SDK Guide for Oracle Siebel eStatement Manager

Version 4.7

May 31, 2007

ORACLE

Contents

4 User Management

6 Line Item Dispute and Annotations

1 Preface

About Customer Self-Service and eaSuite™

Oracle 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 consists 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 Oracle's 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.

- **eStatement Manager**™ 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 eStatement Manager with its data access layer, composition engine, and security, enrollment and logging framework to power complex Customer Self-Service applications.
- **ePayment Manager**™ is the electronic payment solution that decreases payment processing costs, accelerates receivables and improves operational efficiency. EPayment Manager 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.

Oracle's **Development Tools** are visual development environments for designing and configuring Oracle's 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

The Oracle Software Developers Kit allows developers to write custom code against Oracle applications. This SDK guide is intended for Oracle system integrator partners, senior developers with an Oracle client company, and Oracle Professional Services representatives.

This guide is intended for The SDK assumes you have:

- Completed a Statement Mastering Plan
- Installed and configured eStatement Manager and the sample application Sample

This SDK assumes in-depth understanding of and practical experience with:

- eStatement Manager system architecture, installation, deployment, application design, and administration
- Java 2 Enterprise Edition (J2EE), Enterprise JavaBeans (EJBs), servlets, and JSPs
- Packaging and deploying J2EE applications for WebLogic or WebSphere
- Directory services including the Java Naming Directory Interface (JNDI) and the Lightweight Directory Access Protocol (LDAP)
- HTML and XML, Web server administration, and Web browsers

Related Documentation

This guide is part of the eStatement Manager documentation set. For more information about using eStatement Manager, see the following guides:

Print Document	Description
Installation Guide for Oracle Siebel eStatement Manager	How to install and configure eStatement Manager in a distributed environment.
Migration Guide for Oracle Siebel eaSuite	How to migrate an existing eStatement implementation to the current version.
Deploying and Customizing J2EE Applications Guide for Oracle Siebel eStatement Manager	How to customize J2EE Web applications for deployment with the eaSuite.
Data Definition (DefTool) Guide for Oracle Siebel eStatement Manager	How to create data extraction and definition rules for an eStatement Manager application with the DefTool.
Presentation Design (Composer Guide) for Oracle Siebel eStatement Manager	How to design data presentment for an eStatement Manager application with the Composer.

Print Document	Description
Administration Guide for Oracle Siebel eStatement Manager	How to set up and run a live eStatement Manager application in a J2EE environment.
Reporting Guide for Oracle Siebel eStatement Manager	Describes the tasks required to use the reporting analytics feature to create reports.
Troubleshooting Guide for Oracle Siebel eaSuite	How to initiate the troubleshooting process, identify critical information about what was happening in your system and applications when the problem occurred, and suggests ways to resolve the problem.

.

2 Overview of eaSuite SDK

Deploying and Customizing J2EE Applications

Information about deploying and customizing J2EE applications is contained in the *Deploying and Customizing J2EE Applications Guide for Oracle Siebel eStatement Manager.* The guide contains details about:

- The components of a J2EE Web application for eStatement Manager, which can include customized JSPs, HTML pages, EJBs, and Java APIs and class files.
- The components of an eStatement Manager application (dataset), which includes a data source file, data definition and application logic files, and HTML templates.
- Introduction to the customizable Web application **`Sample.`**
- How to build a custom Web application for eStatement Manager by customizing sample components of **`Sample`** and **`Training.`**
- How to deploy your custom Web application to your application server.

Implementing a User Management Framework

The Implementing a user management framework chapter describes:

- Concepts in naming services and directory access.
- Core decisions surrounding customer login and data access.
- How to plan and select a directory framework for user management.
- How to build and manage a hierarchical user profile schema with the Oracle Common Directory Access (CDA) framework.
- How to implement non-directory access enrollment models.

About User Management

Defines the concepts and goals of the Oracle User Management Framework and identifies the core decisions surrounding customer login and data access needed to plan and design your user management framework. Describes components of the Sample J2EE Applications shipped with eStatement Manager, particularly Java Server Pages (JSP) and servlets in the sample Web applications. Describes how to authenticate user identity and add and modify user accounts using the **`IAccount`** interface for Oracle enrollment. Also defines the core interfaces and methods in the package **`com.edocs.enrollment.user,`** including **`IAccount`** and **`IAccountResolver`**, and describes methods for contexts, objects and attributes, searching, and authentication.

Introduction to Directory Access

Provides basic concepts and definitions for directory access services, naming systems, naming services, and namespaces, directory contexts, directory information trees (DIT), directory entries, objects, and attributes, directory schema, and distinguished names. Choosing a Directory Access Interface compares features of the Java Naming and Directory Interface™ (JNDI), Lightweight Directory Access Protocol (LDAP), and Common Directory Access (CDA) Interface.

Using the Common Directory Access (CDA) Framework

Introduces the Oracle user management framework shipped with eStatement Manager, including the Oracle default CDA schema and DIT used in the Training Web application. Discusses how to deploy and configure Training as a template with National Wireless. Defines how Training uses CDA and outlines the Training CDA process flow. Describes how to modify the CDA EJB for your application and how to use **`IAccount`** with CDA to obtain attributes and their values, manage user levels, and search for attributes.

Using the CDA Client

Using the CDA Client introduces the Oracle tool for creating and managing directory schemas. Describes how to start the CDA Client and parse CDA commands, navigate within a DIT, and work with directory contexts and user attributes. Also discusses how to integrate with existing user management systems by exporting a schema as LDIF and importing an LDIF schema into CDA.

Using a Non-Directory Access Implementation

Discusses implementing the Oracle user management framework outside of CDA. Describes how to deploy and configure **`UMFsample`** as a Template with National Wireless. Defines how **`UMFsample`** does enrollment and outlines the **`UMFsample`** Enrollment Process Flow. Also discusses how to create a custom an application based on **`UMFsample`** by modifying the enrollment source files, defining a custom enrollment EJB, and building the new custom version of **`UMFSample`**

Two Appendices suggest additional background reading and describe the User Management Framework API packages shipped with eStatement Manager.

Content Access

The content access chapter describes:

- How to plan and design data access for retrieving and presenting statements
- How to customize summary and detail Web views
- How to record and present Web-time activity, such as when a customer last viewed a statement

Planning Your Content Access Interface

Planning Your Content Access Interface defines the concepts and goals of content access and provides basic concepts about XML, XSL, and XSLT as used with eStatement Manager.

Introduction to Oracle Content Access

Describes the new XML views and jobs for eStatement Manager and the XML Templates available to customize input and output.

XML and eStatement Manager

Discusses how eStatement Manager uses XML, how to map a DDF to XML, and provides further reading about XML, XSL, and XSLT.

Extracting Detail Data to the Database

Describes the Detail Extractor Job and how to customize the XML templates provided.

Transforming Data with XSLT

Describes the XSLT View Type and how to customize the XML templates provided.

Extracting Data with XML Queries

Describes the XML Query View and how to customize the XML templates provided.

Application Programming Interfaces (API) for Content Access

Provides a description of package **`com.edocs.app.user`** methods, including the User and UserMain methods.

Using Content Access APIs

Describes how to use Content Access APIs by calling user methods in correct sequence to retrieve and present statement summaries, retrieve and present statement detail, retrieve and present XML, and record and present Web-time activity.

Element ID and Composition Hints

Defines Element ID and Composition Hints, the rich language of XML metadata provided with eStatement Manager. Defines syntax and tag attributes for Element ID and defines values and DTD for the Composition Hints language.

Appendix A lists sample code for the National Wireless XML Templates for Detail Extractor, XSLT View, XML Query View, and an example DDF to XML Mapping.

Line Item Dispute and Annotation

The line item dispute and annotation chapter describes:

- How to extract and retrieve line item detail
- How to dispute all or part of a line item

- How to add annotations to line items

Introduction to Line Item Dispute and Annotation

Defines goals of adding disputes and annotations to eStatement Manager statement data, and compares the features of disputes with annotations.

Components of Line Item Dispute and Annotation

Provides an architectural overview of dispute and annotation features and components and outlines the task flow of configuration, composition, production, and Web application tasks. Also gives step-by-step procedures for viewing examples of National Wireless sample data in the Sample Web application, both shipped with eStatement Manager.

Configuring Dispute and Annotation Services

Defines JMS settings and database tables required for dispute and annotation.

Composition and Production for Dispute and Annotation

Provides an architectural overview of composition and production components, including Element ID, edit XSLT templates for Detail Extractor and XML Query views, how to create and configure a Detail Extractor job and publish three required XML Query dynamic Web views.

Web Application Components for Dispute and Annotation

Provides an architectural overview of Web application components, including the Manage Statement JSPs for line item detail, dispute, and annotation. Describes how to use Apache XTags with Dispute and Annotation JSPs.

Application Programming Interfaces (API) for Dispute and Annotation

Gives an overview of data flow for Annotation and Dispute Services. Defines the main classes and methods in the packages **`com.edocs.direct.dispute`** and **`com.edocs,direct.annotation`**, including the **`submit`** and **`getDocument`** methods.

Using the Dispute and Annotation APIs

Implementing Line Item Dispute and Annotation discusses the APIs needed to submit, retrieve, update, and delete disputes and annotations, including an overview of retrieving line item detail with the Content Access API **`com.edocs.user`**.

Appendix A lists Sample JSPs and National Wireless XML templates for dispute and annotation.

Auditing Data Streams

The auditing data streams chapter describes:

- How to create an audit trail to review, accept, or reject a volume of statement data before presentment
- How to query which applications are deployed on an eStatement Manager server

Introduction to Auditing Data Streams

Describes when and why to audit data streams of online statements before presentment with the Verify API.

Using the Verify API

Defines Verify methods and signatures to retrieve a list of all applications, retrieve a list of indexed volumes, retrieve a list of account numbers, retrieve account summary information, accept or reject an indexed volume, and update summary information.

Building Custom Jobs

Describes when and why to customize a Command Center job type with the Shell Command Task, and provides examples of how to script and configure your new job type.

The custom jobs chapter describes:

- How to add custom job types to the Oracle Command Center with shell commands
- How to customize and schedule Command Center jobs and tasks

Charting

The charting chapter describes:

- How to present statement data as a graphical chart in a dynamic HTML page
- How to customize the eStatement Manager charting servlet

Introduction to Charting

Gives an overview of data flow and components for formatting statement data as a graphical chart in a dynamic HTML page.

Configuring Charting for Your Server

Discusses the procedures for configuring display devices and permissions, including how to configure a "headless" server (without a dedicated display) for charting.

Composing Charts in Statements

Describes how to use the Composer to create an Application Logic File (ALF) and a chart properties file to display charts in statements, and how to simulate the appearance of your published chart at runtime with the Simulator tool.

Customizing Chart Properties

Describes how to adjust chart settings in ALF attributes and chart properties, and how to preview your customized chart.

Publishing Charts

Describes how to create and configure a Chart view in the Command Center, and how to test chart viewing in your Web application.

Designing Custom Charts with the Charting Servlet

Discusses the default charting servlet, **`charting.java`**, shipped with eStatement Manager and provides suggestions and tips for writing your own custom chart servlet and integrating it into your Web application.

Troubleshooting Charts

Lists solutions to common problems with charting.

Application Programming Interfaces (APIs) for Charting

Defines the main classes and methods in the package **`com.edocs.app.charting`**, including the chartClient and chartData, Charter, Publisher, and Simulator classes.

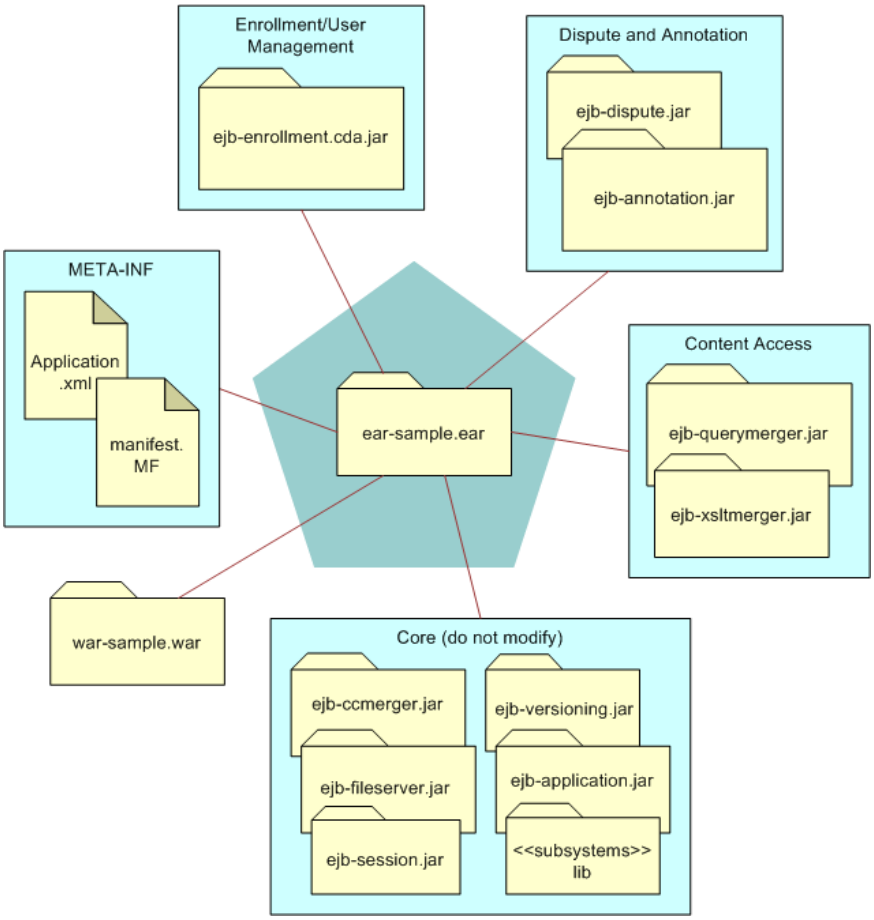
3 The Sample J2EE Application

About Sample

The Sample EAR file contains sample code components to demonstrate the features of eStatement Manager. For a list of the components in Sample, see the *Deploying and Customizing J2EE Applications Guide for Oracle Siebel eStatement Manager.*

Customizing Sample

Depending on what you plan to customize, you will work with one or more of the JAR files shown in this diagram.



eStatement Manager SDK Specification

User Management

com.edocs.app provides the interface **LoginRequired** and base servlet classes for eStatement Manager. Class **App** is the base class for all eStatement Manager application servlets. Class **AppConstant** holds string fields for request attributes. Class **AppServlet** is a servlet that extracts and dispatches the name of the APP requested. Class **LoginApp** is the base class for login application servlets, providing session-based management of **IAccount**.

com.edocs.app.enrollment provides utility classes for enrollment and default implementations of Login and Logout servlets.

com.edocs.enrollment provides a set of utility classes and exceptions to support user authentication with **IAccount**. Interface **EnrollmentConstants** supports implementation of the class **AccountUtils** in **com.edocs.enrollment.user**. Class **Encrypt** provides a method to encrypt a user ID with a password. Class **NameValue** represents an attribute as a name/value pair, manipulated by the helper class Parameters.

com.edocs.enrollment.user provides interfaces and classes for Oracle user management framework. Includes two primary interfaces. **IAccount** defines directory services for user enrollment and authentication. **IAccountResolver** provides a batch interface for retrieving attributes for a directory context, for example to return the e-mail address for an account number.

com.edocs.enrollment.user.jndi provides an implementation of the **IAccount** interface, which accesses Oracle's Common Directory Access (CDA). Includes the interface **JNDIAccountAttributes** and classes **JNDIAccount**, Login, and **JNDIAccountResolver**. Intended for advanced application development.

com.edocs.jndi.cda provides an interface and classes for implementations of Common Directory Access (CDA), the Oracle limited implementation of an LDAP-like JNDI service provider. Includes the interface **CDAConstants** and classes **CDANameParser**, **CDAFactory**, and **cdaURLContextFactory**.

com.edocs.jndi.cda.cli provides the command-line interface to CDA, Oracle's Common Directory Access. This tool provides the ability to create and manage directory schemas.

com.edocs.services.session provides the interface **ISession** and classes to obtain and return the Oracle Session object of a servlet when **com.edocs.app** calls **App.getSession**.

Content Access

com.edocs.app.user provides classes **User** and **UserMain** for access to statement summary and detail. The **User** class is the content access interface to the eStatement Manager core. Its methods retrieve and send statement data for a given user account, as well as sorting, subtotaling, and updating optional fields at presentment. Class **UserMain** implements two interfaces. The interface **com.edocs.app.LoginRequired** informs the Oracle servlet framework that the requesting client must authenticate itself before accessing **UserMain**. The interface Servlet defines basic methods that any servlet must implement. Its methods **doGet** and **doPost** support, respectively, HTTP requests for GET and POST.

Audit to Verify

com.edocs.app.verify provides the Verify class and methods for auditing indexed volumes of data before releasing them for presentment. **getIndexedVolumeList** retrieves a list of indexed volumes available for audit, while **getAccountList** retrieves all the account numbers in a volume. **getDDNList** retrieves all DDNs. Two signatures of **getHitList** retrieve all Description items either for a given volume, or for a given account. **acceptIndexedVolume** or **rejectIndexedVolume** respectively accept or reject a volume for presentment to customers. **updateDescriptionInfo** supports updates to the optional information field **(Y_#)** on a statement page.

Shell Commands for Custom Jobs

com.edocs.tasks.shellcmd provides the **ShellCmdTask** class as a task that executes an external shell command, for example to create custom Command Center jobs.

Line Item Dispute and Annotation

com.edocs.direct.annotation provides the **Annotation** class and methods **submit** and **getDocument** to allow users to create, update, or cancel an annotation to line item detail, and to retrieve annotation data from the database.

com.edocs.direct.dispute provides the **Dispute** class and methods **submit** and **getDocument** to allow users to submit a dispute to line item detail, and to retrieve disputes from the database.

Hierarchy

com.edocs.hierarchy provides an interface to define the values of constants for hierarchy fields, and utility classes for creating a hierarchy as a directory information tree (DIT).

com.edocs.hierarchy.app provides servlet classes to extend and override the example servlet classes in **com.edocs.app** for user login and content access to account data mapped to the hierarchy.

com.edocs.hierarchy.navigate provides an abstract interface and sample implementation class to filter and display content in the Hierarchy Manager.

com.edocs.hierarchy.render provides an interface and classes to design and display the user interface of the hierarchy console as HTML.

com.edocs.hierarchy.taglib provides a custom tag library of JSP tags for presenting hierarchy data.

Charting

com.edocs.app.chart Charting in eStatement Manager is achieved using third party **Kavachart's** charting utility, an Oracle wrapper to set and interpret chart properties, and by extending the eStatement Manager publisher to accept chart views.

Other

`com.edocs.common.web.validation` 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.

4 User Management

Overview

What is a User Management Framework?

