one or more J2EE deployable modules.

- **name element**

```
<!ELEMENT name (#PCDATA)>
```

The name of a role reference that must be mapped to a user or group name. It must match the name of a role reference entry in the deployment descriptor (if it does not, a warning is generated at deployment).

- **order element**

```
<!ELEMENT order (#PCDATA)>
```

Identifies the order of deployment of the modules. The smaller the number the higher its deployment priority. Modules without the order element are deployed after ordered modules.

- **principalList element**

```
<!ELEMENT principalList (el+)>
```

The principalList element specifies the name of the principals to be mapped to a role.

- **rarJar element**

```
<!ELEMENT rarJar (version?, isEnabled?, rarJarName?,  
resourceAdapterName, extractDirectory?, connectionPoolList?)>
```

The main RAR element.

- **roleMap element**

```
<!ELEMENT roleMap (roleMapping+)>
```

Container for EAR-level role mappings.

You might consider mapping security roles at the EAR level and not within individual modules (EJB JARs, WARs, and so on). This allows you to combine and simplify the security settings for the constituent J2EE modules.

Here are the basic rules:

- If the individual modules do not contain any role maps, the EAR-level role map is used
- If some or all individual modules and the EAR both contain a role map:

And the roles are unique, the role map used for each module is a UNION of the EAR-level role map and the module

And one or more of the roles are not unique, the module-level role map takes precedence for the duplicate role and the unique roles are added to the role map

This example illustrates how a role map is determined when roles are not unique. If the EAR-level role map contains the following values:

This role	Is mapped to this userOrGroupName
EJBAdmin	Zack
User	Mary

and an EJB module within the EAR contains this role map:

This role	Is mapped to this userOrGroupName
Admin	Joe
User	Helen

then the role map used for the EJBs will look like this:

This role	Is mapped to this userOrGroupName
Admin	Joe
User	Helen
EJBAdmin	Zack

In this case, the EJB module's User role takes precedence. The Admin role is added to the EJB's role mapping, because it is inherited from the EAR's role map.

- **roleMapping element**

```
<!ELEMENT roleMapping (name, userOrGroupName)>
```

Maps a single role to a user or group name.

- **userOrGroupName element**

```
<!ELEMENT userOrGroupName (#PCDATA)>
```

The name of a principal in a security policy domain or a user group in the operational environment. This is mapped to a role reference name from the deployment descriptor. You can map a role reference to any of the security domains supported by the SilverStream server.

- **userLibJars element**

```
<!ELEMENT userlibJars (el+)>
```

Lists any additional JAR files that should be made available to the application. The JARs must reside in the server's userlib directory. The value is a StringArray of JAR names, relative to (and contained within) the userlib directory.

- **usesJars element**

```
<!ELEMENT usesJars (el+)>
```

List of JARs used by this EAR.

- **version element**

```
<!ELEMENT version (#PCDATA)>
```

Specifies the version number of this deployment plan. If it is not present, the version number will be assumed to be 0.0. Deployment plans with version 0.0 are associated with SilverStream eXtend Application Server Version 3.7.2 and lower.

- **warJar element**

```
<!ELEMENT warJar (version?, warJarName, deployAs?, isEnabled?,
deployToFilesystem?, sessionTimeout?, urls?, deployedObject?, usesJars?,
excludedJSPs?, contextParamsList?, servletsList?, environmentList?,
beanReferenceList?, resourceReferenceList?, roleMap?)>
```

The main WAR element.

4 SilverCmd Reference

This chapter describes the SilverCmd commands. It describes the purpose, syntax, and arguments for each command, along with how to run SilverCmd—plus associated security and authentication issues. It includes these sections:

- Command locator
- About SilverCmd
- Alphabetical list of commands

NOTE Not all SilverCmds listed in this reference are included with all editions of the SilverStream eXtend Application Server.

Command locator

Click a command to display complete information:

Command	Description
AddCP	Adds a connection pool to the server
AddDatabase	Deprecated. Registers a database with the specified server
Build	Deprecated. Compiles SilverStream application components
BuildWAR	Deprecated. Creates and deploys a J2EE-compatible Web application archive from a local directory
ClearDefaultURL	Clears a database or server default URL
ClearLog	Clears records from the HTTP log, the error log, or the trace log
ComGen	Deprecated. Generates COM-access classes from a COM Typelib
ConvertEJB	Deprecated. Converts an EJB1.0-compatible JAR file to an EJB1.1-compatible JAR file
CreatePackage	Deprecated. Creates Java packages in the Objects directory
Delete	Deprecated. Deletes application components from the server

Command	Description
DeployCAR	Deploys a J2EE-compatible client application archive to a server
DeployEAR	Deploys a J2EE-compatible Enterprise application archive (EAR) to a server
DeployEAR12	Deprecated. Deploys a J2EE 1.2 EAR to a server
DeployEJB	Deploys a J2EE 1.2 or 1.3 EJB JAR to a server
DeployEJB11	Deprecated. Deploys an EJB1.1 JAR to a server
DeployRAR	Deploys a J2EE-compatible RAR file to the specified SilverStream server
DeployWAR	Deploys a J2EE-compatible Web archive (WAR) to a SilverStream server
ExportSource	Deprecated. Copies the Java source file for a business object or a package to the local file system
GetConsole	Redirects server console output to the local terminal
GetDefaultURL	Displays the default URL for a database or server
ImportClass	Deprecated. Imports class files to the server as business objects
ImportMedia	Deprecated. Imports a media object (such as an image or a sound) to a server
ImportPage	Deprecated. Imports a static HTML page to a server, optionally associating a set of URLs with the page
ImportSource	Deprecated. Copies a Java source file to the Objects directory of a server
ListCP	Lists the active connection pools on a server
ModifyCP	Modifies a subset of the configuration properties for a connection pool
ModifyTableList	Deprecated. Modifies the set of database tables available to the SilverStream server for a specific database
Prefs	Updates various compiler settings for the preferences file

Command	Description
PrintLog	Displays records from the HTTP log, error log, or trace log
Publish	Deprecated. Publishes items from one database or server to another
PublishFromFile	Deprecated. Copies items from a location on disk to a server and database
PublishToFile	Deprecated. Copies Java source files for any SilverStream application component (such as forms, views, and media) and its design metadata (if desired) from a server to the local file system Use with PublishFromFile when you want to copy the application components—as a complete unit—from one SilverStream server to another. The resulting file format is only usable with PublishFromFile
QueryCP	Displays configuration properties for a connection pool
RebuildJAR	Deprecated. Rebuilds SilverStream-generated JAR files
RemoveCP	Removes a connection pool from a server
RemoveDatabase	Deprecated. Removes a database from a server's list of accessible databases
ServerState	Tests whether a server is running or shuts down the server
SetDefaultURL	Sets the default URL for a database or server
SetSecurity	Sets Read, Write, Protect, Select, and Execute security on application objects
SetUserGroupInfo	Creates, deletes, and sets properties for SilverStream users and groups
SourceControl	Deprecated. Performs source control tasks
Undeploy	Undeploys any of the following J2EE archive types: EAR, EJB, RAR, or WAR
ValidateEAR	Validates the deployment descriptor within an EAR

Command	Description
ValidateEJB	Validates an EJB JAR for correctness
ValidateEJB11	Validates an EJB 1.1 JAR for correctness

About SilverCmd

SilverCmd provides a way to perform SilverStream operations from the command line. You can use SilverCmd to automate many of the tasks associated with managing the components of a SilverStream server.

Separate ports The SilverStream server lets you define separate runtime, design, and administration ports for different types of users and operations. Some commands require you to specify the **design** port, and others require you to specify the **administration** port. Specifying an inappropriate port will result in a security error code. When necessary, the reference section for each command lists the type of port you must specify.



For more information about using separate ports, see the chapter on running the server in the *Administrator's Guide*.

Running SilverCmd

SilverCmd is located in the server's bin directory. If you will be using SilverCmd frequently, consider adding the server's \bin directory to your system path for convenience.

Authentication If your SilverStream server is running in a restricted production environment (as opposed to an unrestricted design environment), you will need to authenticate yourself using the -U and -P options to run commands that access the server.

Running SilverCmd from the command prompt To run SilverCmd from the command prompt, use this syntax:

```
SilverCmd command arguments
```

To display the list of commands type:

```
SilverCmd -?
```

```
OR
```

```
SilverCmd -h
```

To display the usage for a particular command use:

```
SilverCmd command -?
```

Normally when SilverCmd encounters an error, it stops execution, generates detailed error messages explaining the failure, and displays the messages in the command prompt window. If you specify `-i`, SilverCmd ignores the errors and continues execution. Here's how you use the `-i` option:

```
SilverCmd command -i
```

Running SilverCmd in execute mode You can run one or more SilverCmds from a file. This is called **execute mode**. To run in execute mode, use this syntax:

```
SilverCmd Execute command-file
```

- **Command-file format** The command file is a text file, and it must be structured so that each command is on its own line and contains the appropriate arguments. Do not specify the SilverCmd keyword on each line in the command file—you specify SilverCmd only at the command line. For example:

```
ClearLog localhost:80 -E -U myusername -P mypassword
Undeploy localhost:80 SilverMaster40 myEAR -U myusername -P mypassword
RemoveCP localhost:80 myPool -U myusername -P mypassword
```

The file can have any extension, but the you may want to use `.SCD`—because the SilverStream installation program automatically creates a file association for files with an `.SCD` extension for these files.

The commands execute in the order in which they appear in the file. Running in execute mode is like repeatedly calling SilverCmd, except that when you run in execute mode you can avoid any performance penalty associated with starting SilverCmd repeatedly.

- **Using the `i` option in execute mode** When you use the `-i` option in execute mode, you can ignore errors from any one command and proceed with the command that follows in the command-file. To use `-i`, specify it after the command-file name, like this:

```
SilverCmd Execute command-file -i
```

A subset of commands also allow you to specify the `-i` option. When used this way, `-i` means to continue on error within the set of operations of that command—for example:

```
SetSecurity localhost mydb -f myfile.xml -i
```

- **Using the -U and -P options** The -U and -P options specify a user name and password combination for SilverStream server authentication. When the server is running in a restricted production environment, you must be authenticated to run commands that access the server.

When you run SilverCmd in execute mode, you can run all of the command in the command file as a single authenticated user by specifying the -U and -P options after the command file, like this:

```
SilverCmd Execute command-file -U myusername -P mypassword
```

You can also run each command as a different authenticated user by specifying the -U and -P options with each command like this:

```
ClearLog localhost:80 -E -U myusername -P mypassword  
Undeploy localhost:80 SilverMaster40 myEAR -U myusername -P mypassword  
RemoveCP localhost:80 myPool -U myusername -P mypassword
```

When you do not specify the -U and -P for each command, then the one specified at the command line is used.



For more information about supplying these values for each supported security realm, see the chapter on setting up security in the *Administrator's Guide*.

Logging messages to a file By default, SilverCmd logs informational messages, warnings, and errors to the command window. You can write the messages to a file using the standard redirect symbol (>). For example:

```
SilverCmd Execute command-file -i > SilverCmd.log  
SilverCmd RebuildJAR localhost MyApp myJar > SilverCmd.log
```

Permission to write temporary files when deploying The SilverCmd deployment commands (such as DeployEAR) generate temporary files on disk. These files are created in the server's installation directory, unless you have defined a HOME environment variable. If you have a HOME variable, the temporary files are created in %HOME%\silverstream. So if you have a HOME environment variable defined, it must point to a reachable and writable location in order to deploy successfully.

Specifying values in input files and deployment plans

Some commands require an input file or deployment plan (specified using the `-f` option); for other commands, the `-f` option is optional and provided as a convenience. For example:

Command	Description
AddDatabase	Requires you to supply the database name, the database type, the user name, the password, and the JDBC driver in an input file
Prefs	Does not require an input file; you can specify a single preference at the command line, or you can specify a list of preferences within an input file
Undeploy	Does not take an input file and will generate an error message if you attempt to use it

Input file and deployment plan format Input files and deployment plans must be in XML format and must include a DOCTYPE statement. You do not need to be an XML expert to create input files—there are sample XML files for many of the commands, and you can copy and paste the required DOCTYPE statement from the appropriate sample into your own XML input file:

Sample file	Location
The DTD for each input file or deployment plan	The server's <code>\Resources\DTDCatalog</code> directory
The XML sample for each file	The server's <code>\samples\SilverCmd</code> directory

The samples and DTDs are self-documenting. See them for the most up-to-date requirements.

Command line versus input file For commands where values can be specified at both the command line and within an input file or deployment plan, values specified at the command line override input file settings.

Alphabetical list of commands

AddCP

Adds a connection pool to the SilverStream server.

Server permissions	DTD and sample input file
Modify server configuration	None

Syntax

```
SilverCmd AddCP server[:port] poolName poolTypeFlag dataSourceOptions
[options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the name of the target server and the administration port
<i>poolName</i>	Specifies the logical name for the pool
<i>poolTypeFlag</i>	Specifies the type of pool to create. Values are: -J—to create a JDBC connection pool -C—to create a Connector connection pool
options	Specifies any operating criteria for the command

The valid options for all pool types are:

Option	Description
-? or -h	Displays the usage message
-A <i>username</i> and -W <i>password</i>	Specifies user name and password for connection pool resource manager authentication

Option	Description
<i>-m minconn</i>	The minimum number of connections. The pool manager will attempt to maintain this minimum number of transactions (this is a soft limit)
<i>-t timeout</i>	The idle timeout in seconds. The default is 60 seconds. When set to -1, idle timeout is disabled and no idle connections are ever closed
<i>-U username</i> and <i>-P password</i>	Specifies user name and password for SilverStream authentication
<i>-w timeout</i>	The connection wait timeout in seconds. The default is 30 seconds. When set to -1, clients are forced to wait until a connection becomes available
<i>-v log level</i>	Specifies the logging level. The logging levels are: <ul style="list-style-type: none"> • 0 - logging disabled • 1 - basic ConnectionFactory operations and settings • 2 - level 1 plus detailed output from connection pool manager • 3 - level 2 plus exception stack traces and logging information from underlying JDBC driver or Connector resource adapter
<i>-N</i>	When used, the connections returned by the pool are not enlisted in XA transactions
<i>-x max conn</i>	The maximum number of connections allowed by the pool. The default is 10. Use -1 to create a pool with no maximum

Adding JDBC1.0 connection pools To create a data source for a JDBC1.0 connection pool, you can specify either the JDBC driver class name or the LDS Key.

To specify the **JDBC driver class** name, use these options:

Option	Description
<i>-d driver</i>	Specifies the fully qualified name of your JDBC driver class

Option	Description
<i>-j url</i>	Specifies the JDBC URL string defined by the driver vendor to connect to your database
<i>-a attributes</i>	(Optional) Specifies any additional URL attributes defined by the vendor that you can use to customize the driver connection. For example: <code>cache=100</code>

To specify the **LDS Key**, use these options:

Option	Description
<i>-l lds key</i>	Specifies the LDS key
<i>-j url</i>	Specifies the JDBC URL string defined by the driver vendor to connect to your database
<i>-a attributes</i>	(Optional) Specifies any additional URL attributes defined by the vendor that you can use to customize the driver connection. For example: <code>cache=100</code>

Adding JDBC2.0 connection pools To create a data source for a JDBC2.0 connection pool, you can specify the JDBC LDS Key, the CPDS class name, or the XADS class name.

To specify the **LDS key**, use these options:

Option	Description
<i>-l lds key</i>	Specifies the LDS key
<i>-p properties</i>	Specifies the data source configuration properties The format for these properties is: <code>name=value</code> For example: <code>name1=value1, name2=value2, name3=value3, . . .</code> For names and property values, see your driver documentation


To specify the CPDS class name, use this option:

Option	Description
-k <i>class name</i>	Specifies the fully qualified Connection Pool DataSource class name

To specify the XADS class name, use this option:

Option	Description
-g <i>class name</i>	Specifies the fully qualified name of the XA DataSource class

Adding connector connection pools To create a data source for a connector connection pool, use these options:

Option	Description
-r <i>adapter</i>	Specifies the resource adapter name
-p <i>properties</i>	Specifies the properties for the ManagedConnectionFactory using the format: <p style="text-align: center;"><i>name=value</i></p> For example: name1=value1, name2=value2, name3=value3, . . .  See your Resource Adapter documentation for these values

AddDatabase

Classic use only. This command is available for use with SilverStream Classic applications only.

Deprecated. Registers a SQL database with the specified SilverStream server.

Server permissions	DTD and sample input file
Modify server configuration	DTD: add_database.dtd Sample: add_database_sample.xml

Syntax

```
SilverCmd AddDatabase server[:port] -f file [options]
```


The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the name of the target server and the administration port
<i>-f file</i>	Specifies the input file containing the database and connection information If <i>-s</i> is also specified, the XML file must contain a System Tables node describing the connection information for the database containing the system tables
options	Specifies any operating criteria for the command

The valid options are:

Option	Definition
<i>-s</i>	Indicates that you want to store SilverStream's system tables in a different database from the one you are adding If you want to store the system tables in a different database, the input file must include a System Tables node that specifies the database name and connection information for the different database
<i>-U username</i> and <i>-P password</i>	Specifies user name and password for SilverStream authentication
<i>?</i>	Displays the usage message

AddDatabase input file The AddDatabase command requires an input file. The input file includes the following entries:

Entry	Description
<i>Database name</i>	Specifies the fully qualified database name
<i>Username and Password</i>	<p>Specifies the user account used by the SilverStream server when accessing this database</p> <p>The account must have read/write permissions for the database</p>
<i>Platform</i>	<p>Specifies the vendor name for the database—for example: Oracle, DB2, or Sybase System 11</p> <p> For more information, see “Valid database connection types” on page 108</p>
<i>Driver set</i>	<p>Specifies the driver type for the database</p> <p>Each database type has a default connection type that SilverStream assumes if you do not specify a value</p> <p>If you are using a value that is not in the list of database type/connection type values, you must specify Other JDBC Driver plus:</p> <ul style="list-style-type: none"> • The fully qualified package name for the JDBC driver • The JDBC URL that tells the driver where to connect to the specified database • The URL attributes (which include properties like cache size) <p>This syntax is driver-dependent</p>

The following nodes are optional:

Entry	Description
<i>Table list</i>	<p>Specifies a subset of the tables that should be made available to the SilverStream server</p> <p>You can modify this value later using the <code>ModifyTableList</code> command or the SilverStream Designer</p> <p>The table list can include:</p> <ul style="list-style-type: none"> • A list of table names (which must exactly match the names in the database) • A list of table name patterns <p>NOTE When you specify a pattern, you can use the % symbol to match any number of characters, and the underscore (_) character to specify that you want to match any one character</p> <ul style="list-style-type: none"> • A combination of names and patterns
<i>System tables</i>	<p>Stores SilverStream's system tables in a different database from the one you are adding</p> <p>This section must include the following information about the database where you want to store the system tables: the database, the user name, the password, the database platform, and the driver set or the LDS key</p> <p>This entry is required when you specify the <code>-s</code> argument at the command line; otherwise, it is ignored</p>

Valid database connection types To connect to a database, the SilverStream server needs **either** of the following:

- (a) The combination of:
 - **Platform name.** Identifies the vendor and version of the database. For example, Oracle 7 or Microsoft SQL Server 7.
 - AND
 - **Driver set.** Identifies the database driver. It must include the package name so the server can locate it.
- OR**
- (b) **LDS key.** The logical data source key. You can use this in place of the platform name and driver set.

SilverStream has defined strings that resolve to these values. See `add_database_sample.xml` in the server's `\samples\SilverCmd` directory for the complete listing of valid values.

Build

Classic use only. This command is available for use with SilverStream Classic applications only.

Deprecated. Compiles the items in an entire database, a directory within the database, or a specified item within a directory. You can only build the items located in the Forms, Pages, Views, and Objects directories.

The Build command:

- Copies the specified item (or set of items) from the server to the following directory on the local disk: ***SilverStream/compilecache/server/database/source***
- Compiles the objects (using the Java compiler specified in the Designer's preferences file). Generates warning messages for items that cannot be compiled
- Saves (or uploads) the successfully compiled items (class files) back to the server

Server permissions	DTD and sample input file
Read/Write	DTD: itemlist.dtd Sample: build_sample.xml

Syntax

```
SilverCmd Build server[:port] database [item] [options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the name of the target SilverStream server and the design-time port
<i>database</i>	Specifies the name of the target SilverStream database

Argument	Description
<i>item</i>	Specifies the name of a single item to build. It must include the SilverStream directory. For example: <code>SilverCmd Build localhost MyApp Forms/frmAccountType</code>
options	Specifies operating criteria for the command

The valid options are:

Option	Definition
-?	Displays the usage message
-U <i>username</i> and -P <i>password</i>	Specifies user name and password for SilverStream authentication
-c	Removes any class files from the compile cache directory on the local machine before executing the current command. If you are building the Forms directory of your Examples database, SilverCmd removes the class files from the compile cache directory under: <code>\host\database\classes\com\sssw\gen\forms</code>
-f <i>file</i>	Specifies an input file that contains the list of items to build The input file format is based on the ItemList DTD The file name must include a string array of item names. The file can have any extension and can reside in any directory that is accessible to SilverCmd. For example, you can specify: <code>SilverCmd Build localhost MyApp -f c:\build_sample.xml</code>
-i	Continues on error

BuildWAR

Deprecated. Constructs a Web archive (WAR) file using the contents of a specified directory on the local file system (including files and subdirectories). It also constructs the web.xml deployment descriptor and by default adds an entry for each JSP page in the directory. You can optionally provide a more detailed web.xml that may include information about any servlets or tag libraries in the WAR. When you provide a web.xml, you must also include the information about the JSP pages.

