edocs

Deploying and Customizing J2EE Applications

ea**Direct**

Table of Contents

Preface

About Customer Self-Service and eaSuite™

edocs has developed the industry's most comprehensive software and services for deploying Customer Self-Service solutions. **eaSuite™** combines electronic presentment and payment (EPP), order management, knowledge management, personalization and application integration technologies to create an integrated, natural starting point for all customer service issues. eaSuite's unique architecture leverages and preserves existing infrastructure and data, and offers unparalleled scalability for the most demanding applications. With deployments across the healthcare, financial services, energy, retail, and communications industries, and the public sector, eaSuite powers some of the world's largest and most demanding customer self-service applications. eaSuite is a standards-based, feature rich, and highly scalable platform, that delivers the lowest total cost of ownership of any self-service solution available.
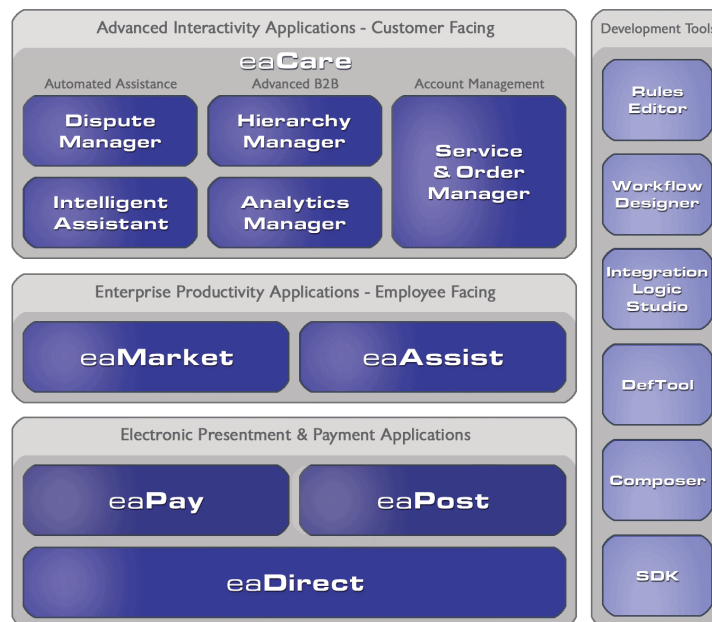
eaSuite is comprised of four product families:

- Electronic Presentment and Payment (EPP) Applications

- Advanced Interactivity Applications

- Enterprise Productivity Applications

- Development Tools

Electronic Presentment and Payment (EPP) Applications are the foundation of edocs'
Customer Self-Service solution. They provide the core integration infrastructure between
organizations' backend transactional systems and end users, as well as rich e-billing, e-
invoicing and e-statement functionality. Designed to meet the rigorous demands of the
most technologically advanced organizations, these applications power Customer Self-
Service by managing transactional data and by enabling payments and account
distribution.

> **eaDirect**™ is the core infrastructure of enterprise Customer Self-Service solutions
> for organizations large and small with special emphasis on meeting the needs of
> organizations with large numbers of customers, high data volumes and extensive
> integration with systems and business processes across the enterprise. Organizations
> use eaDirect with its data access layer, composition engine, and security, enrollment
> and logging framework to power complex Customer Self-Service applications.

> **eaPay**™ is the electronic payment solution that decreases payment processing costs,
> accelerates receivables and improves operational efficiency. eaPay is a complete
> payment scheduling and warehousing system with real-time and batch connections to
> payment gateways for Automated Clearing House (ACH) and credit card payments,
> and payments via various payment processing service providers.

> **eaPost**® is the account content distribution system that handles all the complexities
> of enrollment, authentication and secure distribution of summary account information
> to any endpoint, while also bringing customers back the organization's Website to
> manage and control their self-service experience.

Advanced Interactivity Applications are a comprehensive set of advanced customer-
facing self-service capabilities that enable the full range of business and consumer
customer service activities. These sophisticated modules have the flexibility to
completely customize the Customer Self-Service solution to meet vertical industry and
specific company requirements.

> **eaCare**™ consists of a rich set of sophisticated self-service modules – Dispute
> Manager, Intelligent Assistant, Hierarchy Manager, Analytics Manager, and Service
> and Order Manager - for automated assistance, advanced business-to-business
> applications and account management. These capabilities come together to create a
> web self-service dashboard for customers to access all service offerings from a
> single, easy-to-use interface. eaCare's modularity accelerates time to market with
> components that can be deployed incrementally in a phased approach.

Enterprise Productivity Applications are employee-facing solutions that empower customer service representatives, sales agents, account managers, marketing managers, broker-dealers and channel partners within an organization and external partner organizations to facilitate self-service and to support assisted service. Employees leverage edocs' Customer Self-Service solution to deliver customer service, access information, create and deploy marketing and customer service content, and perform activities for the benefit of customers.

eaAssist™ reduces interaction costs and increases customer satisfaction by enabling enterprise agents – customer service representatives (CSRs), sales agents, broker-dealers and others – to efficiently access critical account data and service-related information to effectively service customers. Through its browser interface designed especially for the enterprise agent, eaAssist enables agents to take advantage of customer-facing online capabilities to provide better service by more efficiently resolving customer account inquiries at the point of customer contact.

eaMarket™ is the personalization, campaign and content management solution that enables organizations to increase revenue and improve customer satisfaction by weaving personalized marketing and customer service messages throughout the Customer Self-Service experience. The transactional account data that provides the foundation for a Customer Self-Service solution – such as transaction activity, service or usage charges, current task and prior service history – bring valuable insight into customers and can help optimize personalized marketing and customer service campaigns. eaMarket leverages that data to present relevant marketing and customer service messages to customers.

edocs' **Development Tools** are visual development environments for designing and configuring edocs' Customer Self-Service solutions. The Configuration Tools encompass data and rules management, workflow authoring, systems integration, and a software development kit that makes it easy to create customer and employee-facing self-service applications leveraging eaSuite.

About This Guide

This Guide describes the tasks required to deploy and customize the J2EE applications provided by eaDirect. It also provides a step by step description of how to deploy the eaSample application provided by eaDirect, and how to validate that it is set up correctly by running a job through your installed eaDirect environment.

Related Documentation

This guide is part of the eaDirect documentation set. For more information about implementing your eaDirect application, see one of the following guides:

Print Document	Description
eaDirect Installation and Configuration Guides	How to install eaDirect and configure it in a distributed environment.

Print Document	Description
eaDirect Data Definition and Presentation Design Guides	How to design and create an eaDirect application using the DefTool and Composer tools.
eaDirect Data Presentation Production Guide	How to set up and run a live eaDirect application in a J2EE environment.

Obtaining edocs Software and Documentation

You can download edocs software and documentation directly from Customer Central at https://support.edocs.com. After you log in, click on the Downloads button on the left. When the next page appears, you will see a table displaying all of the available downloads. To search for specific items, select the Version and/or Category and click the Search Downloads button. If you download software, an email from edocs Technical Support will automatically be sent to you (the registered owner) with your license key information.

If you received an edocs product installation CD, load it on your system and navigate from its root directory to the folder where the software installer resides for your operating system. You can run the installer from that location, or you can copy it to your file system and run it from there. The product documentation included with your CD is in the Documentation folder located in the root directory. The license key information for the products on the CD is included with the package materials shipped with the CD.

If You Need Help

Technical support is available to customers who have valid maintenance and support contracts with edocs. Technical support engineers can help you install, configure, and maintain your edocs application.

edocs provides global Technical Support services from the following Support Centers:

US Support Center
Natick, MA
Mon-Fri 8:30am – 8:00pm US EST
Telephone: 508-652-8400

Europe Support Center
London, United Kingdom
Mon-Fri 9:00am – 5:00 GMT
Telephone: +44 20 8956 2673

Asia Pac Rim Support Center
Melbourne, Australia
Mon-Fri 9:00am – 5:00pm AU
Telephone: +61 3 9909 7301

Customer Central
https://support.edocs.com

Email Support
mailto:support@edocs.com

When you report a problem, please be prepared to provide us the following information:

- What is your name and role in your organization?

- What is your company's name?

- What is your phone number and best times to call you?

- What is your e-mail address?

- In which edocs product did a problem occur?

- What is your Operating System version?

- What were you doing when the problem occurred?

- How did the system respond to the error?

- If the system generated a screen message, please send us that screen message.

- If the system wrote information to a log file, please send us that log file.

If the system crashed or hung, please tell us.

1

About edocs Sample Applications

Before You Start

Before you can begin to develop an application for eaDirect, you will need to install and deploy eaDirect to your application server using the instructions described in the eaDirect Installation and Configuration Guides and the Release Notes. Please verify that eaDirect is configured and running correctly before proceeding.

For more specific deployment information about your application server platform, please consult your application server product documentation.

About ear-eadirect.ear

The **ear-eadirect.ear** J2EE application contains the core functionality of eaDirect, including the Command Center and the eaDirect engine. It also contains placeholder files for other components of the eaSuite.

> **Caution**   You should NOT modify the ear-eadirect.ear file, and you should always keep a backup in a safe place.

Migration Issues

eaDirect 3.4 and later versions support specifying a data source for each DDN, or eaDirect application. The default datasource EJB is **edx/ejb/EdocsDataSource**. When migrating the database from 3.2 or earlier versions to 3.4 or later, you must redeploy **ear-eadirect.ear** (to populate the data source mapping for eaDirect) to view statements for eaDirect applications created before migration.

For more information about DDN to Datasource mapping, please see the *eaDirect Data Presentation Production Guide.*

About the Samples Directory

Your edocs installation includes the folder **EDCSbd/samples**, which contains J2EE, Web, and eaDirect applications for use with the eaSuite. This topic defines each sample and directs you to further information on its use.

In eaDirect, the term *application* can refer to three different types of file components: J2EE applications, web applications, and eaDirect applications. Each is defined in the following sections.

J2EE Applications (EAR files) and Web Applications (WAR files)

A *J2EE application* bundles enterprise Java components and services for deployment into a J2EE Server container. The J2EE server provides underlying services to handle transaction and state management, multithreading, resource pooling, and other complex low-level details.

For an introduction to J2EE application components, see http://java.sun.com/j2ee/sdk_1.2.1/techdocs/guides/ejb/html/Overview5.html#10106.

J2EE applications are packaged and deployed in an *Enterprise Archive* (EAR) file. An EAR file contains an XML file for its deployment descriptor and one or more EJB .jar and .war files.

- Each EJB JAR file contains a deployment descriptor, the enterprise bean files, and related files.

- Each application client JAR file contains a deployment descriptor, the class files for the application client, and related files.

- Each WAR file contains a deployment descriptor, the Web component files, and related resources.

A *web application,* sometimes called a *web app,* is a customized software system of web services and code components with a business state that is changed by user input, usually through a web interface. Each unique combination of business features and functionality requires a unique web application.

Web applications are packaged and deployed in a *Web Application Archive* (WAR) file within the EAR file. The WAR file contains JSPs, servlet classes, HTML templates and image files, EJBs specific to the WAR file, and XML deployment descriptors for each component.

For more information, see Sample J2EE and Web Applications for eaSuite.

eaDirect Applications (DDF, ALF, HTML, and XML files)

An *eaDirect application* is a customized set of eaDirect data files (DDFs, ALFs, HTML templates, and XML/XSLT files) and Command Center jobs, created to extract and present statements online from a particular dataset. Each unique combination of data, business rules, and presentment requires a unique (eaDirect) application.

Your eaDirect installation includes an example eaDirect application dataset called *National Wireless*, a fictional telecommunications company. Your team will use the **NatlWireless** sample files and data with one or more J2EE web applications to test your installation of eaDirect; practice the eaDirect toolset including the DefTool, Composer, and Command Center; and customize the sample web applications for your own environment. For more information about National Wireless, see Components of National Wireless dataset and the eaDirect *Data Definition* and *Data Presentation Production Guides*.

Sample J2EE and Web Applications for eaSuite

When you install eaDirect and the SDK, you will find enterprise application archives (EAR files) for *eaDirect* itself and for *eaSample*, *eaTraining*, and *UMFsample* (SDK only) in the **EDCSbd/samples/j2eeapps** directory of your eaDirect installation. Each is defined in the following sections.