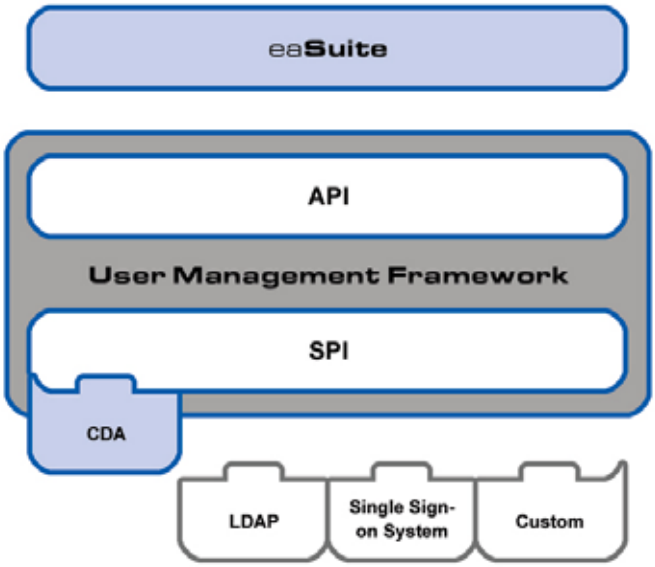
Online account management provides **users**, or customers, with securely **authenticated** access to online statements. Defining a framework for managing account and authentication information allows eStatement Manager to present online statements independently of where statement data is stored or how it is retrieved. This abstraction is the purpose of the **user management framework** (UMF).

The UMF defines accounts and authenticates login identity for customers who enroll with an On-Line Account Management system, in this case eStatement Manager.

Some of the most popular approaches to user management are based on the concept of **directory access**—programming interfaces to a flexibly structured database optimized for live retrieval. For more information on directory access, see Chapter 4, "User Management."

In deploying an Oracle solution, the user management framework integrates with the **content access framework** to retrieve and present account data for each enrolled customer. For more information on content access, see Chapter 5, Content Access.

As the foundation of each deployment, a user management framework is designed for customization. The customized code of a properly implemented project will integrate seamlessly with Oracle core software. The framework itself is not specific to any deployment, and should not be modified for any specific project.



Goals of the Oracle User Management Framework

The Oracle user management framework supports these core features:

- Add, delete, modify, and search for **user profiles**
- **Authenticate** user identity at login

A customized user management framework can also **integrate** with existing systems, such as importing or exporting directory schemas with an outside directory service such as LDAP.

Planning Your User Management Framework

This guide introduces the basic tools, concepts, and tasks for implementing a user management framework for eStatement Manager using either its Common Directory Access (CDA) interface or a simple non-directory interface. Both implementations described in this guide interact with the common Oracle UMF interface called **IAccount**.

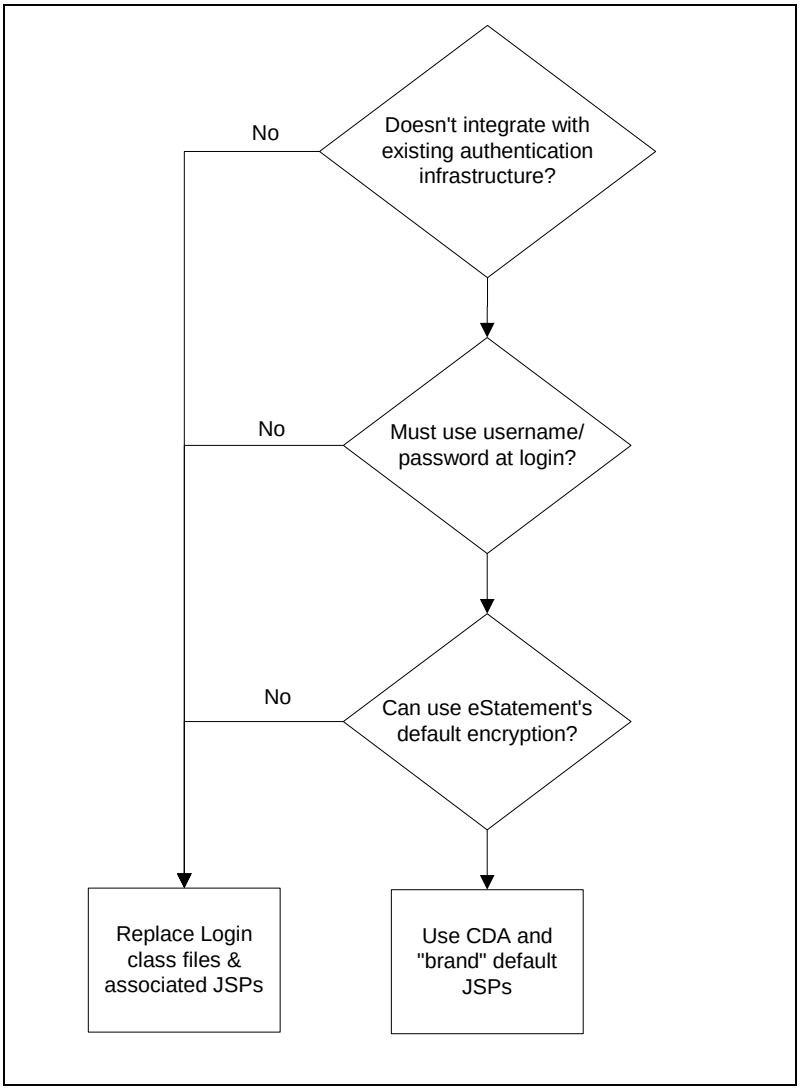
However, these implementations may not meet all of your client requirements. Fortunately, the open design of Oracle user management allows an application designer or developer to modify or even replace the UMF implementation for a deployed application without extensive recoding.

An application designer faces two main decisions when deciding to modify or replace the **IAccount** interface: **login** and **account access**. Each set of requirements may involve business logic, application design, and Web development tasks to customize the application.

Login Decisions

Login decisions include:

- Will eStatement Manager authenticate users without integrating with a separate client authentication system?
- Must the user present a user ID and password at login?
- Will eStatement Manager use its own default encryption, without integrating with client encryption systems?



If the answer to each of these login questions is YES, then the user management framework can use **IAccount** and CDA "out of the box." If the answer to any question is NO, then some or all of the login logic must be replaced. This can involve login class files, any associated JSPs, the corresponding account implementing EJBs, and any single sign-on implementations.

Access Decisions

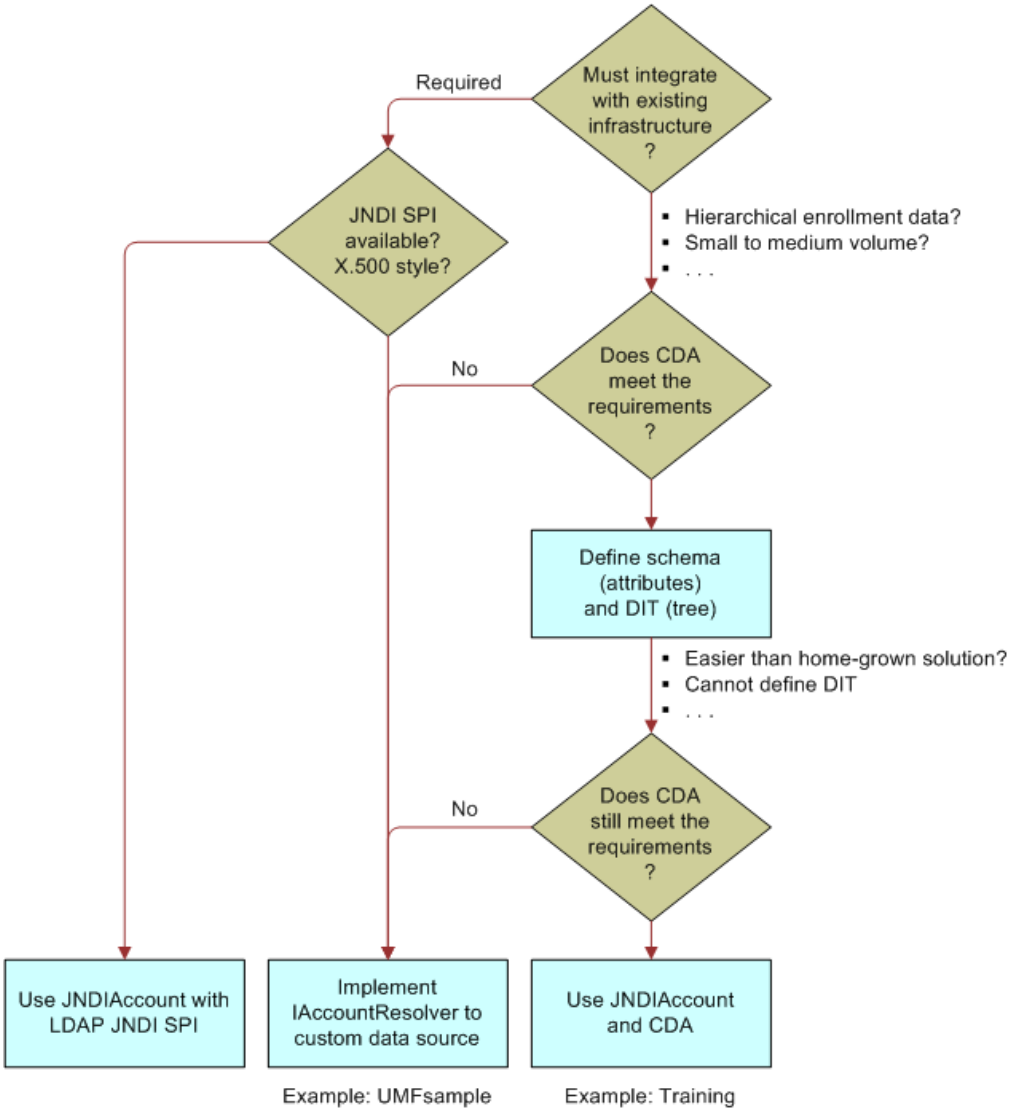
Access decisions include:

- Is account data to be stored in eStatement Manager without integrating with an existing client data source?
- Is the account data of small to medium volume and modeled in a hierarchical directory structure?
- Is there an easily definable schema and directory information tree?

If the answer to each of these access questions is YES, then the user management framework can use **IAccount** and CDA "out of the box." If the answer to any question is NO, then some or all of the access logic must be replaced, involving a JNDI service provider interface (SPI) and any associated JSPs.

Each of these decision trees may involve tasks in business logic, application design, or Web development. For example, any implementation requires business logic skills to analyze and map existing account structure to an x.500 directory schema. If the data structure is X.500 compatible, the resulting schema must then be implemented using either CDA or LDAP. If not, **`IAccount`**and possibly **`IAccountResolver`** must be implemented as interfaces to the custom data source.

Login decisions must also be analyzed, and login logic modified depending on the results. Finally, the result of all of these processes determines the extent of custom JSP code and servlet configuration for each Web application.



About the Sample J2EE Applications

Oracle provides several sample J2EE applications for deployment and as a base for customization. **Training** is based on the default eStatement Manager CDA schema. **UMFsample** is based on accessing a simple database table for enrollment information.

Generic Java Server Pages (JSPs)

The sample applications generally have the following main Java Server Pages (JSPs). These can, and should, be modified and extended to customize your implementation.

- **UserLogin.jsp** takes customer input for login fields, for example user name and password, and posts it to the **User** servlet for authentication.
- **user_subscribe.jsp** checks to see if the customer is logged in. If so, the page presents an editable set of user data, for example user name and password, to view or to change. If the user is not logged in, the page presents a login screen to create a new account.
- **user_update.jsp** is similar to **user_subscribe.jsp**, except it is used only to view or change data.
- **HistoryList.jsp** retrieves summary data through the **User** servlet to present a customer with a list of statement summaries. For more information, see Chapter 5, Content Access.
- **Detail.jsp** retrieves data through the **User** servlet to present a customer with a statement detail. For more information, see Chapter 5, Content Access.
- Servlets in the Sample Web Applications

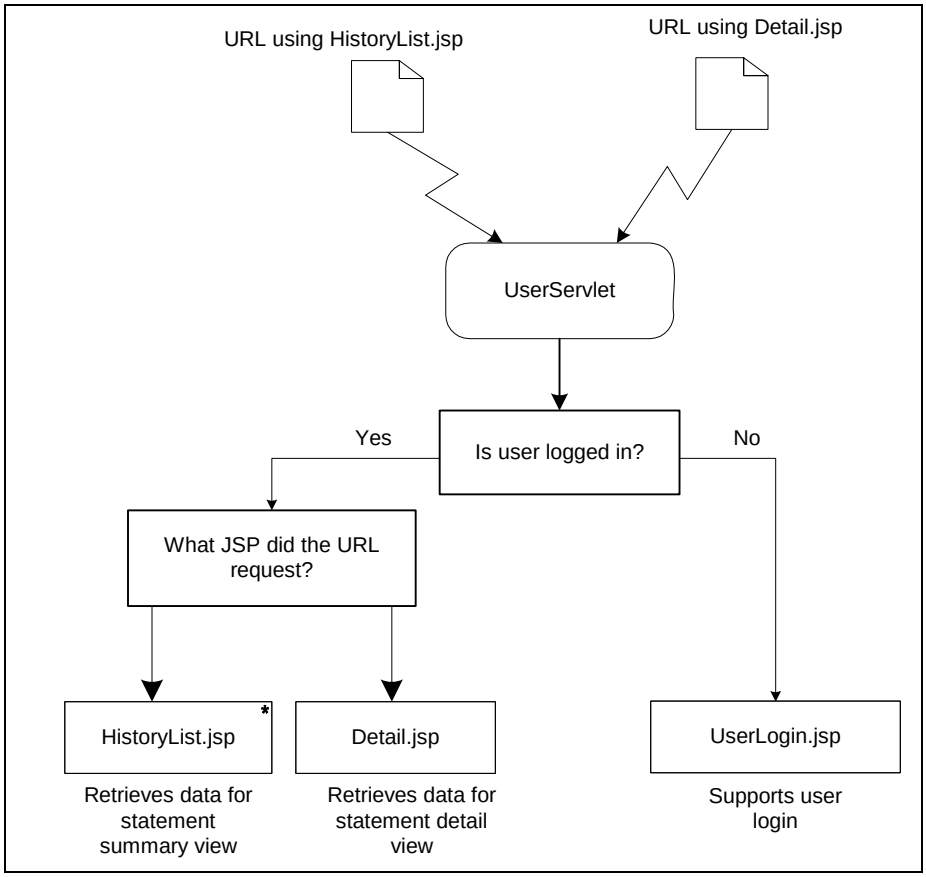
Two primary servlets support the JSPs of an eStatement Manager Web application. These servlets are deployed in the **web.xml** file on the application server.

- **UserServlet** checks to see if the customer is already logged in when requesting a statement summary (**HistoryList.jsp**) or detail (**Detail.jsp**). If so, the customer receives the requested page directly; if not, the servlet redirects the request to the login page.
- **UserEnrollmentServlet** does not check login, allowing a new customer to go directly to the enrollment page (**user_subscribe.jsp**) to sign up for online account management. This servlet also supports customer updates to their existing profiles through **user_update.jsp**.

Authenticate User Identity

In the sample Web applications, the Java Server Page **UserLogin.jsp** authenticates the identity of a user against the eStatement Manager database. The entry point for each Web application is the User servlet in the **web.xml** file, which implements the interface **com.edocs.app.LogInRequired** to present **UserLogin.jsp**.

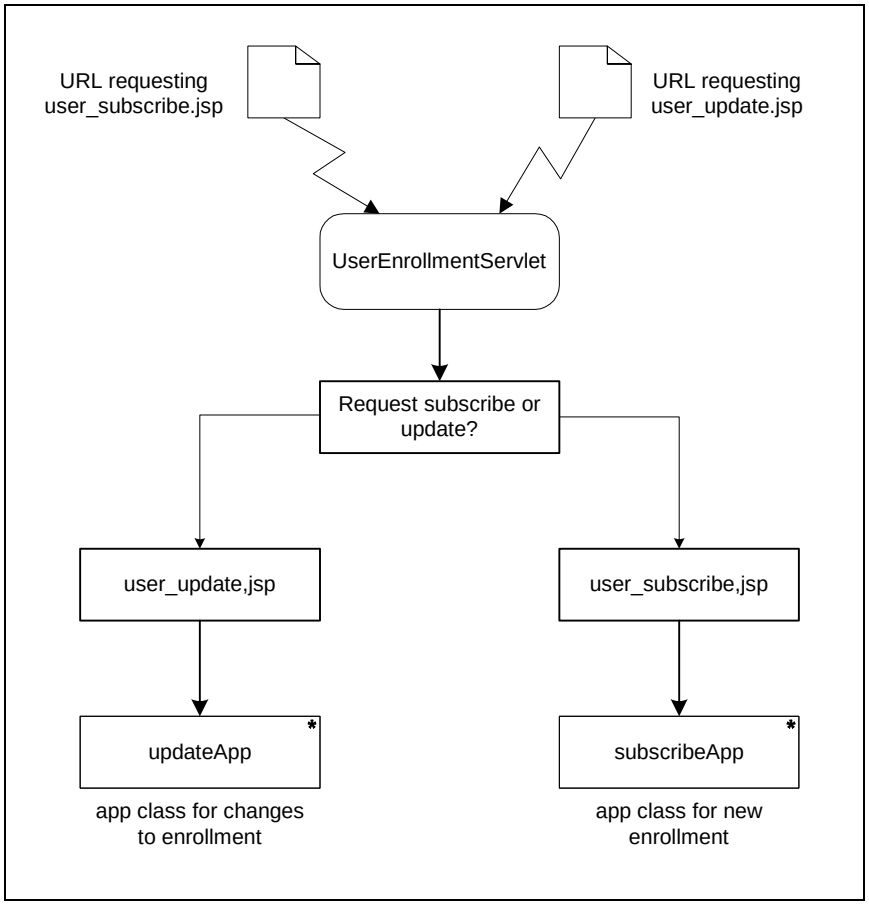
When the customer enters login data (typically user ID and password), the **LoginRoot** of the User servlet passes this information to the **Login** class for authentication against the eStatement Manager user database. If the login data matches an existing user profile, **UserLogin.jsp** creates an **IAccount** object so that an identification cookie can be placed on the user's computer for the session. If there is no match for the login data, **LogInRequired** throws an exception and **UserLogin.jsp** presents an error message.



Add and Modify User Accounts

In the sample Web applications, the Java Server Page **`user_subscribe.jsp`** can add a new user to the database. To add a new user, **`user_subscribe.jsp`** posts user input to the constructor **`com.edocs.app.enrollment.SubscribeApp`** via the **`UserEnrollmentServlet`**. To modify an existing user, **`user_update.jsp`** posts user input to the constructor **`com.edocs.app.enrollment.UpdateApp`** via the **`UserEnrollmentServlet`**.

APIs for User Management



Application Programming Interfaces (APIs) for User Management

This section provides a package description summary for each of the APIs required for user management. For more information on any package, see the Javadoc.

IAccount and IAccount Resolver Packages

Package com.edocs.enrollment.user Description

Provides interfaces and classes for Oracle enrollment and authentication in implementations of the CDA user management framework. It includes two primary interfaces. **`IAccount`** defines directory services for CDA user enrollment and authentication. **`IAccountResolver`** retrieves attributes from a given match in the same directory context, for example to return the e-mail address for an account number.

Also includes the classes **AccountBean, AccountImpl, AccountNameParser, AccountResolverBean, AccountResolverImpl, AccountUtils, IAccountFactory,** and **IAccountResolverFactory**.

JNDI Packages for Common Directory Access (CDA)

Package com.edocs.jndi.cda Description

Provides an interface and classes for implementations of Common Directory Access (CDA), the Oracle emulation of an LDAP service provider. Includes the interface **CDAConstants** and classes **CDADataSourceFactory**, **CDANameParser**, **CDAOracleFactory**, and **cdaURLContextFactory**.

Package com.edocs.jndi.cda.cli Description

Provides the command-line interface to the CDA Client, the Oracle tool for creating and managing directory schemas. Includes the interfaces **Command** and **LDIFParserConstants**, and classes **ASCII_CharStream**, **LDIFParserTokenManager**, **Main**, and **Token**. Also includes a **ParseException** and the **TokenMgrError**.