You can use the `-n` option to specify the location where BuildWAR should create the WAR file. When you do not specify `-n`, the WAR is created in the TEMP directory. It is named `agwarXXXX.war` (where `XXXX` is any four digits).

You can use the `-d` option to deploy the WAR to a SilverStream server after the WAR is constructed. When you specify `-d`, the server and database arguments are required along with the `-u` option and one or more URLs. If you do not specify the `-u` option, you must supply them in a deployment plan file using the `-f` option.

Server permissions	DTD and sample input file
Write (when used to deploy to the server)	DTD: <code>deploy_war.dtd</code>

Syntax

```
SilverCmd BuildWar server[:port] database rootDir [options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the name of the target SilverStream server and the design-time port Optional unless <code>-d</code> is specified
<i>database</i>	Specifies the name of the target database Optional unless <code>-d</code> is specified
<i>rootDir</i>	Specifies the directory of files to be placed in the JAR
<i>options</i>	Specifies any operating criteria for the command

The valid options are:

Option	Description
-?	Displays the usage message
-U <i>username</i> and -P <i>password</i>	Specifies the user name and password for SilverStream server authentication
-u <i>baseURLList</i>	One or more base URLs for the WAR They should be specified in a semicolon-separated list (for example: url1;url2)
-d	Specifies that once the WAR is created, it should be deployed on the target SilverStream server and database Server and database arguments are required when -d is specified
-n	Specifies the name for the WAR that is created If not specified, a WAR is created in the temp directory and is named AgwXXXX.war, where XXXX is any four digits
-x	Specifies that the web.xml file already exists in the web-inf directory and should not be overwritten
-o	Specifies that an existing WAR of the same name on the server should be overwritten with the current WAR
-f	Specifies the name of the SilverStream deployment plan file

ClearDefaultURL

Clears the default URL for the server or a database.

Server permissions	DTD and sample input file
Modify server settings	None

Syntax

```
SilverCmd ClearDefaultURL server[:port] [database] [options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the name of the target server and the administration port
<i>database</i>	Specifies the SilverStream database whose default URL you want to clear. If the database is not specified, ClearDefaultURL clears the server's default URL
options	Specifies operating criteria for the command

The valid options are:

Option	Definition
-? or -h	Displays the usage message
-U <i>username</i> and -P <i>password</i>	Specifies user name and password for SilverStream authentication

ClearLog

Removes records from the HTTP log, the error log, or the trace log. ClearLog can only delete records when server logging output is specified as database (not file or user defined).



For more information on specifying logging output using the SMC, see the chapter on running the server in the *Administrator's Guide*.

Server permissions	DTD and sample input file
Modify server configuration	None

Syntax

```
SilverCmd ClearLog server[:port] logTypeFlags [options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the name of the target server and the administration port
<i>logTypeFlags</i>	Values are: -E—Removes records from the error log -H—Removes records from the HTTP log -T—Removes records from the trace log You can specify any combination in a space or comma separated list
options	Specifies operating criteria for the command

The valid options are:

Option	Definition
-? or -h	Displays the usage message
-U <i>username</i> and -P <i>password</i>	Specifies user name and password for SilverStream authentication

ComGen

Deprecated. Generates COM-access classes from a COM Typelib.

Server permissions	DTD and sample input file
None	None

Syntax

```
SilverCmd ComGen typelib_path [options]
```

The valid arguments are:

Argument	Description
<i>typelib_path</i>	The type library for the COM component. The type library can be a separate TLB file or a resource that is linked into the DLL or EXE
options	Specifies operating criteria for the command

The valid options are:

Option	Description
-? or -h	Displays the usage message
-l	Lists all registered type libraries on your system This option does not generate any code
-d <i>dir</i>	Output directory The output directory is the root directory under which the package directory tree is created
-p <i>pkg</i>	Output package The Java package name for the generated classes

ConvertEJB

Deprecated. Converts EJB 1.0 serialized deployment descriptors to EJB 1.1 XML deployment descriptors.

Server permissions	DTD and sample input file
None	None

Syntax

```
SilverCmd ConvertEJB OldEjbJarPath [NewEjbJarPath] [options]
```

The valid arguments are:

Arguments	Description
<i>OldEjbJarPath</i>	Specifies the path for the existing EJB 1.0 JAR file
<i>NewEjbJarPath</i>	Specifies the path for newly created EJB 1.1 JAR file
options	Specifies operating criteria for the command

The valid options are:

Option	Description
-?	Displays the usage message
-v	Prints verbose output to the console window

CreatePackage

Classic use only. This command is available for use with SilverStream Classic applications only.

Deprecated. Creates a Java package of the specified name in the Objects directory of the target database.

Server permissions	DTD and sample input file
Write	None

Syntax

```
SilverCmd CreatePackage server[:port] database package [options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the name of the target SilverStream server and the design-time port
<i>database</i>	Specifies the name of the target database

Argument	Description
<i>package</i>	Specifies the name of the package to create
options	Specifies operating criteria for the command

The valid options are:

Option	Definition
-? or -h	Displays the usage message
-U <i>username</i> and -P <i>password</i>	Specifies a user name and password for SilverStream authentication

This example creates the package com.myco.foo in the mydb database on the localhost server:

```
SilverCmd CreatePackage localhost mydb com.myco.foo
```

Delete

Classic use only. This command is available for use with SilverStream Classic applications only.

Deprecated. Deletes the specified component(s) from the specified database and server.

NOTE Do not use Delete to remove deployed J2EE archives. Use Undeploy.

Server permissions	DTD and sample input file
Write	DTD: itemlist.dtd Sample: None

Syntax

```
SilverCmd Delete server[:port] database [item] [options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the name of the target SilverStream server and the design-time port
<i>database</i>	Specifies the name of the target database
<i>item</i>	<p>Specifies the name of the item to delete:</p> <ul style="list-style-type: none"> To delete a form, page, or view, specify: <i>dir/item</i> where <i>dir</i> is Forms, Pages, or Views To delete a business object, specify: <i>Objects/packageName/item</i> To delete a media item, specify: <i>Media/dir/item</i> where <i>dir</i> is JARs, Images, Sounds, or General To delete multiple items, use <i>-f file</i>
options	Specifies operating criteria for the command

The valid options are:

Option	Description
<i>-?</i> or <i>-h</i>	Displays the usage message
<i>-f file</i>	<p>Specifies an input file that contains the list of items to delete</p> <p>The input file format is based on the ItemList DTD</p> <p>The file name must include a string array of item names. The file can have any extension and can reside in any directory accessible to SilverCmd</p>
<i>-U username</i> and <i>-P password</i>	Specifies the user name and password for SilverStream authentication
<i>-r</i>	Recursively deletes children of packages

DeployCAR

Deploys a J2EE-compatible application client archive file to the specified SilverStream server.

The archive file must comply with the J2EE specification and must contain a manifest file that includes a Main-Class entry.

The deployed object is given the same name as the archive unless you specify the -d option.

Server permissions	DTD and sample input file
Write to the EJB JARs & Media directory of the target database	DTD: deploy_car.dtd

Syntax

```
SilverCmd DeployCAR server[:port] database jarfile
[-f deployment_plan] [options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the name of the target SilverStream server and the design-time port
<i>database</i>	Specifies the name of the target SilverStream database
<i>jarfile</i>	Specifies the name (and path) of the client application archive to deploy
<i>options</i>	Specifies any operating criteria for the command

The valid options are:

Option	Description
-? or -h	Displays the usage message
-d	Specifies an alternate name for the deployed object Use this option when you want to use the same client archive for multiple deployments

Option	Description
<i>-f deployment plan</i>	Specifies the name (and path) of the deployment plan If the location is not specified, DeployCAR looks in the CAR's META-INF/silverstream.xml file by default A deployment plan is only necessary when the application references EJBs or external resources
<i>-o</i>	Overwrites an existing deployed object of the same name
<i>-U username</i> and <i>-P password</i>	Specifies user name and password for authentication by the server
<i>-v verbose-level</i>	Specifies the level of messages to output. Values for verboseLevel are 0 for no messages (default) to 5 for the most messages

DeployEAR

Deploys an enterprise archive file (EAR) to a SilverStream server. You can use this to deploy EARs of any supported J2EE version (unlike SilverCmd DeployEAR12, which deploys J2EE 1.2 EARs only).

DeployEAR performs these tasks:


1. Opens the EAR file and extracts all files to a local temporary directory.
2. Validates any EJBs (by calling SilverCmd ValidateEJB).
3. Examines the deployment plan's DOCTYPE statement and performs the appropriate SilverCmd DeployXXX corresponding to each archive and DOCTYPE. See "Deployment Plan DTDs" on page 49 for information about DOCTYPE statements.
4. Passes the *-o* flag (if specified) to each of the DeployXXX commands:
 - You might **want** to specify *-o* if you are redeploying an existing EAR.
 - You might **not want** to specify *-o* if you are deploying an EAR name which has not previously been deployed. This ensures that you will get an error if there's one there already that you might not want to overwrite.

Server permissions	DTD and sample input file
Write to the EJB JARs & Media directory of the target database	DTD: See "EAR deployment plan DTD" on page 88 for more information

Syntax

```
SilverCmd DeployEAR server[:port] database [EARFile]
-f deploymentPlan [options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the name of the target SilverStream server and the design-time port
<i>database</i>	Specifies the name of the target SilverStream database
<i>EARFile</i>	Specifies the name and location (on disk) of the EAR file to deploy This value can also be specified in the deployment plan (specified with -f). When specified in both places, the command-line value is used
<i>-f deploymentPlan</i>	Specifies the name and location (on disk) of the XML-based deployment plan  For more information on the structure of this file, see Chapter 3, “Deployment Plan DTDs”
<i>options</i>	Specifies any operating criteria for the command

The valid options are:

Option	Description
<i>-? or -h</i>	Displays the usage message
<i>-i</i>	Ignores errors when compiling JSP pages and deploys whatever builds successfully
<i>-l</i>	Specifies the name for the deployed EAR object on the SilverStream server By default, the EAR name is used

Option	Description
-n	Specifies that EJB validation should be skipped during deployment. If not present, SilverCmd ValidateEJB is executed before the EAR is deployed
-o	If a deployed object (or remoteJar object for 1.2 deployments only) already exists on the server, this flag forces that object to be overwritten
-t	Used for debugging JSP pages using non-Latin-1 character sets: if specified, the JSP compiler outputs into the compile cache an additional Java file with the extension -local (for example, along with date_jsp_XXXXXXXXXX.java, you will also find date_jsp_XXXXXXXXXX-local.java). The version of the file with -local is in the machine's local character set, instead of UTF-8 By default, you will find these files in compilecache/server/database/temp/sources//archive/com/sssw/gen/jsps
-U <i>username</i> and -P <i>password</i>	Specifies user name and password for authentication by the server
-v <i>verbose-level</i>	The level of messages to write to the SilverCmd console window. Values for verboseLevel are 0 for no messages (default) to 5 for the most messages

DeployEAR12

Deprecated. Deploys an EAR file for J2EE 1.2 to a SilverStream server. DeployEAR performs these tasks:

1. Opens the EAR file and extracts all of the files to a local temporary directory.
2. Passes the `-o` flag (if specified) to each of the DeployXXX commands:
 - You might **want** to specify `-o` if you are redeploying an existing EAR.
 - You might **not want** to specify `-o` if you are deploying an EAR name which has not previously been deployed. This ensures that you will get an error if there's one there already that you might not want to overwrite.


Server permissions	DTD and sample input file
Write to the EJB JARs & Media directory of the target database	DTD: <code>deploy-ear_1_2.dtd</code> Sample: <code>deploy_ear_sample.xml</code>

Syntax

```
SilverCmd DeployEAR12 server[:port] database [EARFile 1.2 File]
-f deploymentPlan [options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the name of the target SilverStream server and the design-time port
<i>database</i>	Specifies the name of the target SilverStream database
<i>EAR 1.2 File</i>	Specifies the name and location (on disk) of the EAR file to deploy This value can also be specified in the deployment plan. When specified in both places, the deploymentPlan value takes precedence

Argument	Description
<i>-f deploymentPlan</i>	Specifies the name and location (on disk) of the XML-based deployment plan  For more information on the structure of this file, see Chapter 3, “Deployment Plan DTDs”
options	Specifies any operating criteria for the command

The valid options are:

Option	Description
<i>-? or -h</i>	Displays the usage message
<i>-i</i>	Ignores errors when compiling JSP pages and deploys whatever builds successfully
<i>-l</i>	Specifies the name for the deployed EAR object on the SilverStream server By default, the EAR name is used
<i>-o</i>	If a deployedObject or remoteJar object of the same name already exists on the server, this flag forces that object to be overwritten
<i>-t</i>	Used for debugging JSP pages using non-Latin-1 character sets: if specified, the JSP compiler outputs into the compile cache an additional Java file with the extension <i>-local</i> (for example, along with <i>date_jsp_XXXXXXXXXX.java</i> , you will also find <i>date_jsp_XXXXXXXXXX-local.java</i>). The version of the file with <i>-local</i> is in the machine's local character set, instead of UTF-8 By default, you will find these files in <code>compilecache/server/database/temp/sources//archive/com/sssw/gen/jsps</code>
<i>-U username</i> and <i>-P password</i>	Specifies user name and password for authentication by the server

DeployEJB

Deploys an EJB JAR on the specified SilverStream server.

You can use DeployEJB to deploy EJBs to either the J2EE 1.2 or J2EE 1.3 container (unlike DeployEJB11, which only deploys EJBs to the 1.2 container). The EJB is deployed to the appropriate container based on the DOCTYPE of the deployment plan. See “Deployment Plan DTDs” on page 49 for information about which DOCTYPE statement corresponds to each EJB container.

DeployEJB performs these tasks:

- Validates the EJB JAR (by calling SilverCmd ValidateEJB)
 - Examines the DOCTYPE statement to determine the container to which to deploy the EJB
 - Creates implementation classes for interfaces, and generates wrapper classes that handle security and transactions
 - For deployment to the 1.3 container, generates a deployed object and uploads it to the server
 - If the ejb-client-jar element is present in the deployment descriptor, generates stub classes and puts them in the ejb-client-jar and uploads it to the server
- NOTE** For external Java clients, you must manually copy this ejb-client-jar to each client that needs to access the deployed EJBs
- For deployment to the 1.2 container, processes the EJB JAR the same way as DeployEJB11
 - Enables the deployedObject for client access when allowed by the deployment plan

Server permissions	DTD and sample input file
Write to the EJB JARs & Media directory of the target database	<p>DTD: For deployment to the EJB 2.0 container, use <code>deploy-ejb_2_0.dtd</code>. For deployment to the EJB 1.1 container, use <code>deploy-ejb_1_1.dtd</code> or <code>deploy-ejb.dtd</code></p> <p>Sample: For how to use the <code>deploy-ejb.dtd</code>, see <code>deploy_ejb_sample.xml</code></p>

Syntax

```
SilverCmd DeployEJB server[:port] database [EJBFile] [-f deploymentPlan]
[options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the name of the target SilverStream server and the design-time port
<i>database</i>	Specifies the name of the target database
<i>EJBFile</i>	Specifies the name of the EJB archive to deploy If not specified at the command line, the EJB archive must be specified in the deployment plan
<i>-f deploymentPlan</i>	Specifies the name of the deployment plan See “Deployment Plan DTDs” on page 49 for information about how the deployment plan should be structured You can create this file using Workbench’s Deployment Plan Editor
<i>options</i>	Specifies operating criteria for the command

The valid options are:

Option	Description
<i>-? or -h</i>	Displays the usage message
<i>-U username</i> and <i>-P password</i>	Specifies user name and password for server authentication
<i>-o</i>	Specifies that when the <i>ejbDeployedObject</i> or <i>ejbRemoteJar</i> values are used in the <i>deploymentPlan</i> and the object exists on the server, the objects will be overwritten
<i>-n</i>	Specifies that validation should be skipped. If not present, SilverCmd <i>ValidateEJB</i> is executed before <i>DeployEJB</i>
<i>-t</i>	Specifies that <i>DeployEJB</i> should write temporary Java files in the local character set for debugging
<i>-v verboseLevel</i>	The level of messages to write to the SilverCmd console window; values for <i>verboseLevel</i> are 0 for no messages (default) to 5 for the most messages

DeployEJB11

Deprecated. Deploys an EJB 1.1 JAR on the specified SilverStream server.

DeployEJB performs these tasks:

- Creates implementation classes for the home and remote interfaces, and generates wrapper classes that handle security and transactions
- Generates a deployed object that contains the implementation classes used by the SilverStream server (this object remains on the server)
- Generates stub classes and puts them into a remote JAR that is used by the bean's clients (a copy of this remains on the server)

NOTE For external Java clients, you must manually copy this remote JAR to each client that needs to access this deployed EJB, or you can specify the `-R` option to automatically copy the remote JAR to a single client machine.

- Enables the deployedObject for client access when allowed by the deployment plan

Syntax

```
SilverCmd DeployEJB11 server[:port] database [ejbJar] [-d
ejbDeployedObject] [-r ejbRemoteJar] -f file [-R remoteJarPath] [options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the name of the target SilverStream server and the design-time port
<i>database</i>	Specifies the name of the target database
<i>ejbJar</i>	Specifies the name of the EJB archive to deploy If not specified at the command line, the EJB archive must be specified in the deployment plan
<i>-f file</i>	Specifies the name of the deployment plan See “Deployment Plan DTDs” on page 49 for information about how the deployment plan should be structured You can create this file using Workbench’s Deployment Plan Editor
<i>options</i>	Specifies operating criteria for the command

The valid options are:

Option	Description
-? or -h	Displays the usage message
-U <i>username</i> and -P <i>password</i>	Specifies user name and password for server authentication
-o	If an <code>ejbDeployedObject</code> or <code>ejbRemoteJar</code> argument is given and the object exists on the server, this flag forces the object to be overwritten
-d <i>ejbDeployedObject</i>	Specifies the name for the deployed object created by the command You can specify the name in the deployment plan or at the command line. Values specified at the command line override deployment plan values. If the name is not specified, SilverCmd generates a default name by appending Deployed to the current name (for example, <code>xxxDeployed</code>). If a JAR of this name already exists on the server, you must specify the <code>-o</code> parameter to overwrite it
-r <i>ejbRemoteJar</i>	Specifies the name to use for the remote JAR created by this command. You can specify this name in the deployment plan or at the command line. Values specified at the command line override deployment plan values If the name is not specified, SilverCmd generates a default name by appending Remote to the current name (for example, <code>xxxRemote.JAR</code>). If a JAR of this name already exists on the server, you must specify the <code>-o</code> parameter to overwrite it. The remote JAR file is enumerated only if the input file specifies that the JAR should be enabled
-R <i>remoteJarPath</i>	Creates a copy of the generated <code>ejbRemoteJAR</code> on the local drive in the directory path given. The JAR will have the same name as the remote JAR generated on the server
-v <i>verboseLevel</i>	The level of messages to output. Values for <code>verboseLevel</code> are 0 for no messages (default) to 5 for the most messages

DeployRAR

Deploys a resource adapter archive (RAR) to the specified server and creates the associated connection pool(s).