Platform	EAR Installed to:	Deploy EAR to:
WebLogic	EDCSbd/samples/J2EEApps/weblogic	**WebLogic-home/config /mydomain/applications**
WebSphere	EDCSbd/samples/J2EEApps/websphere	**WebSphere_home /installedApps/**

eaSample

The **ear-easample.ear** J2EE application demonstrates the core features of eaDirect, including:

- Non-hierarchical enrollment model using edocs Common Directory Access (CDA)

- Content access to statement summary and detail data

- Line item disputes

edocs recommends that you use eaSample as the skeleton for developing your own custom web applications. Each SDK module uses eaSample to demonstrate how to customize eaDirect features.

For a list of the components in eaSample, please see Appendix A: Component Lists for Sample Web Applications.

Tip   If you are also using eaPay, edocs recommends customizing one of the eaPay web application EAR files instead of eaSample.

For more information about enrollment models, please see the *User Management Frameworks* SDK guide.

For more information about content access to statement summary and detail data, please see the:*Content Access* SDK guide.

Caution   Before modifying any sample EAR file, you should always keep a backup in a safe place.

---

eaTraining

The **ear-eatraining.ear** J2EE application demonstrates more advanced features of eaDirect, including hierarchical user enrollment using edocs Common Directory Access (CDA) and the use of sub-documents.

For more information about eaTraining, please see Appendix A: Components of EAR and WAR Samples.

For more information about hierarchical enrollment and CDA, please see the *User Management Frameworks* SDK guide.

umfsample

The **ear-umfsample.ear** J2EE application implements a non-directory enrollment model that customizes the edocs user management framework as an interface to enrollment information already stored in a separate repository.

umfsample does *not* ship with eaDirect, but is installed separately from the SDK CD. For details of working with **umfsample**, see *User Management Frameworks* SDK guide.

Sample Datasets for eaSuite

The samples directory also contains sample data and design files (DDF , ALF, and HTML templates) for use with sample web applications.

Datasets are optimized to implement features of sample web applications. The following table shows which datasets to use with each sample web application.

Dataset/Web App	eaSample	eaTraining	umfsample
NatlWireless	X	X	X
NW_LDDetail	X	X	

About Jobs, Views, and Version Sets

Most features of eaDirect require that you use the edocs Command Center to publish a specific type of *view*, sometimes with particular parameters and settings. A *view* is a set of design files that result in a particular presentation of statement data. The view files enable a user to dynamically display formatted statements live on the Web, receive email notifying them that an online statement is available, or to view account data in a static format online.

A feature may also depend on a particular sequence of *jobs*; for example, you must have loaded detail data into the database in order to dispute and annotate line items.

When creating and configuring your eaDirect application, you *publish* the files that make up each dynamic Web view in your application. Publishing adds a timestamp to the set of view files. A *version set* is a dated set of design files for a dynamic Web view. Publishing a view is also called creating a version set.

Tip   If you cannot see a particular feature in one of the samples, check to make sure that you have published the necessary views and run the required jobs in the correct sequence.

For more information about Jobs, Views, and Version Sets, see the *eaDirect Data PresentationProduction Guide.*

2

Deploying and Using eaSample

About eaSample

eaSample is a sample J2EE application that eaDirect provides as part of its software distribution. You can use it as a framework for developing a custom EJB application, as it contains all the Java Server Pages (JSPs), HTML, image files, scripts, and templates you need to get started. eaSample deploys as *ear-easample.ear*.

You can use eaSample to view the sample NatlWireless and NW_LDDetail applications provided with eaDirect. You can use the data and design files in these sample applications to become familiar with eaDirect by creating sample billing applications and jobs, publishing data and design files in the form of *version sets*, and scheduling the jobs to run in the Command Center, the administrative 'hub' for the eaDirect production environment.

The following steps describe how to deploy and use eaSample to view the sample eaDirect application called **NatlWireless**. NatlWireless is a set of example design and data files that demonstrate the features of an eaDirect presentment application.

You must set up NatlWireless in the eaDirect Command Center (production environment), then enroll and log in to eaSample to view sample bills.

Deploying the eaSample J2EE Web Application

When you install eaDirect, you will find the Enterprise Application aRchive (EAR file) for eaSample in the **/EDCSbd/samples/j2eeapps** directory of your eaDirect installation. You will deploy this application using your application server's administrative console.
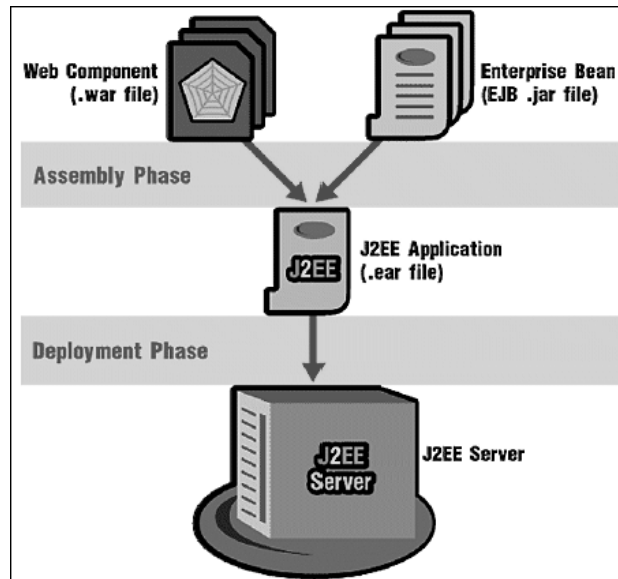
In general, deploying a web application involves three distinct phases:

1. Component creation, typically done by application developers

2. Application assembly, typically done by application developers (although they may not have participated in the 'component creation' phase)

3. Application deployment, typically done by both application developers and system administrators

During development and testing, it is common for web developers to deploy their own applications. However, when the application has been assembled and is ready for production, a system administrator most likely will deploy it.

Deployment Using WebLogic

This section provides and example of how to deploy eaSample using the WebLogic application server.

> **Caution** The specific steps for your application server may differ from the ones described below. You should consult your System Administrator and application server documentation for complete details of how to deploy a J2EE application based on your system's configuration.

The steps below summarize how to deploy using WebLogic Server 7.0. When you deploy applications to WebLogic Server 7.0, you use the Administration Console to deploy using the new two-phase deployment protocol. WebLogic Server 8.1 may vary slightly. Please consult your BEA WebLogic documentation for specific instructions on this task.

Note that WebLogic Server will not deploy an application that has errors in its deployment descriptor. Previous versions of WebLogic Server would deploy an application that had errors in its deployment descriptor.

To deploy a web application to your application server:

1. Start WebLogic Server, if it isn't already running.

2. Open a web browser and enter a URL with the syntax:

    `http://host:port/console`

    For example:

    `http://audi:7001/console`

    The Enter Network Password dialog appears. (Note that the dialog might take a few seconds to display.)

3. Enter your WebLogic Server user name and password. The default user name is 'system' and the password is the one you specified during installation of WebLogic Server.

4. Click **OK**. The WebLogic Server Console main menu appears.

5. From your domain in the left pane, select **Deployment** and then **Applications**.

6. Click the **Configure New Application** link.

7. Follow the instructions of the wizard to locate and upload the eaDirect J2EE application *ear-easample.ear*. By default, this file is located in *EDX_HOME/J2EEApps/weblogic*.

8. After uploading the EAR file, follow the instructions in the wizard to configure and deploy it to your application server.

Deployment Using WebSphere

This section describes how to deploy eaSample using the WebSphere application server.

Caution	The specific steps for your application server may differ from the ones described below. You should consult your System Administrator and application server documentation for complete details of how to deploy a J2EE application based on your system's configuration.

The first step in setting up the application server is to generate the deployment code for the eaSample application that you will deploy on WebSphere. This is done though the Application Assembly Tool.

The following instructions describe how to invoke the Application Assembly Tool from the WebSphere 4 Administrative Console. You can al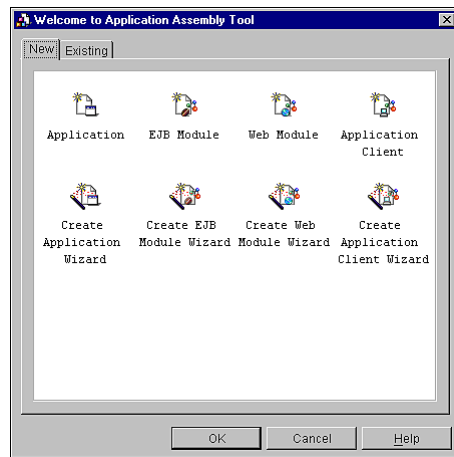so start it from a command line window; WebSphere 5 requires you to start the Application Assembly Tool from the command line (see your WebSphere documentation).

1. From the WebSphere Administrative Console, open the Tools menu and select **`Application Assembly Tool`**.
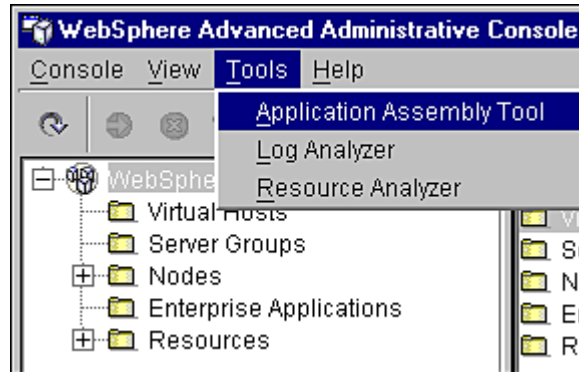
2. The Application Assembly Tool appears.

Tip On UNIX, the Application Assembly Tool appears in an X-window, so you will have to have X-window software installed and you might have to set your display (in your *.profile* file) to the local machine if you are trying to invoke the Application Assembly tool remotely. For example:

set DISPLAY=montero:0.0
export DISPLAY

To generate deployment code for eaDirect J2EE applications:

1. When the Application Assembly Tool appears, click the **`Existing`** tab.

2. Click **`Browse`** at the bottom of the dialog, and navigate to the directory that contains the eaDirect J2EE applications. For example:
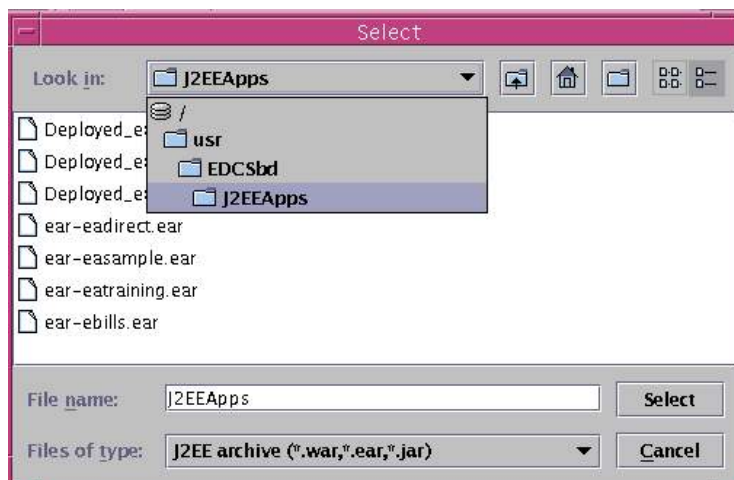
3. Select **ear-easample.ear** from the list of J2EE applications and click **Select**.

4. Click **OK**. The eaDirect application opens in the Application Assembly Tool.

5. In the left pane, right click on **easample**, and select **Generate code for deployment** from the menu.

    The **Generate Code for deployment** dialog appears.

6. In the **Dependent classpath** field, enter the java classpath for the edocs system, client, and common JAR files separated by colons (for Windows the separator is a semi-colon), for example:

    ```
    /opt/EDCSbd/lib/edx_system.jar:/opt/EDCSbd/lib/edx_client.jar:/opt/ED
    CSbd/lib/edx_common.jar
    ```

> **Tip**   Be sure to include the leading slashes so that these classpaths are taken
> from the root.