Package com.edocs.enrollment.user.jndi Description

Provides an interface and classes for JNDI implementations of the CDA interface **IAccount.** Includes the interface **JNDIAccountAttributes** and classes **JNDIAccount**, **Login**, and **JNDIAccountResolver**. Intended for advanced application development.

Other User Management Packages

Package com.edocs.app Description

Provides the interface **LoginRequired** and base servlet classes for eStatement Manager. Class **App** is the base class for all eStatement Manager application servlets. Class **AppConstant** holds string fields for request attributes. Class **AppServlet** is an HTTP servlet that extracts and dispatches the name of the **App** requested. **LoginApp** is the base class for login application servlets, providing session-based management of **IAccount**.

TIP: **IAccount** is the primary interface for implementing a user management framework in the eaSuite.

Package com.edocs.app.enrollment Description

Provides utility classes for enrollment. Includes classes **EnrollmentAppConstants, HttpRequestParameters, Login, Logout, SubscribeApp, UpdateApp.**

Package com.edocs.enrollment Description

Provides a static interface and classes to store and manipulate user enrollment data, and to support user authentication with **IAccount**. Interface **EnrollmentConstants** supports implementation of the class **AccountUtils** in **com.edocs.enrollment.user.** Class **Encrypt** provides a method to encrypt a user ID with a password. Class **NameValue** represents an attribute as a name/value pair, manipulated by the helper class **Parameters**.

The method **IAccount.authenticate** takes a parameter of a **NameValue** result, and may throw authentication errors of **AccountNotFoundException, DuplicateEnrollmentException, ExpiredAccountException, InvalidAccountException, InvalidLoginException,** and **NoSuchAccountException**.

Package com.edocs.services.session Description

Provides the interface **ISession** and classes to obtain and return the **Session** object of a servlet when **com.edocs.app** calls **App.getSession**. Also includes **NoSuchBindingException** and **SessionExpiredException**.

About IAccount

com.edocs.enrollment.user.IAccount is a public interface extending the Java public interface **javax.ejb.EJBObject**. For example, the **IAccount** interface can support Oracle enrollment and authentication using CDA as a JNDI interface to any X.500 type directory. It can also support access to non-directory enrollment models. Oracle refers to an implementation of the **IAccount** interface as an **enrollment model**, which requires:

- An implementation of **IAccountImpl** as an **IAccount** class object.
- An implementation of **IAccountResolverImpl** as an **IAccountResolver** class object.
- An implementation of a Login class to instantiate **IAccount** for a given user.

Developers working at this level need to understand the implementation of the underlying database or directory service. For example, Training uses the CDA framework that requires the Oracle JNDI implementation of **IAccount**, **JNDIAccount,** to access and modify the directory. Other customized versions of **IAccount** may dispense with most or all of the methods to handle authentication on their own, such as UMFsample which uses a separate implementation of **IAccount** called **SampleAccount** to access and modify an enrollment table.

CAUTION: Custom implementations should not modify any attribute, string, or array object passed as a parameter to any method.

The following are some of the features the IAccount methods support:

- Authenticate users
- Enroll and update users
- Manage user levels (if any are defined)
- Obtain attribute information for a user
- Search for users based on defined attributes

The IAccount methods you can use for these features are described in the sections that follow. For a complete listing of **IAccount** methods, see the Javadoc. For more details on JNDI, see "Additional Reading Sources" on page 94.

Required Imports

Any JSP using **IAccount** must import these base classes and packages:

```
java.util.*
```

```
javax.naming.*
javax.naming.directory.*
com.edocs.app.App
com.edocs.enrollment.*
com.edocs.enrollment.user.*
com.edocs.app.enrollment.EnrollmentAppConstant
```

Names as Arguments

Each name passed as an argument to an **IAccount** method is relative to that context. The empty name is used to name the context itself. The name parameter may never be null. Names are always represented as String objects, so that each client application must create and parse names to be consistent with any customized implementation.

Exceptions in IAccount

At a minimum, all **IAccount** methods throw **NamingException** or any of its subclasses or a **RemoteException**. As for JNDI, API methods only define **NamingException** in the throw clause, but when the clients can expect specific subclass exceptions thrown for certain conditions it is documented in these subclass exceptions API's method comments.

Implementations may throw **OperationNotSupportedException** in any **IAccount** APIs such that a working subset of methods remains.

CAUTION: CDA relies on database constraints when throwing exceptions. For example, **CreateSubcontext name** may throw **NamingException** instead of **javax.naming.NameAlreadyBoundException.**

Methods for Contexts

- **String composeName(String name, String prefix)**

  Composes the name of this context with a name relative to this context.

- **void rename(String oldName, String newName)**

  Renames and/or moves a context from **oldName** to **newName**.

- **String getNameInNamespace()**

  Retrieves the full name of this context within its own namespace.

- **void setContext(String name)**

  Sets the underlying context to the node identified by the name relative to the current context.

- **void switchContext(String name)**

  Sets the underlying context to the node identified by the absolute name within a current namespace.

- **void resetContext()**

  Resets the underlying context to the home context.

- **void createSubcontext(String name, Attributes attrs)**

Creates and binds a new context, along with associated attributes.

- **`void destroySubcontext(String name)`**

  Destroys the named context and removes it from the namespace.

- **`void close()`**

  Releases all resources immediately, instead of waiting for the garbage collector.

Methods for Objects and Attributes

- **`Attributes getAttributes(String name, String[] attrIds)`**

  Retrieves selected attributes associated with a named object.

- **`void modifyAttributes(String name, int mod_op, Attributes attrs)`**

  Modify specified attributes associated with a named object.

- **`void modifyAttributes(String name, ModificationItem[] mods)`**

  Modify attributes associated with a named object in the specified order.

JNDIAccountAttributes for GetAttributes

`JNDIAccountAttributes` is a public interface in the package **`com.edocs.enrollment.user.jndi.`** These attributes provide data retrieval parameters for the **`getAttributes`** methods of **`IAccount`**.

Field	Detail	Description
`JNDI_DN`	**`public static final String JNDI_DN`**	Retrieves the name value pair for the attribute named dn.
`JNDI_UID`	**`public static final String JNDI_UID`**	Retrieves the name value pair for the attribute named uid.
`JNDI_USER_PASSWORD`	**`public static final String JNDI_USER_PASSWORD`**	Retrieves the name value pair for the attribute named userPassword.

JNDI_DDN	**public static final String JNDI_DDN**	Retrieves the name value pair for the attribute named ddn.
JNDI_ACCOUNT_NUMBER	**public static final String JNDI_ACCOUNT_NUMBER**	Retrieves the name value pair for the attribute named accountNumber.
JNDI_USER_EMAIL	**public static final String JNDI_USER_EMAIL**	Retrieves the name value pair for the attribute named mail.

Methods for Searching

- **String[] list(String name)**

  Returns an array of object names bound in the named context.

- **SearchResult[] search(String name, Attributes matchingAttributes, String[] attributesToReturn)**

  Retrieves names and attributes for objects matching search criteria.

Methods for Authentication

- **void authenticate(Hashtable env)**

  Authenticates an enrollment context.

- **void reAuthenticate(Hashtable env)**

  Re-authenticates an **IAccount** object.

About IAccountResolver

IAccountResolver is the batch access interface to the enrollment information. **IAccountResolver** can retrieve enrollment attributes from a given match in the same top-level context, for example to return the e-mail address for an account number in order to send batch e-mail to users. Its **IAccount** analogue is:

```
IAccount.search(java.lang.String, javax.naming.directory.Attributes,
java.lang.String[]).
```

IAccountResolver is useful when:

- The **IAccount** interface is otherwise not needed.

- A search is not possible without fixating a schema.
- The **IAccountResolver** implementation is easier to optimize by itself.
- The **IAccountResolver** implementation must return custom objects.

This method retrieves attributes from objects matching the search criteria, for example:

```
search(Attributes matchingAttributes, String[] attributesToReturn)
```

Note: This more general method is intended for advanced application developers. Rather than use **IAccount** directly, advanced implementations can extend the **IAccountResolver** class to create their own version of **IAccount**, or extend the base adapter class **AccountResolverImpl**.

Two examples of its use are the following job types in the eStatement Manager Command Center:

- Detail Extractor
- Email Notification

For each of these jobs, you need to specify the IAccountResolver enrollment model in the job configuration. For more information about the Command Center and its jobs, see the *Administration Guide for Oracle Siebel eStatement Manager*.

When the above Email Notification job is run, a routine is eventually called that gets the Email addresses by the account numbers to send any notification emails.

Example

```
private void sendUnSentMails(MailQueueObject[]
mailQueueObject) throws Exception {
   if(emailResolver == null)
         throw new Exception("Email resolver bean is not
available");
   setup();
   InputStream is = null;
   IMerger merger = null;
   String _viewType="HTML";
   for(int i =0; i < mailQueueObject.length; i ++)
   {
         String[] emailList =
emailResolver.getEmailAddressesByAccountNumber(configProp.ddn,
         mailQueueObject[i].acctId);


         if(emailList != null && emailList.length != 0)
         {
               final String _ddn = configProp.ddn;
               final String _docid = mailQueueObject[i].docId;
```

```
                final String _ddfPath = ddfFilePath;
                final String _alfPath = alfFilePath;
                final String _hints =
   hints.urlDecode(_ddn,_docid,getHintsString(_ddn,_docid));
                final String _sessionId = new
                java.rmi.server.UID().toString();
                String[] _accounts = new String[1];
                _accounts[0] =    mailQueueObject[i].acctId;
                merger =
IMergerFactory.createByViewType(_viewType,_sessionId);
                HashMap hm = new HashMap();
                hm.put(IConstants.HINTS,_hints);
          IDistributedInputStream dis = merger.getRawDocument(
                              _ddn,
                              _accounts,
                              _docid,
                              _sessionId,
                              hm,
                              _alfPath,
                              _ddfPath);
          is = new RemoteInputStream(dis, true);
   handleResults(mailQueueObject[i], is, emailList);

   try   {
          resultq.nReqs = 1;
          handleResults ();
          }
          catch (Exception e) {
                // handle it      serious error for this batch
                // Log it
                System.out.println(" error occured at
handleResults
                      in sendUnSentMail"+ e);
                      throw e;
          }
          }
```

```
        else {
        if (DEBUG)
System.out.println("null/empty email list, continuing");
        continue;
        }
} // end for-loop
}
```

Introduction to Directory Access

About Directory Access Services

A **directory** is a special form of database, or a group of objects organized in a hierarchical framework. However, while a database is often optimized for writing, or storing, data, a directory is optimized for the **directory access service** of retrieving data quickly and flexibly. For those new to directory access, this section defines common terms and core concepts including **naming systems** and **namespaces**, **contexts**, **schemas** and **information trees**, and **distinguished names**.

Naming Systems, Naming Services, and Namespaces

A **naming system** is a connected set of contexts of the same type (they have the same naming convention) and provides a common set of operations. For example, a system that communicates using the LDAP is a naming system.

A naming system provides a **naming service** to its customers for naming-related operations. A naming service is accessed through its own interface. For example, the LDAP offers a naming service that maps LDAP names to LDAP entries. A file system offers a naming service that maps filenames to files and directories.

A **namespace** is the set of names in a naming system. For example, the UNIX file system has a namespace consisting of all of the names of files and directories in that file system. The LDAP namespace contains names of LDAP entries. A directory information tree (DIT) is one example of a namespace.

Directory Contexts

A **context** is a set of name-to-object bindings. Every context has an associated naming convention. A context provides a lookup (resolution) operation that returns the object and may provide operations such as those for binding names, unbinding names, and listing bound names. A name in one context object can be bound to another context object (called a **subcontext**) that has the same naming convention.

For example, a file directory, such as **`/usr`**, in the UNIX file system is a context. A file directory named relative to another file directory is a subcontext (some UNIX users refer to this as a subdirectory). That is, in a file directory **`/usr/bin`**, the directory **`bin`** is a subcontext of **`usr`**. In another example, an LDAP entry, such as **`c=us`**, is a context. An LDAP entry named relative to another

LDAP entry is a subcontext. For example, in the LDAP entry **o=sun, c=us,** the entry **o=sun** is a subcontext of **c=us**.

CAUTION: Unlike LDAP, CDA does not support transactional contexts (a transaction updates two directories, such as a payroll transfer involving a withdrawal from one data source and a deposit to another). If an update operation fails, the content of the directory is unknown, although the tree will remain stable and useable.

Directory Information Tree (DIT)

Directories typically arrange their objects in a hierarchy. For example, LDAP arranges all directory objects in a tree, called a **directory information tree** (DIT). Within the DIT, an organization object, for example, might contain group objects that might in turn contain person objects. When directory objects are arranged in this way, they serve as naming contexts as well as containers of attributes.

Directory Entries, Objects, and Attributes

A directory stores information as **entries**. Each entry is a named **object** containing one or more **attributes**. Each attribute is a **name/value pair** of the syntax **<name>=<value>,** where the name is a unique object identifier and the value has a defined syntax.

Directory Schema

A directory **schema** defines the rules for **distinguished names**, and for what attributes a directory entry must or may not contain. A schema defines **object classes** of mandatory and optional **attributes**, and every entry in the directory has an associated object class.

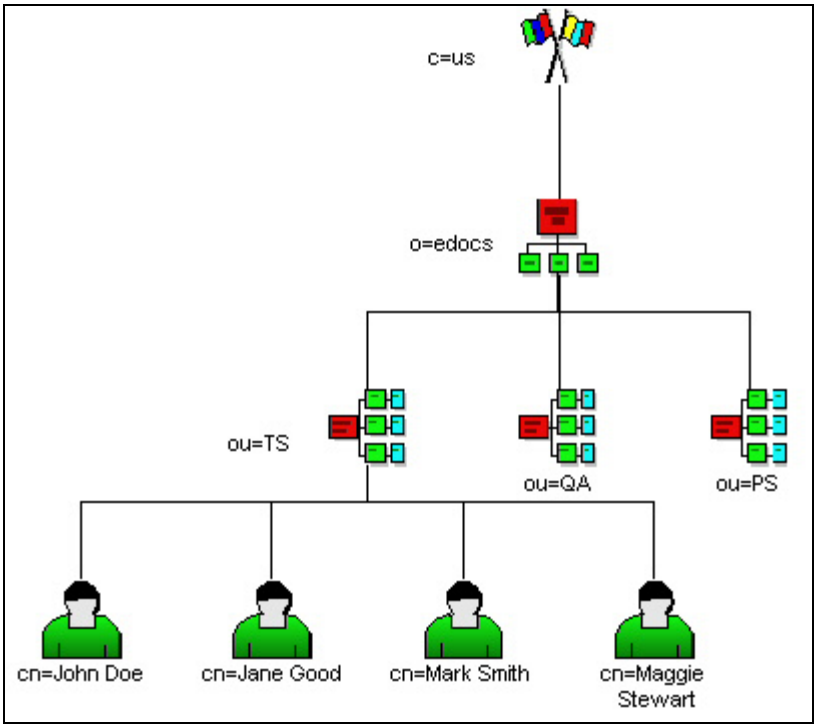
TIP: CDA does not enforce a directory schema as LDAP does. However, treating CDA like an LDAP service minimizes code changes when migrating to LDAP. Oracle strongly recommends modeling a CDA schema on LDAP and adhering to it.

Distinguished Names

A directory information tree organizes entries by **distinguished name** (DN). A distinguished name concatenates attributes in a unique path from the named entry up to the root of the tree, separating each attribute with a comma. For example, the distinguished name

```
cn=John Doe, ou=TS, o=edocs, c=us
```

implies the directory tree illustrated below.



CAUTION: Distinguished names whose attributes contain a comma must be enclosed in either single (') or double (") quotes**.**

Entries whose distinguished name contains the distinguished name of another entry as a suffix are considered subentries in the hierarchy, making the namespace **hierarchical**.

Choosing a Directory Access Interface

Common directory access interfaces include the Microsoft Active Directory Service Interfaces (ADSI) and the **Lightweight Directory Access Protocol** (LDAP). Oracle has developed its own directory access interface, **Common Directory Access** (CDA), included with the eaSuite. All Oracle applications use the **Java Naming and Directory Interface**™ (JNDI) to connect with directory access. This section discusses JNDI, LDAP, and CDA and compares features to consider when choosing a directory protocol.

The Java Naming and Directory Interface™

The **Java Naming and Directory Interface**™ (JNDI) is an industry standard extension to the Java™ platform, providing Java-enabled applications with a powerful and portable interface to enterprise naming and directory services. As part of the Java Enterprise API set, JNDI can seamlessly connect many and varied directory services. For more information about JNDI, see "Additional Reading Sources" on page 94.

Lightweight Directory Access Protocol (LDAP)

The **Lightweight Directory Access Protocol** (LDAP) provides directory access to networked databases. LDAP support is being implemented in Web browsers and e-mail programs that can query an LDAP-compliant directory. LDAP is expected to provide a common method for searching e-mail

addresses on the Internet, eventually leading to global white pages. LDAP is a sibling protocol to HTTP and FTP and uses the **`ldap://`** prefix in its URL.

Based on the X.500 directory access model defined in 1988, LDAP improves performance by running over TCP/IP or other "out-of-the-box" network transport; simplifying queries and other directory operations; and encoding elements more efficiently to reduce code size and complexity. For more information about LDAP, see "Additional Reading Sources" on page 94.

LDAP integrates with any existing infrastructure based on JNDI. To implement an LDAP solution for the Oracle eaSuite, this SDK provides the APIs **`IAccount`** and **`JNDIAccountAttributes`**.

Common Directory Access (CDA) Interface

The **Common Directory Access** (CDA) interface was developed by Oracle as a self-contained subset of LDAP that ships with eStatement Manager. Since LDAP servers are third-party solutions requiring high cost and maintenance, CDA supports Oracle directory access for customers without an existing directory access interface in place, or for deployments not requiring the advanced features available in LDAP.

CDA allows developers to:

- Define a directory schema and attributes
- Create a hierarchical user management framework of small to medium volume
- Integrate customized Java Server Pages (JSPs) with your schema

Like LDAP, CDA integrates with any existing infrastructure based on JNDI. To implement a CDA solution for the Oracle eaSuite, this SDK provides the APIs **`JNDIAccount`** and **`JNDIAccountAttributes`**. For more information, see "Using the Common Directory Access (CDA) Framework" on page 43.

Comparing LDAP and CDA Features

Feature	LDAP	CDA	CDA Notes
Add-on cost	Yes	No	Ships with eStatement Manager
Aliases	Yes	No	Attributes can store `DirContext`
APIs for read/write access	Yes	Yes	
Attribute syntax	Yes	Limited	
Character encoding of names	Yes	No	
Directory information tree with country at top, users at bottom	Yes	Yes	
Distinguished names	Yes	Yes	
Enforced access control	Yes	No	
Enforced schema	Yes	No	
Entries as objects	Yes	No	
Hierarchical enrollment model	Yes	Yes	
LDAP extensions and referrals to JNDI	Yes	No	
Map to hierarchical namespace	Yes	Yes	
Native batch uploads (LDIF)	Yes	Yes	
Replication	Yes	No	
Search Filters	Yes	No	Implements searches in `DirContext`
Standalone directory protocol	Yes	No	
Transactional context	Yes	No	Ensure referential integrity
User authentication	Yes	No	

Using the Common Directory Access (CDA) Framework

What is CDA?

One part of the Oracle user management framework uses the **Common Directory Access** (CDA) interface to emulate the core features of an LDAP service provider. CDA implements the JNDI public

interface **DirContext** to map a hierarchical **namespace** onto a directory. **DirContext** contains methods for examining and updating attributes associated with objects, and for searching the directory. For more information about **DirContext**, see the JNDI documentation provided by Sun at http://java.sun.com .