Server permissions	DTD and sample input file
Write to the EJB JARs & Media directory of the target database	None

Syntax

```
SilverCmd DeployRAR server[:port] database RARFile [-f deployment plan]
[options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the name of the target SilverStream server and the design-time port
<i>database</i>	Specifies the name of the target database
<i>RARFile</i>	Specifies the name of the RAR to deploy
-f <i>deploymentPlan</i>	Specifies the name of the deployment plan. If a file is not specified, DeployRAR looks for a file named silverstream.xml in the RAR's META-INF directory You can create the RAR deployment plan using Workbench's Deployment Plan Editor
options	Specifies operating criteria for the command

The valid options are:

Option	Definition
<i>-U username</i> and <i>-P password</i>	Specifies user name and password for authentication by the server
<i>-o</i>	When specified, overwrites a resource adapter of the same name deployed in the same database
<i>-n resource adapter name</i>	Specifies the name that the resource adapter will be deployed as on the server. This can also be specified in the deployment plan
<i>-d extract directory</i>	Specifies the directory (on the deployer's machine) the contents of the RAR file are extracted to during deployment. This value can also be specified in the deployment plan Files are extracted to the compilecache directory if <i>-d</i> is not specified
<i>-v verboseLevel</i>	The level of messages to output. Values for VerboseLevel are 0 for no messages (default) to 5 for the most messages

DeployWAR

Deploys a J2EE-compatible Web archive (WAR) to a SilverStream server.

DeployWAR performs these tasks:

- Compiles all JSP pages in the WAR into Java source files and then compiles these Java sources
- Adds the compiled Java class files to the WAR file
- Uploads the WAR file to the SilverStream server

The deployed WAR can also contain standard servlet classes and other supporting Java source files that were compiled separately, as well as HTML documents, images, and any other files required by the application.

Server permissions	DTD and sample input file
Write to the EJB JARs & Media directory of the target database	None

Syntax

```
SilverCmd DeployWAR server[:port] database [WARFile]
-f deploymentPlan [options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the name of the target SilverStream server and the design-time port
<i>database</i>	Specifies the name of the target database
<i>WARFile</i>	Specifies the name of the WAR file to deploy This value can be specified at either the command line or in the deployment plan. Values specified at the command line override deploymentPlan settings
-f <i>deploymentPlan</i>	An XML-based file that specifies the SilverStream-specific deployment information
options	Specifies any operating criteria for the command

The valid options are:

Option	Description
-? or -h	Displays the usage message
-i	Ignores errors when compiling JSP pages and deploys whatever builds successfully
-o	If a deployedObject or remoteJar object already exists on the server, this flag forces it to be overwritten

Option	Description
-t	Used for debugging JSP pages using non-Latin-1 character sets: if specified, the JSP compiler outputs into the compile cache an additional Java file with the extension <code>-local</code> (for example, along with <code>date_jsp_XXXXXXXXXX.java</code> , you will also find <code>date_jsp_XXXXXXXXXX-local.java</code>). The version of the file with <code>-local</code> is in the machine's local character set, instead of UTF-8 By default, you will find these files in <code>compilecache/server/database/temp/sources/archive/com/sssw/gen/jsps</code>
<code>-U username</code> and <code>-P password</code>	User name and password for authentication by the server

ExportSource

Classic use only. This command is available for use with SilverStream Classic applications only.

Deprecated. Copies the Java source file of a single business object from a SilverStream application database to the specified location on disk as a text file.

Server permissions	DTD and sample input file
Read	DTD: <code>export_source.dtd</code> Sample: <code>export_source_sample.xml</code>

Syntax

```
SilverCmd ExportSource server[:port] database [classname] [javafile]
[is-file] [options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the name of the source SilverStream server and the design-time port
<i>database</i>	Specifies the name of the source database
<i>classname</i>	Specifies the name of the class to export
<i>javafile</i>	Specifies the name of the local file If you do not specify a fully qualified file name, ExportSource creates the file in the current working directory
<i>is-file</i>	Specifies a file that will be populated by ExportSource The resulting file is in XML format and includes the metadata required to recreate the object on a SilverStream server using the ImportSource command For example, if the object is a triggered business object, it includes the metadata about the trigger and the object's lifetime; if the file does not exist, SilverCmd creates it If you do not specify this file for a triggered business object, you must manually create the import options file when you want to reimport the object It is not necessary to specify this option for nontriggered business objects, because they do not have any associated metadata
<i>options</i>	Specifies operating criteria for the command

The valid options are:

Options	Description
<i>-? or -h</i>	Displays the usage message
<i>-f file</i>	Specifies the file name of an input file This input file can contain the item or directory to export as well as the export source and options files

Options	Description
-U <i>username</i> and -P <i>password</i>	User name and password for SilverStream authentication
-o	Specifies that ExportSource should overwrite the Java file if it already exists on disk

GetConsole

Displays the contents of the specified server's console in the SilverCmd console window.

Server permissions	DTD and sample input file
Read server configuration	None

Syntax

```
SilverCmd GetConsole server[:port] [options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the name and optionally the administration port number of the target SilverStream server
options	Specifies operating criteria for the command

The valid options are:

Options	Description
-? or -h	Displays the usage message
-p	Specifies the port to use for the server console socket connection
-U <i>username</i> and -P <i>password</i>	Specifies user name and password for SilverStream authentication

GetDefaultURL

Displays the default URL for the specified database or server

Server permissions	DTD and sample input file
Read server configuration	None

Syntax

```
SilverCmd GetDefaultURL server[:port] database [options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the name and optionally the administration port of the target server
<i>database</i>	Specifies the name of the database whose default URL you want to get
<i>options</i>	Specifies operating criteria for the command

The valid options are:

Options	Description
-? or -h	Displays the usage message
-U <i>username</i> and -P <i>password</i>	Specifies user name and password for SilverStream authentication

ImportClass

Classic use only. This command is available for use with SilverStream Classic applications only.

Deprecated. Copies Java class files to the Objects directory of the specified database.

You can use ImportClass to copy any kind of Java class including utility classes and SilverStream's triggered objects. You cannot use this command to import the class files for any other type of SilverStream object (such as forms, views, or pages).

To import some types of triggered business object, you must provide additional information about the object's trigger. For example: a mail triggered business object requires data about the mail account to which it responds, a servlet requires the URL specification, and so on. Values for mail and table listeners can be specified at the command line; all others must be specified within the input file. Server, cluster, and invoked business objects do not require additional information. Values specified at the command line (specifically, the -t, -m, and -p options) override settings in the input file.

Server permissions	DTD and sample input file
Write	DTD: import.dtd Sample: import_sample.xml


Syntax

```
SilverCmd ImportClass server[:port] database classfile [options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the name and optionally the design-time port number of the target SilverStream server
<i>database</i>	Specifies the name of the target SilverStream database
<i>classfile</i>	Specifies the name of the class file to import
options	Specifies operating criteria for the command

The valid options are:

Option	Definition
-? or -h	Displays the usage message
-f <i>file</i>	<p>Specifies the file name of the input file that contains the metadata information about the object you are importing</p> <p>Used only if the object is a triggered business object</p> <p> For more information, see import.dtd in the server's Resources\DTDCatalog directory and import_sample.xml in the samples\SilverCmd directory</p>
-m <i>account</i>	<p>Specifies the mail account if loading an AgiMailListener object</p> <p>Overrides input file specifications</p>
-o	Specifies that ImportClass should overwrite the class file if it already exists on the server
-p	<p>Makes the triggered business object a server-lifetime object; that is, it is instantiated once, not for each event (objects are event-lifetime by default)</p> <p>Overrides input file specifications</p>
-t <i>table</i>	<p>Specifies a database table name if loading an AgiTableListener object</p> <p>Overrides input file specifications</p>
-U <i>username</i> and -P <i>password</i>	Specifies the user name and password for SilverStream authentication

ImportMedia

Classic use only. This command is available for use with SilverStream Classic applications only.

Deprecated. Imports objects to the appropriate Media subdirectory on the SilverStream server. The objects can include images, standard JARs, EJB JARs, and sounds.

NOTE You cannot modify imported JARs using the SilverStream Jar Designer. You must make any modifications to the JAR file before importing it into SilverStream.

Importing JAR files that reference files in other JARs You may have to import a JAR file that contains references to classes in other JAR files. The JAR you want to import must contain a Class-path entry that lists the JARs containing the referenced classes. When this is true, you:

1. Import the JARs containing the referenced files (the referenced JARs).
2. Import the primary JAR.
3. Add the primary JAR to any forms, pages, business objects, or deployment plans that use the components in the primary JAR (added via the **File>Jar files** menu item).

If the JAR you are importing does not include a manifest file with a Class-path entry listing the referenced JARs, do the following before trying to import the JAR:

1. Extract the manifest of the primary JAR (making sure that the path on extract matches the manifest's path within the JAR).
2. Add the Class-path attribute to the manifest of the primary JAR.
3. Update the primary JAR with the new manifest file.
4. Follow the steps for importing JARs that already contain the Class-path entry outlined above.

Server permissions	DTD and sample input file
Write	None

Syntax

```
SilverCmd ImportMedia server[:port] database fileOrDir... [options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the name of the target SilverStream server and the design-time port
<i>database</i>	Specifies the name of the target SilverStream database into which you want to import the media files
<i>fileOrDir...</i>	Specifies the name of the file to import or the directory whose files are to be imported You can supply multiple files or directories at the command line
<i>options</i>	Specifies operating criteria for the command

The valid options are:

Option	Description
-? or -h	Displays the usage message
-U <i>username</i> and -P <i>password</i>	Specifies user name and password for SilverStream authentication
-i	Continues on error
-j	Imports all specified files or the contents of all directories as JAR files regardless of their extensions
-o	If the media object exists on the server, this flag forces it to be overwritten
-r	For specified directories, recurses into the subdirectories

ImportPage

Classic use only. This command is available for use with SilverStream Classic applications only.

Deprecated. Imports a static HTML page to a SilverStream database. You can optionally associate the HTML page with one or more URLs.

Server permissions	DTD and sample input file
Write	None

Syntax

```
SilverCmd ImportPage server[:port] database htmlfile [options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the name of the target SilverStream server and the design-time port
<i>database</i>	Specifies the name of the target SilverStream database
<i>htmlfile</i>	Specifies the name of the HTML file to import
<i>options</i>	Specifies operating criteria for the command

The valid options are:

Option	Description
<i>-e URLElements</i>	Specifies a semicolon-separated list of database-relative URLs to associate with the HTML file
<i>-o</i>	Specifies that ImportPage should overwrite an existing HTML file of the same name if necessary

ImportSource

Classic use only. This command is available for use with SilverStream Classic applications only.

Deprecated. Imports Java files to the business objects directory of the specified database and server.

You can use ImportSource to copy source code for utility, EJB, and SilverStream's triggered objects. You cannot use this command to import the source for any other type of SilverStream object (such as forms, views, or pages).

To import some types of triggered business objects, you must provide additional information about the object's trigger. For example: a mail-triggered business object requires data about the mail account to which it should respond, a servlet requires the URL specification, and so on. Values for mail and table listeners can be specified either at the command line or in an input file; all other values must be specified within the input file. Server, cluster, and invoked business objects do not require additional information. Values specified at the command line (specifically the -t, -m, and -p options) override settings in input file. You can also set or modify these values once the object is imported.

Server permissions	DTD and sample input file
Write	DTD: import.dtd Sample: import_sample.xml


Syntax

```
SilverCmd ImportSource server[:port] database javafile [options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the name of the target SilverStream server and the design-time port
<i>database</i>	Specifies the name of the target SilverStream database
<i>javafile</i>	Specifies the location and name of the Java file to import
<i>options</i>	Specifies operating criteria for the command

The valid options are:

Option	Description
-? or -h	Displays the usage message
-f <i>file</i>	<p>Specifies the file name of the input file that contains the metadata information about the object you are importing</p> <p>Used only if the object is a triggered business object</p> <p> For more information, see import.dtd in the Resources\DTDCatalog directory and import_sample.xml in the samples\SilverCmd directory</p>
-k	Specifies that ImportSource should not overwrite existing metadata if the class already exists on the server
-m <i>account</i>	<p>Specifies the mail account if loading an AgiMailListener object</p> <p>Overrides input file specifications</p>
-o	Specifies that ImportSource should overwrite the source file if it already exists on the server
-p	<p>Makes the triggered business object a server-lifetime object; that is, it is instantiated once, not for each event (objects are event-lifetime by default)</p> <p>Overrides input file specifications</p>
-t <i>table</i>	<p>Specifies a database table name if loading an AgiTableListener object</p> <p>Overrides input file specifications</p>
-U <i>username</i> and -P <i>password</i>	Specifies the user name and password for SilverStream authentication

ListCP

Lists the connection pools that are active on the specified server.

Server permissions	DTD and sample input file
Read server configuration	None

Syntax

```
SilverCmd ListCP server[:port] [options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the name of the server and the administration port for which you want the connection pool listing
[<i>options</i>]	Specifies operating criteria for the command

The valid options are:

Option	Definition
-? or -h	Displays the usage message
-U <i>username</i> and -P <i>password</i>	Specifies the user name and password for SilverStream authentication

ModifyCP

Modifies a subset of connection pool properties. To change properties not listed, you must recreate the connection pool.

Server permissions	DTD and sample input file
Modify server configuration	None

Syntax

```
SilverCmd ModifyCP server[:port] poolName [options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the name of the target server and the administration port
<i>poolName</i>	Specifies the name of the connection pool whose properties you want to modify
options	Specifies operating criteria for the command

The valid options are:

Option	Definition
-? or -h	Displays the usage message
-m <i>min conn</i>	The minimum number of connections. The pool manager will attempt to maintain this minimum number of transactions (this is a soft limit)
-x <i>max conn</i>	Specifies the maximum number of connections allowed by the pool. The default is 10. Use -1 to create a pool with no maximum
-t <i>timeout</i>	Specifies the idle timeout in seconds. The default is 60 seconds. When set to -1, idle timeout is disabled and no idle connections are ever closed
-w <i>timeout</i>	The connection wait timeout in seconds. The default is 30 seconds. When set to -1, clients are forced to wait until a connection becomes available

Option	Definition
<code>-U <i>username</i></code> and <code>-P <i>password</i></code>	Specifies the user name and password for SilverStream authentication
<code>-v <i>log level</i></code>	Specifies the logging level. The logging levels are: <ul style="list-style-type: none"> • 0 - logging disabled • 1 - basic connection factory operations and settings • 2 - level 1 plus detailed output from connection pool manager • 3 - level 2 plus exception stack traces and logging information from underlying JDBC driver or Connector resource adapter
<code>-N</code>	Disables debugging

You must remove and recreate the connection pool to change these values.

ModifyTableList

Classic use only. This command is available for use with SilverStream Classic applications only.

Deprecated. Defines the set of tables available to SilverStream for a specified database. The ModifyTableList command requires an input file to specify the table names.

Modifying the set of tables for SilverMaster is not supported.

Server permissions	DTD and sample input file
Modify server configuration	DTD: <code>modify_table_list.dtd</code> Sample: <code>modify_table_list_sample.xml</code>

Syntax

```
SilverCmd ModifyTableList server[:port] database -f file [options]
```

The valid arguments are:

Argument	Definition
<i>server[:port]</i>	Specifies the name of the target SilverStream server and the administration port
<i>database</i>	Specifies the name of the database whose table list you want to modify
<i>-f file</i>	Specifies a file containing the names of the database tables See the <code>modify_table_list_sample.xml</code> file in the <code>samples\SilverCmd</code> directory for an example
options	Specifies operating criteria for the command

The valid options are:

Option	Definition
<i>-?</i> or <i>-h</i>	Displays the usage message
<i>-U username</i> and <i>-P password</i>	Specifies user name and password for SilverStream authentication

Specifying tables in the input file Your input file specifies the tables you want to include or make accessible to the server. The input file can include either a list of individual table names to include, a list of table name patterns, or a combination.

When you specify a pattern, you can use the `%` symbol to match any number of characters and the underscore (`_`) character to specify that you want to match any one character.

Prefs

Updates the following compiler settings in the preferences file:

- Compiler name and directory
- Compiler flags
- Compile cache directory
- SilverStream debug flags

The preferences file contains additional information that is not settable from the Prefs command line. Any compiler values not specified via the command line or in an input file are left unchanged in the preferences file. To change any other settings, such as the default browser, you must use the **Edit>Preferences** menu option in the Main Designer.

NOTE You should stop the Designer before executing the Prefs command. If the Designer is running, the changes will not take effect.

Server permissions	DTD and sample input file
None	DTD: prefs.dtd Sample: prefs_sample.xml

Syntax

```
SilverCmd Prefs [options]
```

The valid options are:

Option	Description
-? or -h	Displays the usage message
-a <i>flags</i>	Sets the compiler-specific flags. It must be a quoted string If compiler flags start with a hyphen (-), eliminate the space between the -a and the flags. For example: <pre>SilverCmd Prefs -c sj "-a-nodeprecated -noinline"</pre>

Option	Description
<i>-c name</i>	Sets the compiler type. The value must be one of the following: <ul style="list-style-type: none"> • SunJavacInProc • javac • sj • jikes
<i>-d dir</i>	Sets the compiler's directory
<i>-f file</i>	Specifies an input file that contains the new compiler preferences Values specified at the command line override input file settings See the <code>prefs_sample.xml</code> file in the <code>samples\SilverCmd</code> directory for an example that shows how to create one of these files
<i>-g true/false</i>	Turns debugging information on or off Debug is off (false) by default
<i>-l</i>	Lists existing preferences to the console You cannot specify -l with any other options; if you do, the other options will not take effect
<i>-s file</i>	Saves existing preferences to the specified file You cannot specify -s with any other options, if you do, the other options will not take effect
<i>-t dir</i>	Sets the compile-cache directory
<i>-r true/false</i>	Runs the rmi2iop compiler in process (true) or not (false)

Setting debug flags The `DebugFlags` option is a directive not to the compiler but to the server. It specifies whether or not `AgfForm.debugPrint()` should print. An existing preferences file might list the value as 0 or 1, but when you set this flag you should always set it to a boolean value (true or false). You cannot change the debug flags option at the command line; you must set it via the XML file specified, using the `-f` option.

PrintLog

Displays records from the HTTP log, error log, or trace log to the SilverCmd console window. Use the standard redirect symbol (>) to write the records to a file. PrintLog can only display records when server logging output is specified as database (not file or user defined).



For more information on specifying logging output using the SMC, see the chapter on running the server in the *Administrator's Guide*.

Server permissions	DTD and sample input file
Read server configuration	None

Syntax

```
SilverCmd PrintLog server[:port] logTypeFlags [options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the name of the target server and the administration port
<i>logTypeFlags</i>	Values are: -E—Removes records from the error log -H—Removes records from the HTTP log -T—Removes records from the trace log You can specify any combination in a space or comma-separated list
<i>options</i>	Specifies operating criteria for the command

The valid options are:

Option	Definition
-? or -h	Displays the usage message
-U <i>username</i> and -P <i>password</i>	Specifies user name and password for SilverStream authentication

Publish

Classic use only. This command is available for use with SilverStream Classic applications only.

Deprecated. Publishes items from one database or server to another.

Server permissions	DTD and sample input file
Read for the source database and write for the destination database	DTD: itemlist.dtd Sample: publish_sample.xml

Syntax

```
SilverCmd Publish sourceserver[:port] sourcedatabase targetserver[:port]
targetdatabase -f file [options]
```

The valid arguments are:

Argument	Definition
<i>sourceserver[:port]</i>	Specifies the name of the source SilverStream server and the design-time port
<i>sourcedatabase</i>	Specifies the name of the source database
<i>targetserver[:port]</i>	Specifies the name of the target SilverStream server and the design-time port
<i>targetdatabase</i>	Specifies the name of the target database
options	Specifies operating criteria for the command

The valid options are:

Option	Definition
-? or -h	Displays the usage message
-U <i>username</i> and -P <i>password</i>	Specifies user name and password for source SilverStream server for authentication

Option	Definition
-V and -Q	Specifies user name and password for destination SilverStream server for authentication; if not specified, values given for -U and -P are used
-a	Publishes the contents of the database
-f <i>file</i>	Specifies the name of a file that specifies the publish list You can specify -f only when -a is not specified
-c	Forces an overwrite of read-only items in the destination database This setting is ignored when the destination database is not part of source control
-s	Specifies that Publish should strip design information from the items published

PublishFromFile

Classic use only. This command is available for use with SilverStream Classic applications only.

Deprecated. Publishes SilverStream components from the specified directory on disk to the appropriate directory on the specified SilverStream server and database. You can **only** publish items that were downloaded using PublishToFile (not exported) from a SilverStream server—because they must be in the expected file format. SilverCmd places the objects in the appropriate server directory. For example: forms are uploaded to the forms directory, pages to the page directory, and so on.



For more information, see “PublishToFile” on page 153.

If you are publishing items without their class files, you must build the objects after uploading them.

Server permissions	DTD and sample input file
Write	DTD: filelist.dtd Sample: publish_from_file_sample.xml

Syntax

`SilverCmd PublishFromFile server[:port] database [fileorDir] [options]`

The valid arguments are:

Argument	Definition
<i>server[:port]</i>	Specifies the name of the target SilverStream server and the design-time port
<i>database</i>	Specifies the name of the target database
<i>fileorDir</i>	Specifies the source file or directory to publish
<i>options</i>	Specifies operating criteria for the command

The valid options are:

Option	Definition
<i>-?</i> or <i>-h</i>	Displays the usage message
<i>-U username</i> and <i>-P password</i>	Specifies user name and password for SilverStream authentication
<i>-f file</i>	Specifies an input file that contains the list of items to upload The file should be in the format specified by the FileList.DTD in the SilverStream/DTDs directory
<i>-s</i>	Specifies that SilverStream should copy items from the specified directory recursively

PublishToFile

Classic use only. This command is available for use with SilverStream Classic applications only.

Deprecated. Downloads the components of an application database from the server to a specified location on disk, or to the current working directory if no directory is specified. This command is intended to be used in conjunction with PublishFromFile when you want to transfer application components indirectly between servers. The resulting file format is not usable for any other purpose.

Suppose, for example, that you have created an application that you want to install (not Publish) on another server. You would use PublishToFile to obtain all of the application's files including all the necessary components (such as Java code, metadata information, and associated data like HTML for pages, and so on). You might then create a script that runs the SilverCmd PublishFromFile command to put them on the target server.