7. From the Database type drop-down menu, select the database you are using, such as Oracle or DB2.

8. Click **Generate Now**. During generation of the deployment code, status information is shown in the window at the bottom of the dialog.

9. Wait for the progress bar at the bottom of the Application Assembly Tool to complete. Some applications might take several minutes to deploy, depending on the speed of your machine.

10. Click **Close** to exit from the Generate code for deployment dialog.

11. Click **Exit** in the File menu to close the Application Assembly Tool.

To deploy a web application with WebSphere 5:

1. From the WebSphere Administrative Console, expand the Applications branch and click on **Install New Application**.

2. Select "Server Path" and then type in the absolute path to where your application is installed, for example,

   `/opt/EDCSbd/J2EEApps/websphere5/Deployed_ear-easample.ear`

3. Keep clicking **Next** until you reach the last page, and then click **Finish**. Use the default settings for each step.

4. Save to the Master Configuration.

To deploy a web application with WebSphere 4:

1. From the WebSphere Administrative Console, expand the WebSphere Administrative Domain view.

2. Right-click the **Enterprise Applications** folder, and then select **Install Enterprise Applications** on the menu.
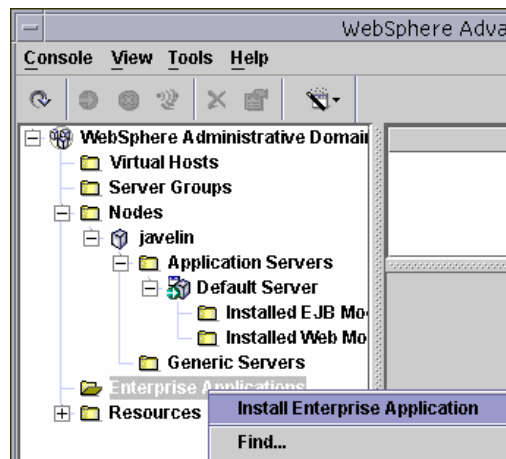

   The Install Enterprise Application Wizard dialog appears.

3. Click the **Install Application** radio button, and confirm that the correct node has been chosen in the Browse for file on node field.

4. Click **Browse**. The Open dialog appears.

5. Navigate to *$EDX_HOME/J2EEApps/websphere*, and select **Deployed_ear-easample.ear**. The name of the file appears in the File name field.

6. Click **Open**. The Install Enterprise Application Wizard dialog appears with the name of the deployed EAR file in the InstallApplication (*.ear) Path field.

7. Click **Next** about nine times until you come to the Selecting Application Servers screen. Highlight all the modules for selection by clicking the first and last module in the list, while holding down the **Shift** key.

8. Click **Select Server**. The Select a Server or Server Group dialog appears.

9. Select the Default Server. In the sample screen below, the default server is named javelin.
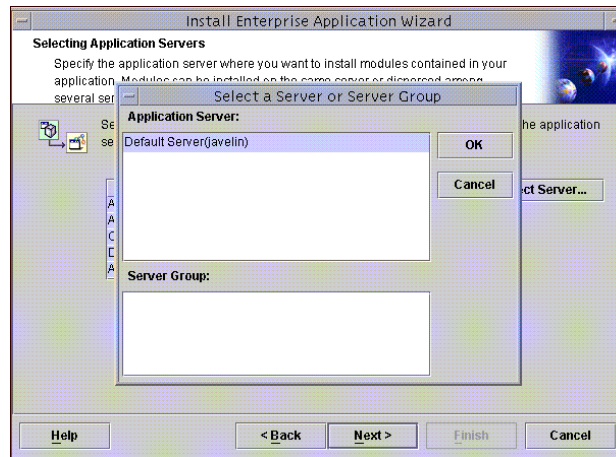
10. Click **OK** to close the Select a Server of Server Group dialog. The Install Enterprise Application Wizard dialog lists the modules and the server on which they will be installed.

11. Click **Next** and then **Finish**.

12. When asked if you want to regenerate code, click **No**. The installation of eaDirect takes place on the application server.

13. Click **OK** to close the Information dialog reporting that the installation was successful.

14. Stop the server on which you installed the eaDirect J2EE application. Select (expand) **Nodes** and <**your_node**> and **Application Servers**, and then right-click the server name in the left pane. Select **Stop** on the menu.

15. An Information dialog notifies you that the application server has stopped successfully.

16. Click **OK** to close the Information dialog.

17. Restart the application server by right-clicking on the its name, and selecting **Start** on the menu.
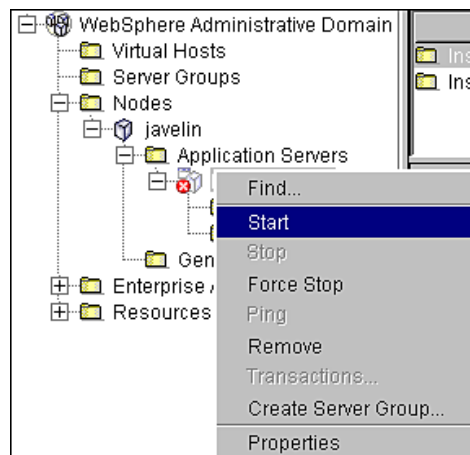
18. An Information dialog notifies you that the application server has started successfully.

19. Click **OK** to close the Information dialog.

20. Start the eaSample J2EE application by expanding the Enterprise Applications folder, right clicking on the application name (for example, *easample*), and then selecting **Start** from the menu.

21. An Information dialog notifies you that the application has started successfully. Click **OK** to close the Information dialog.

Setting Up NatlWireless

1. Create a new **application** for NatlWireless in the eaDirect Command Center.

2. Create a new **Indexer job**, publishing the application's indexing DDF for the job to use, configure the four tasks that run sequentially as part of the Indexer job, and run the job. Then publish the NatlWireless application (dynamic HTML view) files designed to display the statement summary.

3. Create and configure a **Detail Extractor job**, publishing the DDF, database table XML file, and statement XSLT stylesheet view files designed for the Detail Extractor job, and run the job. Then publish the three dynamic XML Query files (views) designed to display the extracted NatlWireless data and demonstrate the disputes and annotations features.

To create a new eaDirect application for NatlWireless:

1. Start your application server and the Scheduler, if not already running.

2. Open a web browser and enter the URL to the eaDirect Command Center, for example:

   ```
   http://<HOST>:<PORT>/eaDirect
   ```

3. Create a new application for NatlWireless. Click **Create New Application** at the Main Console. The Create New Application screen appears.

4. Enter **NatlWireless** as the application name.

5. Enter the JNDI name of the datasource EJB as follows:

   edx/eaSample/ejb/EdocsDataSource

6. Click **Create Application and Continue**. eaDirect displays the Create New Job screen.

To create and configure an Indexer job:

1. The Create New Job screen appears after you create a new application:

2. Enter **Indexer** for the Indexer job name, select the **Indexer** job type from the drop-down menu. Click **Launch Publisher** to publish the indexing DDF for NatlWireless. Click **Create**. The Publisher displays the Select a Version Set Type screen:

3. Under Batch Jobs, next to Indexer, click **0** (Number of Auxiliary files). The Publisher displays the Create a Version Set For Indexer screen:

4. Select **NatlWireless** from the list of application names, and browse to the *$EDX_HOME/samples/NatlWireless* directory and select **Indexerjob\NatlWirelessIndexer.DDF** file for the Indexer job.

5. Click **Submit**. The Publisher displays the Submission screen with details about the DDF file. Close the Publisher window.

6. At the Create New Job screen in Command Center, click `Configure Job and Continue`. eaDirect displays the job configuration screen. For each task, specify the configuration parameters listed below:

Task 1: Scanner Task Configuration	
Input File Path	Use the default (`EDX_HOME/Input/NatlWireless`).
Input File Name	Specify `NatlWireless.txt`.
Output File Path	Use the default (`EDX_HOME/Data/NatlWireless`).
Task 2: Indexer Task Configuration	
DDF Path	(Not editable.)
Task 3: IXLoader Task Configuration	
Load Method	Use the default (`Direct`)
Task 4: AutoIndexVolAccept Task Configuration	
Action on Index Volume	Use the default (`Auto Accept`).

7. When finished entering the configuration parameters, click `Submit Changes and Schedule`. eaDirect asks "OK to submit this configuration?" Click `OK`. eaDirect submits the job configuration parameters and displays the Schedule screen.

8. In the left pane, click `Main Console`. On the Main Console's left pane, click `Publisher`, and then click `Create`. The Publisher displays the Select a Version Set Type screen:

9. Next to HTML under Dynamic Web Views, click **0**. The Publisher displays the Create a Version Set for HTML screen:

10. Select **NatlWireless** from the drop-down list of application names. Enter **HtmlDetail** for the view name. Browse to *$EDX_HOME/samples/NatlWireless* and select the **NatlWireless.DDF, NatlWireless.ALF,** and **NatlWireless.HTM** design files. Then click **Submit**. The Publisher displays the Submission screen showing the files you published:
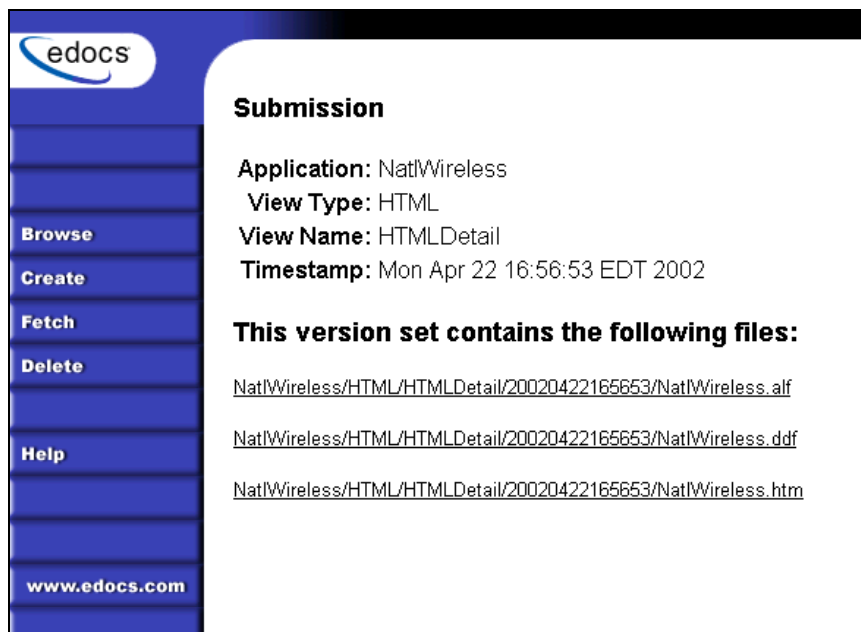
11. Close the Publisher.

12. Move the NatlWireless data file (*NatlWireless.txt)*, which is located in
    $EDX_HOME/samples/NatlWireless/datafile, to *$EDX_HOME/Input/NatlWireless*.
    This is the same data file that you specified when you configured the job.

13. On the Main Console, click the **Run Now** button next to the NatlWireless Indexer
    job. Monitor the job's progress by clicking **Refresh** on the Main Console window.
    The Indexer job completes successfully when the job status on the Main Console
    changes to "Done."

To create and configure a Detail Extractor job:

1. On the Main Console, click the application name, **NatlWireless**, listed under
   Applications in the table.

2. Click **Add New Job**. eaDirect displays the Create New Job screen.

3. Enter a job name (the job name can be whatever you want it to be), and then select
   job type **Detail Extractor**.

4. Click **Launch Publisher**. eaDirect displays the Publisher screen. Click **Create**.
   The Publisher displays the Select a Version Set Type screen.

5. Under Batch Jobs, next to Detail Extractor, click **0** (Number of Auxiliary files). The
   Publisher displays the Create a Version Set For Detail Extractor screen.

6. Select the **NatlWireless** application from the drop-down list. Enter the view name
   dtlextr (this name is hard coded in several JSPs for detail, disputes, and
   annotations).

7. Browse to select **NatlWireless.DDF**. (The default location for this file is *$EDX_HOME/samples/NatlWireless/NatlWireless.DDF*.)