Common Directory Access (CDA) supports hierarchical enrollment schemas that nest users in subaccounts. The **IAccount** API emulates the JNDI interface **DirContext** to afford the flexibility and power of JNDI and LDAP. The **IAccount** API replaces JNDI methods that return instances of **DirContext** with similar methods that either return context names or reset context state inside the account object.

The Default CDA Schema and DIT

The eStatement Manager default CDA schema extends the well-defined LDAP schema, in which each object is a set of attributes. The DIT developer determines which attribute names the object, whether objects may include themselves, and which attributes are required. Basic objects and their rules include the following:

- Countries **(c)** may contain Organizations.
- Organizations **(o)** may contain Organization Units.
- Organization Units **(ou)** may nest three levels deep, and may not contain an Organization.

For more information on LDAP DIT rules, see "Additional Reading Sources" on page 94.

The CDA model supports attributes of the value types **String** and **DirContext.** CDA attribute and value names are encoded to the ISO-8859-1 data standard only. Attribute names are limited to 255 characters and values are limited to 1024 characters, encoded according to the schema.

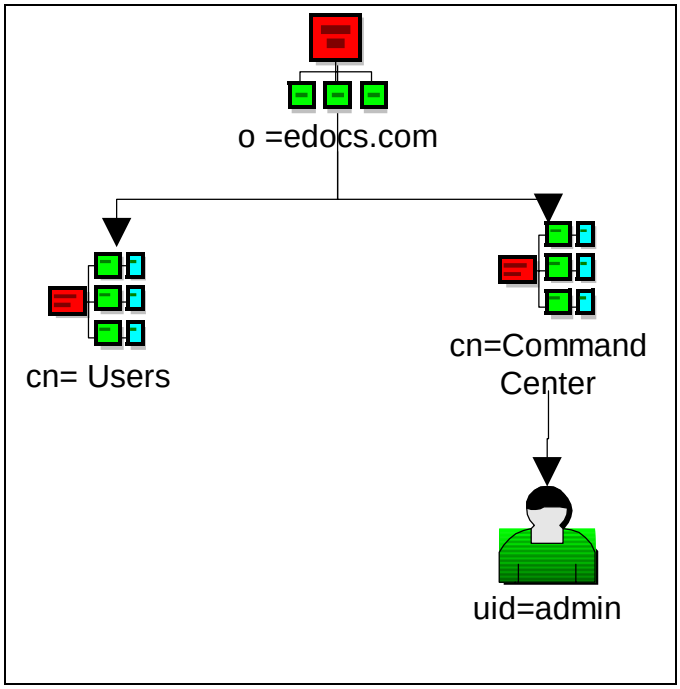
The Oracle default CDA schema is based on LDAP attributes defined by the RFC2256 standard, though Oracle has added three attributes specific to eaSuite:

- **accountNumber**
- **DDN**
- **status**

For more information on default schema attributes, see the Javadoc.

Your schema will probably require additional attribute names, which you must add, or **bind,** to the schema before giving a value to an attribute name. For more information about customizing schemas, see "Using the CDA Client" on page 58.

When you first install eStatement Manager, it creates a default CDA DIT to handle the enrollment of Admin users to the Command Center, and it provides a subcontext (**cn=Users**) to handle the enrollment of users for the Training application. Initially, the entities under this node do not exist until their specific enrollment using the **user_subscribe.jsp**. This process is described in more detail later in this chapter.



Using Training as a Template

eStatement Manager provides several sample J2EE applications for you to deploy that use different enrollment models. One of these, Training, you can use as a template to begin your own custom J2EE application using the CDA user management framework with hierarchical levels of enrollment.

In addition to the JSPs described in Chapter 2: *About User Management*, Training provides the following JSPs:

- **`ManageUsers.jsp`** provides links to add new users or add direct reports in the hierarchical model.
- **`AddDirectReports.jsp`** provides search capabilities and can be used as a tool to build hierarchical enrollment schemas.

How the JSPs interact with the CDA user management framework is described in the next major section. By learning how they work should give you the understanding of how to create your own custom CDA implementations. However, before you can use the Training JSPs to see how the enrollment works, you need to deploy and configure Training and then run it through the National Wireless sample application files.

Deploying and Configuring Training

Before you can use Training, you need to successfully install and configure eStatement Manager using the instructions in the *Installation Guide for Oracle Siebel eStatement Manager* for your operating system and application server.

The deployment instructions in that guide for the **`Sample.ear`** should work equally well for the **`Training.ear`** application. You can find **`Training.ear`** along with the other sample EAR files in the **`Samples`** directory where you installed eStatement Manager.

You also need to run the **create_training_schema** script provided by eStatement Manager in the directory **EDX_HOME/samples/Training/J2EEApps/weblogic/** (or **/websphere**), where EDX_HOME is the location where you installed eStatement Manager. The **create_training_schema** file contains the CDA Client Tool commands to modify the default CDA schema to run Training. It adds the following attributes that are used to denote a hierarchy between users at the same context level:

- **role**: defines a user role such as "supervisor"
- **supervisor**: defines the DN (distinguished name in CDA) for a user's supervisor

Configuring the CDA Client

UNIX users need to set environment variables in the **create_training_schema** script. Both UNIX and Windows users then pass the script to the CDA Client Tool, using the correct JDBC and database values for your platform.

TIP: The **java** command examples provided in this section and elsewhere in this guide presume the location of your systems JDK *bin* directory is included in its PATH setting. If not, you should explicitly specify the java command, for example **\bea\jdk141_05\bin\java**.

To configure CDA on UNIX:

1. Go to the location where the *config_training_tool* script and *create_training_schema* file reside on your system (by default, EDX_HOME/samples/Training/schema).
2. Execute the *config_training_tool*:

   ./config_training_tool

   This script prompts you for additional parameters (such as the DB port).

   config_training_tool takes the *create_training_schema* file as a parameter inside the script. It also takes all the environment-specific parameters from *EDX_HOME/config/edx_env*.

To configure CDA on Windows:

1. Go to the location where the *create_training_schema* script and *config_training_tool.bat* file reside on your system (by default, EDX_HOME/samples/Training/schema).
2. Open an MS-DOS Command Prompt window.
3. Edit the *config_training_tool.bat* with the appropriate parameters. You must define the environment variable necessary for your database:

   For Microsoft SQL Server:

   @set MSSQL_SVR_IP= <DB IP address>
   @set MSSQL_SVR_PORT= <DB port> default is 1433
   @set MSSQL_DB_USER =<DB user name>
   @set MSSQL_DB__PASSWD= <DB password>

   For Oracle:

   @set ORACLE_SVR_IP=<DB IP address>
   @set ORACLE_SVR_PORT=<DB port> default is 1521
   @set ORACLE_DB_ALIAS=<SID / alias>

```
@set ORACLE_DB_USER=<DB user name>
@set ORACLE_DB__PASSWD=<DB password>
@set ORACLE_HOME=<Oracle home>
```

4 Execute *config_training_tool.bat*:

config_training_tool.bat

config_training_tool.bat takes all additional environment-specific parameters from *EDX_HOME/config/edx_env.bat*.

Using Training with National Wireless

After deploying and configuring Training, you can use it with the National Wireless sample application input files provided with eStatement Manager. The eStatement Manager guides provide information about how to process the National Wireless files using Training as the DDN.

However, for users to view their statements, you must enroll them into the CDA tables using the Training JSPs as follows:

1 Enter the following URL in your browser:

```
http://your-server:port/Training/user/jsp/ index.jsp
```

Where **your-server** and **port** are the values you defined for eStatement Manager when you installed and configured it.

2 Scroll down to the bottom of the screen where the Enroll User button is and click on it. Do not specify a DDN.

3 Add the following super user to the CDA database:

User: super

Password: oracle

User Level: Supervisor

Email: super@oracle.com

Primary Account: 0331734

Secondary Account: 0331734

4 Exit the application.

5 Enter the following URL:

```
http://your-server:port/Training/User?app=UserMain
&jsp=/user/jsp/HistoryList.jsp&ddn=NatlWireless
```

6 Log in as User: **super** and Password: **oracle**.

7 Click on **Manage Users** and then on **Enroll User**.

8 Enroll the following users:

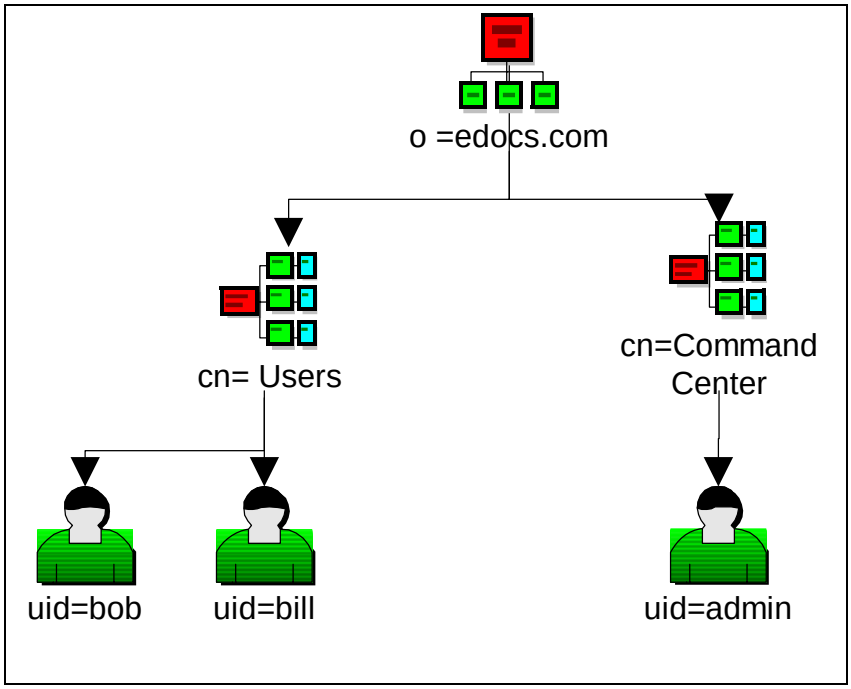
User	Password	User Level	Email	Primary Account	Secondary Account
0331734	oracle	User	<Name>@oracle.com	0331734	0331734
4191463	oracle	User	<Name>@oracle.com	4191463	4191463
8611250	oracle	User	<Name>@oracle.com	8611250	8611250
9001203	oracle	User	<Name>@oracle.com	9001203	9001203
0407200	oracle	User	<Name>@oracle.com	0407200	0407200
3069725	oracle	User	<Name>@oracle.com	3069725	3069725
4694878	oracle	User	<Name>@oracle.com	4694878	4694878
1710123	oracle	User	<Name>@oracle.com	1710123	1710123
9424090	oracle	User	<Name>@oracle.com	9424090	9424090

How this enrollment works is described in the next section.

How does Training Use CDA?

As mentioned earlier in this chapter, eStatement Manager provides a default CDA schema to be used by Training for enrollment purposes. This structure is created upon installation and configuration of eStatement Manager, and its primary purpose is to handle the enrollment of Admin users for the Command Center. You can create your own enrollment DIT using the instructions provided in Using the CDA Client.

However, creating a DIT using the CDA Client Tool provides a "static" environment to add contexts to an enrollment system. What is required is a mechanism to invoke **`IAccount`** to enroll users dynamically through JSPs and EJBs.



eStatement Manager provides the **SubscribeApp** and **UpdateApp** class files along with the **ejb-enrollment-cda.jar** EJB to handle this process. Through the sample EARs, it also provides servlets and JSPs to interact with those class files and EJB; Training is just one implementation that uses these files to support the enrollment of the National Wireless application and others.

It is expected that when you create your own J2EE application, you need to create custom JSPs and Servlets. Also, it is likely that you will need to modify the **ejb-enrollment-cda.jar** to reflect a new context root directory for any new DIT structure you define.

The **SubscribeApp** and **UpdateApp** files are not to be modified when using CDA, as they are flexible enough to handle any DIT structure through the use of schema attributes. For example, Training requires the use of two additional attributes to handle the use of User Roles in the **ManageUsers,jsp** and **AddDirectReports.jsp** files. That is why it is required that you run the **create_training_schema** file to modify the default CDA schema before using Training.

It is recommended that you look at the contents of the Training EAR (specifically the WAR file) as the following sections describe how they work. You can extract the contents using the **jar** command (UNIX) or WinZip utility (Windows). The *Deploying and Customizing J2EE Applications Guide for Oracle Siebel eStatement Manager* provides several examples to do this, along with an explanation of the basic elements inside the EAR.

Training CDA Process Flow

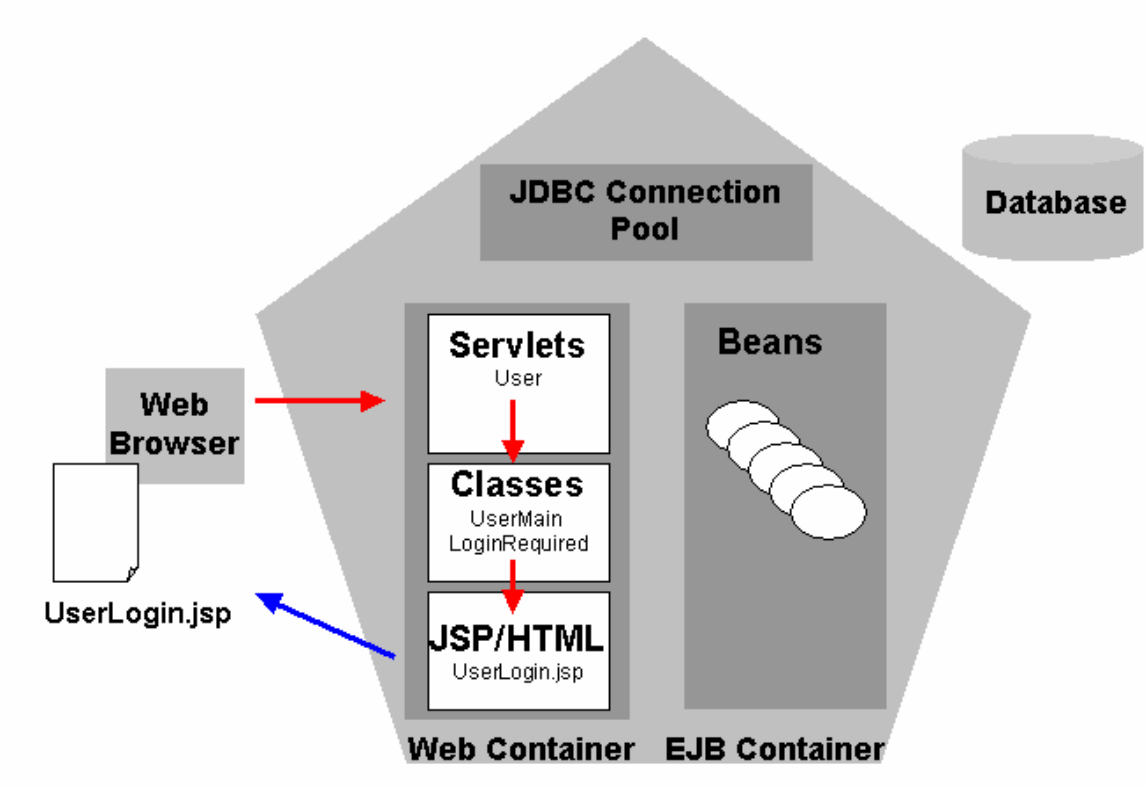
The enrollment process for Training can be broken down into two parts:

- Authenticate the user: Determine if the user is already logged in, and if not, ask the user to log in. After the user provides the log in information, validate that information against the enrollment information stored in the CDA database. If it is valid, assign an IAccount instance (stored in a cookie) to the user and proceed to the requested page. If it is not valid, return an error to the user.

- Enroll the user: If they have the correct permissions, provide the user with a subscription page to enter as their log in information and stored in the CDA database. The next time they attempt to access Training, they will be authenticated against this enrollment information. Training also provides a mechanism to update enrollment information.

Training checks the authentication of a user prior to showing them any page on their browser. As long as the IAccount instance is still valid, the user can proceed as usual. The moment the instance becomes invalid (such as through a session timeout), the user needs to log back in before proceeding.

The following diagram shows how the process works for the first part of authentication:

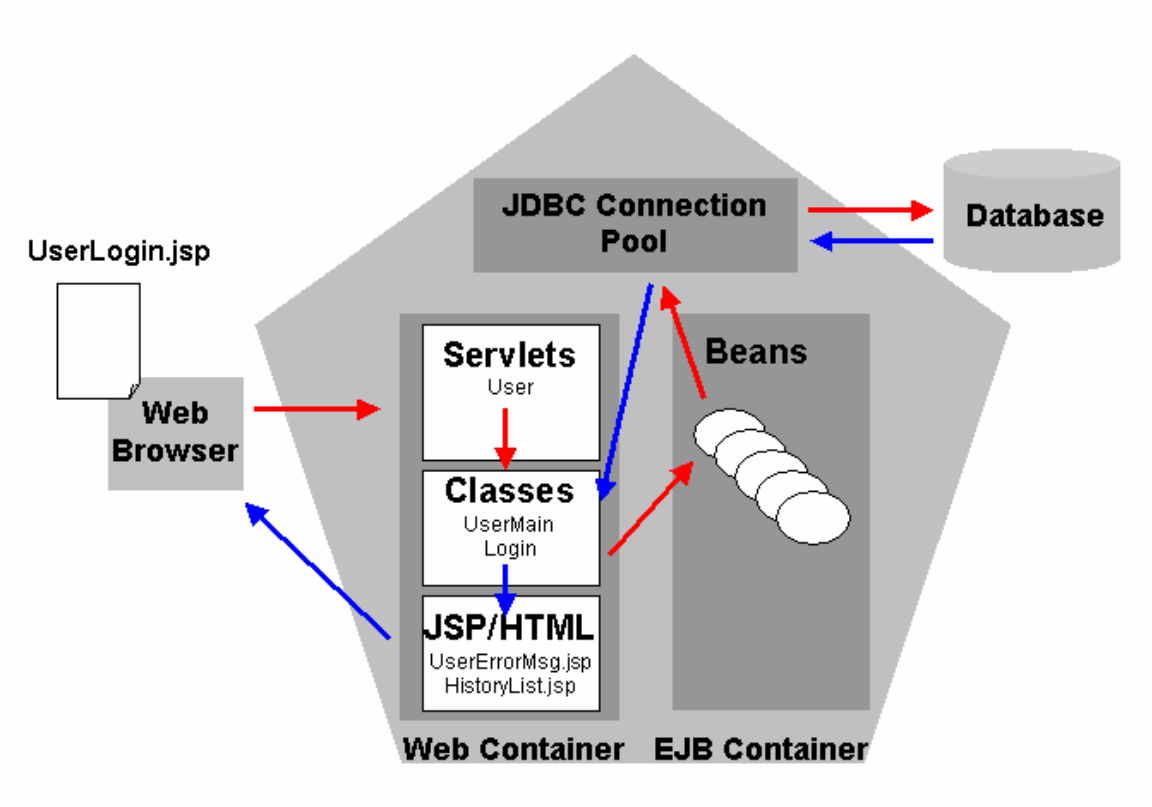


The user through the specified URL requests the User servlet and **`UserMain`** class file that determines whether the user is already logged on. The following example URL (used previously in this chapter) shows this request:

```
http://your-server:port/Training/User?app=UserMain
&jsp=/training/jsp/HistoryList.jsp&ddn=NatlWireless
```

Built into **`UserMain`** is the **`LoginRequired`** class file that indicates the user is not logged in and will be given the **`UserLogin.jsp`** to do so. In the contents of the Training's WAR file, you can find the User servlet definition in the **`WEB-INF/web.xml`** file, and the **`UserLogin.jsp`** in the **`training/jsp`** directory. **`UserMain.class`** resides in **`WEB-INF/classes/com/edocs/app/user`** directory.