The PublishToFile operation maintains the server's directory structure for the items it downloads. For example, if you specify that the items should be downloaded to the C:\test directory, SilverCmd downloads forms to the c:\test\Forms, pages to the c:\test\Pages, business objects to the c:\test\Objects, and so on.

Server permissions	DTD and sample input file
Read	DTD: <code>filelist.dtd</code> Sample: <code>publish_to_file_sample.xml</code>

Syntax

```
SilverCmd PublishToFile server[:port] database [item] [options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the name of the target SilverStream server and the design-time port
<i>database</i>	Specifies the name of the target database

Argument	Description
<i>item</i>	<p>The name of a single item to publish</p> <p>Specifications include:</p> <ul style="list-style-type: none"> • For forms, views, or pages, use: <i>dir/itemname</i> • For business objects, use: <i>Objects/packageName/itemname</i> • For items in the Media directory: <i>Media/dir/itemname</i> <p>where <i>dir</i> can be JARs, Images, Sounds, or General</p>
options	Specifies operating criteria for the command

The valid options are:

Option	Definition
-? or -h	Displays the usage message
-U <i>username</i> and -P <i>password</i>	Specifies user name and password for SilverStream authentication
-c	Specifies that the download should exclude the class files
-d <i>dir</i>	Specifies the output directory. SilverCmd creates the directory if it does not exist
-f	Specifies an input file that contains a list of items to download

Option	Definition
-s	Specifies that SilverStream should strip the design metadata and the Java source code
-x	Specifies that the download should be in XML format instead of binary (binary is the default) Binary format is more compact and faster to read and write than XML format. If you were building an installer (by downloading the objects and then using Upload at install time to install them), you might want to use binary format

QueryCP

Displays configuration properties for a connection pool.

Server permissions	DTD and sample input file
Read server configuration	None

Syntax

```
SilverCmd QueryCP server[:port] poolName [options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the name of the target SilverStream server and the design-time port
<i>poolName</i>	Specifies the connection pool name
options	Specifies operating criteria for the command

The valid options are:

Option	Definition
-? or -h	Displays the usage message
-a	Displays all of the properties for a connection pool. When -a is not specified, only the settable properties are displayed. Settable properties include connection wait timeout, idle timeout, minimum, connections, maximum connections, and debug
-U <i>username</i> and -P <i>password</i>	Specifies user name and password for SilverStream authentication

RebuildJAR

Classic use only. This command is available for use with SilverStream Classic applications only.

Deprecated. Rebuilds SilverStream-generated JAR files.

You might use RebuildJAR when certain components of your application that are part of a JAR have changed or when you want to insert new packages or business objects into your JAR. For example, you might have changed a GIF or added a new package as a subpackage. The RebuildJAR operation discards the old items and inserts the new ones. You might think of RebuildJAR as “edit existing JAR and save changes.”

You cannot use RebuildJAR to add or remove objects from the JAR. RebuildJAR takes the most recent versions of the objects that exist in the current JAR. This means that for specific items (such as **Forms/form1** or **Views/view1**), the newer versions of these items (if any) replace the existing ones. For directories (such as Forms, Views and so on), only the items that were in those directories at the time of the JAR’s creation will be reinserted. This is **not** so for packages, whose contents are dynamically regenerated each time you rebuild. For example, if the user adds any new classes to a package, they will get inserted into the JAR—older versions of existing classes substituted for newer ones and any deletions reflected in warning messages.

Restrictions You cannot rebuild JAR files created outside the SilverStream IDE.

Server permissions	DTD and sample input file
Read/Write	None

Syntax

```
SilverCmd RebuildJAR server[:port] database jarfile [options]
```

The valid arguments are:

Argument	Definition
<i>server[:port]</i>	Specifies the name of the source SilverStream server and the design-time port
<i>database</i>	Specifies the name of the source database
<i>jarfile</i>	Specifies the name of the JAR file to rebuild
options	Specifies operating criteria for the command

RebuildJar supports the following options:

Options	Description
-? or -h	Displays the usage message
-U <i>username</i> and -P <i>password</i>	Specifies user name and password for SilverStream authentication

RemoveCP

Shuts down the specified connection pool and removes it so that the server does not try to restart the connection pool during a server restart.

Server permissions	DTD and sample input file
Modify server configuration	None

Syntax

```
SilverCmd RemoveCP server[:port] poolName [options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the target server and administration port
<i>poolName</i>	Specifies the name of the connection pool to remove
options	Specifies operating criteria for the command

The valid options are:

Option	Definition
-? or -h	Displays usage message
-U <i>username</i> and -P <i>password</i>	Specifies user name and password for SilverStream authentication

RemoveDatabase

Classic use only. This command is available for use with SilverStream Classic applications only.

Deprecated. Removes the database from the server's list of accessible databases.

Server permissions	DTD and sample input file
Modify server configuration	None

Syntax

```
SilverCmd RemoveDatabase server[:port] database [options]
```


The valid arguments are:

Argument	Definition
<i>server[:port]</i>	Specifies the name of the source server and the administration port
<i>database</i>	Specifies the name of the database to remove
options	Specifies operating criteria for the command

The valid options are:

Option	Definition
-? or -h	Print the usage message
-U <i>username</i> and -P <i>password</i>	Specifies user name and password for SilverStream authentication

ServerState

Manages the server's state. Use it to shut down a server or test whether the server is currently running.

NOTE **ServerState** can be run on any (runtime, design-time, or administration) port with the **isrunning** action—if you have configured separate ports. If you run **ServerState** with the shutdown action, you must specify the administration port.

Server permissions	DTD and sample input file
Modify server configuration when the action is shutdown Read server configuration with the action is isrunning	None

Syntax

```
SilverCmd ServerState server[:port] action [options]
```

The valid arguments are:

Argument	Definition
<i>server[:port]</i>	Specifies the name of the source SilverStream server and the port. The required port depends on which of the following two actions you specify
<i>action</i>	Specifies one of the following: <ul style="list-style-type: none"> • isrunning—Returns a message when the server is (or is not) running. You can run ServerState with the isrunning action on any (runtime, design-time, or administration) port. This action must be run as part of a batch file or script • shutdown—Gracefully shuts down the SilverStream server. You must enter the administration port when running this action; otherwise, a security error code is returned
options	Specifies operating criteria for the command

The valid options are:

Option	Definition
-? or -h	Displays the usage message
-U <i>username</i> and -P <i>password</i>	Specifies user name and password for SilverStream authentication
-d	Deactivate the target server. For use only when <i>action</i> is shutdown
-r	Restart the target server. For use only when <i>action</i> is shutdown

SetDefaultURL

Sets the default URL for a server or database.

Server permissions	DTD and sample input file
Modify server configuration	None

Syntax

```
SilverCmd SetDefaultURL server[:port] [database] [options] -e URL
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the target SilverStream server and the design-time port
<i>database</i>	Specifies the target SilverStream database; include only if setting a database-default URL (see the examples below)
<i>options</i>	Specifies operating criteria for the command

The valid options are:

Option	Definition
-? or -h	Displays the usage message
-U <i>username</i> and -P <i>password</i>	Specifies user name and password for SilverStream authentication
-e <i>URL</i>	Specifies a server-relative URL (<i>database/URL</i>) or a database-relative URL (<i>URL</i>) Here is a server-relative URL: /MyDatabase/SilverStream/Pages/pgHome.html Here is a database-relative URL: /SilverStream/Pages/pgHome.html

Examples

Setting server-default URLs When setting a server-default URL, do not specify a database name as an argument, and specify an URL using a server-relative URL.

For example, the following command sets a server-default URL for the server **myServer**:

```
SilverCmd SetDefaultURL myServer  
-e /MyDatabase/SilverStream/Pages/pgHome.html
```

Setting database-default URLs When setting a database-default URL, specify the database name as an argument, and use a database-relative URL.

For example, the following command sets a database-default URL for the database **myDatabase** on the server **myServer**:

```
SilverCmd SetDefaultURL myServer myDatabase  
-e /SilverStream/Pages/pgHome.html
```

SetSecurity

Sets Read, Write, Protect, Select, and Execute security permissions on the SilverStream server, a database, a directory, or one or more objects. Certain permission types are applicable only for certain types of items. For example, the Select permission is only applicable to tables.

You can also set permissions on the Security directory of a server. The Read permission on this resource rules who can have access to user and group information such as lists of users and groups and user/group properties. The Protect permission rules who can set the permissions on the Security directory.

Server permissions	DTD and sample input file
Set Permissions	DTD: set_security.dtd
Read Users and Groups	Sample: set_security_sample.xml

Syntax

```
SilverCmd SetSecurity server[:port] [database] -f file [options]
```

The valid arguments are:

Argument	Definition
<i>server[:port]</i>	Specifies the name of the SilverStream server and the administration port
<i>database</i>	Specifies the name of the database Do not specify this value when setting server permissions
<i>-f file</i>	Specifies the fully qualified name for an input file whose contents specify the security permissions information
options	Specifies the operating criteria for the command

The valid options are:

Option	Definition
<i>-? or -h</i>	Displays the usage message
<i>-U username and -P password</i>	Specifies user name and password for SilverStream authentication
<i>-i</i>	Continues on error

SetUserGroupInfo

Creates and deletes SilverStream users and groups, adds users to groups, and sets properties for both. This command has eight optional actions (listed in “Actions” on page 164).

NOTE If you have configured separate ports, you will need to specify the **administration** port when running **SetUserGroupInfo** to change settings.

Server permissions	DTD and sample input file
Modify server settings	DTD: set_user_group_info.dtd Sample: set_user_group_info_sample.xml

Syntax

```
SilverCmd SetUserGroupInfo server[:port] [action action-parameters]
[options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the name of the SilverStream server and the administration port
<i>action</i>	Specifies the action to perform—for example, CreateUser or DeleteUser For a list of actions, see “Actions” below
action-parameters	Specifies any special operating criteria for the action
options	Specifies operating criteria for the command

The valid options are:

Option	Description
-? or -h	Displays the usage message
-f <i>file</i>	Specifies the name of a file containing data for the SetUserGroupInfo command
-i	Continues on error This is valid in batch mode only
-U <i>username</i> and -P <i>password</i>	Specifies user name and password for SilverStream authentication

Actions The SetUserGroupInfo actions are:

Action	Description
AddUserToGroup	Adds a user to a SilverStream security group
CreateGroup	Creates a SilverStream security group

Action	Description
CreateUser	Creates a SilverStream user or certificate
DeleteGroup	Deletes a SilverStream group from the server
DeleteUser	Deletes a SilverStream user from the server
DeleteUserFromGroup	Deletes a SilverStream user from the specified SilverStream group
SetGroupProperties	Specifies properties for an existing SilverStream group
SetUserProperties	Specifies properties for an existing SilverStream user


AddUserToGroup action

Adds an existing user from a known security realm (such as NT) to an existing SilverStream security group.

Syntax

```
SilverCmd SetUserGroupInfo server[:port] AddUserToGroup username
groupname [options]
```

The action-parameters are:

Action-parameter	Description
<i>username</i>	<p>Specifies the name of the user to add. The name must be in a valid login format</p> <p>If the name includes spaces, it must be enclosed in quotes</p> <p> For more information about supplying these values for the security realms, see the chapter on setting up security in the <i>Administrator's Guide</i></p>
<i>groupname</i>	<p>Specifies the name of the SilverStream group to which you want to add the user</p> <p>If the name includes spaces, it must be enclosed in quotes. It is case-sensitive and must exactly match an existing groupname</p>

Examples

This example shows how to add the NT user **admin** to the SilverStream group **Admins**:

```
SilverCmd SetUserGroupInfo localhost AddUserToGroup ntDomain1\admin Admins
```

This example shows how to add the NT user **admin** to the SilverStream group **Our NT Administrators**:

```
SilverCmd SetUserGroupInfo localhost AddUserToGroup ntDomain1\admin "Our NT Administrators"
```

CreateGroup action

Creates a SilverStream security group for the specified server.

Syntax

```
SilverCmd SetUserGroupInfo server[:port] CreateGroup -g groupname [-d description]
```

The valid arguments are:

Argument	Description
-g <i>groupname</i>	Specifies the name of the group This value is required. If the groupname includes spaces, it must be enclosed in quotes
-d <i>description</i>	Specifies a description for the group This value is optional. If the description includes spaces, it must be enclosed in quotes

Examples

The following examples show how to create three distinct SilverStream groups: one called **Developers**, one called **Our Administrators**, and one called **Finance**:

```
SilverCmd SetUserGroupInfo localhost CreateGroup -g Developers -d "Research and Development Group"
```

```
SilverCmd SetUserGroupInfo localhost CreateGroup -g "Our Administrators" -d "Our Admins"
```

```
SilverCmd SetUserGroupInfo http://myserver CreateGroup -g Finance
```


CreateUser action

Creates a SilverStream user by specifying a user name/password or a certificate user.

Syntax

```
SilverCmd SetUserGroupInfo server[:port] CreateUser -u username [-p
password] [-n full-name] [-d description]
```

OR

```
SilverCmd SetUserGroupInfo server[:port] CreateUser -c client-certificate-
file
```

The valid arguments are:

Argument	Description
<i>-u username</i>	Specifies the name by which the new user will be known to the SilverStream server. This value is required except when specifying a client certificate file Note that this value is different from the -U and -P (uppercase) parameters used for authenticating the user running SilverCmd
<i>-p password</i>	Specifies the user's SilverStream password. This value is optional Note that this value is different from the -U and -P (uppercase) parameters used for authenticating the user running SilverCmd
<i>-n full-name</i>	Specifies the user's full name. If the name includes spaces, it must be enclosed in quotes. This value is optional
<i>-d description</i>	Specifies a description for the user. If the description includes spaces, it must be enclosed in quotes. This value is optional
<i>-c client-certificate-file</i>	Specifies the client certificate file

Examples

This example shows how to create a new user:

```
SilverCmd SetUserGroupInfo http://myserver CreateUser -u user1 -p  
MyPassword -n "John Doe" -d "Applications Developer"
```

```
SilverCmd SetUserGroupInfo localhost CreateUser -u user1 -n "John Doe"
```

```
SilverCmd SetUserGroupInfo localhost CreateUser -u user1
```

This example shows how to create a certificate user, given a client certificate file:

```
SilverCmd SetUserGroupInfo localhost CreateUser -c c:\certs\ClientCert1.cer
```

DeleteGroup action

Deletes a SilverStream group.

Syntax

```
SilverCmd SetUserGroupInfo server[:port] DeleteGroup groupname
```

The valid arguments are:

Argument	Description
<i>groupname</i>	Specifies the name of the group to delete. It must exactly match the existing groupname (it is case-sensitive). If the name includes spaces, it must be enclosed in quotes

For example:

```
SilverCmd SetUserGroupInfo localhost DeleteGroup TestGroup
```

DeleteUser action

Deletes a SilverStream user from the system.

Syntax

```
SilverCmd SetUserGroupInfo server[:port] DeleteUser username
```

The valid arguments are:

Argument	Description
<i>username</i>	Specifies the name of the user to delete. It must exactly match an existing username (it is case-sensitive). If the name includes spaces it must be enclosed in quotes

For example:

```
SilverCmd SetUserGroupInfo http://myserver DeleteUser testUser1
```

To delete a certificate user:

```
SilverCmd SetUserGroupInfo localhost DeleteUser "CERT\\Jack Brown,
DigitalID Class 1 - Microsoft Full Service, VeriSign, Inc.
(28f52c889e8d6d8cf21d932d9b71z705) "
```

You must specify the complete distinguished name of the certificate user:

- The constant CERT\\ must be prepended to the distinguished name in order to disambiguate it (CERT means Certificate Security Realm).
- The \\ signifies a security authority that is not present in this case.
- With NT, for example, the name would look something like this:
NT\domainname\user1
- With Certificate Security Realm, no authorities are specified.

You can specify the default security realm (via the SMC or by setting the AgiAdmServer.PROP_DEFAULT_SECURITY_REALM property on the server object). If you set the default to **Certificate Security Realm** (for example, by setting the property value to **CERT**), there would be no need for the **CERT** part, because it would be assumed by default.

DeleteUserFromGroup action

Deletes a user from a SilverStream group.

Syntax

```
SilverCmd SetUserGroupInfo server[:port] DeleteUserFromGroup username
groupname
```

The valid arguments are:

Argument	Description
<i>username</i>	Specifies the name of the user to delete from the group. It must exactly match an existing username (it is case-sensitive). If the name includes spaces, it must be enclosed in quotes
<i>groupname</i>	Specifies the name of the SilverStream group from which to delete the user. It must exactly match an existing groupname (it is case-sensitive). If the name includes spaces, it must be enclosed in quotes

For example:

```
SilverCmd SetUserGroupInfo localhost DeleteUserFromGroup ntDomain1\admin  
Admins
```

SetGroupProperties action

Sets properties for a SilverStream group. Any properties that are not specified retain their original values.

Syntax

```
SilverCmd SetUserGroupInfo server[:port] SetGroupProperties -g groupname [-  
d description -l "is-locksmith"]
```

The valid arguments are:

Argument	Description
<i>-g groupname</i>	Specifies the name of the group. It must exactly match an existing groupname (it is case-sensitive). If the name includes spaces, it must be enclosed in quotes

Argument	Description
<code>-d <i>description</i></code>	Provides a description of the group. If the name includes spaces, it must be enclosed in quotes
<code>-l <i>is-locksmith</i></code>	<p>Specifies the locksmith value, which may be true or false</p> <p>You may set is-locksmith on any type of group, not just SilverStream groups</p> <p>You can only grant Locksmith privileges if you have them—for example, if you are a Locksmith</p> <p>The value true means grant the privilege, and false means revoke the privilege</p>

Examples

```
SilverCmd SetUserGroupInfo myserver SetGroupProperties -g testGroup -d
"This is a test group"
```

```
SilverCmd SetUserGroupInfo myserver SetGroupProperties -g "Our
Administrators" -l false
```

SetUserProperties action

Modifies properties for a SilverStream or certificate user. Values not specified are not modified.

Syntax

```
SilverCmd SetUserGroupInfo server[:port] SetUserProperties -u username -p
password -n full-name [-d description] [-l is-locksmith]
```

OR

```
SilverCmd SetUserGroupInfo server[:port] SetUserProperties -c certificate-
file [-l is-locksmith]
```

The valid arguments are:

Argument	Description
<code>-u <i>username</i></code>	Specifies the user name for the user whose properties you want to change. This value is required and is not configurable
<code>-p <i>password</i></code>	Specifies a new password for the user
<code>-n <i>full-name</i></code>	Specifies a full name for the user. If the name includes spaces, it must be enclosed in quotes
<code>-d <i>description</i></code>	Provides a description of the user. If the name includes spaces, it must be enclosed in quotes
<code>-l <i>is-locksmith</i></code>	Specifies the Locksmith privilege value. You can give Locksmith privileges to any type of user, not just SilverUser or CertificateUser. You can only grant Locksmith privileges if you have them—for example, if you are a Locksmith. The value true means grant the privilege and false means revoke the privilege
<code>-c <i>certificate-file</i></code>	Specifies the certificate file for updating the certificate users. This is a required value for certificate files and is not configurable

Examples

```
SilverCmd SetUserGroupInfo localhost SetUserProperties -u jsmith -p "new
password" -l false
SilverCmd SetUserGroupInfo localhost SetUserProperties -u jsmith -n
"Jonathan H. Smith"
SilverCmd SetUserGroupInfo localhost SetUserProperties -u jsmith -d
"Principal Engineer" -p "new pwd"
```

SourceControl

Classic use only. This command is available for use with SilverStream Classic applications only.

Deprecated. Performs source control tasks.

Before you can run the SourceControl command, you must have one of the SilverStream supported source control software packages installed and your SilverStream application database set up to use it. You cannot set up a database for source control using the SourceControl command-line tool.

Server permissions	DTD and sample input file
Read/Write	DTD: itemlist.dtd Sample: source_control_sample.xml

Syntax

```
SilverCmd SourceControl server[:port] database action [item] [options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	Specifies the name of the target SilverStream server and the design-time port
<i>database</i>	Specifies the name of the target database that contains your SilverStream application components This database must already be configured to use a source control system
<i>action</i>	Specifies the source control action to execute Valid values are Get, Checkin, Checkout, and Undocheckout These values are not case-sensitive

Argument	Description
<i>item</i>	<p>Specifies the name of the SilverStream application object on which you want to perform the source control operation</p> <p>The name must be specified using the SilverStream server's directory structure—for example, to specify perform a source control operation on a form, use:</p> <p style="text-align: center;"><i>Forms/formname</i></p> <p>To perform an operation on a JAR, use:</p> <p style="text-align: center;"><i>Media/Jars/jarname</i></p> <p>To work with multiple items, you must specify them in an input file, which you specify using the <i>-f file</i> argument</p>
options	<p>Specifies operating criteria for the action</p> <p>If you supply an option that is not valid for the current action, SilverCmd executes the action and ignores any invalid options</p> <p>See the table below for more information on the valid options</p>

You can customize the source control operation (Get, Checkin, and so on) using the options described in the following table:

Option	Description
-? or -h	Displays the SourceControl usage message
-U <i>username</i> and -P <i>password</i>	Specifies user name and password for SilverStream authentication
-c <i>comment</i>	<p>Specifies the comment to use for check-in operations. The text must be in quotes. For example:</p> <p style="text-align: center;"><code>SilverCmd SourceControl Checkin -c "This is a comment" localhost MyApp Forms/myForm</code></p> <p>This option is only valid for check-in operations</p>

Option	Description
<i>-d date</i>	<p>Gets an object checked in on a specific date</p> <p>Valid formats are:</p> <pre>-d 4/4/98 -d 4/4/1998 -d 1998-4-4 -d "4/4/98 3:33 pm" -d "4/4/98 15:33" -d "1998-4-4 15:33:33"</pre> <p>You can use period (.) or hyphen (-) in place of slash (/) anywhere in the date portion. The month/day order is determined by the current locale. Two-digit years are handled appropriately (98 means 1998; 02 means 2002)</p> <p>This option is only valid with Get operations</p>
<i>-f file</i>	<p>The name of an input file that specifies the items on which to perform the source control operation. This is useful when you want to perform an operation on multiple files</p>
<i>-l label</i>	<p>Gets an object that has the specified label. The label must be in quotes. For example:</p> <pre>SilverCmd SourceControl Get -l "Version 3.0 Beta" localhost Examples3 Forms/myForm</pre>
<i>-p password</i>	<p>Specifies the source control password. This must match the password exactly as configured in your source control system</p> <p>The user name cannot be specified on the command line; it must be set in the Source Control Settings dialog</p> <p>If your source control access was set up to save your password, you do not need to supply one. You can check the Source Control settings from the Designer by choosing Source>Source Control Settings</p> <p>SilverCmd generates an error message if the -p option is required but not specified</p>

Option	Description
<code>-v version</code>	Gets an object by its source control version
<code>-o</code>	Confirms an action that will overwrite an existing file—for example, a Get of a checked-out file SilverCmd fails if the specified operation will overwrite an existing file and the <code>-y</code> option is not specified

This example illustrates how to check in a form called frmCasting:

```
SilverCmd SourceControl Checkin -c "updated code in FormActivate
event"localhost MyApp Forms/frmCasting
```

Undeploy

Undeploys a J2EE deployed object from a specified SilverStream server. You can use this command to undeploy EARs, EJBs, RARs, CARs, and WARs.