8. Browse to select **summary_info.XML**, the database table XML view file created for this job. (The default location for this file is *$EDX_HOME/samples/NatlWireless/DetailExtractor*.)

9. Browse to select **summary_info.XSL**, the statement XSLT stylesheet. The default location for this file is *$EDX_HOME/samples/NatlWireless/DetailExtractor*.

10. Click **Submit** and close the Publisher.

11. On the Create New Job screen in the Command Center, click **Configure Job and Continue**. eaDirect displays the Detail Extractor job configuration screen.

12. Specify the configuration parameters (below) for each of the three tasks in the Detail Extractor job:

Task 1: IVNScanner Task Configuration	
Field	**What to enter/select**
Index Volume Status	Choose the default, **Accepted**.
Scan Starting From (Number of Days)	Use the default (**7**).
Task 2: StatementsToIR Task Configuration	
Field	**What to enter/select**
View Name	**dtlextr**
Enroll Model	Leave blank.
Output File Path	Use the default (data output directory, specified in the Scanner task for the Indexer job).

Task 3: DXLoader Configuration	
Field	**What to enter/select**
Load Method	Use the default (**Direct**).

13. Click **Submit Changes and Schedule**. eaDirect asks "OK to submit this configuration?" Click **OK**. eaDirect submits the job configuration parameters and displays the Schedule screen.

14. On the Schedule screen, click **Run Now**.

15. On the left pane, click **Main Console**.

16. Publish the XMLQuery dynamic web views that use the data extracted by the Detail Extractor; click **Publisher**.

17. Click **Create**. The Publisher displays the Select a Version Set Type screen. Under Dynamic Web Views, click the **0** next to the XML Query job type. The Publisher displays the Create a Version Set for XML Query screen.

18. Select the **NatlWireless** application. Enter **DetailQuery** as the view name, and browse *%EDX_HOME%\samples\NatlWireless\XMLQuery* to select the **detail_sql.xml** XML query file. (The DetailQuery view name is hard coded in your JSP HTML pages as the specific name the Web browser looks for in the code.) Click **Submit**. The Publisher displays the Submission screen

19. Click **Create** and repeat the previous two steps twice to publish two additional XML Query views (both view names are hard coded in your JSP HTML pages as the specific names the Web browser looks for in the code):

View Name	File
DisputeQuery	*dispute_sql.xml*
AnnotationQuery	*annot_sql.xml*

20. Close the Publisher. You can proceed to use eaSample to display the data.

21. On the Main Console, click the **Run Now** button next to the NatlWireless Detail Extractor job. Monitor the job's progress by clicking **Refresh** on the Main Console window. The Detail Extractor job completes successfully when the job status on the Main Console changes to "Done."

Viewing NatlWireless Statements in eaSample

When the Indexer job completes successfully (status changes to "Done"), you are ready to view your online statements in eaSample.

To use eaSample to view NatlWireless statements:

1. Open a Web browser and access eaSample, substituting your own server name (host) and port number:

   ```
   http://<HOST>:<PORT>/eaSample/User?app=UserMain&jsp=
   /user/jsp/HistoryList.jsp&ddn=NatlWireless
   ```

   The eaSample User Login page appears.

2. Click the `Enroll Now` link. The sample enrollment page appears.

3. You can enter any user name and any password. However, you must enter a valid email address and a valid NatlWireless customer account number, such as one of the following: 0331734, 4191463, or 8611250. (Use **Reset** to clear the text fields, if necessary.) Click **Submit** to save the subscription information. eaDirect displays a message to let you know you have subscribed successfully. Click **OK** to display the User Login page.

4. Enter the username (Subscriber ID) and password (the same combination you entered during enrollment).

5. Click **Submit**. The sample statement summary page for the account appears. (**Note:** You must have eaPay, the eaDirect payment module, installed to view the payment screens and functionality.)
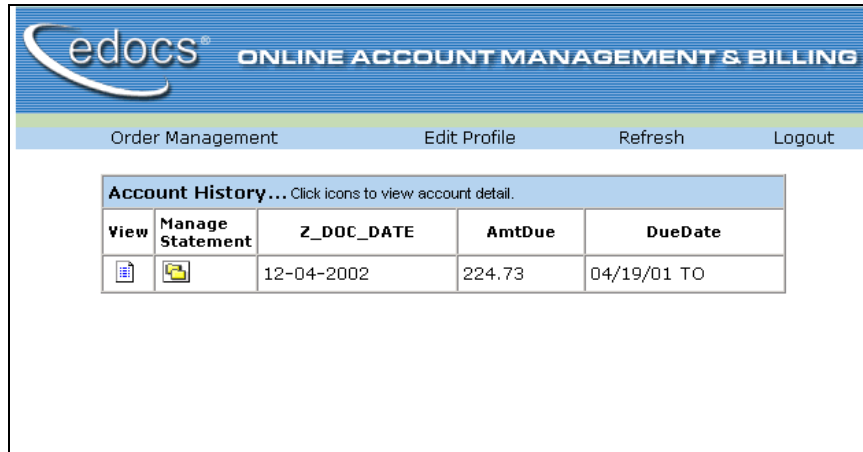
6. To view the statement summary, click the View icon.
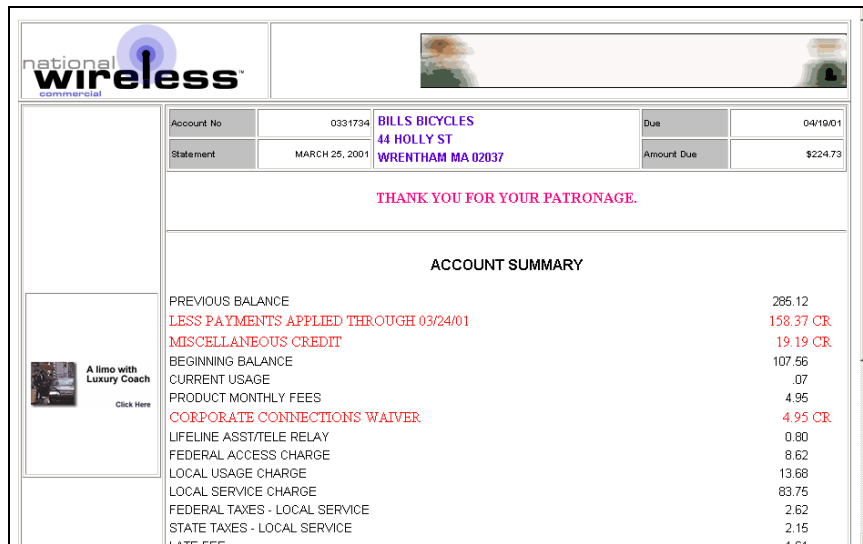
7. To view the Manage Statement page, click from the History page.

8. Click ✎ next to an item to display the Add Note page where you can add comments (annotations) regarding that item.

9. Click ● to display the Dispute Your Statement page where you can dispute the item:

Dispute your statement:	
Disputed Item:	Summary Info Amount
Current Amount:	4.95
Adjusted Amount:	
Comments:	

Submit   Reset

3

Renaming eaSample to a New J2EE Application

About Application Contexts

To rename your web application, you will need to change its *context name* and modify the *deployment descriptors* for each main component. This is the first step toward customization of any sample web application.

The following chapter, Creating Your Own Custom Web Application for eaDirect, gives an overview of some common customization choices and directs you to the appropriate SDK module for working with those components. Most customization involves the WAR file only, though if you are customizing enrollment, you may be working with an enrollment EJB, for example **ejb-enrollment-cda.jar**. Other EAR file EJBs should not be customized beyond modifying their deployment descriptors as shown below.

In this chapter, we will simply change the *context root* of eaSample and update the *JNDI name* of each deployment descriptor to point to the new context root.

About the Context Root

A *context root* is a name that maps to the document root of a Web client. If your client's context root is **easample**, then the request URL

```
http://host:8000/easample/index.html
```

will retrieve the file **index.html** from the document root **easample**.

You will need to change the context root **easample** to the name of your new web application.

For an introduction to context root and web client configuration, see The J2EE Tutorial at http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/WCC4.html.

About JNDI Names

A *JNDI name* is a people-friendly name for an object in the Java Naming and Directory Interface API (JNDI). These names are bound to their objects by the J2EE server naming and directory service.

A *resource reference* is an element in a deployment descriptor that identifies the component's coded name for the resource. More specifically, the coded name references a connection factory for the resource. A *connection factory* is an object that produces connection objects that enable a J2EE component to access a resource.

The JNDI name of a resource and the name of the resource reference are *not the same*. You will need to change both the JNDI name and the resource reference for eaSample to the name of your new web application.

For an introduction to JNDI names, see The J2EE Tutorial at http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Resources2.html.

About XML Deployment Descriptors

Each J2EE application, Web component, enterprise bean, and J2EE application client has a *deployment descriptor*. A deployment descriptor is an XML document that describes a component's deployment settings. An enterprise bean deployment descriptor, for example, declares transaction attributes and security authorizations for the bean. Deployment descriptors are located in the META-INF directory of each component.

> **Tip** Deployment descriptors for WebLogic have the extension `.xml`.
> Descriptors for WebSphere have the extension `.xmi`.

For the XML DTDs for EAR, WAR, and EJB files in the J2EE specification, see http://java.sun.com/j2ee/dtds/.

Define Your New Context Name

Define a *context name* for your new web application. This name will be the web context for all URL references; a prefix to the JNDI name references in all deployment descriptors and part of the name of the WAR and EAR files. Be consistent: use the same context name everywhere.

For the eaSample EAR file, the context name is **eaSample**. This example will use **eaNewapp** as its context name, but you should use your own descriptive name.

Write your context name here: _____

Edit EAR File Deployment Descriptors

Extract the eaSample EAR File

Before you can modify eaSample, you must extract the contents of the enterprise archive to a working directory. This allows you to view and change individual components of the archive, in this example the context root and JNDI names only. Once you have made your changes, you must repackage the archive and redeploy it to your application server for the change to take effect.

To extract ear-easample.ear to a working directory:

1. Create a working directory on your machine, for example:

   /EDCSbd/ear-eanewapp

2. Copy the file **ear-easample.ear**

   From **/EDCSbd/j2eeapps**

   To **/EDCSbd/ear-eanewapp**

3. Extract the files from the EAR file, for example:

   ```
   jar xvf ear-easample.ear
   ```

Windows users can extract the contents of an EAR file with WinZip.

Once you have extracted the EAR file to your working directory, delete the **ear-easample.ear** file from that directory, or move it to a backup location. DO NOT delete the version in the j2eeapps directory!

Tip   The examples described in this guide presume you are deploying to a production environment, so you should clean up your working directory after each step before proceeding.

After the extraction, your working directory will contain the files listed in Components of ear-easample.ear and shown in the following diagram.

Architecture of eaSample EAR

For a complete listing and definition of components in **ear-easample.ear**, please see Appendix A: Component Lists for Sample Web Applications.

To rename your web application, you must change *only the deployment descriptors* for the EAR file, the WAR file, and each EJB in the EAR file. Deployment descriptors are located in the META-INF directory of EAR and JAR files, and the WEB-INF directory of WAR files. The following sections describe each task in detail.

Edit application.xml

The **application.xml** deployment descriptor defines runtime properties of the J2EE application EAR file. Deployment descriptors are located in the WEB-INF directory of the EAR file. The following diagram illustrates the XML elements defining runtime properties of eaSample.

Tip   XML illustrations in this SDK were created with Anova XML Spy. For information about XML Spy, see http://www.xmlspy.com/.