The next diagram shows what happens after the user submits the log in information through **`UserLogin.jsp`**:

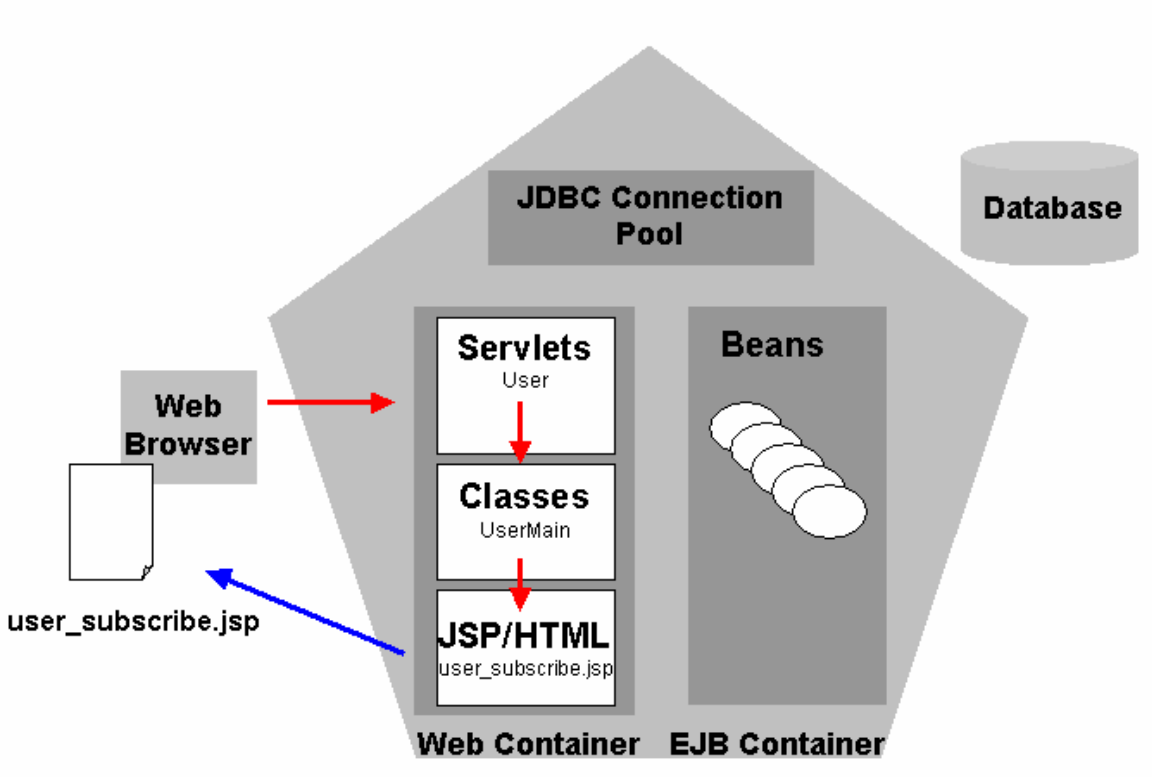


Basically, the User servlet and **`UserMain`** class file invokes the **`Login`** class file to validate the log in information against the CDA database through a JDBC connection pool (established during installation and configuration of eStatement Manager). **`Login.class`** resides in **`WEB-INF/classes/com/edocs/app/enrollment`** directory.

`TIP`: When using CDA, the **`UserMain.class`** and **`Login.class`** files do not need to be modified.

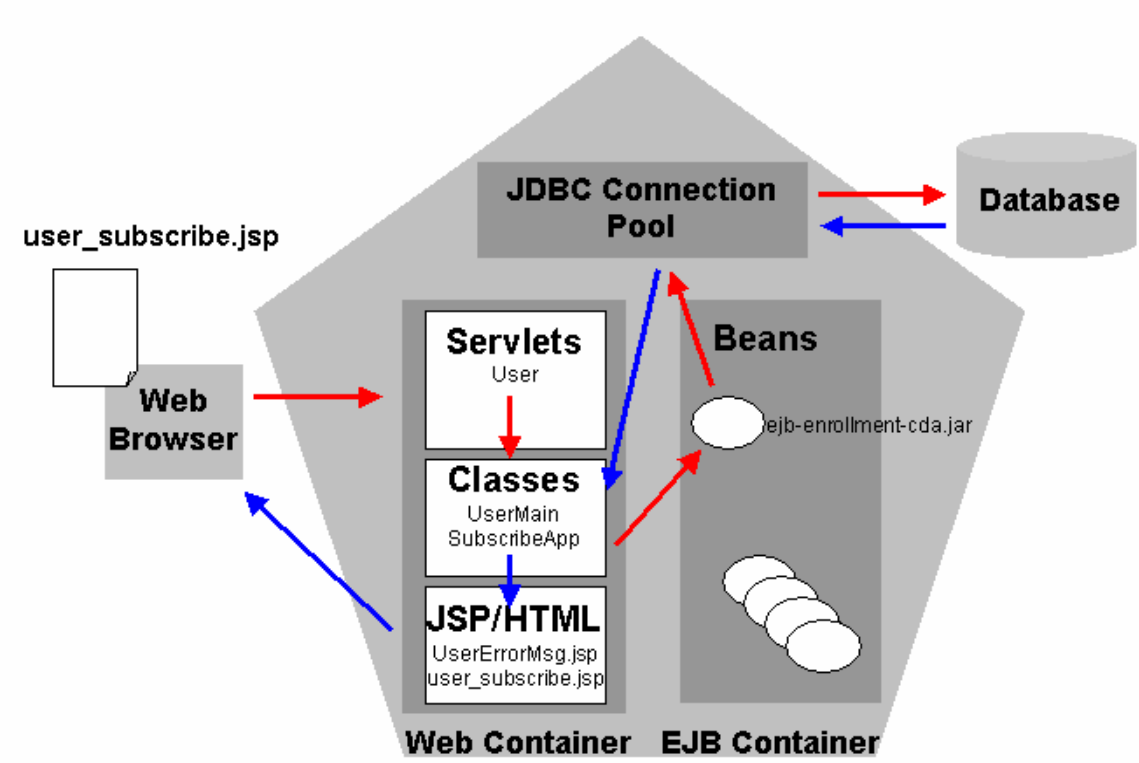
If the information is valid, it returns the **`HistoryList.jsp`** (or other file that was requested by the original URL request) to the user's browser. Otherwise, it returns **`UserErrorMsg.jsp`** with any error message generated by eStatement Manager.

The next two diagrams shows how the enrollment works in Training:



Training handles the enrollment of new users through the **`ManageUsers.jsp`** page that is only accessible by someone that is already enrolled and has the correct User Level permission (such as the Supervisor role). This is the reason in the previous example that the step to create a Supervisor user (Step 3) was done before adding any of the other users.

When you click on the **`Enroll User`** button, eStatement Manager returns the **`user_subscribe.jsp`** to the user to enter the enrollment information. After the user submits the information, the following diagram shows how the information is processed:



The information of the new user is posted through the User servlet and **`UserMain`** class file to the **`SubscribeApp`** class file. It in turn invokes the **`JNDIAccount`** implementation of CDA (called **`CDAAccount`** in the EJB descriptor file) from the **`ejb-enrollment-cda.jar`** EJB to add the enrollment information to the CDA tables in the database. If the enrollment is successful, eStatement Manager returns the **`user_subscribe.jsp`** to the user in order to enroll another person. Otherwise, it returns the error information through **`UserErrorMsg.jsp`**.

In addition to **`user_subscribe.jsp`**, Training provides the **`user_update.jsp`** that goes through the same process described above to update the enrollment information for existing users, except it uses the **`UpdateApp`** class file.

You can find the **`SubscribeApp.class`** and **`UpdateApp.class`** files in the **`WEB-INF/classes/com/edocs/app/enrollment`** directory of the Training WAR file. However, like the **`UserMain`** class file, these two files should not be modified when using CDA. In contrast, the **`user_subscribe.jsp`** and **`user_update.jsp`** should be part of any customization, and you should use the Training ones as template examples. You can find them at **`training/jsp`**.

Modifying the CDA EJB for Your Application

The **`ejb-enrollment-cda.jar`** EJB descriptor files will probably need to change when your application implements your own DIT. See "Using the CDA Client" on page 58 for an example of creating a new DIT. To modify these descriptor files, you need to unjar or unzip the **`ejb-enrollment-cda.jar`** file into a temporary directory. For examples about extracting the contents of a JAR file, see the *Deploying and Customizing J2EE Applications Guide for Oracle Siebel eStatement Manager*.

The two descriptor files you modify are:

- **`ejb-jar.xml`**

- **`weblogic-ejb-jar.xml`** (WebLogic) or **`ibm-ejb-jar-bnd.xmi`** (WebSphere)

The first descriptor file (**`ejb-jar.xml`**) is common across J2EE application servers and contains the enterprise beans Session descriptors for the CDA enrollment model. Specifically, it defines the **`CDAAccount`** EJB which is an implementation of **`JNDIAccount`**, and it defines the **`CDAAccountResolver`** EJB which is an implementation of **`JNDIAccountResolver`**. Both **`CDAAccount`** and **`CDAAccountResolver`** are configured specifically for Training.

For your own custom DIT, you should modify these Session descriptors to specify the new context root value of it. The following code sample highlights the portion of **`CDAAccount`** that you change. If your DIT contains many context levels in its enrollment hierarchy, you need to begin the entry description with the lowest directory entry, followed by the next highest directory context, and so forth until you reach the highest one (usually **`o=something.com`**). Remember that the lowest directory entry is where you expect **`SubscribeApp`** to dynamically add the entries of users you want to enroll.

```
<session id="Session_1">
<description>CDA enrollment model account</description>
<display-name>CDAAccount</display-name>
<ejb-name>CDAAccount</ejb-name>
<home>com.edocs.enrollment.user.IAccountHome</home>
<remote>com.edocs.enrollment.user.IAccount</remote>
<ejb-class>com.edocs.enrollment.user.AccountBean</ejb-class>
<session-type>Stateful</session-type>
<transaction-type>Bean</transaction-type>
<env-entry id="EnvEntry_CDAAccount_1">
<env-entry-name>accountImpl</env-entry-name>
<env-entry-type>java.lang.String</env-entry-type>
<env-entry-value>com.edocs.enrollment.user.jndi.JNDIAccount</env-entry-
value>
</env-entry>
<env-entry id="EnvEntry_CDAAccount_2">
<description>enrollment context root</description>
<env-entry-name>contextRoot</env-entry-name>
<env-entry-type>java.lang.String</env-entry-type>
<env-entry-value>cn=Users,o=edocs.com</env-entry-value>
</env-entry>
```

```
<env-entry id="EnvEntry_CDAAccount_3">
<description>javax.naming.Context.INITIAL_CONTEXT_FACTORY</description>
<env-entry-name>java_naming_factory_initial</env-entry-name>
<env-entry-type>java.lang.String</env-entry-type>
<env-entry-value>com.edocs.jndi.cda.CDADataSourceFactory</env-entry-
value>
</env-entry>
<env-entry id="EnvEntry_CDAAccount_4">
<description>javax.naming.Context.PROVIDER_URL</description>
<env-entry-name>java_naming_provider_url</env-entry-name>
<env-entry-type>java.lang.String</env-entry-type>
<env-entry-value>java:comp/env/jdbc/DataSource</env-entry-value>
</env-entry>
<resource-ref id="ResRef_CDAAccount_1">
<res-ref-name>jdbc/DataSource</res-ref-name>
<res-type>javax.sql.DataSource</res-type>
<res-auth>Container</res-auth>
</resource-ref>
</session>
```

You also need to modify **`CDAAccountResolver`** in the same way.

The second descriptor file (**`weblogic-ejb-jar.xml`** for WebLogic or **`ibm-ejb-jar-bnd.xmi`** for WebSphere) contains reference descriptors for **`CDAAccount`** and **`CDAAcountResolver`** that mention Training. For example:

```
<ejb-name>CDAAccount</ejb-name>
<stateful-session-descriptor>
<stateful-session-cache>
<max-beans-in-cache>500</max-beans-in-cache>
<idle-timeout-seconds>900</idle-timeout-seconds>
</stateful-session-cache>
<stateful-session-clustering>
<home-is-clusterable>true</home-is-clusterable>
<replication-type>None</replication-type>
</stateful-session-clustering>
</stateful-session-descriptor>


<reference-descriptor>
<resource-description>
<res-ref-name>jdbc/DataSource</res-ref-name>
<jndi-name>edx.databasePool</jndi-name>
</resource-description>
</reference-descriptor>
<jndi-name>edx/Training/ejb/CDAAccount</jndi-name>
```

You need to change these entries to your new J2EE application name. Normally this is done as part of the configuration and deployment of your custom application as described in the *Deploying and Customizing J2EE Applications Guide for Oracle Siebel eStatement Manager*. In addition, after making the changes to the above descriptor files, you must re-jar or re-zip the files you extracted back into a new **`ejb-enrollment-cda.jar`** file and replace the previous one in your EAR.

Using IAccount with CDA for Other Tasks

In addition to authenticating, enrolling, and updating users in the CDA database, the CDA implementation of IAccount also provides directory-like services to perform the following tasks:

- Obtain attributes (name/value pairs) for a user context
- Manage user levels within a DIT structure
- Search for user contexts that match an attribute criteria

Each of these tasks can be implemented at the JSP layer of your application, and the following sections provide code examples of how to do it.

Obtain Attributes and their Values

With **`IAccount`**, you can return the attributes in name/value pairs using the method **`getAttributes`**. For example, the following code returns the **`rights`** for the current user in a name/value pair:

```
account = (IAccount)request.getAttribute
("com.edocs.enrollment.user.IAccount");

Attributes attrs = account.getAttributes("", new String[] {
"rights", JNDIAccountAttributes.JNDI_USER_EMAIL
});
```

You can also return the name or value only using the method **`unescapeAttributeValue`** from the Class **`CDANameParser`**. For example, continuing from the above code, you can obtain the individual attributes as follows:

```
Attribute a = (Attribute)attrs.get("rights");

if(a != null && a.size() > 0)
String rights = CDANameParser.unescapeAttributeValue
((String)a.get(0));
```

If the authenticated user had **`rights`** that equal supervisor (“rights” = “supervisor”), the above method call stores the string “supervisor” in the variable **`rights`**.

Manage User Levels

CDA through the use of attributes can further define a user hierarchy without changing the DIT structure. The Training sample application already does this through the role attribute that defines Supervisors from Users. You can use this functionality to manage user levels as follows:

1. Return the entire name/value pair for the attribute role of the current user
2. Extract only the value of the name/value pair

   Based on that value, turn on or off any appropriate links

   For example, the following code sample from Training allows only Supervisors to have the link to the Manage Users page (and subsequently enroll new users):

```
// Return name value pair for role and extract the value only
```

```
<%
Attribute a = (Attribute)attrs.get("role");
if(a != null && a.size() > 0)
   role = CDANameParser.unescapeAttributeValue((String)a.get(0));
%>

.
.
.

// Based on value turn on or off appropriate links

<%
if (role.equals("supervisor"))
{
%>
<td class="TDtext"><font size = -1><a href="User?app=UserMain
&jsp=/enrollment/jsp/ManageUsers.jsp&forwardto=<%= request.
getAttribute("FORWARDURL")%>&<%= returnInfo %>"><b>Manage
Users</b></a></font></td>
<%
}
%>
logUser = URLEncoder.encode(logUser);
```

Search for Attributes

IAccount provides the search method that searches a CDA DIT for specific name/value pairs. The search name can only search attributes in the tree that are not part of the distinguished name (DN). For a template example that uses this feature, see the **AddDirectReports.jsp** in the Training sample application.

The process to search for attributes is as follows:

1. Obtain the name of the attribute(s) you want to search based on some user input
2. Specify the attributes you want to return for any hit.

   For example:

```
…//The name for the attribute/s you are searching by based on user
input or searchKey

String key = (String)request.getParameter("searchKey");

//The value for the attribute/s you want to search by based on user
input

String sValue = (String)request.getParameter("searchValue");

...

//The attributes you want to return for the hit (UID, ACCOUNT_NUMBER
and EMAIL)

String [] attrIds = {JNDIAccountAttributes.JNDI_UID,
JNDIAccountAttributes.JNDI_USER_EMAIL,
JNDIAccountAttributes.JNDI_ACCOUNT_NUMBER};

...
```

```
//Set up an Attributes variable to store the attributes you are
searching by //(In this case //based on user input)

Attributes matchAttrs = new BasicAttributes(true);

...

//Put the attribute name/value pair to search by into matchAttrs

if (sValue == null || sValue.equals(""))
matchAttrs.put(new BasicAttribute(key));
else
if (sType != null && sType.equals("ic"))
matchAttrs.put(new BasicAttribute(key, sValue.toLowerCase()));
else
matchAttrs.put(new BasicAttribute(key, sValue));

...

//Perform Search return UID , ACCOUNT_NUMBER and User mail (attrIds)

s = root.search("", matchAttrs, attrIds);
```

Using the CDA Client

About the CDA Client

The CDA Client is the Oracle utility for creating and managing CDA schemas and directory contexts. Use the CDA Client to:

- Navigate within a directory tree
- Create, modify, or delete directory contexts
- Create or delete schema attributes
- Export nodes in a directory tree and import them into another database

For more information about the CDA Client, see the Javadoc.

Starting the CDA Client

Starting the CDA Client involves three steps at the command line interface:

1. Set up your environment for the CDA Client.
2. Run the command **`com.edocs.jndi.cda.cli.Main`** with appropriate parameters, or use a shell script to start the CDA Client.
3. Troubleshoot your environment as necessary.

This section describes how to start the CDA Client using different J2EE application and database servers and operating systems.

Setting Your Environment for the CDA Client

Before you begin, review the procedures for defining your eStatement Manager environment in the *Installation Guide for Oracle Siebel eStatement Manager*. That guide describes how to set environment variables for your Oracle, application server, and database server home directories and your default classpath. It also describes how to pass environment variables to your application server at server startup.

To run the CDA Client, your classpath must include **edx_common.jar**. Java class files for CDA and the CDA client are packaged and installed by default in **EDX_HOME/lib/edx_common.jar.** If you have customized your installation directory, look for **edx_common.jar** in the **/lib** file of your Oracle home directory (EDX_HOME).

The CDA Client also requires a database connection. This can be set either directly to the JDBC driver or to one of the database connection pools you defined when configuring your application server.