Server permissions	DTD and sample input file
Modify server configuration	None

Syntax

```
SilverCmd Undeploy server[:port] database archive [options]
```

The valid arguments are:

Argument	Description
<i>server[:port]</i>	The target SilverStream server
<i>database</i>	The database containing the deployed J2EE archive
<i>archive</i>	The name of the J2EE object to undeploy. Use the name exactly as shown in the SMC's Deployed Object panel (accessed via the Deployment icon).
options	Specifies operating criteria for the command

The valid options are:

Option	Definition
-? or -h	Displays the usage message
-U <i>username</i> and -P <i>password</i>	Specifies user name and password for SilverStream authentication
-v <i>verboseLevel</i>	The level of messages to output. Values for VerboseLevel are 0 for no messages (default) to 5 for the most messages

ValidateEAR

Validates the deployment descriptor within the specified EAR file. It reports any deployment descriptor problems, missing application assembly components, and class-related problems. The problems are written to the command window by default.

Use this command when you want to verify that the descriptor is correct before attempting to deploy the EAR on the SilverStream server with DeployEAR.

Server permissions	DTD and sample input file
None	None

Syntax

```
SilverCmd ValidateEAR earfile [options]
```

The valid arguments are:

Argument	Description
<i>earfile</i>	Specifies the name of the EAR file to validate
options	Specifies any operating criteria for the command

The valid options are:

Option	Description
-? or -h	Displays the usage message

ValidateEJB

Validates the beans, the deployment plan, and the deployment descriptor. It is automatically called by SilverCmd DeployEAR and DeployEJB and writes any errors or warnings to the SilverCmd console window.

Server permissions	DTD and sample input file
None	None

Syntax

```
SilverCmd ValidateEJB ejbJarFile deploymentPlan [options]
```

The valid arguments are:

Argument	Description
<i>ejbJarFile</i>	Specifies the EJB JAR file whose beans are to be validated
<i>deploymentPlan</i>	Specifies the SilverStream server deployment plan
options	Specifies operating criteria for the command

The valid options are:

Option	Description
-? or -h	Displays the ValidateEJB usage message
-v <i>verboseLevel</i>	The level of messages to output. Values for verboseLevel are 0 for no messages (default) to 5 for the most messages

ValidateEJB11

Validates the deployment descriptor within the specified EJB 1.1 JAR file. It reports any deployment descriptor problems, missing application assembly components, and class-related problems. The problems are written to the console window by default.

You might want to use this command if you are building your own EJB JAR and its descriptor outside the SilverStream IDE and want to verify that the descriptor is correct before importing the JAR to the SilverStream server.

Server permissions	DTD and sample input file
None	None

Syntax

```
SilverCmd ValidateEJB11 ejbJarFile [options]
```

The valid arguments are:

Argument	Description
<i>ejbJarFile</i>	Specifies the EJB JAR file whose beans are to be validated
options	Specifies any operating criteria for the command

The valid options are:

Option	Description
-? or -h	Displays the ValidateEJB usage message
-c <i>max</i>	Maximum number of catastrophic errors allowed per bean The default is 1
-t <i>max</i>	Maximum number of fatal errors allowed per bean The default is 5

Option	Description
<i>-n max</i>	Maximum number of nonfatal errors allowed per bean The default is 10
<i>-w max</i>	Maximum number of warnings allowed per bean The default is 15

If *-c*, *-t*, *-n*, or *-w* is not specified, the default maximum for the given type of error is used.

If a numeric value is specified for the switch, the SilverStream server allows up to that many errors of the given type per bean and stops validating the bean when the limit is reached. If the string value of **all** is specified for the switch, the SilverStream server registers all errors of the given type.

5

SilverJ2EEClient and SilverJRunner

This chapter describes the facilities provided with the SilverStream eXtend Application Server to host Java-based clients: SilverJ2EEClient (for J2EE applications) and SilverJRunner (for classic SilverStream applications). Topics include:

- About SilverJ2EEClient
- About SilverJRunner
- Installing SilverJRunner and SilverJ2EEClient
- Starting SilverJRunner and SilverJ2EEClient
- Using startup options
- Passing application arguments
- Supporting access to secured EJBs

About SilverJ2EEClient

If your production environment includes **J2EE application clients**, the user machines running them will need to install and use **SilverJ2EEClient**. SilverJ2EEClient is a low-administration J2EE application client container that hosts your clients with a robust set of supporting J2EE services (including deployment, JNDI namespace access, and security authentication).



To learn about J2EE application clients and how to write them, see the eXtend Workbench help.

SilverJ2EEClient features

Users invoke SilverJ2EEClient to run J2EE application clients you've deployed to the SilverStream server. The features that SilverJ2EEClient provides to support these clients include:


- **Container installation** SilverJ2EEClient is easy to download from the SilverStream server and install on user machines.
- **Client deployment** When the user starts a particular client, SilverJ2EEClient checks whether the user machine already has the appropriate versions of that client's deployed JAR files. If not, it automatically downloads them from the SilverStream server.

- **Communication protocols** When downloading JARs from the SilverStream server to a user machine, SilverJ2EEClient uses HTTP (or HTTPS) as the communication protocol. In all other cases (such as access to EJBs and other resources), it uses RMI-IIOP or RMI-IIOP over SSL (for secured EJBs). Beyond that, you're free to code your client classes to use any communication protocols necessary for particular tasks.
- **User authentication** When SilverJ2EEClient connects to the SilverStream server to download client JARs, it automatically handles any user authentication required (such as by prompting for user name and password). Alternatively, it lets you pass user name and password command-line options (to support authentication during RMI-IIOP communication).
- **Server access** When accessing the SilverStream server, SilverJ2EEClient automatically takes care of establishing an RMI session. You don't need to write any code for this in your client classes.
- **Namespace access** SilverJ2EEClient automatically provides a JNDI namespace for your client. This gives your client classes access to environment entries, EJB references, and resource references.
- **Client portability** Because SilverJ2EEClient handles the housekeeping, you are insulated from having to write vendor-specific code in your client classes. Of course, you can still choose to code vendor specifics (such as SilverStream API calls) if the situation requires it (for example, when the functionality needed is not specified by J2EE). In that case, consider a modular approach to isolate anything that isn't standard J2EE.

About SilverJRunner

If your production environment includes **SilverStream forms**, the user machines running them will need to install and use **SilverJRunner**. SilverJRunner is a low-administration Java client that hosts your forms with support for the full set of form features, as well as encryption and application file caching.

SilverJRunner is self-updating, so you just need to install it once per user machine.

 To learn about SilverStream forms, see the chapter on form basics in the *Programmer's Guide* of the server's Classic Development Help.

Installing SilverJRunner and SilverJ2EEClient

To minimize administration effort, the SilverStream server provides an installer that users can download to set up the current version of SilverJRunner and SilverJ2EEClient. It also provides an install page that users can browse to for easy selection of the appropriate installer flavor (Windows, UNIX, or Linux).

The install page and installers are named after SilverJRunner, but they include SilverJ2EEClient as well.

Providing the SilverJRunner install page

By default, the Enterprise Edition of the SilverStream server is set up to provide the SilverJRunner install page and installers. The Developer Edition can be set up to provide them, although by default it does not.



For more information, see the SilverStream eXtend Application Server Installation Guide.

Going to the SilverJRunner install page

By default, the SilverJRunner install page is available from your SilverStream server at the following URL:

```
http://servername/SilverStream/Pages/SilverJRunner.html
```

You can instruct users to type this URL directly in their browsers, or you can supply a page that links to it for easier access.

Using the install page Once displayed, the SilverJRunner install page provides links that users can click to download:

- A Windows version of the SilverJRunner installer
- UNIX or Linux versions of the SilverJRunner installer (for supported platforms)

This page also includes instructions for using the installers and for starting SilverJRunner or SilverJ2EEClient once installed.

Customizing the install page By default, the SilverJRunner install page is named SilverJRunner.html and stored in the SilverMaster database under Pages. You're free to customize this page as necessary to suit the requirements of your environment and user audience.

If you need to restrict access to the SilverJRunner installers, you can secure this page or even delete it from the server.

Providing direct access to the installers You may want to let users download a SilverJRunner installer directly, without going to the SilverJRunner install page first. In that case, you can instruct them to type one of the following URLs, or you can link to it from your own pages.

To link directly to	Use this URL
Windows version of the SilverJRunner installer	SilverJRunnerInstall.exe
UNIX (Solaris) version of the SilverJRunner installer	SilverJRunnerInstallSolaris.sh
Linux version of the SilverJRunner installer	SilverJRunnerInstallLinux.sh

These URLs all begin with:

```
http://servername/SilverStream/silverJRunnerInstall/
```

Installing on Windows

Once you download the Windows version of the SilverJRunner installer, you'll have the following file on your local machine:

```
SilverJRunnerInstall.exe
```

Run this file to install SilverJRunner, SilverJ2EEClient, and the supporting components they require.

What gets installed When the installer is done, you'll have a directory for SilverJRunner on your local machine that contains:

- SilverJRunner and SilverJ2EEClient executables (SilverJRunner.exe and SilverJ2EEClient.exe)
- SilverStream runtime files (ZIPs and JARs of API packages and supporting files)
- Java 2 Runtime Environment (JRE)
- Java HotSpot Performance Engine
- jBroker ORB (a Java-based CORBA ORB, set up as the default ORB for the machine)

SilverJRunner and SilverJ2EEClient should now be ready to use on this machine.

Installing on UNIX or Linux

Once you download a UNIX or Linux version of the SilverJRunner installer, you'll have the following file on your local machine: **SilverJRunnerInstallPlatform.sh**. Run this file to install SilverJRunner, SilverJ2EEClient, and the supporting components they require.


For example:

- In Solaris, type `sh SilverJRunnerInstallSolaris.sh`
- In Linux, type `sh SilverJRunnerInstallLinux.sh`

What gets installed When the installer is done, you'll have a directory for SilverJRunner on your local machine that contains:

- SilverJRunner and SilverJ2EEClient executables (SilverJRunner and SilverJ2EEClient)
- SilverStream runtime files (ZIPs and JARs of API packages and supporting files)
- Java 2 Runtime Environment (JRE)
- jBroker ORB (a Java-based CORBA ORB, set up as the default ORB for the machine)

SilverJRunner and SilverJ2EEClient should now be ready to use on this machine.

 For detailed information about UNIX or Linux platform support, see the SilverStream eXtend Application Server Release Notes.

Installing from your SilverStream product CD

Another way to install SilverJRunner and SilverJ2EEClient is to use the SilverStream installation (Setup) program from your SilverStream product CD. It provides choices to install them (and supporting files) alone or along with other SilverStream components.

Installing SilverJRunner and SilverJ2EEClient this way is primarily for developers. The result is the same as accessing the SilverJRunner install page (SilverJRunner.html) from the SilverStream server to do the installation.

 For more information on using the SilverStream installation program, see the following chapters in the *Installation Guide*:

- Installing SilverStream on Windows
- Installing SilverStream on UNIX (also applies to Linux)

Starting SilverJRunner and SilverJ2EEClient

You can start SilverJRunner and SilverJ2EEClient in several different ways, depending on the platform you're using (Windows, UNIX, or Linux).

Running on Windows

In Windows, you can start SilverJRunner and SilverJ2EEClient by doing either of the following:

- Using the executables
- Using SJR and SJC files


Using the executables You can start SilverJRunner from the command prompt by invoking the executable program SilverJRunner.exe. Type:


```
installdirectory\bin\SilverJRunner [options]
[protocol://]hostname[:port] dbname formname [appargs]
```

You can start SilverJ2EEClient from the command prompt by invoking the executable program SilverJ2EEClient.exe. Type:

```
installdirectory\bin\SilverJ2EEClient [options]
[protocol://]hostname[:port] dbname clientname [appargs]
```

where:

For this value	You specify
<i>installdirectory</i>	The root directory where SilverJRunner and SilverJ2EEClient are installed on the user machine. For example: c:\SilverJRunner
<i>options</i>	(Optional) Startup options for controlling the execution of SilverJRunner or SilverJ2EEClient. Specify zero or more of these. For example: -ss_username=sam -ss_password=icecream  See "Using startup options" on page 195.
<i>protocol</i>	(Optional) One of the following HTTP protocols: <ul style="list-style-type: none"> • http:// (default) • https:// (for SSL connections)

For this value	You specify
<i>hostname</i>	The host name (or Internet address) of the SilverStream server to access. For example: corporate
<i>port</i>	(Optional) The TCP/IP port number that server uses. For example: 8080 The default is 80.
<i>databasename</i>	The name of the SilverStream database containing the object (form/view or client deployment) to run. For example: sales
<i>formname</i>	(For SilverJRunner only) The name of the form (or view) to run. For example: frmMonthlyQuota
<i>clientname</i>	(For SilverJ2EEClient only) The name of the J2EE application client deployment to run. For example: quotaclient
<i>appargs</i>	(Optional) Application-specific arguments that you want to pass through to your form or client for processing. Specify zero or more of these. For example: myarg -myswitch -y 2001  See “Passing application arguments” on page 200.

For example:

```
c:\SilverJRunner\bin\SilverJRunner
-ss_username=sam -ss_password=icecream
http://corporate:8080 sales frmMonthlyQuota
myarg -myswitch -y 2001
```

or

```
c:\SilverJRunner\bin\SilverJ2EEClient
-ss_username=sam -ss_password=icecream
http://corporate:8080 sales quotaclient
myarg -myswitch -y 2001
```

If you don't want to type the command every time, you can create a batch (BAT) file to issue it.

Using SJR and SJC files Another alternative to starting SilverJRunner and SilverJ2EEClient from the command prompt is to use SJR and SJC files:

- **SJR**s are SilverJRunner application files in which you store all the arguments you'd otherwise type. Opening an SJR file automatically invokes the SilverJRunner executable SilverJRunner.exe and uses those arguments.
- **SJC**s are SilverJ2EEClient application files in which you store all the arguments you'd otherwise type. Opening an SJC file automatically invokes the SilverJ2EEClient executable SilverJ2EEClient.exe and uses those arguments.

The associations between the SJR file extension and SilverJRunner.exe and between the SJC file extension and SilverJ2EEClient.exe are automatically set up in Windows when you install SilverJRunner and SilverJ2EEClient.

➤ To create an SJR file:

1. Open a new text file in an editor of your choice.
2. Type the following on a single line:

```
[options] [protocol://]hostname[:port] databasename formname  
[appargs]
```

For example:

```
-ss_username=sam -ss_password=icecream  
http://corporate:8080 sales frmMonthlyQuota  
myarg -myswitch -y 2001
```

3. Save this text file with the extension SJR. For example:

```
quota.sjr
```

➤ To create an SJC file:

1. Open a new text file in an editor of your choice.
2. Type the following on a single line:

```
[options] [protocol://]hostname[:port] databasename clientname  
[appargs]
```

For example:

```
-ss_username=sam -ss_password=icecream  
http://corporate:8080 sales quotaclient  
myarg -myswitch -y 2001
```

3. Save this text file with the extension SJC. For example:

```
quota.sjc
```

Once you have an SJR or SJC file, you can:

- Create a Windows shortcut to launch it
- Send it to someone in e-mail
- Link to it from any HTML page—for instance:

```
<a href="quota.sjr">Monthly Quota Form</a>
```

or

```
<a href="quota.sjc">Quota Client</a>
```

Running on UNIX or Linux

In UNIX or Linux, you can start SilverJRunner and SilverJ2EEClient by doing either of the following:

- Using the executables
- Executing the JRunner class yourself


Using the executables You can start SilverJRunner from the command prompt by invoking the executable program SilverJRunner. Type:


```
installdirectory/bin/SilverJRunner [options]  
[protocol://]hostname[:port] databasename formname [appargs]
```

You can start SilverJ2EEClient from the command prompt by invoking the executable program SilverJ2EEClient. Type:

```
installdirectory/bin/SilverJ2EEClient [options]  
[protocol://]hostname[:port] databasename clientname [appargs]
```

where:

For this value	You specify
<i>installdirectory</i>	The root directory where SilverJRunner and SilverJ2EEClient are installed on the user machine. For example: <code>/export/home/sam/SilverJRunner</code>
<i>options</i>	(Optional) Startup options for controlling the execution of SilverJRunner or SilverJ2EEClient. Specify zero or more of these. For example: <code>-ss_username=sam -ss_password=icecream</code>  See “Using startup options” on page 195.
<i>protocol</i>	(Optional) One of the following HTTP protocols: <ul style="list-style-type: none"> • <code>http://</code> (default) • <code>https://</code> (for SSL connections)
<i>hostname</i>	The host name (or Internet address) of the SilverStream server to access. For example: <code>corporate</code>
<i>port</i>	(Optional) The TCP/IP port number that server uses. For example: <code>8888</code> The default is 8080.
<i>databasename</i>	The name of the SilverStream database containing the object (form/view or client deployment) to run. For example: <code>sales</code>
<i>formname</i>	(For SilverJRunner only) The name of the form (or view) to run. For example: <code>frmMonthlyQuota</code>

For this value	You specify
<i>clientname</i>	(For SilverJ2EEClient only) The name of the J2EE application client deployment to run. For example: <code>quotaclient</code>
<i>appargs</i>	(Optional) Application-specific arguments that you want to pass through to your form or client for processing. Specify zero or more of these. For example: <code>myarg -myswitch -y 2001</code>  See “Passing application arguments” on page 200.

For example:

```
/export/home/sam/SilverJRunner/bin/SilverJRunner
-ss_username=sam -ss_password=icecream
http://corporate:8888 sales frmMonthlyQuota
myarg -myswitch -y 2001
```

or

```
/export/home/sam/SilverJRunner/bin/SilverJ2EEClient
-ss_username=sam -ss_password=icecream
http://corporate:8888 sales quotaclient
myarg -myswitch -y 2001
```

If you prefer to work in the desktop environment, you can set up an icon to issue the command or run from your file manager.

Executing the JRunner class yourself The SilverJRunner and SilverJ2EEClient executables both run the class `com.ssw.jrunner.JRunner` (from the `SilverJRunner.jar` file in your SilverJRunner lib directory). An alternative way to start SilverJRunner or SilverJ2EEClient is to execute this class yourself, either from the command prompt or in your own script.