XML		
= version	1.0	
= encoding	UTF-8	

DOCTYPE application	
ID PUBLIC	"-//Sun Microsystems, Inc.//DTD J2EE Application 1.2//EN" "http://java.sun.com/j2ee/dtds/application_1_2.dtd"

application		
() display-name easample		
() description (C)Copyright 1999-2002 edocs(R), Inc. All Rights Reserved.		
module (14)		

		() web	() ejb
1	web		
		() web-uri war-easample.war	
		() context-root eaSample	
2			ejb-application.jar
3			ejb-enrollment-cda.jar
4			ejb-fileserver.jar
5			ejb-ccmerger.jar
6			ejb-xsltmerger.jar
7			ejb-querymerger.jar
8			ejb-annotation.jar
9			ejb-dispute.jar
10			ejb-session.jar
11			ejb-oc-cc.jar
12			ejb-oc-telco.jar
13			ejb-oc-health.jar
14			ejb-versioning.jar

From this example, we can see that the name of the J2EE application appears in three XML elements: the **display-name** of the application and both the **web-uri** and **context root** of its web application WAR file.

To rename eaSample in application.xml, search for all occurrences of **eaSample** or **easample** (note case-sensitivity) and replace with **Newapp** or **newapp**.

The next step is to rename individual deployment descriptors inside each WAR and EJB component of the EAR file. Once you have finished renaming these components, you will Repackage and Deploy the EAR File.

Edit EJB Deployment Descriptors

About eaSample EJBs

As we can see in **application.xml**, eaSample contains thirteen EJB JAR files. For a definition of each JAR file and which SDK features use it, see Appendix A: Components of EAR and WAR Samples. To rename your application, you will edit deployment descriptors for *each* of these JAR files.

Each EJB contains three deployment descriptors that contain structural and application assembly information for each enterprise bean. Two of these, **ejb-jar.xml** (for the EJB itself) and **sun-j2ee-ri.xml** (for the J2EE specification), are the same for all platforms. The application server descriptor for WebLogic is **weblogic-ejb-jar.xml**; for WebSphere, **ibm-ejb-jar-bnd.xmi**.

The following sections describe how to edit **ejb-jar.xml** and either **weblogic-ejb-jar.xml** or **ibm-ejb-jar-bnd.xmi**. You need not modify **sun-j2ee-ri.xml**.

Caution	EJB files contain class files required by eaDirect for each J2EE application. DO NOT MODIFY these class files while you are renaming your application context. Edit only the XML files.

Extract descriptors from the EJB JAR File

For each EJB in eaSample, you should follow these steps:

1. Extract **ejb-jar.xml** from the JAR file.

2. Extract **weblogic-ejb-jar.xml** (WebLogic) or **ibm-ejb-jar-bnd.xmi** (WebSphere) from the JAR file.

3. Change all occurrences of **eaSample** to **Newapp**, and **easample** to **newapp** (note case-sensitivity) in both files.

4. Repackage the JAR file with its two new descriptors.

5. Clean up your working directory before proceeding.

The following sections give a detailed example of each step.

Tip	As long as eaSample is deployed on your application server, you may find that a new J2EE application works fine without renaming these files. However, to make your code more robust and prevent errors (for example, your session unexpectedly timing out), use the best practice of mapping all application server deployment descriptors to the JNDI name of your web application.

Edit ejb-jar-xml

The **ejb-jar.xml** deployment descriptor defines runtime properties of the enterprise bean. Deployment descriptors are located in the WEB-INF directory of the EJB. The following diagram illustrates the XML elements defining runtime properties of **ejb-annotation.jar** in eaSample.

From this example, we can see that this **ejb-jar** deployment descriptor refers to a JAR file for a stateless session bean. This JAR is named **AnnotationJAR**, and its session properties define its home, remote, class, session, and reference types. These properties should not be modified.

This bean is mapped to the JMS Publisher name of its web application in the third annotation entry of the **env-entry** property. The JMS publisher name contains the JNDI name of the bean itself, in this example **jms/edx/eaSample/annotation/publisherName.**

You need to change this JNDI mapping of **eaSample** to **newapp**, or the name of your own web application. *You should make this change for every EJB in the web application, in both the EAR and WAR files.*

Edit weblogic-ejb-jar.xml (WebLogic)

The **weblogic-ejb-jar.xml** deployment descriptor defines runtime properties of the enterprise bean for the BEA WebLogic application server. Deployment descriptors are located in the WEB-INF directory of the EJB. The following diagram illustrates the XML elements defining WebLogic properties of **ejb-annotation.jar** in eaSample.

From this example, we can see that this **weblogic-ejb-jar** deployment descriptor refers to a WebLogic enterprise bean. This bean is named Annotation, and it is a stateless session bean that allows up to 500 instances in the free pool and may be clustered across multiple application servers. These properties should not be modified.

For more information about XML elements in **weblogic-ejb-jar**, see http://edocs.bea.com/wls/docs61/ejb/reference.html.

This bean is mapped to the JNDI name of its web application in two places:

- the JNDI name of the bean itself, in this example
  `edx/eaSample/ejb/Annotation`

- the JNDI name of its reference descriptor, `edx/eaSample/ejb/AppIndexVolMgr.`

To rename eaSample, change *both* these JNDI mappings of **eaSample** to **newapp** or the name of your own web application.

Edit ibm-ejb-jar-bnd.xmi (WebSphere)

The **ibm-ejb-jar-bnd.xmi** deployment descriptor defines runtime properties of the enterprise bean for the IBM WebSphere application server. Deployment descriptors are located in the WEB-INF directory of the EJB. The following diagram illustrates the XML elements defining WebSphere properties of **ejb-annotation.jar** in eaSample.
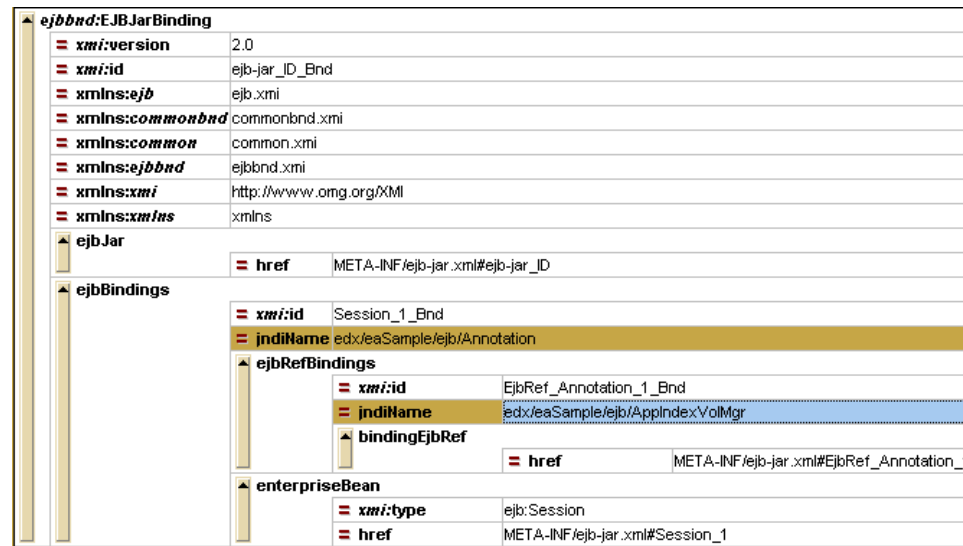
From this example, we can see that this **ibm-ejb-jar-bnd.xmi** deployment descriptor refers to a WebSphere enterprise bean (by noting that its root element is **EJBJarBinding**). WebSphere defines fewer properties in its EJB descriptor, and apart from the JNDI name, these properties should not be modified.

For more information about EJB bindings for WebSphere, see http://www-3.ibm.com/software/webservers/appserv/doc/v40/ae/infocenter/was/003502.html.

This bean is mapped to the JNDI name of its web application in two places:

- the JNDI name of the bean itself, in this example
  edx/eaSample/ejb/Annotation.

- the JNDI name of its EJB reference bindings,
  edx/eaSample/ejb/AppIndexVolMgr.

You need to change *both* these JNDI mappings of **eaSample** to **newapp**, or the name of your own web application.

Repackage the EJB Jar File

Once you have modified both EJB descriptors, repackage the JAR file using the following command:

```
jar cvf ejb-application.jar com META-INF/weblogic-ejb-jar.xml
META-INF/ejb-jar.xml _WL_GENERATED META-INF/sun-j2ee-ri.xml
```

Clean up your working directory before proceeding.

Caution   Windows users should *not* use WinZip to repackage components, as
this can cause your application to deploy incorrectly.

Repeat for each EJB in the EAR and WAR files

*Remember to rename EJB descriptors for every EJB in the web application, in both the
EAR and WAR files.* You may want to use [Appendix A: Components of EAR and WAR
Samples](#) as a checklist.

Tip   FOR WEBLOGIC ONLY, the order capture EJBs (`ejb-oc-cc.jar, ejb-oc-health.jar` and `ejb-telco.jar`) have additional deployment
descriptors configuring their RDBMS connection pool settings. For information on
RDBMS settings, see the eaDirect Installation and Configuration Guides.

Edit WAR File Deployment Descriptors

Extract the eaSample WAR File

The `war-easample.war` file contains example web application components that you
can customize or extend. The following chapter, [Creating Your Own Custom Web
Application for eaDirect](#), gives an overview of some common customization choices and
directs you to the appropriate SDK module for working with those components. Most
customization involves individual WAR file components, primarily JSPs and HTML
files. In this section, we will address only customizing the deployment descriptors to the
new context and JNDI name.

Extract the contents of the WAR file to a temporary directory using the following
command:

```
mkdir temp
jar xvf war-easample.war
```

Windows users can extract the contents of an EAR file with WinZip.

For a complete listing of components in each directory of `war-easample.war`, see
[Appendix A: Component Lists for Sample Web Applications](#).

Architecture of the WAR File

After the extraction, your working directory will contain the files listed in [Components of
war-easample.war](#) and shown in the following diagram.

eaSample Architecture: WAR File Components
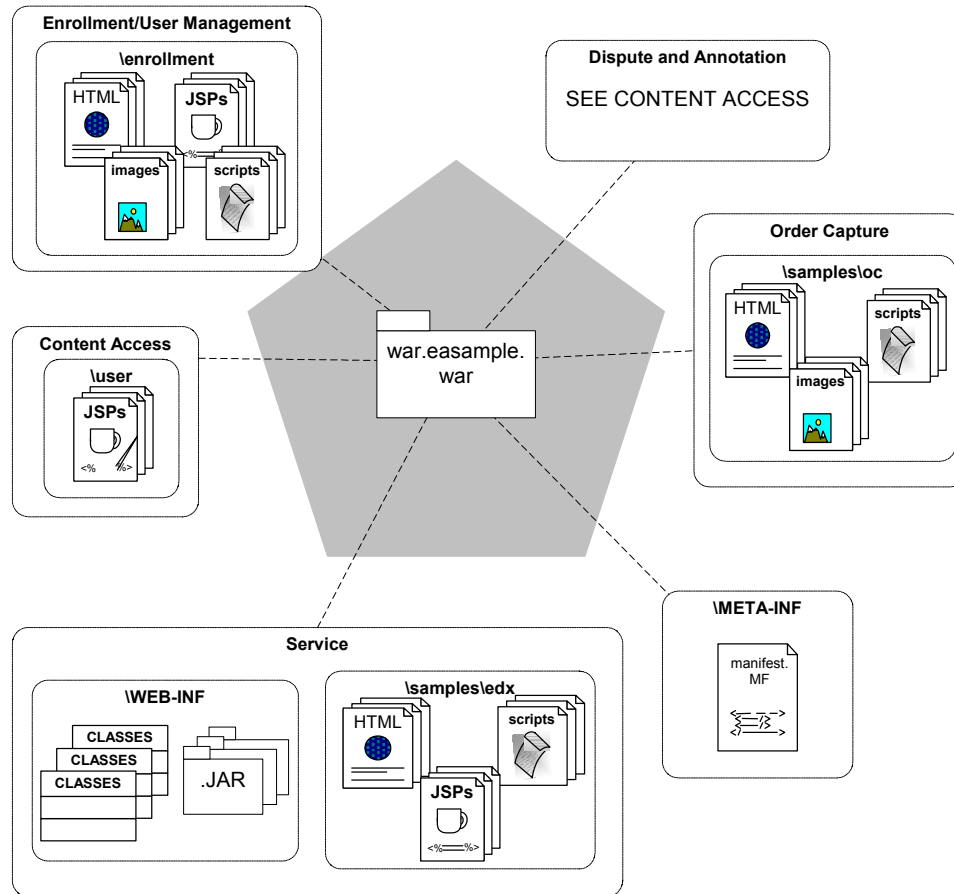
These include:

- *deployment descriptors* in the `WEB-INF` directory. You will need to customize each of these deployment descriptors as described in the next section.