To set your environment for the CDA Client:

1. Start your application server if it is not running.
2. Switch users to your application server owner.
3. Capture your environment with the **edx_env** shell script and pass it to your application server as described in the *Installation Guide for Oracle Siebel eStatement Manager.*

Method Signatures

The CDA Client command **Main** has three "signatures," shown here as they might be entered on the command line. Your choice of signature depends on which variables you know for your environment. Study the examples in this section for comparisons.

```
java com.edocs.jndi.cda.cli.Main jndi url

java com.edocs.jndi.cda.cli.Main jdbc url username password

java com.edocs.jndi.cda.cli.Main
java.naming.factory.initial=jdbc factory
java.naming.provider.url=jdbc url
java.naming.security.principal=username
java.naming.security.credentials=password
```

Parameters

Name	Description	Syntax
jdbc factory	Initial naming factory for CDA or for your application server. See examples.	java.naming.factory.initial=**jdbc factory**
jdbc url	The URL for your JDBC driver, consisting of the database instance, local host and port, and username. See examples.	java.naming.provider.url=**jdbc url**
jndi url	The JNDI name of a database pool created when you configured your application	

Name	Description	Syntax
	server. See examples.	
password	The password of your database administrator.	java.naming.security.credentials= **password**
username	The user name of your database administrator.	java.naming.security.principal= **username**

Running the CDA Client with a Shell Script

These examples follow the shell script **create_training_schema**, installed in **EDX_HOME/samples/Training/schema**. Study this file to learn more about CDA Client options for your platform. You can customize this script, substitute your own script, or omit the file and enter CDA Client commands at the prompt.

To run the CDA client with a shell script:

1. Run the command line setup shell script.
2. Run the CDA client tool as shown in the examples below. You should receive a command prompt.
3. Test the CDA client by displaying the schema list:

```
sl
```

Example: Starting the CDA Client with a Direct Database Connection

TIP: For each example shown, enter the command entirely on one line without line breaks (shown here for clarity). If you have already set your classpath as shown in the preceding section, you can omit the **-classpath** parameter.

This example for **Oracle** connects the CDA client directly to the database driver with the **-Djdbc.drivers** parameter.

```
java -classpath $EDX_HOME/lib/edx_common.jar:$CLASSPATH
-Djdbc.drivers=oracle.jdbc.driver.OracleDriver
com.edocs.jndi.cda.cli.Main
java.naming.factory.initial=com.edocs.jndi.cda.CDAJDBCFactory
java.naming.provider.url=jdbc:oracle:thin:@localhost:1521:edx0
java.naming.security.principal=edx_dba
java.naming.security.credentials=edx
```

This example for **Microsoft SQL Server** connects the CDA client directly to the database driver with the **-Djdbc.drivers** parameter.

```
java -classpath %EDX_HOME%\lib\edx_common.jar:%CLASSPATH%
-Djdbc.drivers= com.inet.pool.PoolDriver com.edocs.jndi.cda.cli.Main
java.naming.factory.initial=com.edocs.jndi.cda.CDAJDBCFactory
java.naming.provider.url=jdbc:inetpool:inetdae7://localhost:1433:edx0
java.naming.security.principal=edx_dba
java.naming.security.credentials=edx
```

This example for **DB2** connects the CDA client directly to the database driver with the **-Djdbc.drivers** parameter.

```
java -classpath $EDX_HOME/lib/edx_common.jar:$DB2_HOME/sqllib/java12/
db2java.zip:$CLASSPATH
-Djdbc.drivers=COM.ibm.db2.jbdc.net.DB2Driver
com.edocs.jndi.cda.cli.Main
java.naming.factory.initial=com.edocs.jndi.cda.CDADB2Factory
java.naming.provider.url=jdbc:db2://localhost:6789/edx0
java.naming.security.principal=db2inst1
java.naming.security.credentials=db2inst1
```

Example: Starting the CDA Client with a Database Pool Connection

This example for **WebLogic** connects the CDA client to the database through a database connection pool using a datasource EJB on the application server.

```
java -classpath $EDX_HOME/lib/edx_common.jar:$CLASSPATH
-Djava.naming.factory.initial=weblogic.jndi.WLInitialContextFactory
-Djava.naming.provider.url=t3://localhost:7001
com.edocs.jndi.cda.cli.Main cda://edx.user.databasePool
```

This example for **WebSphere** connects the CDA client to the database through a database connection pool using a datasource EJB on the application server.

```
java -classpath $EDX_HOME/lib/edx_common.jar:/usr/WebSphere/AppServer/lib/
websphere.jar:/export/home/db2inst1/sqllib/java12/db2java.zip:$WAS_CLASSPA
TH
-Djava.library.path=/export/home/db2inst1/sqllib/java12
-
java.naming.factory.initial=com.ibm.websphere.naming.WsnInitialContextFact
ory
-Djava.naming.provider.url=iiop://10.2.1.99:900
com.edocs.jndi.cda.cli.Main cda://jdbc/Oracle
```

Command Parsing in the CDA Client

The CDA Client parser splits input lines into words at spaces and tabs. For example:

Raw Input	Parsed Input
`ls o="edocs.com"`	`ls` `o=edocs.com`
`ls o=edocs.com`	`ls` `o=edocs.com`

The pound sign (#) begins a comment. The line after the pound sign is ignored. For example:

Raw Input	Parsed Input
# this is a comment	

To parse a string containing special characters, enclose the string in single (') or double (") quotes. For example:

Raw Input	Parsed Input
`ls 'o="edocs.com"'`	`ls` `o="edocs.com"`

Note that this returns the same output as the example below.

A backslash (\) is the escape character for special characters, including quotes, pound signs, spaces, and tabs. For example:

Raw Input	Parsed Input
`ls o=\"edocs.com\"`	`ls` `o="edocs.com"`

A newline preceded by a backslash (\) without quotes is equivalent to a space. To return a newline inside a string, place the string in quotes. For example:

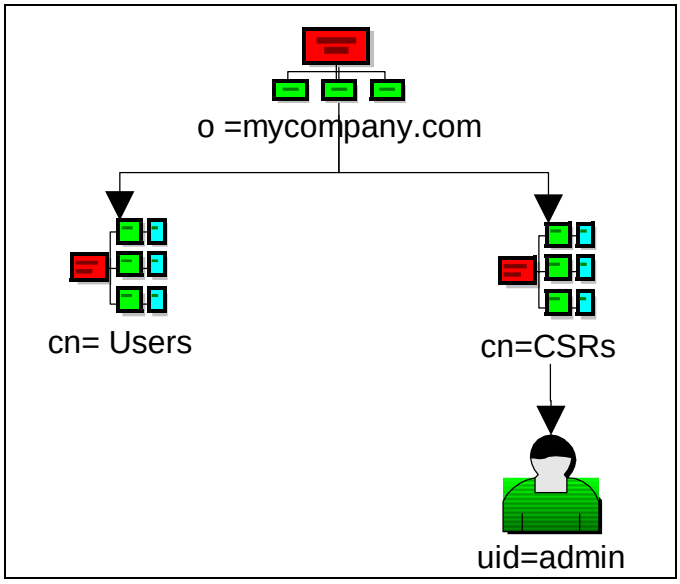
Raw Input	Parsed Input
`ls \` `o="edocs.\ com"`	`ls` `o=edocs.\ com`

The following topics discuss commands by task.

Creating and Populating a Directory Information Tree (DIT)

Creating a New DIT

This section provides an example of how to use the CDA Client to create the following DIT:



To create a new DIT:

This design is similar to the default DIT provided for Training (described above), and it should help you understand that structure if you plan to emulate it with your own CDA DIT. Follow these steps:

1. Start the CDA Client.
2. Define any schema attributes to be used by your DIT before creating any contexts. Use **SchemaList** to see all currently defined attributes:

   ```
   sl
   ```

3. Enter any schema attributes using the **SchemaBind** command. For example, if the above DIT requires a **privilege** attribute to define how much access a Customer Service Representative needs and a **groupid** attribute if you plan to further classify CSR users, you can add them as follows:

   ```
   sb privilege syntax STRING
   sb groupid
   ```

   STRING is optional (it is the default value).

   If you later decide that you do not need an attribute, you can remove it with **SchemaUnbind**:

   ```
   su groupid
   ```

   To verify your changes, you can always run **SchemaList**.

4. Make the root context node for the DIT with the **CreateSubcontext** command as follows:

   ```
   mk o=mycompany.com
   ```

5. Add the next levels of the DIT as follows:

   ```
   mk cn=Users,o=mycompany.com
   mk cn=CSRs,o=mycompany.com
   ```

   These commands use the full DIT pathname to create the subcontexts. You could also navigate to the context **o=mycompany.com** using the **pushd** or **cd** commands and then omit the **o=mycompany.com** part with the **mk** command.

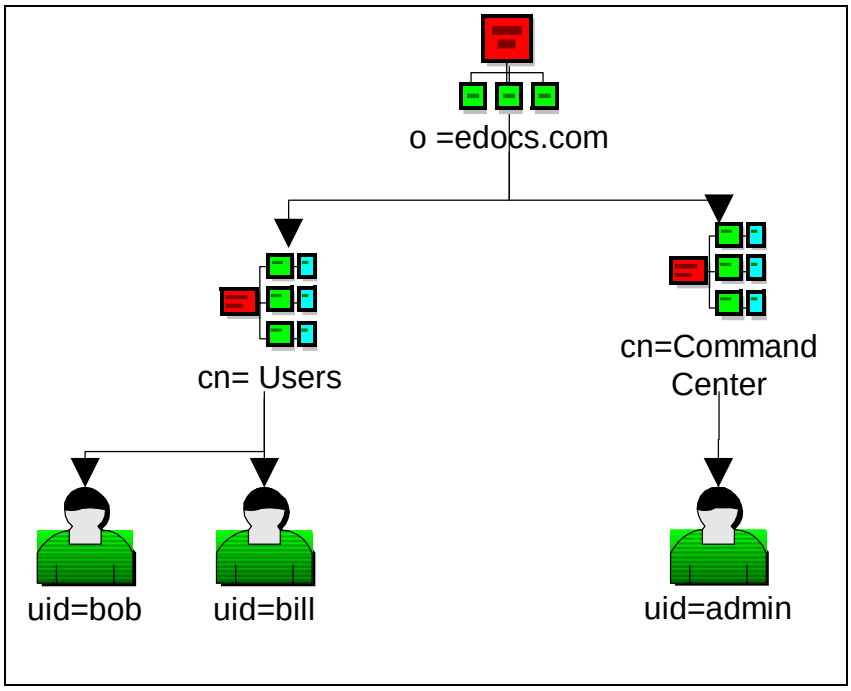
6 The final subcontext is the Admin super user that will be used to log into the system initially to add other CSR users, so this information must be pre-enrolled with CDA Client. It will also need some additional attributes defined to distinguish it such as a password and privilege. For example:

```
mk uid=admin,cn=CSRs,o=mycompany.com uid admin userPassword
D2B71E9C2E21C5F2 privilege all mail admin@mycompany.com
```

Note that even though the **`uid`** attribute is specified in the context name, it is also included as a subsequent attribute value in order to be searchable (see the **`find`** command above).

Adding a New User to a DIT

This section provides an example of how to use CDA Client to add a user context to the DIT used by Training:



To add a new user to the eStatement Manager DIT:

1 Start the CDA Client tool as described in Chapter 3: **`Deploying and Configuring Training`** for your platform.

2 Navigate to the **`DirContext`** that contains the user subcontexts:

```
cd o=edocs.com
cd cn=Users
```

3 Create the new subcontext for a user named "jack":

```
mk uid=jack
```

4 At this point you could add any attributes to "jack". For example, you could add the email and role attribute values as follows:

```
aa uid=jack email jack@boardwalk.com role user
```

Navigating a Directory Information Tree

The following table lists the CDA Client commands used to navigate a CDA directory information tree.

Command	Description
exit **bye** **quit**	Terminates.
alias [<name> [<word>]]	Without arguments, prints all aliases. With name, prints alias name. With name and word, defines an alias.
unalias <name>	Removes alias name.
path [<package> ...]	Without arguments, prints the current path. With arguments, sets the path to the list of packages.
pushd <name> **cd <name>**	Pushes the current working **DirContext** onto the context stack and changes to name.
popd **cd..**	Pops the context stack and changes to it.
dirs	Prints the context stack.
time <command>	Executes command and prints the elapsed time to completion.
source <filename>	Executes the commands in filename.

Working with Directory Contexts

CreateSubcontext

```
CreateSubcontext <name> [<attribute_name> <attribute_value> ...]
mk <name> [<attribute_name> <attribute_value> ...]
```

Creates a new subcontext name and associates all specified attribute names and values with the new subcontext.

DestroySubcontext

```
DestroySubcontext <name>
rm <name>
```

Removes subcontext name. Note: Removing the current working context results in undefined behavior.

CAUTION: While CDA does not as a rule support features nonstandard to LDAP, it does support deletion of contexts, or subtrees, containing data. In LDAP, **`DestroySubcontext`** throws a **`ContextNotEmptyException`** unless the context is empty. In CDA, **`DestroySubcontext`** will delete the context and all its subcontexts. Use this powerful feature with care**.**

List

`List [<name>]`

`ls [<name>]`

Prints the sub-contexts of name. If name is not specified, prints the top-level contexts.

ListBindings

`ListBindings [<name>]`

Prints the sub-bindings of name. If name is not specified, prints the top-level contexts.

GetAttributes

`GetAttributes <name>`

`la <name>`

Prints the attributes associated with a context name.

AddAttributes

`AddAttributes <name> <attribute_name> <attribute_value> ...`

`aa <name> <attribute_name> <attribute_value> ...`

Adds all specified attributes to the context name.

RemoveAttributes

`RemoveAttributes <name> <attribute_name> [<attribute_value>]`

`ra <name> <attribute_name> [<attribute_value>]`

Removes all attributes from context name. If an attribute value is not specified, removes all the values of **`attribute name`**.

Rename

`Rename <old_name> <new_name>`

`mv <old_name> <new_name>`

Renames old name to new name.

Working with User Attributes

Add New Attribute Names with SchemaBind

Before assigning a value to an attribute name in a directory, you must add, or **bind**, the attribute name to the schema. In the CDA Client, add a new attribute name to the schema with the command **SchemaBind** (**sb**).

```
sb <attribute_name> <syntax>
```

SchemaBind takes two parameters, attribute name and syntax. There are only two valid values for the syntax parameter: **String** (default) and **Distinguished Name** (**DN**). You must specify syntax of **DN** (upper case required) when adding an attribute of the Distinguished Name type.

To add an attribute named "employee" with the default syntax of **string**, issue the following command:

```
sb employee
```

To add an attribute named "employee" with a syntax of distinguished name, issue the following command:

```
sb employee syntax DN
```

CAUTION: Once added to the schema, an attribute name cannot be modified. It can then only be deleted (unbound) from the schema if it is not used. Plan your schema carefully before assigning attribute names.

Remove Attribute Names with SchemaUnbind

You can remove, or **unbind**, an attribute name from the schema in the CDA Client. In the CDA Client, remove an attribute name from the schema with the command **SchemaUnbind** (**su**).

```
su <attribute_name>
```

For example, to remove the attribute named "employee," issue the following command:

```
su employee
```

List Attribute Names with SchemaList

The CDA directory SchemaList contains the names of all attributes declared in the schema. In the CDA Client, you can list all attribute names in the schema with the command **SchemaList (sl).**

TIP: You can make a schema more readable by associating a description with each attribute name.

List Attributes with SchemaGetAttributes

In the CDA Client, you can print all attributes associated with an attribute name in the schema with the command **SchemaGetAttributes (sga).**

Search for Attributes with Find

There are two ways to search for attributes in an Oracle schema. For simple searches, for example when designing your schema, you can use the **Find** command in the CDA Client to print the matching attributes to the screen. The Search method of **IAccount** provides more powerful searching, for example when you need the results of a search as a parameter in a JSP. This method is described in the previous chapter.

TIP: Although CDA does not support search filters, it does implement the search method **DirContext.search(javax.naming.Name,javax.naming.directory.Attributes,String[])**.

The **Find** command prints the contexts and their attributes that match the search.

```
find name [<attribute_name> [<attribute_value>] ...]
```

Integrating With Existing User Management Systems

You may need to exchange data with another directory service, for example an LDAP server. In the CDA Client, you can export a directory schema and its attribute values in LDIF (LDAP Directory Interchange Format), or import existing attributes into an LDAP directory.

Export a Schema as LDIF

The **Export** command exports attributes in LDIF format.

```
export <file_name>
```

The **Export** command takes two parameters, **context** and **filename** (optional). **Context** defines the top-level directory context from which to export, so that you can export only a subcontext of a tree. Omitting this parameter exports from the directory root. **Filename** defines the name of the target export file. Oracle recommends the format ***.ldif.**

```
export o=edocs.com edocs.ldif
```

The example above exports all contexts of the current schema to the file **edocs.ldif** in the current directory.

Import an LDIF Schema into CDA

The **Import** command imports attributes in LDIF format into CDA.

```
import <file_name>
```

TIP: If the LDIF file you import contains any attributes whose names are not defined in the target directory schema, you must add them to the schema (with **SchemaBind**) before importing.

Using a Non-Directory Access Implementation

In addition to the CDA interface, the Oracle user management framework is flexible enough for you to modify when using other non-directory enrollment models. For example, you may have the enrollment information already stored in a separate repository for your customers. In this case, your requirement of the user management framework is to access this repository in order to authenticate existing customers or even enroll new ones.

This chapter describes how you can re-implement the IAccount interface to access such a repository. Although it provides only one example, the framework it describes allows you to tailor it for your specific application, as you will be able to supply the interface java code to interact with the repository application, such as a separate database.

Using UMFsample as a Template

eStatement Manager provides another sample J2EE application for you to deploy that uses a non-directory access enrollment models. It is called UMFsample, and you can use as a template to begin your own custom J2EE application using the Oracle user management framework.

How the JSPs interact with the user management framework defined in UMFsample is described in the next major section. By learning how they work should give you the understanding of how to create your custom implementation. However, before you can use the UMFsample JSPs to see how the enrollment works, you must deploy and configure UMFsample and then run it through the National Wireless sample application files.

Deploying and Configuring UMFsample

Before you can use UMFsample, you must successfully install and configure eStatement Manager using the instructions in the *Installation Guide for Oracle Siebel eStatement Manager* for your operating system and application server. You also need to install the UMFsample application files provided by the installation program supplied with the SDK.

After installing the **`UMFsample`** application, you can find its EAR file in the **`<EDX_HOME>/Samples/umfsample/J2EEApps/weblogic`** (or **`/websphere)`** directory where you installed eStatement Manager. To see how UMFsample works, you must deploy it. The deployment instructions in the *Deploying and Customizing J2EE Applications Guide for Oracle Siebel eStatement Manager* for the **`Sample.ear`** should work equally well for the **`UMFsample.ear`** application.

After deploying **`UMFsample`**, you must run the **`create_sample_table`** script located in the database subdirectory for your platform in **`Samples/umfsample/db`**. This file contains the database commands to add the sample repository enrollment information used by **`UMFsample`**.

The sections that follow describe how to run the script for a specific database.

For an Oracle Database:

As an Oracle user, run the following script in **`EDX_HOME/samples/umfsample/db/oracle`** and pass it the parameter values for the database ID (ORACLE_SID), database user name, and database password:

```
./create_sample_table.sh edx0 edx_dba edx
```

After you enter the information, it connects to the database and executes **`create_sample_table.sql`**.

For a DB2 Database:

As a DB2 user, run the following script in **`EDX_HOME/samples/umfsample/db/oracle`** and pass it the parameter values for the database ID, database user name, and database password:

```
./create_sample_table.sh edx0 edx_dba edx
```

After you enter the information, it connects to the database and executes **`create_sample_table.sql`**.

For a Microsoft SQL Server Database:

In **`EDX_HOME\samples\umfsample\db\oracle`**, run the following script and pass it the parameter values for the database ID, database user name, and database password. For example:

```
create_sample_table.bat edx0 edx_dba edx
```

After you enter the information, it connects to the database and executes **`create_sample_table.sql`**.

Using UMFsample with National Wireless

After deploying and configuring UMFsample, you can use it with the National Wireless sample application input files provided with eStatement Manager. The eStatement Manager guides provide information about how to process the National Wireless files.