This sample script starts SilverJRunner:

```
export SILVERSTREAMROOT=/export/home/sam/SilverJRunner
export JRE_HOME=$SILVERSTREAMROOT/jre
export PATH=$JRE_HOME/bin:$PATH
$SILVERSTREAMROOT/jre/bin/java -cp
  $SILVERSTREAMROOT/lib/SilverJRunner.jar:
  $SILVERSTREAMROOT/lib/jndi.jar:
  $SILVERSTREAMROOT/lib/ejb.jar:
  $SILVERSTREAMROOT/lib/activation.jar:
```

```
$SILVERSTREAMROOT/lib/mail.jar:
$SILVERSTREAMROOT/lib/pop3.jar:
$SILVERSTREAMROOT/lib/xerces.jar:
$SILVERSTREAMROOT/lib/xml4j.jar:
$SILVERSTREAMROOT/lib/javax_sql.zip:
$SILVERSTREAMROOT/lib/javax_trans.zip"
com.sssw.jrunner.JRunner -ss_root="$SILVERSTREAMROOT"
-ss_noconsole "$@" -SS_FORM
```

Note that:

- This script adds **SilverJRunner.jar** (as well as various other JARs and ZIPs) to the user's classpath. For SilverJRunner, make sure this is the only SilverStream Silver*.jar file on the classpath. For SilverJ2EEClient, you must also add **SilverApplication40.jar** to the classpath.
- "\$@" includes any command-line arguments provided by the user (startup options, server host, database, form/client, application arguments). If you execute the JRunner class from the command prompt, be sure to type those arguments directly.
- **-SS_FORM** indicates that you want to run SilverJRunner. To run SilverJ2EEClient (the default), simply remove **-SS_FORM**.

Displaying a console window

This section describes various ways to display console information on the screen when running SilverJRunner or SilverJ2EEClient.

Using -ss_showconsole with SilverJRunner Normally, SilverJRunner displays windows only for the forms (and views) that make up your application. But when necessary, you can force it to display a Java console window as well. The console enables you to see the System.out and System.err output from your application, which can be particularly useful if you need to debug a problem on the client.

Include the option `-ss_showconsole` when starting SilverJRunner if you want to display the console window.



For details, see “Using startup options” on page 195.

Coding forms to display the console When developing a form in the SilverStream Form Designer, you can use the `showConsole()` method of the `agGeneral` instance variable (`com.sssw.rt.form.PvHelperGeneral` class) to display the Java console programmatically. In that case, you won't need the `-ss_showconsole` option when starting SilverJRunner.

Using the console version of SilverJRunner The SilverStream development environment on Windows provides another way to display standard output and error messages from SilverJRunner. SilverJRunner_c.exe (in the SilverStream server's bin directory) is a console version of SilverJRunner that you can invoke instead of the usual end-user executable (SilverJRunner.exe). You can start it from the command prompt by typing:

```
silverstreamdirectory\bin\SilverJRunner_c [options]
[protocol://]hostname[:port] databasename formname [appargs]
```

For example:

```
c:\SilverStream\bin\SilverJRunner_c
-ss_username=sam -ss_password=icecream
http://corporate:8080 sales frmMonthlyQuota
myarg -myswitch -y 2001
```

Any standard output and error messages from SilverJRunner then display in the command prompt window you're running from. An advantage of this approach is that you can see messages related to startup issues.

Using the console version of SilverJ2EEClient The SilverStream development environment on Windows provides a way to display standard output and error messages from SilverJ2EEClient. SilverJ2EEClient_c.exe (in the SilverStream server's bin directory) is a console version of SilverJ2EEClient that you can invoke instead of the usual end-user executable (SilverJ2EEClient.exe). You can start it from the command prompt by typing:

```
silverstreamdirectory\bin\SilverJ2EEClient_c [options]
[protocol://]hostname[:port] databasename clientname [appargs]
```

For example:

```
c:\SilverStream\bin\SilverJ2EEClient_c
-ss_username=sam -ss_password=icecream
http://corporate:8080 sales quotaclient
myarg -myswitch -y 2001
```

Any standard output and error messages from SilverJ2EEClient then display in the command prompt window you're running from. That includes messages related to startup issues.

Displaying your own splash screen

SilverJRunner and SilverJ2EEClient enable you to specify a JPG file of your choice as the splash screen to display whenever they start up.

➤ To specify the splash screen image:

1. Install your JPG file on the user machine in one of the following ways:

- In the file system, on a path that ends with

```
com\sssw\jrunner\Splash.jpg
```

- In a JAR or ZIP file containing that path

2. Set the AGCLASSPATH environment variable to specify either:

- The path leading up to the com directory

```
set AGCLASSPATH=c:\myapp
```

- The path of the JAR or ZIP file

```
set AGCLASSPATH=c:\myapp\mysplash.jar
```

How SilverJRunner updates itself

Whenever you start SilverJRunner, it automatically checks that it's the correct version for the SilverStream server you're accessing. It compares the local SilverApplicationXX.jar file (located in the user machine's SilverJRunner lib directory) with the one on the server. If they don't match, it downloads the server's version of SilverApplicationXX.jar into the local cache and uses that. (If the local cache already contains that version, the download is skipped.)

After SilverJRunner downloads a version of SilverApplicationXX.jar, it starts a second VM to reload its classes from that file.

SilverJ2EEClient in the development environment

When you invoke SilverJ2EEClient (or SilverJ2EEClient_c) in your development environment to test a client deployment, make sure the current directory is not the one containing your local development version of that client (compiled classes, JARs, and so on). Doing so causes classpath confusion that may prevent the client from running properly.

Using startup options


There are two kinds of options you can provide when starting SilverJRunner or SilverJ2EEClient:

- **- options** These are options specific to SilverStream. They are passed to the JRunner class.
- **+ options** (Windows only) Some of these options are passed directly to the Java interpreter (at which time the + characters are changed to - characters). Others are handled by the SilverJRunner or SilverJ2EEClient executable to launch the Java interpreter.


Using - options

The following table describes the - options you can provide when starting SilverJRunner or SilverJ2EEClient:

Option	Description
<code>-ss_username=<i>username</i></code>	<p>Specifies the user name to use when logging in to a SilverStream server. For example:</p> <pre>-ss_username=sam</pre> <p>If you don't supply this option and user authentication is required, you'll get a dialog prompting for user name and password.</p>
<code>-ss_password=<i>password</i></code>	<p>Specifies the password to use when logging in to a SilverStream server. For example:</p> <pre>-ss_password=icecream</pre> <p>If you don't supply this option and user authentication is required, you'll get a dialog prompting for user name and password.</p>



Option	Description
<p><code>-ss_proxy=<i>proxyserver</i></code></p>	<p>Specifies a proxy server that you want to use. Include the proxy server name and, optionally, its port number (the default is 80). Examples:</p> <pre data-bbox="639 395 943 444">-ss_proxy=corpproxy -ss_proxy=corpproxy:8080</pre> <p>Using SSL To use SSL with a proxy server, simply specify the HTTPS protocol for your SilverStream server:</p> <pre data-bbox="639 545 1098 593">SilverJRunner -ss_proxy=corpproxy https://finance payroll frmSalary</pre> <p>or</p> <pre data-bbox="639 664 1133 713">SilverJ2EEClient -ss_proxy=corpproxy https://finance payroll salaryclient</pre> <p> To learn about configuring a SilverStream server for SSL, see the chapter on setting up security in the <i>Administrator's Guide</i>.</p>
<p><code>-ss_dialect=<i>locale</i></code></p>	<p>(For SilverJRunner only) Sets the locale for use with resourced strings. For example:</p> <pre data-bbox="639 939 919 960">-ss_dialect=de_DE_EURO</pre> <p>Valid locales are defined by the Java VM. The default locale is en_US for US English.</p>

Option	Description
-ss_keepalive	<p>(For SilverJRunner only) Used on Windows NT to prevent forms from hanging when the network connection goes down.</p> <p>How it works -ss_keepalive sets the socket option SO_KEEPALIVE to true for TCP connections (the default is false). A TCP packet (keepalive probe) is then sent out periodically from SilverJRunner to the server to confirm that the TCP connection is still alive. If the TCP connection has died, a socket exception is thrown to prevent the thread from blocking on socket read. This dead connection is then released from SilverJRunner.</p> <p>Setup requirements To use -ss_keepalive, you must set the following registry key on the Windows NT machine running SilverJRunner:</p> <pre>HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\ Services\Tcpip\Parameters\KeepAliveTime</pre> <p>KeepAliveTime is a socket parameter for TCP (don't confuse it with the SilverStream server's KeepAlive property). It controls how often the connection is tested. For example, you might set it to 60000 milliseconds (one minute); the default is two hours. Reboot the machine after you set KeepAliveTime.</p> <p>Catching the exception You can code your forms to catch the exception that's thrown if the connection dies:</p> <pre>java.net.SocketException: Connection reset by peer: JVM_rcv in socket input stream read</pre> <p>Then you can perform whatever processing is needed for the situation.</p>

Option	Description
-ss_showconsole	<p>(For SilverJRunner only) Displays a Java console window for the SilverJRunner session.</p> <p> For more information, see “Displaying a console window” on page 192.</p>
-c <i>cache</i> size	<p>Specifies the size (in bytes) of the disk cache for SilverJRunner or SilverJ2EEClient. This cache is used to hold any downloaded JAR files needed to run a form/view or client.</p> <p>You can type the size in any of the following formats:</p> <pre data-bbox="639 651 768 725">-c 2000000 -c 2000K -c 2M</pre> <p>The default is 5M (usually enough for most applications).</p>
-v	<p>(For SilverJRunner only) Required to run a view. This option tells SilverJRunner that the <i>formname</i> you’ve supplied is the name of a view.</p> <p>If you omit -v, SilverJRunner won’t find your view (because it will look for a form with that name).</p>
-? -h -help	<p>Displays usage information about SilverJRunner or SilverJ2EEClient and how to start it.</p>

Using + options

The following table describes the SilverStream + options you can provide when starting SilverJRunner or SilverJ2EEClient in Windows:

Option	Description
+classic, +client, and +server	Specifies the VM to use.  For more information, see <i>Specifying the VM to use</i> .
+profile	Turns on profiling for the session. HotSpot troubleshooting If you're running a version of the Sun HotSpot JVM that doesn't support the Java Virtual Machine Profiler Interface (JVMPi), you'll need to turn off HotSpot to do profiling. Specify the following additional option to turn off HotSpot and use the classic JVM instead: <code>+classic</code> If you upgrade the server Normally, SilverJRunner is self-updating, which means at startup it checks to see if there is a newer version on the server; if so, the newer version is automatically downloaded to the client machine. However, if you start SilverJRunner with the +profile option, it will not self-update. So if you upgrade the SilverStream server, you should run SilverJRunner without +profile one time to get the updated client.  For more information, see the SilverStream Profiler chapter in the <i>Tools Guide</i> of the server's Classic Development Help.
+verbose:vmopts	(For SilverJRunner_c or SilverJ2EEClient_c) Specifies that you want to output only startup options to the screen, without all the other information generated in verbose mode.

You can also pass standard VM options directly to the Java interpreter by specifying them as + options. The + character is automatically converted to - as the option is passed.

For example, specifying:

```
+verbose
```

passes this option to the Java interpreter as:

```
-verbose
```

Passing application arguments

When developing a SilverStream form or a J2EE application client, you can code it to look for one or more application-specific arguments at runtime and then perform some processing based on those arguments. To help you do that, this section describes how to:

1. Pass application arguments to a form when starting SilverJRunner or to a client when starting SilverJ2EEClient
2. Enable the form or client to access and use those arguments

Specifying the arguments to pass

When starting SilverJRunner or SilverJ2EEClient, add any application arguments to the end of the command (after the name of the form or client to run). For instance, to pass the arguments **-dollars** and **2** to the frmRates form (in the service database on the custserv server), you specify:

```
SilverJRunner custserv service frmRates -dollars 2
```

To pass them to the client named rateclient, you specify:

```
SilverJ2EEClient custserv service rateclient -dollars 2
```

Here are some additional rules to keep in mind:

- **Arguments before the form/client name** are assumed to be startup options and are not passed to the form or client.
- **Arguments that begin with -ss_** are assumed to be startup options and are not passed to the form or client (even if they are after the form/client name).
- **Startup options not beginning with -ss_** (such as -c and -v) must be before the form/client name. Otherwise, they are assumed to be application arguments and are passed to the form or client.



For more information on the command syntax, see “Starting SilverJRunner and SilverJ2EEClient” on page 186.

Accessing the arguments from a form

You can access the application arguments that SilverJRunner passes by coding the `agGeneral.getArguments()` method on your form (`agGeneral` is a variable that's automatically included on your form to provide an instance of the `PvHelperGeneral` class from `com.sssw.rt.form`). The `getArguments()` method returns a string array containing the passed application arguments.

For example, you might code the following for the form's `formActivate` event:

```
// Get the arguments passed in from SilverJRunner
String[] passedArgs;
passedArgs = agGeneral.getArguments();

// Check those arguments, then use them
if (passedArgs != null && passedArgs.length == 2) {
    Field1.setText(passedArgs[0]);
    Field2.setText(passedArgs[1]);
}
else {
    agDialog.showMessageDialog("Please supply exactly 2 args");
}
```

Accessing the arguments from a client

When `SilverJ2EEClient` starts a client, it invokes the `main()` method of the client's main class and passes any application arguments to that method as a `String` array. You can then read them just as you would in any Java application.




For sample client code that accesses passed application arguments, see the documentation on J2EE application clients in the eXtend Workbench help.

Supporting access to secured EJBs

An Enterprise JavaBean (EJB) deployed on the SilverStream server may be secured so that an IIOP over SSL connection is automatically used when someone tries to access it. To support this kind of connection, SilverJRunner and SilverJ2EEClient include the following JAR file of CA (Certificate Authority) certificates: **agrootca.jar** (located in the SilverJRunner lib directory).

If you need to use a CA certificate that isn't in this file, you must add it. Use the JAR editing tool of your choice (such as the Sun JAR utility or WinZip).

 For more information on secured connections to EJBs, see the chapter on setting up security in the *Administrator's Guide*.

6

Server Implementation Notes

This chapter presents some implementation details about the SilverStream eXtend Application Server that you may need to reference. Topics include:

- J2EE containers
- Session-level failover
- CORBA support
- XML support
- Internationalization support

J2EE containers

At the heart of the J2EE component model are **containers**. Containers are the runtime environments supplied by J2EE platform providers such as SilverStream. Containers provide life-cycle management and other services so that application developers can concentrate on the presentation and business logic of their applications.

The SilverStream eXtend Application Server implements each of the three kinds of J2EE container:

- Web container
- EJB containers
- Client container

Web container

The Web container contains **Web applications**, which are deployed in Web archive (WAR) files. Each WAR you upload to a SilverStream server functions as a complete, standalone application. The WAR file must contain all of the JSP pages, servlets, JavaBeans components, utility classes, static HTML pages, images, and sounds used by the application.

Once you've uploaded a WAR file to the SilverStream server, each JSP page in the WAR behaves like a servlet. The page is associated with an URL. When a Web client or SilverStream object performs an operation on that URL, the SilverStream server finds the associated servlet, instantiates it, and calls the `init()` and `service()` methods associated with the servlet. When the servlet is about to be unloaded, the server calls the `destroy()` method.

The servlet context for any JSP page (or servlet) within the WAR is the WAR itself. That means a JSP page or servlet cannot forward to (or include) a JSP page or servlet that resides in a different WAR, since the WAR defines the boundaries of the application. (Similarly, a JSP page cannot forward to (or include) a classic SilverStream dynamic page.)

JSP pages running on a SilverStream server are not persistent. A new instance of a JSP page might be created for each HTTP request (depending on whether the JSP page is defined as `threadsafe` in the deployment descriptor).

SilverStream uses the Servlet API to implement response buffering. SilverStream uses the following methods of the `ServletResponse` interface to buffer response data:

- `getBufferSize()`
- `setBufferSize()`
- `isCommitted()`
- `reset()`
- `flushBuffer()`

If a servlet does not specify the length of its content by calling `setContentLength()`, the SilverStream server uses HTTP 1.1 chunking to transfer the content as a series of chunks.

How URLs are processed

When a Web client or SilverStream object requests a resource in a WAR, the SilverStream server breaks the request URL down into several components:

Component	Description
context path	Identifies the WAR
servlet path	Identifies the requested item (JSP page, servlet, or other resource) in the WAR
pathinfo	(Optional) Provides extra data to be passed to the servlet. In general, performance is better when query parameters are passed to a request instead of pathinfo data (because pathinfo slows down the lookup process, but parameters do not).

The following example shows the components of the URL for a WAR file deployed at **myWar**:

```
http://host/db/path/to/war/myWar/foo.jsp/pathinfo/for/jsp
```

Here the context path is **/db/path/to/war**, the servlet path is **/myWar/foo.jsp**, and the pathinfo is **/pathinfo/for/jsp**.

Dispatching requests within a WAR

A JSP page or servlet can forward to (or include) any other JSP page or servlet that resides in the same WAR.

If you use the `<jsp:forward>` or `<jsp:include>` action, the target URL can be context-relative or page-relative. A **context-relative** URL begins with a slash (/) and is interpreted relative to the WAR. A **page-relative** URL does not begin with a slash and is interpreted relative to the current page.

Suppose the complete URL for a JSP page is

http://localhost/myDatabase/jsptests/myjsps/test.jsp, and the URL you give to the WAR at deployment time is **jsptests**. In this case you can forward a request to the page by embedding the following tag in a JSP page in the same WAR:

```
<jsp:forward page="/myjsps/test.jsp"/>
```

In a servlet, if you use the `getRequestDispatcher()` method of the `ServletContext` object to specify the target URL, the URL is context-relative. It must begin with a slash and is interpreted relative to the WAR. To forward a request to


http://localhost/myDatabase/jsptests/myjsps/test.jsp, you could embed the following code in a servlet in the same WAR:

```
ServletConfig sconfig = getServletConfig();
ServletContext sc = sconfig.getServletContext();
RequestDispatcher rd =
    sc.getRequestDispatcher("/myjsps/test.jsp");
rd.forward(req, res);
```

The `forward()` method call passes the implicit **request** and **response** objects as arguments.

JSP pages and session management

The SilverStream server can use either cookies or URL rewriting to track sessions. SilverStream uses cookies if the browser supports them and uses URL rewriting if the browser does not.

 For more information, see the section on session management in Server Configuration in the *Administrator's Guide*.

Dispatching requests to a WAR from a page or business object

Although each WAR defines a standalone application, the SilverStream server will allow you to forward to a JSP page (or servlet) in a WAR from a classic SilverStream dynamic page or business object running on the same server. In this case the URL you specify is server-relative. The URL must begin with a slash followed by the name of the target database. For example, in the `pageRequestBegin` event for a page, you could add this code to redirect the request to a JSP page:

```
ServletConfig sconfig = getServletConfig();
ServletContext sc = sconfig.getServletContext();
RequestDispatcher rd = sc.getRequestDispatcher
    ("/myDatabase/jspstests/jspurl1/myjsps/test.jsp");
rd.forward(req, res);
```

EJB containers

The EJB container provides the runtime environment for EJB2.0 and EJB 1.1 Enterprise JavaBeans. The runtime environment includes such low-level services as naming services, remote access, security, and transaction support.

This section discusses the EJB 2.0 container and includes these topics:

- About existing EJB1.1 beans
- About the EJB 2.0 container
- About the EJB 1.1 container
- Calling EJBs

About existing EJB1.1 beans

If you have existing EJB 1.1 beans, you can deploy them to the EJB container, but first you'll have to update the deployment plan to use the correct DTD. For more information on EJB deployment plans, see Chapter 3, "Deployment Plan DTDs".

The SilverStream eXtend Workbench Deployment Plan Editor will convert the deployment plan for you. For more information on using the Deployment Plan Editor convert deployment plans, see the Workbench help.

There are a few cases when you cannot deploy an existing EJB 1.1 bean to the SilverStream 2.0 container. For these cases (and to support backward compatibility), SilverStream supplies an EJB 1.1-only container. You must use the EJB 1.1 container if your EJB 1.1 JARs contain CMP entity beans and use SilverStream extensions to specify CMP details (such as expressions, foreign bean mappings, and complex fields). These extensions are supported by the EJB 1.1 deployment plan, but not the 2.0 plan. These extensions are deprecated and should be upgraded as soon as is possible. When you use Workbench to update the deployment plan for CMP entity beans, you will need to update the WHERE Clause (because SilverStream expressions in finder methods are not supported). Workbench converts the SilverStream expressions to a simple expression that includes WHERE clause of the SQL statement and one or more input parameters. This is similar to EJB-QL. The first input parameter from the finder is converted to ?1, the second is converted to ?2, and so on. For example:

```
<sqlWhereClause>  
    WHERE COL1=?1  
</sqlWhereClause>
```



For more information on upgrading, see the Workbench help.

About the EJB 2.0 container

This section describes the features of the EJB2.0 container and includes these topics:

- EJBs supported by the EJB2.0 container
- EJB 2.0 container services

EJBs supported by the EJB2.0 container

The EJB2.0 container supports:

- **Message-driven beans**—Message-driven beans let you access messages from a queue or a topic managed by the JMS message server.
- **Session beans**—Both stateful and stateless session beans.

- **Entity beans**—Supports both bean-managed (BMP) and container-managed (CMP) entity beans. It supports the following entity bean features.