- JSPs, JavaScripts, and HTML pages for *user management*, in the `enrollment` directory. To customize these files, see the SDK guide *User Management Frameworks*.

- JSPs, JavaScripts, and HTML pages for *content access* and *line item dispute and annotation*, in the `user` directory. To customize these files, see the SDK guides *Content Access* and *Line Item Dispute and Annotation.*.

- *Class files* and executable JAR files that reference servlets and EJBs, in the `WEB-INF` directory. You will need to customize the deployment descriptors for each of the JAR files.

Edit web.xml

The **web.xml** deployment descriptor is a complex file that defines the properties of your web application WAR file.

For instructions on writing the **web.xml** deployment descriptor for BEA WebLogic, see http://edocs.bea.com/wls/docs61/webapp/webappdeployment.html#1012209.

For more information about **web.xml** elements for WebSphere, see http://www-4.ibm.com/software/webservers/studio/doc/v40/studioguide/en/html/sdswar.html#webxml

To rename eaSample in **web.xml**, search for all occurrences of **eaSample** or **easample** (note case-sensitivity) and replace with **Newapp** or **newapp**. This will change the elements display name, servlet account name, and some EJB references. For definitions of **web.xml** elements for WebLogic, see http://edocs.bea.com/wls/docs61/webapp/web_xml.html.

The following illustration shows the ID and servlet properties for eaSample. Note that this illustrates only part of **web.xml** for eaSample.

XML					
DOCTYPE **web-app**					
ID PUBLIC	"-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN" "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd"				
web-app					
= id	WebApp_1				
{} display-name	eaSample				
{} distributable					
servlet (5)					
	{} servlet-name	{} display-name	{} servlet-class	{} init-param	{}
1	UserServlet	UserServlet	com.edocs.app.AppServlet	**init-param** (5)	1
				{} param-name	{} param-value
				1 ServletRoot	com.edocs.app.user
				2 ErrorPage	/enrollment/jsp/UserErrorMsg.jsp
				3 LoginRoot	com.edocs.app.enrollment
				4 LoginPage	/enrollment/jsp/UserLogin.jsp
				5 Account.name	edx/eaSample/ejb/CDAAccount
2	UserDispute	UserDispute	com.edocs.app.AppServlet	**init-param** (5)	1
3	UserAnnotation	UserAnnotation	com.edocs.app.AppServlet	**init-param** (5)	1
4	UserEnrollmentServlet	UserEnrollmentServlet	com.edocs.app.AppServlet	**init-param** (5)	
5	ChartServlet	ChartServlet	com.edocs.app.AppServlet	**init-param** (5)	
servlet-mapping (5)					
session-config					
welcome-file-list					
taglib					
env-entry					
ejb-ref (23)					

You will need to change the *display name* from **eaSample** to **Newapp** or the name of your custom web application.

eaSample uses five *servlets* as illustrated. You will also need to change the **Account.name** parameter for each servlet. Here is an XML example of the Account name parameter for the **UserServlet**:

```
<init-param>
<param-name>Account.name</param-name>
<param-value>edx/eaSample/ejb/CDAAccount</param-value>
</init-param>
```

For *each* servlet, you must change the **Account.name** value (shown in bold) from **edx/eaSample/ejb/CDAAccount** to **edx/newapp/ejb/CDAAccount.** For more information on customizing account names, see the SDK guide *User Management Frameworks.*

The eaSample WAR file also references 23 EJBs in various components of the EAR file. Each reference defines the type of bean and the location of its home and remote interface classes. This illustration shows the EJB references for eaSample in **web.xml**.

ejb-ref (23)

	= id	() ejb-ref-name	() ejb-ref-type	() home	() remote
1	EjbRef_1	ejb/AppMgr	Session	com.edocs.services.application.IAppMgrHome	com.edocs.services.application.IAppMgr
2	EjbRef_2	ejb/AppIndexVolMgr	Session	com.edocs.services.application.IAppIndexVolMgrHome	com.edocs.services.application.IAppIndexVolMgr
3	EjbRef_3	ejb/AppVolMgr	Session	com.edocs.services.application.IAppVolMgrHome	com.edocs.services.application.IAppVolMgr
4	EjbRef_4	ejb/EdocsDataSource	Session	com.edocs.services.application.IDataSourceHome	com.edocs.services.application.IDataSource
5	EjbRef_5	ejb/FileServer	Session	com.edocs.services.fileserver.IFileServerHome	com.edocs.services.fileserver.IFileServer
6	EjbRef_6	ejb/CSVMerger	Session	com.edocs.services.merger.IMergerHome	com.edocs.services.merger.IMerger
7	EjbRef_7	ejb/HTMLMerger	Session	com.edocs.services.merger.IMergerHome	com.edocs.services.merger.IMerger
8	EjbRef_8	ejb/XMLMerger	Session	com.edocs.services.merger.IMergerHome	com.edocs.services.merger.IMerger
9	EjbRef_9	ejb/XSLTMerger	Session	com.edocs.services.merger.IMergerHome	com.edocs.services.merger.IMerger
10	EjbRef_10	ejb/XMLQueryMerger	Session	com.edocs.services.merger.IMergerHome	com.edocs.services.merger.IMerger
11	EjbRef_11	ejb/Session	Session	com.edocs.services.session.ISessionHome	com.edocs.services.session.ISession
12	EjbRef_12	ejb/VersionManager	Session	com.edocs.services.versioning.IVersionManagerHome	com.edocs.services.versioning.IVersionManager
13	EjbRef_13	ejb/VersionSetReader	Session	com.edocs.services.versioning.IVersionSetReaderHome	com.edocs.services.versioning.IVersionSetReader
14	EjbRef_14	ejb/Annotation	Session	com.edocs.direct.annotation.IAnnotationBeanHome	com.edocs.direct.annotation.IAnnotationBean
15	EjbRef_15	ejb/Dispute	Session	com.edocs.direct.dispute.IDisputeBeanHome	com.edocs.direct.dispute.IDisputeBean
16	EjbRef_16	ejb/oc/cc/AddJointAcHolderReq	Entity	com.edocs.samples.oc.cc.IAddJointAcHolderReqHome	com.edocs.samples.oc.cc.IAddJointAcHolderReq
17	EjbRef_17	ejb/oc/cc/AdditionalUserReq	Entity	com.edocs.samples.oc.cc.IAdditionalUserReqHome	com.edocs.samples.oc.cc.IAdditionalUserReq
18	EjbRef_18	ejb/oc/cc/CreditLineIncReq	Entity	com.edocs.samples.oc.cc.ICreditLineIncReqHome	com.edocs.samples.oc.cc.ICreditLineIncReq
19	EjbRef_19	ejb/oc/cc/BalanceTransferReq	Entity	com.edocs.samples.oc.cc.IBalanceTransferReqHome	com.edocs.samples.oc.cc.IBalanceTransferReq
20	EjbRef_20	ejb/oc/cc/PaymentInquiryReq	Entity	com.edocs.samples.oc.cc.IPaymentInquiryReqHome	com.edocs.samples.oc.cc.IPaymentInquiryReq
21	EjbRef_21	ejb/oc/telco/TelcoOrder	Entity	com.edocs.samples.oc.telco.ITelcoOrderHome	com.edocs.samples.oc.telco.ITelcoOrder
22	EjbRef_22	ejb/oc/telco/AddOrderToAccount	Entity	com.edocs.samples.oc.telco.IAddOrderToAccountHome	com.edocs.samples.oc.telco.IAddOrderToAccount
23	EjbRef_23	ejb/oc/health/EnrollEmployee	Session	com.edocs.samples.oc.health.IEnrollEmployeeHome	com.edocs.samples.oc.health.IEnrollEmployee

You need not change these EJB references at this stage, but you will need to update them if you add custom components to your WAR file. Note that these reference *must* map to the EJB references in **weblogic.xml** or **ibm-ejb-jar-bnd.xmi** as shown in the following sections.

Edit weblogic.xml (WebLogic)

The **weblogic.xml** deployment descriptor binds each EJB in the EAR file to its JNDI name in the WAR file. The following illustration shows the EJB reference descriptions for eaSample in WebLogic. Note that each maps to an EJB reference in **web.xml**, which in turn points back to the bean interface classes.

XML

DOCTYPE weblogic-web-app

ID. PUBLIC "-//BEA Systems, Inc.//DTD Web Application 6.0//EN" "http://www.bea.com/servers/wls600/dtd/weblogic-web-jar.dtd"

weblogic-web-app

reference-descriptor

ejb-reference-description (23)

	() ejb-ref-name	() jndi-name
1	ejb/AppMgr	edx/eaSample/ejb/AppMgr
2	ejb/AppIndexVolMgr	edx/eaSample/ejb/AppIndexVolMgr
3	ejb/AppVolMgr	edx/eaSample/ejb/AppVolMgr
4	ejb/EdocsDataSource	edx/eaSample/ejb/EdocsDataSource
5	ejb/CSVMerger	edx/eaSample/ejb/CSVMerger
6	ejb/HTMLMerger	edx/eaSample/ejb/HTMLMerger
7	ejb/XMLMerger	edx/eaSample/ejb/XMLMerger
8	ejb/XSLTMerger	edx/eaSample/ejb/XSLTMerger
9	ejb/XMLQueryMerger	edx/eaSample/ejb/XMLQueryMerger
10	ejb/Session	edx/eaSample/ejb/Session
11	ejb/VersionSetReader	edx/eaSample/ejb/VersionSetReader
12	ejb/VersionManager	edx/eaSample/ejb/VersionManager
13	ejb/FileServer	edx/eaSample/ejb/FileServer
14	ejb/Annotation	edx/eaSample/ejb/Annotation
15	ejb/Dispute	edx/eaSample/ejb/Dispute
16	ejb/oc/cc/AddJointAcHolderReq	edx/eaSample/ejb/oc/cc/AddJointAcHolderReq
17	ejb/oc/cc/AdditionalUserReq	edx/eaSample/ejb/oc/cc/AdditionalUserReq
18	ejb/oc/cc/CreditLineIncReq	edx/eaSample/ejb/oc/cc/CreditLineIncReq
19	ejb/oc/cc/BalanceTransferReq	edx/eaSample/ejb/oc/cc/BalanceTransferReq
20	ejb/oc/cc/PaymentInquiryReq	edx/eaSample/ejb/oc/cc/PaymentInquiryReq
21	ejb/oc/telco/TelcoOrder	edx/eaSample/ejb/oc/telco/TelcoOrder
22	ejb/oc/telco/AddOrderToAccount	edx/eaSample/ejb/oc/telco/AddOrderToAccount
23	ejb/oc/health/EnrollEmployee	edx/eaSample/ejb/oc/health/EnrollEmployee

To rename eaSample in **weblogic.xml**, search for all occurrences of **eaSample** or **easample** (note case-sensitivity) and replace with **Newapp** or **newapp**. This will map each EJB to its correct JNDI name.