Note: You must define UMFsample as the DDN in the Command Center and process the National Wireless data files before performing the steps described below. A short example of how to do this is provided in the *Deploying and Customizing J2EE Applications Guide for Oracle Siebel eStatement Manager* for the Sample application.

For users to view their statements, you must enroll them into the database tables using the UMFsample JSPs as follows:

1. Enter the following URL in your browser:

   `http://your-server:port/umfsample/user/jsp/index.jsp`

   Where **`your-server`** and **`port`** are the values you defined for eStatement Manager when you installed and configured it.

2. Specify UMFsample as the DDN and click **`Submit`**.
3. Click on **`Enroll`**.
4. Enroll each user with the following information for the required enrollment fields:

Email	SSN	Account Number
<Name>@oracle.com	0001	0331734
<Name>@oracle.com	0002	4191463
<Name>@oracle.com	0003	8611250
<Name>@oracle.com	0004	9001203
<Name>@oracle.com	0005	0407200
<Name>@oracle.com	0006	3069725
<Name>@oracle.com	0007	4694878
<Name>@oracle.com	0008	1710123
<Name>@oracle.com	0009	9424090

5 The other enrollment fields are optional. You can enter any information for these fields as part of this example.

How this enrollment works is described in the next section.

How does UMFsample Do Enrollment?

As mentioned earlier in this chapter, eStatement Manager provides a generic **`IAccount`** user management interface for enrollment purposes. What is required is a custom mechanism to invoke **`IAccount`** to enroll users dynamically through JSPs and EJBs.

UMFsample provides revised **`SubscribeApp`** and **`UpdateApp`** class files along with the **`ejb-enrollment-umfsample.jar`** EJB to handle this process. It also provides servlets and JSPs to interact with those class files and EJB; UMFsample is just one implementation that uses these files to support the enrollment of the National Wireless application and others.

It is expected that when you create your own J2EE application based on UMFsample, you must create custom JSPs and Servlets. Plus, you must modify the **`ejb-enrollment-umfsample.jar`** to redefine the new IAccount interface to the repository holding the enrollment information.

It is recommended that you look at the contents of the UMFsample EAR as the following sections describe how they work. You can extract the contents using the **`jar`** command (UNIX) or WinZip utility (Windows). The *Deploying and Customizing J2EE Applications Guide for Oracle Siebel eStatement Manager* provides several examples to do this, along with an explanation of the basic elements inside the EAR.

UMFsample Enrollment Process Flow

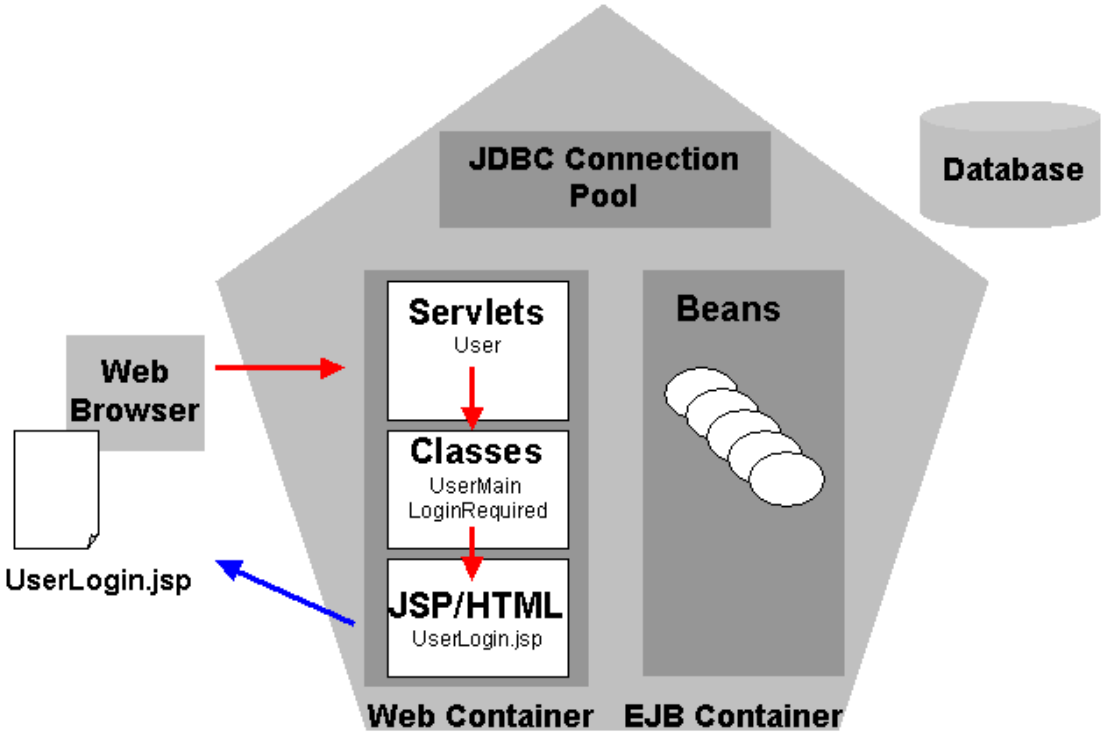
The enrollment process for UMFsample can be broken down into two parts:

- Authenticate the user: determine if the user is already logged in, and if not, ask the user to log in. After the user provides the log in information, validate that information against the enrollment information stored in the database. If it is valid, assign an IAccount instance (stored in a cookie) to the user and proceed to the requested page. If it is not valid, return an error to the user.

- Enroll the user: if they have the correct permissions, provide the user with a subscription page to enter as their log in information and stored in the database. The next time they attempt to access UMFsample, they will be authenticated against this enrollment information. UMFsample also provides a mechanism to update enrollment information.

UMFsample checks the authentication of a user prior to showing them any page on their browser. As long as the IAccount instance is still valid, the user can proceed as usual. The moment the instance becomes invalid (such as through a session timeout), the user needs to log back in before proceeding.

The following diagram shows how the process works for the first part of authentication:

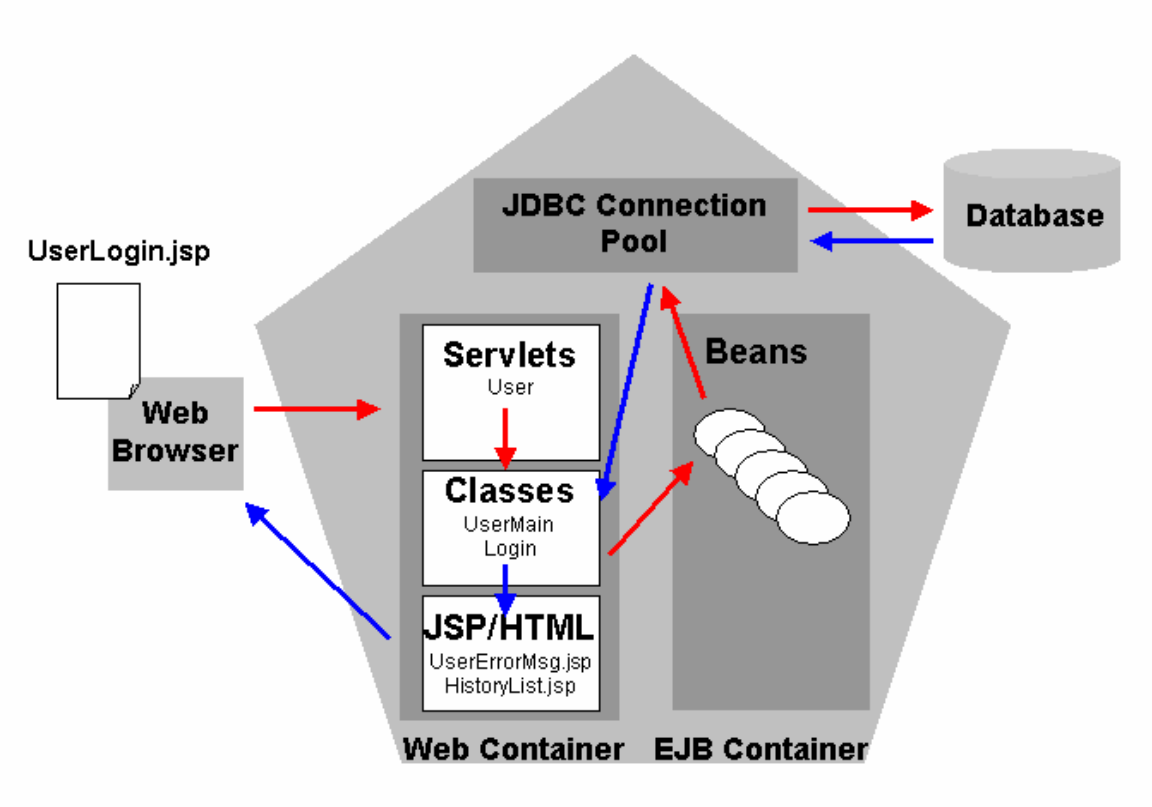


The user through the specified URL requests the User servlet and **`UserMain`** class file that determines whether the user is already logged on. The following example URL shows this request:

```
http://your-server:port/umfsample/User?app=UserMain
&jsp=/user/jsp/HistoryList.jsp&ddn=NatlWireless
```

Built into **`UserMain`** is the **`LoginRequired`** class file that indicates the user is not logged in and will be given the **`UserLogin.jsp`** to do so. In the contents of the UMFsample WAR, you can find the User servlet definition in the **`WEB-INF/web.xml`** file, and the **`UserLogin.jsp`** in the **`enrollment/jsp`** directory. **`UserMain.class`** resides in **`WEB-INF/classes/com/edocs/app/user`** directory.

The next diagram shows what happens after the user submits the log in information through **`UserLogin.jsp`**:

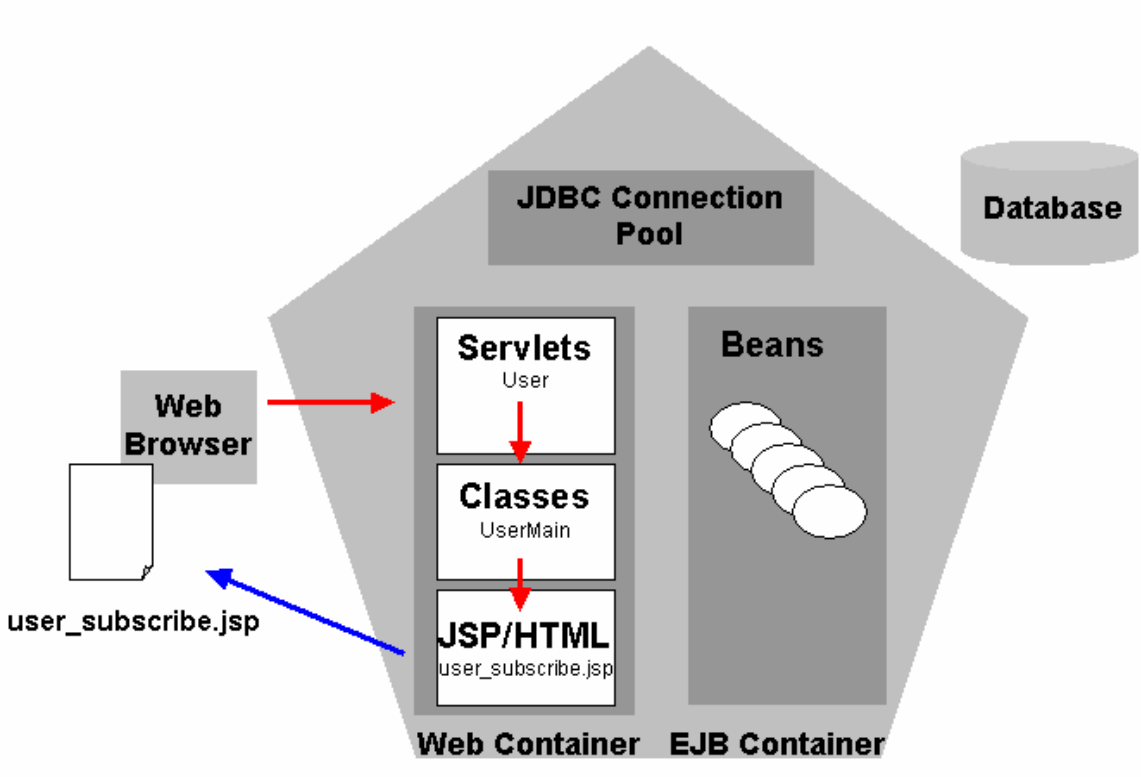


Basically, the User servlet and **`UserMain`** class file invokes the **`Login`** class file to validate the log in information against the repository database through a JDBC connection pool (established during installation and configuration of eStatement Manager). **`Login.class`** resides in **`WEB-INF/classes/com/edocs/app/enrollment`** directory.

`TIP`: When using UMFsample as a template, the **`UserMain.class`** file does not need to be modified, but you need to define a new **`Login.class`** file.

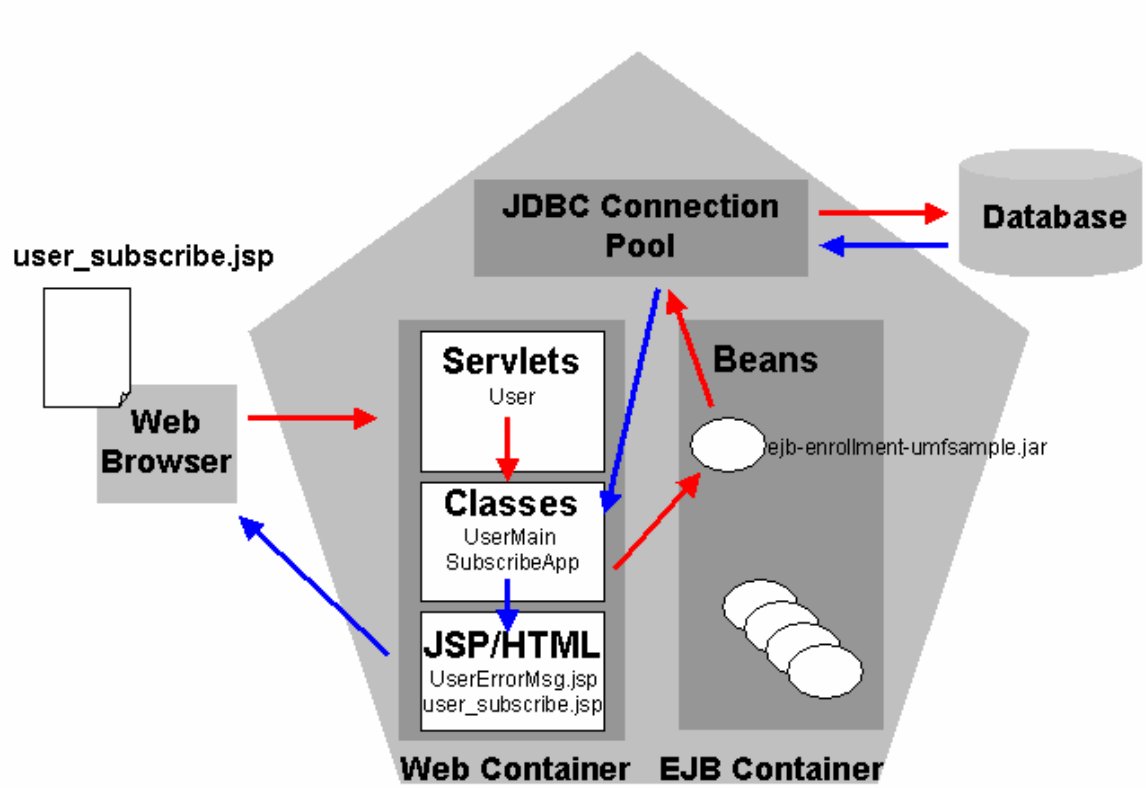
If the information is valid, it returns the **`HistoryList.jsp`** (or other file that was requested by the original URL request) to the user's browser. Otherwise, it returns **`UserErrorMsg.jsp`** with any error message generated by eStatement Manager.

The next two diagrams shows how the enrollment works in UMFsample:



UMFsample handles the enrollment of new users through the **`user_subscribe.jsp`** page.

When a user clicks on the **`Enroll`** button on the **`UserLogin.jsp`**, eStatement Manager returns the **`user_subscribe.jsp`** to the user to enter the enrollment information. After the user submits the information, the following diagram shows how the information is processed:



The information of the new user is posted through the User servlet and **`UserMain`** class file to the **`SubscribeApp`** class file. It in turn invokes the custom **`IAccount`** implementation of UMFsample called **`SampleAccount`** from the **`ejb-enrollment-umfsample.jar`** EJB to add the enrollment information to the tables in the database repository. If the enrollment is successful, eStatement Manager returns the **`user_subscribe.jsp`** to the user in order to enroll another person. Otherwise, it returns the error information through **`UserErrorMsg.jsp`**.

In addition to **`user_subscribe.jsp`**, UMFsample provides the **`user_update.jsp`** that goes through the same process described above to update the enrollment information for existing users, except it uses the **`UpdateApp`** class file.

You can find the **`SubscribeApp.class`** and **`UpdateApp.class`** files in the **`WEB-INF/classes/com/edocs/app/enrollment`** directory of the UMFsample WAR file. However, unlike the versions in the Training EAR, you must modify these two files to communicate with your custom enrollment EJB. The **`user_subscribe.jsp`** and **`user_update.jsp`** will also be part of any customization, and you can use the UMFsample ones as template examples.

Creating an Application Based on UMFsample

When you install UMFsample, you also receive its enrollment Java source files, including the ones mentioned in the previous section. They are packaged in the **`umf_src.jar`** file, and you can extract its contents using the **`jar`** command (UNIX) or WinZip utility (Windows). The following files are included in its **`enrollment`** directory:

- **`Login.java`**

- **Logout.java**
- **SampleAccount.java**
- **SampleAccountResolver.java**
- **SubscribeApp.java**
- **UpdateApp.java**

The **SampleAccount.java** and **SampleAccountResolver.java** source files are part of the **ejb-enrollment-umfsample.jar** EJB, while the other four files are part of the UMFsample WAR file. Although there are many other class files involved in the Oracle user management framework to support enrollment, all the customization required to support your enrollment model can be encapsulated in these six files.

You can choose to simply modify these existing files and re-build them using the utility supplied with this SDK module (described later in this chapter), or you choose to replace and/or rename them using your own Java build tools. However, in the latter case there is significantly more effort to make it work as there are ample references to these files and UMFsample that must be changed in the EAR to reflect the new files.

The sections that follow provide examples of how **Login.java**, **SampleAccount.java**, and **SubscribeApp.java** work. The others are similar to these in function.

Modifying the Enrollment Source Files

This section describes parts of the UMFsample source Java files that are relevant to any modifications you will need to make based on accessing your enrollment repository. However, entire source files are not shown here as you can view them yourself. Many comments have been embedded in the source files to help you understand the program logic.