Feature	Description
Container-managed persistence	Supports CMP via object-relational mapping, and supports access to legacy systems through Resource Adapters. CMP entity beans can comply with the EJB 1.1 or 1.2 persistence model, but they cannot include SilverStream's proprietary features supported in earlier versions (such as foreign bean mapping and WHERE clauses that use SilverStream's expression language).
Autoincrement fields	Supports autoincrement fields in target databases via the autoInc element in the deployment plan. If the target database supports autoincrement fields, you need to add an empty autoInc element to the deployment plan. For databases that do not support autoincrement, you can use the schemaName, autoIncTableName, and columnName elements to generate a unique number. For Oracle databases, you can use the autoIncSequenceName element to specify the sequence name.
Concurrency	Supports both optimistic and pessimistic concurrency. Use the isolationLevel element of the deployment plan to specify the concurrency strategy.
Controlling container-generated SQL	To view the container-generated SQL in the server console, start the server with +DSSSWEJBdebug=1. To override the container-generated SQL, specify the SQL to use instead in the sqlSubstitutionList element in the deployment plan. The container does not perform any error checking on the substituted SQL.
Data loading	Supports both eager and lazy loading. Use the delayInstantiation element in the deployment plan to specify how to load data. The default is false (eager loading).
Link tables	Supports link tables in many-to-many relationships only. Use the linkTable element in the deployment plan.

EJB 2.0 container services

This section describes the services provided when an EJB is deployed to the SilverStream eXtend Application Server EJB 2.0 container.

Debugging

The EJB 2.0 container provides debugging support at both **deployment time** and **runtime**.

Deployment debugging support Deployment debugging support is provided through the command-line tool **SilverCmd**. The following table describes the commands that are most useful for debugging:

SilverCmd command	Description
ValidateEJB	Validates EJBs against the specification Validates the deployment plan and the deployment descriptor Is called by both SilverCmd DeployEAR and DeployEJB Generates errors and warnings
DeployEAR	Allows you to specify verbose level (-v). You can specify three levels of messages: <ul style="list-style-type: none"> • Low: specify 1 • Medium: specify 3 • High: specify 5 To skip validation, use -n
DeployEJB	



For more information on SilverCmd, see Chapter 4, “SilverCmd Reference”

Runtime debugging support Runtime debugging support is provided through a server startup switch or a command shell from the server console. The following table describes the switches and how to use them:

Server switch	Description
SSSWEJBDebug	<p>Lets you specify the types of messages the server should output. Use this syntax when starting the server:</p> <pre>SilverServer +DSSSWEJBDebug=n</pre> <p>Valid values for n are:</p> <ul style="list-style-type: none"> • 1: Shows the SQL the container generates for CMP entity beans (as an alternative, you can use <code>sql.debug</code>) • 2: Shows transaction starts, commits, and rollbacks • 4: Shows any Exceptions • 8: Shows security • 32: Shows the contents of the context pool <p>To specify more than one type of output Use the sum of values. For example, to see SQL and Exceptions, use the value 5.</p> <p>Viewing messages You can view the messages for all deployed beans or for a specific bean. To see values for a subset of beans, use <code>+DSSSWEJBDebugName</code> (described next).</p>
SSSWEJBDebugName	<p>Lets you specify a single bean or a set of beans for which you want to see EJB debug messages. Use this in conjunction with the <code>SSSWEJBDebug</code> flag. The syntax when starting the server is:</p> <pre>SilverServer +DSSSWEJBDebugName=name</pre> <p>where <i>name</i>:</p> <ul style="list-style-type: none"> • is the <code>ejb-name</code> element of the deployment descriptor • is case-sensitive • can be a single letter or a string (debug messages will display for any bean whose name contains the letter or string)

Instance pooling

The EJB 2.0 container supports instance pooling for entity, stateless session, and message-driven beans. Pooling is per bean and is specified in the deployment plan. To reconfigure the pool size, you must redeploy the bean.

For **entity and session beans**, you specify values for the following elements:

Deployment plan element	What you do
Initial pool size	Set this to a number greater than zero to enable pooling
Maximum pool size	Set the maximum number of unused instances in the pool The default is 500 (for session beans), 0 for entity beans, and 5 for message-driven beans
Pooling policy	Define what happens when the maximum pool size is reached Values are CREATE and FAIL (described below)

Entity and stateless session bean pooling policies The EJB 2.0 container does not create the instance pool when you deploy the bean; instead, it populates the pool as the need for instances increases.

- When the poolingPolicy element is set to **FAIL**, the server throws an exception when maximum pool size is exceeded.
- When the poolingPolicy element is set to **CREATE**, the server manages the pool as follows:
 - When the pool is empty, SilverStream does not wait for an instance to be freed; it simply creates a new instance for the caller
 - Until the pool reaches maximum pool size, the container returns all instances to the pool when clients are finished with them
 - When the pool is at maximum pool size, SilverStream discards the instances that are freed
 - When pool size is set to zero, instances are not reused

For **message-driven beans**, the container provides instance pooling by implementing the ServerSessionPool interface.

Load balancing

Load balancing is per session. All EJBs used within a single session reside on one server within the cluster. Load balancing is transparent to the user; the client can do a normal JNDI lookup, and the naming server selects the server for the bean to run on.

When using EJBs within a cluster, you **must** start your servers on different name service ports. The default name service port is 54890. You can configure the name service port using the SMC.



For more information on SilverStream clustering mechanisms, see the *Administrator's Guide*.

Naming service

EJBs are registered in the root context of the Java Naming and Directory Interface (JNDI). If you specify a hierarchical naming structure for JNDI names, bean references, resource references, environment variables, or UserTransactions in the deployment plan, the container creates any intermediate subcontexts that do not already exist.

By default, SilverStream registers bean references, environment variables, and resource references in the **java:comp/env** context. You can follow the recommendations of the EJB specification and store the objects in separate subcontexts, but SilverStream does not enforce these naming conventions—so you can use the naming conventions that work best in your own production environment.

Remote access

SilverStream supports access to EJBs via RMI/IIOP using the jBroker Object Request Broker (ORB). jBroker is an enterprise-class Java-based CORBA ORB.

For portable look ups for EJBs, use the CORBA name syntax like this:

```
corbaname:iiop:host:port#name
```

For example:

```
corbaname:iiop:MyMachine:54520#MyEJB
```

To look up a local bean use:

```
EJBLocalHome/beanName
```



For more information on jBroker, see the documentation included in the jBroker subdirectory of the SilverStream installation directory.

Security

The EJB 2.0 container manages security at runtime using:

- **Authentication** of principals
- **Access authorization** for EJB calls and resource manager access
- **Secure communication** with remote clients

Authentication and caller propagation

When you call an EJB residing on a SilverStream server, the container authenticates the user and uses the caller's identity for the duration of the caller's session on that server. All method calls run with the identity of that session. You can map a role name to a principal (user, group, or list of principals) in the deployment plan.

Access authorization

Access authorization is defined by the security-role and method-permission elements specified in the deployment descriptor. If you secure at least one method of a bean in the EJB JAR, you must secure all methods—or the container assumes **all** methods with unspecified security are restricted and **cannot** be called by **any** user. In addition, you can specify a set of methods that should not be called using the exclude-list element of the deployment descriptor. When a restricted method is called, the container throws an AccessRightsViolation exception. Alternatively, you can choose a nonsecure mode by not securing any methods.

Secure communications

You can establish a secure connection between an EJB client and the SilverStream server using SSL. The jBroker ORB provides the IIOP over SSL support for RSA only.

You **do not** need to be running HTTPS. The following are required:

- An RSA certificate must be installed on the server.
- The deployment plan must specify:
 - One or more cipher suites for integrity and confidentiality (integrity and confidentiality may have different sets of cipher suites)
 - An iorSecurityConfig element for each session and entity bean
- For external and SilverJRunner clients, you also need the agrootca.jar as described in the table in “Accessing the server” on page 219.



For more information on establishing a secure connection between an EJB client and the SilverStream server, see the chapter on setting up security in the *Administrator's Guide*.

Transaction support

The EJB 2.0 container supports distributed transactions via the jBroker Transaction Manager, which fully implements Java Transaction Service (JTS).

If the transaction attribute is not specified in the deployment descriptor, the container uses the Supports attribute as the default transaction attribute.

About the EJB 1.1 container

This section describes:

- EJBs supported by the EJB 1.1 container
- EJB 1.1 container services

EJBs supported by the EJB 1.1 container

The EJB 1.1 container supports the following Enterprise Java Beans:



- **Session beans**—stateful and stateless session beans.
- **Entity beans**—bean-managed (BMP) and container-managed (CMP). It supports the following entity bean features.


Feature	Description
Container-managed persistence	Supports container managed persistence via object-relational mapping. CMP entity beans must comply with EJB1.1 persistence and can contain SilverStream expressions (although beans that use nonstandard container features are not portable.)
Autoincrement fields	Supports autoincrement fields automatically.
Concurrency	Supports optimistic concurrency.
Data loading	Supports both eager and lazy loading. Use the delayInstantiation element in the deployment plan to specify how to load data. The default is false (eager loading).
Removing entity beans	Delays cleaning up remote object references for entity beans until the associated session is invalidated or expires.

EJB 1.1 container services

The following services are provided when an EJB is deployed to the SilverStream eXtend Application Server:

Container service	Description
Instance pooling	<p>Maintains a pool of idle/unused stateless session beans. This allows the container to dynamically assign instances of a stateless session bean to different clients, which allows for better performance and scalability. You can define a maximum pool size for each stateless session bean at deployment. The default is 500.</p> <p>SilverStream does not create the instance pool when you deploy the bean; instead it populates the pool as the need for instances increases. It manages the pool as follows:</p> <ul style="list-style-type: none">• The container returns all instances to the pool (unless the pool has reached max pool size) when clients are finished with them.• If the pool is at max pool size, the container discards the instances that are freed.• When the pool is empty, the container does not wait for an instance to be freed; it simply creates a new instance for the caller. When the client is finished with that instance, the container discards it instead of putting it back in the pool. This way the maximum pool size is never exceeded.• To reconfigure the max pool size, you must redeploy the bean.• It is valid to set the pool size to zero. When it is set to zero, instances are never reused.

Container service	Description
Load balancing	<p>Load balancing is per session. All EJBs used within a single session reside on one server within the cluster. Load balancing is transparent to the user; the client can do a normal JNDI lookup, and the naming server handles the selection of the server to run the bean on.</p> <p>When using EJBs within a cluster, you must start your servers on different name service ports. The default name service port is 54890. You can configure the name service port using the SMC.</p> <p>Failover is your responsibility the developer's responsibility.</p> <p> For more information on SilverStream clustering mechanisms, see the <i>Administrator's Guide</i>.</p>
Naming service	<p>EJBs are registered in the root context of the Java Naming and Directory Interface (JNDI). If you specify a hierarchical naming structure for JNDI names, bean references, resource references, environment variables, or UserTransactions in the deployment plan, the container creates any intermediate subcontexts that do not already exist.</p> <p>By default, the container registers bean references, environment variables, and resource references in the java:comp/env context. You can follow the recommendations of the EJB specification and store the objects in separate subcontexts, but the container does not enforce these naming conventions—so you can use the naming conventions that work best in your own production environment.</p>
Remote access	<p>The SilverStream server supports access to EJBs via RMI/IIOP using the jBroker Object Request Broker (ORB). jBroker is an enterprise-class Java-based CORBA ORB.</p> <p> For more information on jBroker, see the documentation included in the jBroker subdirectory of the SilverStream installation directory.</p>

Container service	Description
Security	<ul style="list-style-type: none"> • Support security at runtime using: Authentication of principals • Access authorization for EJB calls and resource manager access • Secure communication with remote clients <p>Authentication and caller propagation When you call an EJB residing on a SilverStream server, the container authenticates the user and uses the caller's identity for the duration of the caller's session on that server. All method calls run with the identity of that session. You can map a role name to a principal (user or group name).</p> <p>Access authorization Access authorization is defined by the security-role and method-permission elements specified in the deployment descriptor. If you secure at least one method of a bean in the EJB JAR, you must secure all methods—or the container assumes all methods with unspecified security are restricted and cannot be called by any user. When a restricted method is called, the container throws an <code>AccessRightsViolation</code> exception. Alternatively, you can choose a nonsecure mode by not securing any methods.</p> <p>Secure communications You can establish a secure connection between an EJB client and the SilverStream server using SSL. The <code>jBroker ORB</code> provides the IOP over SSL support for RSA only. You do not need to be running HTTPS. The following are required:</p> <ul style="list-style-type: none"> • An RSA certificate must be installed on the server • The deployment plan must specify SSL and one or more valid cipher suites (per bean using IOP over SSL) • For external and SilverJRunner clients, you also need the <code>agrootca.JAR</code> as described in the table in “Accessing the server” on page 219 <p> For more information on establishing a secure connection between an EJB client and the SilverStream server, see the chapter on setting up security in the <i>Administrator's Guide</i>.</p>

Container service	Description
Transactions	<p>The EJB 1.1 container provides transaction support within a single database. It does not support distributed transactions. It supports user transactions via the <code>javax.transaction.UserTransaction</code> interface. The container binds the <code>UserTransaction</code> to the JNDI name RMI/SilverStream-UserTransaction. For session beans, the <code>UserTransaction</code> is available as part of the bean environment context or as a method on the <code>javax.ejb.SessionContext</code>.</p> <p>If the transaction attribute is not specified in the deployment descriptor, the container uses the <code>Supported</code> attribute as the default transaction attribute.</p>

Calling EJBs

You can access EJBs from:

- **J2EE components** (EARs, EJBs, WARs, or application clients) that are deployed on the **same server** as the EJB they are calling
- **J2EE components** deployed on a **different server** from the EJB they are calling
- **Applications** that are **not deployed to a container** (often referred to as standalone or external clients)

Before a J2EE component or external client can find and use a deployed EJB, it **must** connect to the server and establish a session. (The session allows the SilverStream server to authenticate the user of the EJB.) When the application is finished using the EJB, it should end the session.

Accessing the server

How you access the SilverStream server depends on the type and location of your client, as described below:

Client type	What to do
A J2EE component deployed on the same SilverStream eXtend Application Server as the target EJB	No explicit connection is required. The SilverStream server automatically establishes and manages the session.
J2EE components calling EJBs on a different SilverStream eXtend Application Server	<p>The J2EE component must explicitly connect and establish a session by calling a variant of the <code>com.sssw.rt.util.AgRuntime.connect()</code> method—for example:</p> <pre data-bbox="586 756 953 803">AgrServerSession mySess = AgRuntime.connect(otherHost);</pre> <p>Use a full specification to the other server name in the <code>javax.naming.InitialContext.lookup()</code> method.</p> <p>Connecting to a remote server inside a firewall To call an EJB that resides on a server inside a firewall that filters out HTTP traffic, use the <code>connectRMI()</code> method of the <code>com.sssw.rt.util.AgRuntime</code> class instead of <code>connect()</code>. This establishes an RMI/IIOP connection to the server instead of an HTTP connection.</p>

Client type	What to do
External clients	<p>The external client must explicitly establish a session as described above. In addition, an external client accessing an EJB needs a JAR that contains the EJB's interfaces (for compile-time references) and the container-generated stubs (for runtime). The nature of the JAR you use depends on the version of the EJB container you are accessing:</p> <ul style="list-style-type: none">• For the EJB 2.0 container, you should create an EJB client JAR when you develop your EJBs and make sure the deployment descriptor includes the <code>ejb-client-jar</code> element. When this element is present, the container generates stub classes, places the stubs in the client JAR, and uploads the client JAR to the server.• For the EJB 1.1 container, external clients must use the EJB Remote JAR, which is created by the container during deployment. For more information, see Chapter 2, "J2EE Archive Deployment". <p>Downloading the JAR from the server Since both the EJB client JAR and the Remote EJB JAR are generated by the container and reside on the server, you can use the SilverCmd <code>PublishToFile</code> utility to download either JAR from the server to the desired location on disk. Put the disk location of the appropriate JAR on the classpath of the external client. Remember that you must download an updated version each time you make changes to the EJB and redeploy it.</p>

Client type	What to do
External clients using SSL	<p>External clients using SSL must have access to the same JAR as other external clients as described above—and must also have access to agrootca.JAR. The agrootca.JAR file contains every Certificate Authority (CA) that is supported (by default) on the SilverStream server. This file is installed in the SilverStream installation directory under the \lib subdirectory.</p> <p>If you need to use a CA not in this file, you must add it to the file and propagate this change to all SilverJRunner and external clients that require this CA. You can modify this file using any tool that allows you to modify the contents of a JAR file (such as Sun’s JAR utility or Winzip).</p> <p>SilverJRunner clients can automatically locate this file; but for external Java clients, you must specify agrootca.JAR’s location using the AGROOTCA system-level Java property.</p>

Client container

J2EE application clients are the standard way to provide Java-based clients that run on user machines and access J2EE servers. They are hosted by a client container that (at minimum) provides JNDI namespace access. Beyond that, the J2EE specification allows for a wide range of client container implementations, from basic to robust.

The SilverStream server supplies a client container named **SilverJ2EEClient** that users can invoke to run J2EE application clients you’ve deployed to the SilverStream server. SilverJ2EEClient provides a robust set of supporting services, including:

- Easy container installation
- Automated client deployment to user machines
- User authentication and session housekeeping
- JNDI namespace access



For details, see Chapter 5, “SilverJ2EEClient and SilverJRunner”.

Session-level failover

The SilverStream eXtend Application Server supports session-level failover for Web applications (WARs) and stateful session beans (EJB JARs). **Session-level failover** refers to the ability of an application to retain temporary user data (state) across server failures in a cluster. The data is stored in a persistent storage repository (such as a database or file system shared by the servers in the cluster) so that it can be recovered by any server in the cluster in the event of a server failure.

To support session-level failover, you must have a hardware dispatcher installed as the dispatcher for your cluster. All clients accessing the applications configured for session-level failover should access the application via the cluster's (hardware) dispatcher.

Failover support is not a mechanism for load-balancing EJBs belonging to a single session across multiple servers.

EJB support for session-level failover

The EJB container supports session-level failover for stateful session beans (local and remote). The stateful beans must meet these requirements:

- The **recoverable** element in the SilverStream deployment plan must be set to true.
- The session bean must support activate and passivate as described in the section on instance passivation and conversational state in the EJB2.0 specification.
- The session bean's methods must be transactional (specified in the deployment descriptor). Changes to the session bean's state that occur outside of a transaction are not recoverable if the system crashes; changes to the session bean's state that occur in the context of a transaction are recoverable.

How session-level failover works for stateful session beans

If your session beans meet the requirements listed above and a failure occurs, the recovery works like this:

1. At the end of each transaction that includes one or more recoverable stateful session beans, the EJB container passivates and serializes all the recoverable beans used in the transaction and saves them to the database (the AgSessBeans table in the SilverMaster database.)
2. As long as the server is up, the client will continue to use the same server.

3. If the client gets a communication failure on a remote call, it assumes the server failed.
 - If the failure occurs between transactions, the client automatically chooses another server in the cluster and retries the call to it. (It does not require a hardware dispatcher.) The session bean's state is then restored from the database on the new server just as though the bean had been previously passivated.
 - If the failure occurs when there is already a transaction in progress, the call is not automatically retried, instead the transaction is rolled back and the client gets the exception. It is the client's responsibility to recover from a transaction rollback (for example, the client can retry the call).

The performance of your EJB applications might be impacted if the recoverable session bean does a lot of work in the `ejbActivate()` method. For example, if the session bean allocates and caches a database connection.

Server settings to support session-level failover To support session-level failover when using IIOP over SSL, you must also configure a range of ports for IIOP SSL communications for the server.



For more information on setting the range of ports, see the section on specifying ORB settings in the *Administrator's Guide*.

Web application support for session-level failover

The WAR container supports session-level failover. The Web application must meet the following requirements:

- The **distributable** element is present in the deployment descriptor
- The **recoverable** element in the SilverStream deployment plan is set to true
- The components in the WAR follow the rules about distributable objects outlined in the Servlet 2.3 specification.
 - The objects written to the `HTTPSession` object must be `Serializable`. The SilverStream server also supports failover of EJB references and `UserTransaction` objects.
 - State cannot be stored in static or instance variables.

How session-level failover works for Web applications

If your Web application meets the requirements listed above and a failure occurs, the session-level failover recovery works like this:

1. On each HTTP request to the Web application, the server serializes the `HTTPSession` state to the database (the `AgSessBeans` table of the `SilverMaster`) at the end of each request.
Because the container passivates, serializes, and saves the `HTTPSession` state to the database at the end of each HTTP request, this can impact the overall performance of the application.
2. As long as the server is up, the client requests will continue to be directed to the same server.
3. If the server fails between requests and you have a hardware dispatcher, the dispatcher detects the server failure and sends the next request to a different server.
 - The new server restores the `HTTPSession` state from the database and the operation continues without interruption.
 - If the server fails during a request, the browser will eventually timeout the response. When the user resubmits (assuming a hardware dispatcher), the resubmitted request goes to a new server which restores the `HTTPSession` state as described above.