Edit ibm-ejb-jar-bnd.xmi (WebSphere)

The **ibm-ejb-jar-bnd.xmi** deployment descriptor binds each EJB in the EAR file to its JNDI name in the WAR file. The following illustration shows the EJB reference descriptions for eaSample in WebSphere. Note that each maps to an EJB reference in **web.xml**, which in turn points back to the bean interface classes.

webappbnd:WebAppBinding

☰ **xmi:version**	2.0	
☰ **xmi:id**	WebApp_1_Bnd	
☰ **xmlns:xsi**	http://www.w3.org/2001/XMLSchema-instance	
☰ **xmlns:webapplication**	webapplication.xmi	
☰ **xmlns:commonbnd**	commonbndbnd.xmi	
☰ **xmlns:common**	common.xmi	
☰ **xmlns:webappbnd**	webappbnd.xmi	
☰ **xmlns:xmi**	http://www.omg.org/XMI	
☰ **xmlns:xmlns**	xmlns	
▾ **virtualHostName**	xsi:nil=true xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance	
▾ **webapp**	href=WEB-INF/web.xml#WebApp_1	
▲ **ejbRefBindings** (23)		

	☰ **xmi:id**	☰ **jndiName**	◇ **bindingEjbRef**
1	EjbRef_1_Bnd>	edx/eaSample/ejb/AppMgr	▾ **bindingEjbRef** href=WEB-INF/web.xml#EjbRef_1
2	EjbRef_2_Bnd>	edx/eaSample/ejb/AppIndexVolMgr	▾ **bindingEjbRef** href=WEB-INF/web.xml#EjbRef_2
3	EjbRef_3_Bnd>	edx/eaSample/ejb/AppVolMgr	▾ **bindingEjbRef** href=WEB-INF/web.xml#EjbRef_3
4	EjbRef_4_Bnd>	edx/eaSample/ejb/EdocsDataSource	▾ **bindingEjbRef** href=WEB-INF/web.xml#EjbRef_4
5	EjbRef_5_Bnd>	edx/eaSample/ejb/FileServer	▾ **bindingEjbRef** href=WEB-INF/web.xml#EjbRef_5
6	EjbRef_6_Bnd>	edx/eaSample/ejb/CSVMerger	▾ **bindingEjbRef** href=WEB-INF/web.xml#EjbRef_6
7	EjbRef_7_Bnd>	edx/eaSample/ejb/HTMLMerger	▾ **bindingEjbRef** href=WEB-INF/web.xml#EjbRef_7
8	EjbRef_8_Bnd>	edx/eaSample/ejb/XMLMerger	▾ **bindingEjbRef** href=WEB-INF/web.xml#EjbRef_8
9	EjbRef_9_Bnd>	edx/eaSample/ejb/XSLTMerger	▾ **bindingEjbRef** href=WEB-INF/web.xml#EjbRef_9
10	EjbRef_10_Bnd>	edx/eaSample/ejb/XMLQueryMerger	▾ **bindingEjbRef** href=WEB-INF/web.xml#EjbRef_10
11	EjbRef_11_Bnd>	edx/eaSample/ejb/Session	▾ **bindingEjbRef** href=WEB-INF/web.xml#EjbRef_11
12	EjbRef_12_Bnd>	edx/eaSample/ejb/VersionManager	▾ **bindingEjbRef** href=WEB-INF/web.xml#EjbRef_12
13	EjbRef_13_Bnd>	edx/eaSample/ejb/VersionSetReader	▾ **bindingEjbRef** href=WEB-INF/web.xml#EjbRef_13
14	EjbRef_14_Bnd>	edx/eaSample/ejb/Annotation	▾ **bindingEjbRef** href=WEB-INF/web.xml#EjbRef_14
15	EjbRef_15_Bnd>	edx/eaSample/ejb/Dispute	▾ **bindingEjbRef** href=WEB-INF/web.xml#EjbRef_15
16	EjbRef_16_Bnd>	edx/eaSample/ejb/oc/cc/AddJointAcHolderReq	▾ **bindingEjbRef** href=WEB-INF/web.xml#EjbRef_16
17	EjbRef_17_Bnd>	edx/eaSample/ejb/oc/cc/AdditionalUserReq	▾ **bindingEjbRef** href=WEB-INF/web.xml#EjbRef_17
18	EjbRef_18_Bnd>	edx/eaSample/ejb/oc/cc/CreditLineIncReq	▾ **bindingEjbRef** href=WEB-INF/web.xml#EjbRef_18
19	EjbRef_19_Bnd>	edx/eaSample/ejb/oc/cc/BalanceTransferReq	▾ **bindingEjbRef** href=WEB-INF/web.xml#EjbRef_19
20	EjbRef_20_Bnd>	edx/eaSample/ejb/oc/cc/PaymentInquiryReq	▾ **bindingEjbRef** href=WEB-INF/web.xml#EjbRef_20
21	EjbRef_21_Bnd>	edx/eaSample/ejb/oc/telco/TelcoOrder	▾ **bindingEjbRef** href=WEB-INF/web.xml#EjbRef_21
22	EjbRef_22_Bnd>	edx/eaSample/ejb/oc/telco/AddOrderToAccount	▾ **bindingEjbRef** href=WEB-INF/web.xml#EjbRef_22
23	EjbRef_23_Bnd>	edx/eaSample/ejb/oc/health/EnrollEmployee	▾ **bindingEjbRef** href=WEB-INF/web.xml#EjbRef_23

To rename eaSample in **ibm-ejb-jar-bnd.xmi**, search for all occurrences of **eaSample** or **easample** (note case-sensitivity) and replace with **Newapp** or **newapp**. This will map each EJB to its correct JNDI name.

Earlier versions of eaSample contained hard-coded references to its context root on its welcome file, **user/jsp/index.jsp**. This JSP now requests the context root with the method call **String context = request.getContextPath().** The only references to eaSample in **index.jsp** are displayed as HTML text in the title and instructional URL, which will usually be removed as part of customization. edocs recommends calling **request.getContextPath()** as a best practice when customizing welcome or index pages.

Package the WAR File

When you have finished editing deployment descriptors for all WAR file components, repackage the WAR file using this command:

```
jar cvf war-newapp.war enrollment user WEB-INF
```

It is important to clean up your working directory after packaging and BEFORE extracting the next JAR file. If you skip cleanup, the next JAR files you edit may become corrupted with extraneous files.

Caution   Windows users should *not* use WinZip to repackage components, as this can cause your application to deploy incorrectly.

Repackage and Deploy the EAR File

Package the EAR file with a new name using the following command:

```
jar cvf ear-newapp.ear *
```

Deploy your custom application on your application server using the instructions in the *eaDirect Install and Configuration Guide*.

To view your application, follow the *eaDirect Install and Configuration Guide* instructions for eaSample, replacing **eaSample** in the URL with **newapp** or the context name of your new web application.

Tip   Remember, you must index a dataset, publish a dynamic web view, enroll a user, and log in to see your statement data online.

4

Creating a Custom Web Application for eaDirect

Define a Custom Enrollment Model

The steps to defining your custom enrollment model will vary depending on which type you choose: CDA, LDAP, or non-directory access models. In any case, customizing your enrollment model will affect the enrollment EJB you select, for example **enrollment-cda.jar** in eaSample, and the enrollment information for the servlets in the WAR file.

For more information on choosing an enrollment model and customizing required files, see the SDK guide *User Management Frameworks*.

Define Custom Servlets

Servlets in a web application follow the architecture of the WAR file. If you create new class files for servlets, you should place them in a new subdirectory in the **WEB-INF/classes/com** directory.

For example, both the **UserServlet** and **UserEnrollment** servlet use the **AppServlet.class** file located in **WEB-INF/classes/com/edocs/app**. If you want to create a new **AppServlet.class** file, you should create it in a new subdirectory rather than replacing the existing file.

To create a custom servlet:

1. Write and compile your servlet class file, for example, **AppServletNew.class**.

2. Create a new directory in **WEB-INF/classes/com**, for example, **/newapp** and move your class file.

3. Modify the deployment descriptor files as described in preceding sections.

Customize the WAR File

The **enrollment**, **samples**, and **user** directories have the same basic subdirectory structure. Use this structure when creating new directories, for clarity.

- *help*—any help files that are associated with the application.

- *html*—the static display HTML pages to display such as error pages.

- *images*—common GIF or JPEG files that used by the HTML pages or JSPs.

- *jsp*—dynamic web application pages that are the primary interface for your custom application.

- *scripts*—any script files required for processing by a JSP. For example, the sorting feature of eaDirect requires the `script.js` file.

- *templates*—any template files used for your application.

Caution   When creating or modifying web application files, follow the eaDirect API Specification to reference eaDirect APIs correctly.

After completing your changes, [Package the WAR File]. Be sure to include any new WAR file directories in your **jar** command.

Caution   Do not include the JAR file *edx_client.jar* in the *WEB-INF/lib* directory of your customized WAR file. This file belongs only in the *lib* directory of the EAR file, and a copy inside the WAR file could cause your application to fail to deploy correctly.

Then [Edit, Package, and Deploy the EAR File] to test your application.

Implement JSP Validation

About Validation of HTTP Requests

It is common practice to embed HTTP REQUEST parameters into the generated HTML page, for any number of reasons. If such a parameter is hacked to contain HTML content, for example to invoke a JavaScript function, the browser may end up executing malicious code within an authenticated session. This "cross-site scripting can be very damaging.

For background and discussion. see the *CERT® Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests* at http://www.cert.org/advisories/CA-2000-02.html.

To limit this vulnerability, edocs requires that all REQUEST parameters be checked to ensure that they contain no "active content" that might somehow trigger unwanted behavior when delivered back to an HTML browser.

EaDirect sample applications include the JSP code fragment in Validation.jsp, to be included in application-specific JSP pages. This fragment contains a **ValidatorBean** that validates the input and throws an exception in case of danger. Since this fragment will need to be customized per site or per application, some support is provided for helping determine the needed customizations.

Package com.edocs.common.web.validation Description

This API package implements the class **ValidatorBean** to support validation of input to JSP pages. Bean methods capture, set, validate, and write the list of legal and illegal parameter names and values in a ServletRequest object.

You can use a standard server-side include to call this JSP fragment, as illustrated in these lines from the beginning of **Detail.jsp**:

```
// From Detail.jsp
<%@ include file="Validation.jsp" %>
```

About the ValidatorBean

The **ValidatorBean** is intended to be instantiated in a JSP page via the **<jsp:useBean>** tag, and the methods on the bean should validate all parameters for every REQUEST.

The list of illegal substrings is strongly related to the character set encoding. The sample provided assumes ISO-8859-1.

The list of illegal substrings is essentially a collection of the characters '<' and '>' in all the various encodings. There may well be an interaction with regex dynamic patterns.

Clearly, this discussion assumes use in a JSP context. For pure servlet applications, it should be possible to call the bean directly, though this has not been tested.

During development (especially when using the parameter name checking) you may want to set the scope of the bean will page. For production this can be changed to application - which will improve performance by caching the created and initialized bean.

In some limited cases it may be necessary to allow illegal substrings for certain parameters. This can be achieved by designating them as exempt, using the **setExemptParameterNames()** method.

Once a REQUEST has been checked, an attribute is set on the REQUEST object that alerts the validator that the object been checked, and should not be checked again. This prevents infinite recursion when the error page also performs parameter checking, since this recursion has been known to crash application servers.

> **Caution** Forwarding requests between pages and modifying the URL at the same time can introduce security holes, because the REQUEST will not be checked again in the destination JSP.

To create and initialize a ValidatorBean:

```
<jsp:useBean id="validator"
class="com.edocs.common.web.validation.ValidatorBean"
scope="application">
<%
  // note that the entity-reference style encoding appears
without the trailing semicolon
  //
  // the reason is that some browsers are too helpful and will
guess that you
  // meant to include it. Rather than putting both &lt; and
&lt in the list,
  // we really need only the short one...
  //
  // if you use a different encoding, you will need to rethink
this list.
  //
  String[] iso_8859_1_values =
{"<",">","&lt","&gt","%3c","%3e","&#60","&#62"};
  validator.setIllegalParameterSubStrings(iso_8859_1_values);
%>
</jsp:useBean>
```

In general, the only time you'll need to modify this is if you use a different encoding, which might cause the browser to treat more/different characters as "active" in the same way as '<' and '>'.

To invoke a ValidatorBean:

```
<%
  response.setContentType("text/html; charset=ISO-8859-1");
  validator.validateParameters(request);
%>
```

This method checks all REQUEST parameter names and values for illegal substrings and throws a ValidationException if any illegal substring is detected.

Using ValidatorBean for Parameter Name Checking

The **ValidatorBean** can also be used to validate the exact identities of parameter names. Using the ValidatorBean to check identities of parameters requires that you have an exact list of the parameters the application uses. Generating this list can be daunting, so the ValidatorBean implements some support for collecting this list. During DEVELOPMENT, a flag is set to capture parameter names and record them. For production, the flag must be cleared and the captured names put into the Validation.jsp page.

Because the penalty for failing a validation step is an exception, validation will typically be disabled during development, and enabled only when the application is otherwise working. The task of discovering the list of valid parameter names can be eased somewhat by using Capture Mode.

In capture mode, the bean collects all request parameter names in an internal list, and will print them out nicely when asked (via **writeParameterNames()** method). No checks are actually performed, and no exceptions thrown.

To enter Capture Mode, call **setCaptureMode()** with **true** as the argument. The validation methods will not throw an exception, but the **validateParameterNames()** method will accumulate the names of all parameters it sees. These can be output at any time by calling **writeParameterNames()** method. The output can help create the complete list of legal names.

If some parameter is designated as *exempt*, its identity is still checked against the list of legal parameter names, but its *value* is not checked.

In validation mode, required for production, the bean actually performs the validations as required, and throws a **ValidationException** when some parameter name or value is invalid.

The list of legal parameter names in the following example is specific to an eaDirect sample application. In practice, the list of legal parameter names will vary widely among applications, so this procedure must be configurable.

To validate parameter names:

```
<jsp:useBean id="validator"
class="com.edocs.common.web.validation.ValidatorBean"
scope="application">
<%
  // this is the list of all legal request parameter names
  String[] params = { "auth__userPassword", "docId", "ddn",
            "jsp", "app", "auth__uid", "APP_METHOD",
            "errforwardto", "EDOCSLOGIN", "edocs__re-login",
            "auth__dn", "Submit", "viewType", "viewName",};

  validator.setLegalParameterNames(params);

  String[] iso_8859_1_values =
{"<",">","&lt","&gt","%3c","%3e","&#60","&#62"};
  validator.setIllegalParameterSubStrings(iso_8859_1_values);

  // set this during development to accumulate parameter names
  validator.setCaptureMode(true);
%>
</jsp:useBean>

The following JSP fragment shows the invocation of the bean:

<%
  response.setContentType("text/html; charset=ISO-8859-1");

  // call to check that only parameters from the name list are
present
  validator.validateParameterNames(request);

  // call to check for illegal substrings
  validator.validateParameters(request);
```

```
     // call DURING DEVELOPMENT to write out accumulated
parameter names
   validator.writeParameterNames(out);
%>
```

Appendix A: Components of EAR and WAR Samples

Components of ear-easample.ear

The EAR file contains a customizable web application archive (WAR file) and service EJBS to support eaDirect. EJB JAR files should NOT be modified or customized beyond modifying their deployment descriptors as described in this guide.

<root> directory of ear-easample.ear

Name	Purpose
ejb-annotation.jar	Service EJB for annotation of detail data. Used by *Line Item Dispute and Annotation*.
ejb-application.jar	Service EJB for management of DDNs. Used by all modules.
ejb-ccmerger.jar	Service EJB for composition of views. Used by all modules.
ejb-dispute.jar	Service EJB for line item dispute of detail data. Used by *Line Item Dispute and Annotation*.
ejb-enrollment-cda.jar	Java class and descriptor files for defining a User Management EJB with CDA. Used by *Implementing a User Management Framework*.
ejb-fileserver.jar	Service EJB for file transfer from web to application tier. Used by all modules.
ejb-oc-cc.jar	*Customizable* EJB for order capture for the credit card vertical solution. Used by *Order Capture and Management*.
ejb-oc-telco.jar	*Customizable* EJB for order capture for the telecommunications vertical solution. Used by *Order Capture and Management*.
ejb-oc-health.jar	*Customizable* EJB for order capture for the health care vertical solution. Used by *Order Capture and Management*.

Name	Purpose
`ejb-querymerger.jar`	Service EJB for processing XML queries. Used by *Content Access* and *Line Item Dispute and Annotation*.
`ejb-session.jar`	Service EJB for application session management. Used by all modules.
`ejb-versioning.jar`	Service EJB for versioning of views. Used by all modules.
`ejb-xsltmerger.jar`	Service EJB for processing XSLT stylesheets. Used by *Content Access* and *Line Item Dispute and Annotation*.
`war-easample.war`	*Customizable We*b *A*pplication a*R*chive file containing JSPs, servlets, and descriptor files required for presentment. For more information, see [Components of war-easample.war]().

Tip	Note to customers migrating from previous versions: eaSample no longer contains the service EJB `ejb-log.jar`. Logging in eaDirect 2.x is handled internally and no longer requires this web application component.

/lib directory

Name	Purpose
`dom4j.jar`	Service JAR file containing Document Object Model (DOM) Java interface for dynamic access to HTML documents. Used by all modules. For more information, see http://www.w3.org/DOM/.
`edx_client.jar`	Service EJB containing client APIs for access to other EJBs in the archive. Used by all modules. For more information, see the *eaDirect SDK API Specification*.
`edx_common.jar`	Service EJB containing Java classes common to all beans. Used by all modules. For more information, see the *eaDirect SDK API Specification*.
`edx_servlet.jar`	Service JAR file containing Java classes for web APIs into eaDirect. Used by all modules. *Note:* This file is required in the **/lib** directory for each WAR file. It is also included at the EAR level for convenience when building a new web application. Sample web applications include **edx_servlet.jar** in both ***.ear/lib** and ***.ear/*.war/web-inf/lib**.
`xalan.jar`	Service JAR file containing Apache's Java implementation of the DOM API as an XSLT stylesheet processor. eaDirect uses xalan-Java-2.2 D12 as the fastest XSLT processor currently available. For more

Name	Purpose
	information, see http://xml.apache.org/.
xalanj1compat.jar*	Service JAR file containing xalan-Java-1 XSLT processor, which is no longer supported by Apache but is required by the DOM4 API.

/meta-inf directory

Name	Purpose
application.xml	XML deployment descriptor for contents of the EAR file.
manifest.mf	Manifest listing the contents of the EAR file.

Components of war-easample.war

enrollment directory

The **enrollment** directory contains HTML, images, JSPs, and scripts for an example implementation of the edocs user management framework. For more information and a complete listing of files in this directory, the SDK guide *User Management Frameworks*.

Tip	The eaSample enrollment implementation is a simplified flat model using Common Directory Access (CDA). Most custom implementations will want to use **eaTraining** or **umfsample** as a template for your user management framework.

META-INF directory

The **META-INF** directory contains **manifest.MF**, which lists all the components in the EAR file.

samples/edx directory

The **samples/edx** directory contains HTML, JSPs, and scripts for a generic error message page used by edocs sample web applications. Most implementations will replace this directory with custom error message files.

user/jsp directory

The **user/jsp** directory contains JSPs for content access to line item detail, dispute and annotation. For more information and a complete listing of files in this directory, see the SDK guide: *Line Item Dispute and Annotation*.

WEB-INF directory

The **WEB-INF** directory contains XML deployment descriptors, compiled Java class files, and executable .JAR files that support eaDirect.

<root>

Name	Purpose
ibm-web-bnd.xmi (WebSphere only)	XML deployment descriptor specific to the IBM WebSphere application server.
sun-j2ee-ri.xml	XML deployment descriptor specific to Sun systems.
web.xml	XML deployment descriptor specific to your web application's WAR file.
weblogic.xml (WebLogic only)	XML deployment descriptor specific to the BEA WebLogic application server.

/class

This directory contains compiled Java class files for eaDirect APIs in the **/com** subdirectory. For more information on files in this directory, please see the *eaDirect 3.4 SDK API Specification JavaDoc*.

For *WebLogic* web applications, this directory also contains precompiled JSP class files in the **/jsp_servlet** subdirectory. These pages take advantage of the JSP Compiling feature of WebLogic. Since WebLogic compiles JSPs on the fly, precompiled JSPs will load faster when first accessed; if the JSP classes don't exist, WebLogic compiles them at runtime. For more information on compiling JSPs with WebLogic, please see the BEA guide to Using the WebLogic JSP Compiler online at:

http://e-docs.bea.com/wls/docs61////////jsp/reference.html#57794

Tip	You can configure WebLogic Server to precompile JSPs by setting the `precompile` parameter to true in the `<jsp-descriptor>` element of `weblogic.xml` See BEA documentation for details.

/lib

This directory contains utility files used by web application components.

Name	Purpose
edx_servlet.jar	Service JAR file containing Java classes for web APIs into eaDirect. Used by all modules.
	This file is required in the /lib directory for each WAR file; it is also included at the EAR level for convenience when building a new web application. Sample web applications include **edx_servlet.jar** in both ***.ear/lib** and ***.ear/*.war/web-inf/lib**.

Name	Purpose
`javachart.jar`	Service JAR file containing KavaChart charting utility shipped with eaDirect. Used by **Creating Custom Charts**. For more information about KavaChart, see http://www.ve.com/.
`xtags.jar`	Service JAR file containing XTags custom tag libraries (taglibs) for working with XML in JSPs. Used by **Content Access** and **Line Item Dispute and Annotation** For more information about XTags and taglibs, see http://jakarta.apache.org/taglibs/doc/xtags-doc/intro.html.

Appendix B: Components of eaDirect Application Datasets

Components of National Wireless dataset

The National Wireless sample application files are located in the **EDCSbd/samples/NatlWireless** directory. For more information about working with National Wireless, please see the edocs *Data Definition Guide* and the *Presentation Design Guide*.

Filename	Purpose
datafile/NatlWireless.txt	Sample data input file (source).
_DetailExtractor/edx-DE-table.xsd	Custom template for validating XML input to the Detail Extractor job.
DetailExtractor/summary_info.xml	Customizable input file to the Detail Extractor batch job containing business logic rules for extracting summary data as XML.
DetailExtractor/summary_info.xsl	Customizable input file to the Detail Extractor batch job containing XSLT stylesheet rules for presenting extracted XML summary data.
XMLQuery/annot_sql.xml	Customizable input file to the XML Query web view containing SQL statements for extracting annotation detail from database tables.
XMLQuery/detail_sql.xml	Customizable input file to the XML Query web view containing SQL statements for extracting line item detail from database tables.
XMLQuery/dispute_sql.xml	Customizable input file to the XML Query web view containing SQL statements for extracting dispute detail from database tables.
XSLTDownload/summary_info_csv.xsl	Customizable input file to the XSLT Download batch job containing XSLT stylesheet rules for downloading extracted XML summary data as comma-separated values (CSV).
B2B_cust_care_agent.gif	Image file

Filename	Purpose
B2B_DSLBanner.jpg	Image file
B2B_limo_ad_small.gif	Image file
B2C_FreeCallingCardSmallPic.gif	Image file
B2C_hawaii_promo.gif	Image file
Bk2Summary.jpg	Image file
edocslogo.gif	Image file
LDPhoneCard.gif	Image file
Loc_Summary.alf	Contains the customized rules and business logic for mapping and presenting the extracted data in the detail statement view NW_LocSummary.
Loc_Summary.htm	HTML template for presenting the detail statement view NW_LocSummary.
Loc_Summaryn.htm	
LocalSummary.jpg	Image file
NatlWireless.alf	Contains the customized rules and business logic for mapping and presenting the extracted data in an online summary statement.
NatlWireless.ddf	Contains the customized rules and business logic for extracting each field from each page of the original statement file for presenting different views of the data in online statements.
NatlWireless.htm	HTML template for presenting the summary statement view named HtmlDetail.
NatlWireless.tok	Sample file for the Def2TOK utility.
NW_Email.alf	Contains the customized rules and business logic for creating notification email.
NW_Email.htm	Example layout for notification email (its use is determined by business logic).
NW_EmailAlternate.htm	Alternate layout for notification email (its use is determined by business logic).
NW_LocSummary.alf	Contains the customized rules and business logic for presenting the detail statement view NW_LocSummary.
NW_LocSummary.ddf	Contains the customized rules and business logic for extracting each field from each page of the original statement file to present the detail statement view NW_LocSummary.
NW_LocSummary.htm	HTML template for presenting the detail statement view NW_LocSummary.
NW_Logo_B2B.jpg	Image file
NW_Logo_B2C.jpg	Image file

Filename	Purpose
NW_Logo_commercial.jpg	Image file
NW_Logo_consumer.jpg	Image file
PaymentBannerAd.gif	Image file

Index