If you do not have a hardware dispatcher If you use the `SilverStream` software dispatcher (instead of a hardware dispatcher), after the failure, the user will have to manually return to the dispatcher to be re-dispatched. Once re-dispatched, the new server will not automatically restore the state since the session ID cookie will be different.

When failover might not work Because the `HTTPSession` state is not transactional, updates to the `HTTPSession` during a request can be lost under certain circumstances, for example, if the server crashes during a request (but before the state is saved). But if the server crashes after the state is saved, but before returning a reply, then the state can be recovered.

Application client support for session-level failover

A J2EE client application can access Web application components or EJBs that support session-level failover as long as the Web components and EJBs meet the session-level failover requirements described in the previous section. The J2EE client must initially connect to the cluster's (hardware) dispatcher like this:

```
silverj2eeclient dispatcher-name:port database-name application-name
```



For details on SilverJ2EE client, see Chapter 5, “SilverJ2EEClient and SilverJRunner”.

CORBA support

The SilverStream server includes the jBroker ORB. jBroker is an enterprise-class Java-based CORBA ORB. You can use the jBroker ORB from the SilverStream server, SilverJ2EEclient, SilverJRunner, or any browser. You can use it to develop, deploy, and manage Java-based CORBA applications. The SilverStream server uses the following subset of features:

- Bootstrap protocol
- COS naming
- Java objects by value
- Java RMI/IIOP
- Objects by value support for IDL
- IIOP over SSL

 For more information on jBroker, see the jBroker documentation available in the server's Core Help.

XML support

The XML language (eXtensible Markup Language) allows you to create XML documents that can be used to exchange data between computer systems (of different types) and applications on the Web. This section describes SilverStream use of and support for XML documents. It covers the following topics:

- SilverStream XML support
- Resources for learning about XML

SilverStream XML support

SilverStream uses XML documents for the following:

- J2EE archive deployment descriptors and deployment plans
- SilverCmd input files

SilverStream automatically installs the IBM XML for Java parser (XML4J) into the SilverStream lib subdirectory. XML4J supports XML's core features, which you use within your SilverStream applications.

Using XML with SilverStream SilverStream explicitly supports XML for J2EE archive deployment and for SilverCmd input files. If you use XML with SilverStream only for these purposes, you don't need to know anything about the XML parser; for information, see:

- Chapter 2, “J2EE Archive Deployment”
- Chapter 4, “SilverCmd Reference”

If you use the other features of XML4J, you should read the following about the XML parser, and you need to be aware that SilverStream support can change over time. You can only use XML4J for server-side applications. The XML library is not available to the SilverStream client-side runtime environment.


About the XML parser

In 1999, IBM turned over XML4J to the Apache group (xml.apache.org). Apache renamed it Xerces, and they now maintain it through Open Source development (with major contributions from IBM and others). The `com.ibm.xml.parsers` tree was renamed `org.apache.xerces.parsers`.

Apache puts out regular releases, but does not really make a distinction between beta-quality releases and production-quality releases. However, IBM takes certain releases of Xerces, tests them for quality, adds backward-compatibility classes so you can continue to use the old class names, and releases them as XML4J.

NOTE IBM is planning to remove the backward compatibility classes. So if you have been using them, you should start using the `org.apache.xerces.parsers` classes instead.

As of the time of this writing, IBM's current release is XML4J 3.1.0, which is the same code as Xerces 1.2.0. In order to keep the XML support current, this version of the SilverStream eXtend Application Server includes XML4J 3.1.0.

 For more information on XML4J's features and API, see the IBM XML4J online help (provided with SilverStream) or IBM's Web site at <http://www.alphaworks.ibm.com/tech/xml4j>.

Updating from XML4J 2.0.15 If you were using Version 2.0.15 of XML4J with SilverStream eXtend Application Server Version 3.0, read the following information about updating from XML4J Version 2.0.15.

The update from XML4J 2.0.15 to XML4J 3.1.0 should be easy, but there may be some code changes you'll have to make. The Apache group has renamed the `com.ibm.xml` tree to `org.apache.xerces`, but IBM includes compatibility classes with the old names. This should allow most existing code to continue to work.

There were some small changes, however. One change affects the DOM parser, and the other affects the SAX parser.

DOM parser `com.ibm.xml.parsers.DOMParser` no longer implements `org.xml.sax.Parser`. This means that (a) you can't use `ParserFactory.makeParser()` to construct this parser, and (b) you can't reference it as an `org.xml.sax.Parser` to call methods like `setEntityResolver()` or `setErrorHandler()` on it—even though these methods still exist on the class.

However, these problems are easy to work around. Just replacing all instances of **`org.xml.sax.Parser`** with **`com.ibm.xml.parsers.DOMParser`** and directly newing up a **`com.ibm.xml.parser.DOMParser`** instead of using `ParserFactory.makeParser()` should do the trick.

Thus the following old code:

```
org.xml.sax.Parser parser =
    ParserFactory.makeParser("com.ibm.xml.parser.DOMParser");
parser.setErrorHandler(...);
parser.setEntityResolver(...);
parser.parse(...);
org.w3c.dom.Document d = ((DOMParser)parser).getDocument();
```

becomes:

```
com.ibm.xml.parser.DOMParser parser = new
    com.ibm.xml.parser.DOMParser();
parser.setErrorHandler(...);
parser.setEntityResolver(...);
parser.parse(...);
org.w3c.dom.Document d = parser.getDocument();
```

You lose the flexibility that the `Parser` interface and the `ParserFactory` factory provided, but this is unavoidable with the upgrade, since there is no common interface for DOM parsers as there is for SAX parsers.

SAX parser This change has to do with the char arrays passed to your `DocumentHandler` implementation in `characters()` and `ignorableWhitespace()`. In XML4J 2.0.15, the parser sent them in such a way that it was often possible to use and store the char arrays directly, without making a copy of the data. In 3.1.0, however, the char arrays are reused across calls; so if you want to keep the data around, you must copy it. For example, you could create a new char array and use `System.arraycopy` to copy the data into it, or you could simply do **`new String(ch, start, length)`**.

Resources for learning about XML

If you're new to XML or just need to explore a specific XML topic, try the following recommended learning resources:

Resource	Description	Available at
SilverStream DevCenter	An index to XML learning and reference materials with links to many documents and Web sites	http://devcenter.silverstream.com/
Directory of XML resources	—	http://www.xmldir.com

Internationalization support

This section describes the following internationalization topics:

- Database support
- Client-side support

Database support

All JDBC drivers certified for use with the SilverStream server have been fully tested to support Western/Eastern European and Asian languages.

➤ **To use the multibyte version of the SilverStream JDBC-ODBC bridge driver:**

1. Add the following line to AgUserIni.props in your SilverStream/Resources directory:
`com.sssw.srv.ambry.mbcS.AgOdbc=true`
2. Restart the SilverStream server.

Client-side support

The SilverStream server includes runtime language libraries for Simplified and Traditional Chinese, Czech, Dutch, English, French, German, Italian, Japanese, Korean, Norwegian, Portuguese, Spanish, and Swedish.

If you encounter font-mapping problems in SilverJ2EEClient, SilverJRunner, or the SilverStream Designer where the correct characters are not displaying, you can correct the problem by editing the JRE's font.properties file.

You must edit the font.properties.XX file in the jre/lib subdirectory of the SilverStream installation directory, where XX is the two-character language encoding for the language you are interested in. For example, you would change font.properties.ko for Korean. There are two sections of interest in the file that appear one after the other. They are labeled **name aliases** and **for backward compatibility**.

The original version of font.properties.ko is:

```
# name aliases
#
# alias.timesroman=serif
# alias.helvetica=sansserif
# alias.courier=monospaced
# for backward compatibility
timesroman.0=Times New Roman,ANSI_CHARSET
helvetica.0=Arial,ANSI_CHARSET
courier.0=Courier New,ANSI_CHARSET
zapfdingbats.0=WingDings,SYMBOL_CHARSET
```

The **name aliases** section maps nonexistent font names to font mappings defined in the file. You should uncomment those alias lines. This is the preferred way of handling the mapping. The section **for backward compatibility** is the old way of mapping nonexistent font names to fonts described in the file.

NOTE Make sure that you comment the first three lines of this section.

The updated version of the file would then be:

```
# name aliases
#
alias.timesroman=serif
alias.helvetica=sansserif
alias.courier=monospaced
# for backward compatibility
# timesroman.0=Times New Roman,ANSI_CHARSET
# helvetica.0=Arial,ANSI_CHARSET
# courier.0=Courier New,ANSI_CHARSET
zapfdingbats.0=WingDings,SYMBOL_CHARSET
```

Index

A

AddCP command, SilverCmd 102
AddDatabase command, SilverCmd (deprecated) 105
 input file requirements 107
AddUserToGroup action 165
agrootca.jar
 with SilverJ2EEClient 202
 with SilverJRunner 202
application arguments
 passing from SilverJ2EEClient 200
 passing from SilverJRunner 200
application clients 224
 deploying archives 119
 packaging 10
 session-level failover 224
 writing deployment plans 10
archives
 J2EE deployment 9
asContextRequired 35
authentication
 and SilverCmd 98
authMethod 34
autoincrement
 EJBs 15

B

batch mode, SilverCmd 99
Build command, SilverCmd (deprecated) 109
BuildWAR command, SilverCmd (deprecated) 111
business objects
 exporting 132
 importing 136
 importing source code 141

C

CARs (client JARs)
 deploying 9
cipher suites 34

classpath JARs
 specifying on the server 47
ClearDefaultURL command, SilverCmd 112
ClearLog command, SilverCmd 113
client container 221
client JAR deployment plan DTD 50, 85
CMP entity beans
 mapping to a table 15
ComGen command, SilverCmd (deprecated) 114
compiler
 setting preferences 147
Confidentiality
 cipher suites 34
confidentiality 33
consoles
 SilverJ2EEClient 192
 SilverJRunner 192
containers, J2EE
 about 203
 client container 221
 EJB container 206
 Web container 203
ConvertEJB command, SilverCmd (deprecated) 115
CORBA
 SilverStream support 225
CreateGroup action 166
CreatePackage command, SilverCmd (deprecated) 116
CreateUser action 167

D

databases
 removing 158
debugging
 EJBs 209
Delete command, SilverCmd (deprecated) 117
DeleteGroup action 168
DeleteUser action 168
DeleteUserFromGroup action 169
DeployCAR SilverCmd command 119
DeployEAR command, SilverCmd 120
DeployEAR12 command, SilverCmd (deprecated) 123

DeployEJB command, SilverCmd 125
DeployEJB11 command, SilverCmd (deprecated) 127
Deploying
 RARs 42
deploying
 application client archives 9
 EARs 44, 120
 EJB JARs 37, 39
 EJBs 12, 37, 125
 EJBs JARs 125
 J2EE archives 9
 JSP pages to file system 1
 SilverJ2EEClient 181
 SilverJRunner 181
 WARs 40, 130
deployment descriptor
 for EJBs, converting from 1.0 115
 validating 178, 179
deployment plan
 EAR 45
 EJB 12
 EJB tips 15
 for an application client 10
 WAR 40, 43
DeployRAR command, SilverCmd 129
DeployWAR command, SilverCmd 130
dia 138
DOCTYPE statements
 client JAR deployment plan DTD 50, 85
 EAR deployment plan DTD 89
 EJB JAR deployment plan DTD 55
 WAR deployment plan DTD 78
double-byte character set version of JDBC-ODBC bridge
 driver 228
downloading objects 151
DTDs (Document Type Definitions)
 client JAR deployment plan 50, 85
 deployment plan, about 49
 deployment plan, files 49
 deployment plan, reference documentation 49
 deployment plan, samples 49
 EAR deployment plan 88
 EJB JAR deployment plan 55
 WAR deployment plan 78

E

EAR deployment plan DTD 88
EARs
 deploying 44
 deployment plan 45
 role maps 92
EJB JAR deployment plan DTD 55
EJB JARs
 deploying 37, 125
 rebuilding 156
 restructuring after deployment 39
EJBs (Enterprise Java Beans)
 SilverJ2EEClient access via IOP over SSL 202
 SilverJRunner access via IOP over SSL 202
EJBs (Enterprise JavaBeans)
 about 15
 access authorization 213
 asContextRequired 35
 authentication context configuration 32
 authMethod 34
 cipher suites 34
 confidentiality 33
 container 206
 converting from 1.0 115
 debugging 209
 deploying 12, 37, 125
 deploying EJB JARs 37
 deployment plan 12
 establishTrustInClient 33
 instance pooling 211
 integrity 32
 IOR configuration examples 36
 IOR configurations 30
 isolation levels 18
 lazy beans 60
 load balancing 212
 mapping CMP entity beans 15
 mapping persistent fields 17
 packaging 12
 primary key mapping 29
 realm 35
 rebuilding JARs 156
 relationship mapping 19
 restructuring JAR files 39
 restructuring JAR files after deployment 39

- security 213
- security and caller propagation 213, 217
- security attribute context 35
- security attribute context configuration 32
- session-level failover 222
- supporting autoincrement 15
- transaction support 214
- transport configuration 32
- entity beans
 - delaying instantiation 60
- error logging
 - SilverCmd 100
- execute mode
 - SilverCmd 99
- ExportSource command, SilverCmd (deprecated) 132

F

- failover 222
- file system
 - deploying JSP pages to 1
- font.properties files 229
- forms
 - accessing arguments passed from SilverJRunner 200

G

- GetConsole command, SilverCmd 134
- GetDefaultURL command, SilverCmd 135
- groups
 - creating 166
 - deleting 168
 - managing 163
 - setting properties for 170

I

- IBM XML for Java parser 225
- images
 - importing 138
- ImportClass command, SilverCmd (deprecated) 136
- importing
 - objects 153

- ImportMedia command, SilverCmd (deprecated) 138
- ImportPage command, SilverCmd (deprecated) 140
- ImportSource command, SilverCmd (deprecated) 141
- install page, SilverJRunner (and SilverJ2EEClient) 183
- instance pooling
 - EJBs 211
- Integrity
 - attribute 32
 - cipher suites 34
- internationalization support for the server 228
- IOR configurations 30
- isolation levels
 - specifying 18

J

- J2EE
 - application clients, accessing arguments passed from SilverJ2EEClient 200
 - archive deployment 9
 - client container 221
 - containers 203
 - EJB container 206
 - Web container 203
- JARs
 - importing 138
 - on application classpath 47
 - rebuilding 156
- Java packages
 - creating 116
- JRunner
 - installing 183
 - starting 186
 - when to use 181
- JRunner class
 - starting SilverJ2EEClient with 189
 - starting SilverJRunner with 189
- JSP pages
 - compiling 130
 - deploying 130
 - deploying to the file system 1
 - implementation in Web container 203
 - URLs for JSP pages running in SilverStream 204
- JSP/FS 1

L

lazy beans 14
 entity beans 60
link tables 27
ListCP command, SilverCmd 143
load balancing 212

M

managing users and groups 163
message-driven beans
 mapping 30
ModifyCP command, SilverCmd 144
ModifyTableList command, SilverCmd
 (deprecated) 145
multibyte version of JDBC-ODBC bridge driver 228

O

objects, deleting 117

P

pages
 importing static 140
persistent fields
 mapping 17
preferences, setting 147
Prefs command, SilverCmd 147
primary keys 29
PrintLog command, SilverCmd 149
Publish command, SilverCmd (deprecated) 150
PublishFromFile command, SilverCmd
 (deprecated) 151
PublishToFile command, SilverCmd (deprecated) 153

Q

QueryCP command, SilverCmd 155

R

RARs
 deploying 42
 deployment plans 43
 packaging 42
realm
 EJBs 35
RebuildJAR command, SilverCmd (deprecated) 156
relationships
 EJB mapping 19
 general restrictions 29
 link tables 27
 many-to-many unidirectional 28
 one-to-many bidirectional 24
 one-to-many unidirectional 26
 one-to-one bidirectional 20
 one-to-one unidirectional 22
RemoveCP command, SilverCmd 157
RemoveDatabase command, SilverCmd
 (deprecated) 158
removing databases 158
role maps
 EARs 92
running SilverCmd 98
runtime language libraries 229

S

sample input files, for SilverCmd 101
security
 and EJBs 213, 217
 EJBs 213
 IOR configurations 30
 setting permissions 162
security attribute context configuration 35
server
 shutting down 159
ServerState command, SilverCmd 159
session beans
 session-level failover 222
session-level failover 222, 224
 EJB support 222
 web applications 223
SetDefaultURL command, SilverCmd 161

- SetGroupProperties action 170
- SetSecurity command, SilverCmd 162
- SetUserGroupInfo command, SilverCmd 163
 - AddUserToGroup action 165
 - CreateGroup action 166
 - CreateUser action 167
 - DeleteGroup action 168
 - DeleteUser action 168
 - DeleteUserFromGroup action 169
 - SetGroupProperties action 170
 - SetUserProperties action 171
- SetUserProperties action 171
- shutting down servers 159
- SilverCmd 95
 - about 98
 - AddCP command 102
 - AddDatabase command (deprecated) 105
 - AddDatabase driver set 107
 - AddDatabase example input file 107
 - AddDatabase valid database connection types and XML files 108
 - Build command (deprecated) 109
 - BuildWAR command (deprecated) 111
 - ClearDefaultURL command 112
 - ClearLog command 113
 - ComGen command (deprecated) 114
 - ConvertEJB command (deprecated) 115
 - CreatePackage command (deprecated) 116
 - Delete command (deprecated) 117
 - DeployCAR 11
 - DeployCAR command 119
 - DeployEAR 46
 - DeployEAR command 120
 - DeployEAR12 command (deprecated) 123
 - DeployEJB command 125
 - DeployEJB11 command (deprecated) 127
 - DeployRAR command 129
 - DeployWAR 42, 44
 - DeployWAR command 130
 - DTDs 101
 - execute mode 99
 - ExportSource command (deprecated) 132
 - f option 101
 - GetConsole command 134
 - GetDefaultURL command 135
 - ImportClass command (deprecated) 136
 - ImportMedia command (deprecated) 138
 - ImportPage command (deprecated) 140
 - ImportSource command (deprecated) 141
 - input files 101
 - ListCP command 143
 - logging messages 100
 - ModifyCP command 144
 - ModifyTableList command (deprecated) 145
 - Prefs command 147
 - PrintLog command 149
 - Publish command (deprecated) 150
 - PublishFromFile command (deprecated) 151
 - PublishToFile command (deprecated) 153
 - QueryCP command 155
 - RebuildJAR command (deprecated) 156
 - RemoveCP command 157
 - RemoveDatabase command (deprecated) 158
 - running 98
 - running in batch mode 99
 - sample input files 101
 - ServerState command 159
 - SetDefaultURL command 161
 - SetSecurity command 162
 - SetUserGroupInfo command 163
 - SourceControl command (deprecated) 172
 - Undeploy command 176
 - ValidateEAR command 177
 - ValidateEJB command 178
 - ValidateEJB11 command 179
- SilverJ2EEClient
 - about 181
 - accessing secured EJBs 202
 - application arguments 200
 - communication protocols 182
 - console version 192
 - development environment notes 194
 - features 181
 - installing 183
 - JRunner class 189
 - SJC files 186
 - specifying a splash screen 194
 - starting 186
 - startup options 195
 - when to use 181

- SilverJRunner
 - about 182
 - accessing secured EJBs 202
 - application arguments 200
 - console window 192
 - installing 183
 - JRunner class 189
 - SJR files 186
 - specifying a splash screen 194
 - starting 186
 - startup options 195
 - when to use 181
- SilverStream objects, exporting 132
- SJC files
 - starting SilverJ2EEClient with 186
- SJR files, starting SilverJRunner with 186
- sounds
 - importing 138
- SourceControl command, SilverCmd (deprecated) 172
- splash screen
 - specifying for SilverJ2EEClient 194
 - specifying for SilverJRunner 194
- sqlHandler element values 16
- SSL
 - EJB attributes 33
- startup options
 - + startup options, SilverJ2EEClient 195
 - + startup options, SilverJRunner 195
 - startup options, SilverJ2EEClient 195
 - startup options, SilverJRunner 195
 - SilverJ2EEClient 195
 - SilverJRunner 195
- T**
- transactions
 - EJBs 214
- U**
- Undeploy command, SilverCmd 176
- URLs
 - setting default 161
- userlib directory, for application classpath JARs 47
- users
 - adding to groups 165
 - creating 167
 - deleting 168
 - deleting from groups 169
 - managing 163
 - setting properties for 171
- V**
- ValidateEAR command, SilverCmd 177
- ValidateEJB command, SilverCmd 178
- ValidateEJB11 command, SilverCmd 179
- W**
- WAR deployment plan DTD 78
- WARs
 - deploying 40
 - deployment plan 40, 43
- web applications
 - session-level failover 223
- Web container 203
- X**
- XML
 - and SilverCmd input files 101
 - using in SilverStream 225
- XML JARs
 - overriding on the server 47
- XML4J, IBM XML for Java parser 225