As mentioned earlier, the enrollment process flow for UMFsample begins when **UserLogin.jsp** prompts the user for login information. UMFsample requires the user's email and last four social security digits that are posted as hidden URL values, as shown below in **UserLogin.jsp**:

```
<%
                    String appMethod = AppConstants.METHOD ;

                    // Set required hidden values.
                    Enumeration params = request.getParameterNames();
                    while (params.hasMoreElements())
                    {
                      String nameStr = (String) params.nextElement();
                          if (!( nameStr.equals("auth__email") ||
                                 nameStr.equals("auth__dn") ||
                                 nameStr.equals("auth__ssn") ||
                                 nameStr.equals("errforwardto") ||
```

```
                        nameStr.equals(appMethod) ||
                        nameStr.equals("EDOCSLOGIN") ||
                        nameStr.equals("MESSAGE")) &&
                    !( queryParams != null &&
queryParams.containsKey(nameStr))) {


                     out.println("<input type=hidden name=\""
+ nameStr +
                                    "\" value=\"" +
request.getParameter(nameStr) + "\">");
                }
             }
%>


    <input type=hidden name="errforwardto"
value="/enrollment/jsp/UserLogin.jsp">
    <input type=hidden name="<%= AppConstants.METHOD %>"
value="<%= request.getMethod()%>">
    <input type=hidden name="EDOCSLOGIN" value="EDOCSLOGIN">
    <input type=hidden name="edocs__re-login"
value="edocs__re-login">



<table width="500" border="1" cellspacing="0" cellpadding="3"
align="center">
  <tr>
    <td colspan="2" bgcolor="B1D0EE">
   <b><font size="-1" face="Verdana, Arial, Helvetica, sans-
serif">User Login...</font></b><font size="-2" face="Verdana,
Arial, Helvetica, sans-serif">
      Enter your email address, last four digits of your
social security number and click &amp;quot;Submit.&amp;quot;
</font><font size="-2">If you have not subscribed, <a
href="UserEnrollment?app=SubscribeApp&jsp=/umfsample/enrollmen
t/jsp/user_subscribe.jsp&<%= returnInfo %>">Enroll Now</a> to
sign up for your electronic account.</font>
    </td>
  </tr>
  <tr>
    <td align="right" class="label" width="400">
       <b>Email Address:<br></b>
```

```
    </td>
    <td width="400">
      <input type="text"  name="auth__email" size="32"
maxLength="32" onBlur="setDN(this.form);">
      <input type=hidden name="auth__dn" value="default">
    </td>
  </tr>
  <tr>
    <td align="right" class="label" width="400">
       <b>Last 4 digits of SSN:</b>
    </td>
    <td width="400">
      <input type="password"  name="auth__ssn" size="20"
maxLength="20">
    </td>
  </tr>
</table>
```

After submitting the login information, the **UserMain** class file redirects the input to the **Login** class file for authentication. The following is part of the Java source file for **Login**, and it shows where it maps the input parameters to the fields of the table used by UMFsample:

```
    /**
     * Set the key variables in the environment context.
     */
    private Hashtable createContext(HttpServletRequest req) {
        // Map the HTML input parameters to the fields in the
sample_account table.
        Hashtable env = new Hashtable();
        env.put(EnrollmentConstants.ACCOUNT_KEY,
_accountImpl);
        String email = req.getParameter("auth__email");
        String ssn = req.getParameter("auth__ssn");
        String ddn = req.getParameter("ddn");
        env.put("email", email);
        env.put("ssn", ssn);
        env.put("ddn", ddn);
        return env;
    }
```

This is the only significant portion of **Login.java** that needs to change as the rest uses the default **IAccount** implementation.

As shown in the following code sample for **UserLogin.jsp**, there is a link to Enroll new users though the **UserEnrollment** servlet:

```
Enter your email address, last four digits of your social
security number and click &quot;Submit.&quot; </font><font
size="-2">If you have not subscribed, <a
href="UserEnrollment?app=SubscribeApp&jsp=/umfsample/enrollmen
t/jsp/user_subscribe.jsp&<%= returnInfo %>">Enroll Now</a> to
sign up for your electronic account.</font>
```

The URL parameters defined in that link include the **user_subscribe.jsp** that accepts the input values from the user and the **SubscribeApp** class file that calls the custom **IAccount** methods in the enrollment EJB. The primary role of **user_subscribe.jsp** is to ensure all the values are properly collected for insertion to the enrollment database repository. For example, the following code sample from **user_subscribe.jsp** lists the values to be inserted:

```
<%
    // List of user attributes.
    String [] edocsAttr = {
        "email",
        "ssn",
        "ddn",
        "firstname",
        "lastname",
        "zipcode",
        "telephone",
        "address",
        "account"
    };

    Properties props = new Properties();

    for (int i = 0; i < edocsAttr.length; i++) {
        String previousEntry =
request.getParameter(edocsAttr[i]);
        if (previousEntry != null)  {
            props.setProperty(edocsAttr[i], previousEntry);
        } else {
```

```
                props.setProperty(edocsAttr[i], "");
            }
        }

%>
```

As shown earlier, UMFsample uses the **email** and **ssn** values to validate the log in information, with **email** listed first because it is the primary key value in the enrollment table. The other values of the enrollment table are not required for authentication, but depending on your application needs could be used for many other purposes, such as CSR validation of an account while dealing with the customer directly over the phone.

After **user_subscribe.jsp** posts the enrollment input information, **SubscribeApp** begins the process of inserting the information into the enrollment table using the following code:

```
/*
* Inserts new user into database using <i>IAccount<i>
createAccount
*/
protected void doPost(HttpServletRequest req,
HttpServletResponse res) {
        IAccount account = null;
        String forwardto = req.getParameter("returnTo");
        String contextPath = req.getContextPath();

        if (forwardto != null)
            forwardto = forwardto.replace(';', '&');
        else
            forwardto = contextPath + "/User?" +
                    "app=" + req.getParameter("app2") +
                    "&ddn=" + req.getParameter("ddn") +
                    "&jsp=" + req.getParameter("jsp2");

        try {
            account = getAccountObj();

            NameValue[] nameValues = getNameValues(req,
"edocs__");
            req.setAttribute("NAMEVALUEPAIRS", nameValues);
```

```
            createAccountCtx(account, nameValues);


            forwardto += "&" + EnrollmentAppConstants.MESSAGE
+ "=ACCOUNT_CREATED";
            doHttpRedirect(req, res, forwardto);


        } catch (DuplicateEnrollmentException dee) {
            // take them back to the current page and give
them another chance
            req.setAttribute(EnrollmentAppConstants.MESSAGE,
"DUPLICATE_USERID");


            String spath = req.getServletPath();
            if (spath.startsWith("/"))
                spath = spath.substring(1);


            forwardto = contextPath + "/" + spath + "?" +
                    "app=" + req.getParameter("app") +
                    "&jsp=" + req.getParameter("jsp") +
                    "&appRoot2=" +
req.getParameter("appRoot2") +
                    "&ddn=" + req.getParameter("ddn") +
                    "&app2=" + req.getParameter("app2") +
                    "&jsp2=" + req.getParameter("jsp2") + "&"
+
                    EnrollmentAppConstants.MESSAGE +
"=DUPLICATE_USERID";
            try {
                doHttpRedirect(req, res, forwardto);
            } catch (Exception e) {
                doForwardException(req, res, e);
            }
        } catch (Exception e) {
            doForwardException(req, res, e);
        } finally {
            try {
                if (account != null) account.remove();
```

```
            } catch (Throwable t) {
                log("doPost", t);
            }
        }
    }
```

The above method in turn calls the **createAccountCtx** method that ensures the email address value exists and then calls the **SampleAccount** EJB method **createSubcontext**:

```
/**
*  Create the account record in the database.
*/
private void createAccountCtx(IAccount account, NameValue[]
nv)
            throws DuplicateEnrollmentException,
            NamingException,
            Exception {
        Attributes attrs = new BasicAttributes();
        String name = null;

        // Find primary key - email.
        for (int i = 0; i < nv.length; i++) {
            if (nv[i].name.equals("email")) {
                name = nv[i].value.trim();
                break;
            }
        }
        if (name == null) throw new NamingException("Account
parameters missing email address.");

        // Populate the account parameters.
        for (int i = 0; i < nv.length; i++) {
            Attribute attr = attrs.get(nv[i].name);
            if (attr != null) {
                attr.add(nv[i].value);
            } else {
```

```
                attr = new BasicAttribute(nv[i].name,
nv[i].value);
            }
            attrs.put(attr);
        }
        try {
            // Create the account record in the database.
            account.createSubcontext("", attrs);

        } catch (NameAlreadyBoundException nabe) {
            throw new DuplicateEnrollmentException();
        }
    }
```

The **createSubcontext** method defined in **SampleAccount** is as follows:

```
/**
* Insert an account into the DB.
*/
public void createSubcontext(String name, Attributes attrs)
throws NamingException {
        log("createSubcontext(String name, Attributes
attrs)");
        if (name == null || name.length() > 0)
            throw new IllegalArgumentException("Name is null
or not empty.");
        if (attrs == null)
            throw new IllegalArgumentException("Attributes is
null");

        BasicAttributes battrs = (BasicAttributes) attrs;
        BasicAttribute attr = (BasicAttribute)
(battrs.get(SSN));
        if (attr == null)
            throw new NamingException("Must provide a social
security number as password.");
        attr = (BasicAttribute) (battrs.get("account"));
        if (attr == null)
```

```
            throw new NamingException("Must provide an account
number.");
        attr = (BasicAttribute) (battrs.get("ddn"));
        if (attr == null)
            throw new NamingException("Must provide a ddn.");
        attr = (BasicAttribute) (battrs.get("email"));
        if (attr == null)
            throw new NamingException("Must provide a email
address.");

        String emailAddress = (String)attr.get(0);

        Connection cn = null;
        PreparedStatement stmt = null;
        try {
            // Find the account record.
            HashMap record =
retrieveAccountRecord(emailAddress);

            // If found, the account already exists.
            if (!record.isEmpty()) {
                throw new NameAlreadyBoundException();
            }

            // Otherwise, create the insert statement.
            cn = getConnection();
            stmt = cn.prepareStatement(INSERT);

            HashMap tempRecord = new HashMap();

            // Populate the parameters of the
PreparedStatement.
            for (int i = 0; i < accountFields.length; i++) {
                BasicAttribute battr = (BasicAttribute)
(battrs.get(accountFields[i]));
                if (battr == null) {
                    stmt.setObject(i + 1, "");
```

```
                    continue;
                }
                stmt.setObject(i + 1, battr.get(0));
                tempRecord.put(accountFields[i],
battr.get(0));
            }

            // Create the account in database.
            stmt.execute();

            // Set the cache.
            accountRecord = tempRecord;
        } catch (NamingException ne) {
            log(ne);
            throw ne;
        } catch (SQLException se) {
            log(se);
            throw new NamingException(se.getMessage());
        } finally {
            try {
                if (stmt != null) stmt.close();
            } catch (SQLException e) {
                log(e);
            }
            try {
                if (cn != null) cn.close();
            } catch (SQLException e) {
                log(e);
            }
        }
    }
```

In **SampleAccount**, the primary purpose of **createSubcontext** is to add an entry into the enrollment database table using the values initially supplied in **user_subscribe.jsp** and passed down to this method. UMFsample uses a SQL database to store the enrollment information, so this method begins to build the INSERT statement to accomplish the goal (as shown above in **bold**). The structure of each SQL statement used by **SampleAccount** is defined later in the file:

```
// SQL Statements
    private final static String SELECT_BY_EMAIL = "select *
from sample_account
            where email = ?";
    private final static String UPDATE_PREFIX = "update
sample_account set ";
    private final static String UPDATE_PRIMARY_KEY =
UPDATE_PREFIX + " email =
            ? where email = ?";
    private final static String WHERE_SUFFIX = " where email =
?";
    private final static String DELETE_BY_EMAIL = "delete from
sample_account
            where email = ?";
    private final static String INSERT = "insert into
sample_account
(email,ddn,ssn,firstname,lastname,zipcode,telephone," +
            "address,accounttype,account,accountdesc,ddndesc)
" +
            "values (?,?,?,?,?,?,?,?,?,?,?,?,?)";
```

The **createSubcontext** method is just one of many defined in **SampleAccount** to support the enrollment model required by UMFsample. Remember that **SampleAccount** is the custom implementation of **IAccount**, so each method required by your enrollment model will need to change these methods. UMFsample requires the following custom methods:

- **authenticate**: Used to verify the current log in information for a user in the enrollment table.
- **reAuthenticate**
- **getAttributes**: Used to retrieve the account information for an enrolled user.
- **modifyAttributes**: Used to update the account information for an enrolled user.
- **destroySubcontext**: Used to delete the enrollment information for a user.
- **impersonate**: Used by a CSR application to allow a representative to impersonate a user.

For examples of each, see the **SampleAccount** source file.

Defining Your Custom Enrollment EJB

If you choose to re-implement the UMFsample enrollment EJB by adding your own customizations to the files but not changing the names of the files nor their structure, then the process is pretty simple. This SDK module provides the build script to recreate the enrollment EJB and EAR using the source files described earlier. The steps to do this are described in the next section.

The only other major step would be to rename the modified EAR to another name and change the J2EE Web and EJB descriptor files to that new name. This is described in the *Deploying and Customizing J2EE Applications Guide for Oracle Siebel eStatement Manager*.

However, if you choose to rename the enrollment EJB class file from SampleAccount to another name, or change the structure of where those enrollment EJB files reside, then the build environment described in the next section will not work and you will need to use your own environment. This section details some of the changes you will need to be aware of.

The EJB descriptor files will need to change to reflect the new enrollment class file name or file structure. To access these descriptor files, you must unjar or unzip the ejb-**enrollment-umfsample.jar** file into a temporary directory. For examples about extracting the contents of a JAR file, see the *Deploying and Customizing J2EE Applications Guide for Oracle Siebel eStatement Manager*.

The two descriptor files you modify are:

ejb-jar.xml

weblogic-ejb-jar.xml (WebLogic) or **ibm-ejb-jar-bnd.xmi** (WebSphere)

The first descriptor file (**ejb-jar.xml**) is common across J2EE application servers and contains the enterprise beans Session descriptors for the UMFsample enrollment model. Specifically, it defines the **SampleAccount** EJB and the **SampleAccountResolver** EJB, which is an implementation of **IAccountResolver**.

The contents of the **ejb-jar.xml** are shown here, and indicate the parts that you need to change based on your new EJB name and its location (**com.edocs.samples.umf**):

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD
Enterprise JavaBeans 1.1//EN"
"http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">
<ejb-jar id="ejb-jar_ID">
<display-name>Sample Enrollment</display-name>
<enterprise-beans>
<session id="Session_1">
<description>Sample enrollment model account</description>
<display-name>SampleAccount</display-name>
<ejb-name>SampleAccount</ejb-name>
<home>com.edocs.enrollment.user.IAccountHome</home>
<remote>com.edocs.enrollment.user.IAccount</remote>
<ejb-class>com.edocs.enrollment.user.AccountBean</ejb-class>
<session-type>Stateful</session-type>
<transaction-type>Bean</transaction-type>
<env-entry id="EnvEntry_SampleAccount_1">
<env-entry-name>accountImpl</env-entry-name>
```

```
<env-entry-type>java.lang.String</env-entry-type>
<env-entry-value>com.edocs.samples.umf.SampleAccount</env-
entry-value>
</env-entry>
<resource-ref id="ResRef_SampleAccount_1">
<res-ref-name>jdbc/DataSource</res-ref-name>
<res-type>javax.sql.DataSource</res-type>
<res-auth>Container</res-auth>
</resource-ref>
</session>
<session id="Session_2">
<description>Sample enrollment model email
resolver</description>
<display-name>SampleAccountResolver</display-name>
<ejb-name>SampleAccountResolver</ejb-name>
<home>com.edocs.enrollment.user.IAccountResolverHome</home>
<remote>com.edocs.enrollment.user.IAccountResolver</remote>
<ejb-class>com.edocs.enrollment.user.AccountResolverBean</ejb-
class>
<session-type>Stateful</session-type>
<transaction-type>Bean</transaction-type>
<env-entry id="EnvEntry_SampleAccountResolver_1">
<env-entry-name>accountImpl</env-entry-name>
<env-entry-type>java.lang.String</env-entry-type>
<env-entry-
value>com.edocs.samples.umf.SampleAccountResolver</env-entry-
value>
</env-entry>
<resource-ref id="ResRef_SampleAccountResolver_1">
<res-ref-name>jdbc/DataSource</res-ref-name>
<res-type>javax.sql.DataSource</res-type>
<res-auth>Container</res-auth>
</resource-ref>
</session>
</enterprise-beans>
</ejb-jar>
```

The second descriptor file (**weblogic-ejb-jar.xml** for WebLogic or **ibm-ejb-jar-bnd.xmi** for WebSphere) contains reference descriptors for **SampleAccount** and **SampleAcountResolver** that mention UMFsample. For example:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE weblogic-ejb-jar PUBLIC "-//BEA Systems, Inc.//DTD
WebLogic 6.0.0 EJB//EN"
"http://www.bea.com/servers/wls600/dtd/weblogic-ejb-jar.dtd">
<weblogic-ejb-jar>
<weblogic-enterprise-bean>
<ejb-name>SampleAccount</ejb-name>
<stateful-session-descriptor>
<stateful-session-cache>
<max-beans-in-cache>500</max-beans-in-cache>
<idle-timeout-seconds>900</idle-timeout-seconds>
</stateful-session-cache>
<stateful-session-clustering>
<home-is-clusterable>true</home-is-clusterable>
<replication-type>None</replication-type>
</stateful-session-clustering>
</stateful-session-descriptor>
<reference-descriptor>
<resource-description>
<res-ref-name>jdbc/DataSource</res-ref-name>
<jndi-name>edx.databasePool</jndi-name>
</resource-description>
</reference-descriptor>
<jndi-name>edx/umfsample/ejb/SampleAccount</jndi-name>
</weblogic-enterprise-bean>
<weblogic-enterprise-bean>
<ejb-name>SampleAccountResolver</ejb-name>
<stateful-session-descriptor>
<stateful-session-cache>
<max-beans-in-cache>500</max-beans-in-cache>
<idle-timeout-seconds>900</idle-timeout-seconds>
</stateful-session-cache>
<stateful-session-clustering>
```

```
<home-is-clusterable>true</home-is-clusterable>
<replication-type>None</replication-type>
</stateful-session-clustering>
</stateful-session-descriptor>
<reference-descriptor>
<resource-description>
<res-ref-name>jdbc/DataSource</res-ref-name>
<jndi-name>edx.databasePool</jndi-name>
</resource-description>
</reference-descriptor>
<jndi-name>edx/umfsample/ejb/SampleAccountResolver</jndi-name>
</weblogic-enterprise-bean>
</weblogic-ejb-jar>
```

In addition to these EJB descriptor files, you must modify any Web descriptor files that mention **SampleAccount**: **web.xml**, **weblogic.xml** (for WebLogic) and **ibm-web-bnd.xmi** (for WebSphere). For example, the following is a portion of the **web.xml** file included in the UMFsample WAR:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE web-app 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>umfsample</display-name>
<distributable/>
<servlet>
    <servlet-name>UserServlet</servlet-name>
    <display-name>UserServlet</display-name>
    <servlet-class>com.edocs.app.AppServlet</servlet-class>
    <init-param>
      <param-name>ServletRoot</param-name>
      <param-value>com.edocs.app.user</param-value>
    </init-param>
    <init-param>
      <param-name>ErrorPage</param-name>
      <param-value>/common/jsp/UserErrorMsg.jsp</param-value>
```

```
    </init-param>
    <init-param>
      <param-name>LoginRoot</param-name>
      <param-value>com.edocs.samples.umf</param-value>
    </init-param>
    <init-param>
      <param-name>LoginPage</param-name>
      <param-value>/enrollment/jsp/UserLogin.jsp</param-value>
    </init-param>
    <init-param>
      <param-name>Account.name</param-name>
      <param-value>edx/umfsample/ejb/SampleAccount</param-
value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
<servlet>
    <servlet-name>UserEnrollmentServlet</servlet-name>
    <display-name>UserEnrollmentServlet</display-name>
    <servlet-class>com.edocs.app.AppServlet</servlet-class>
    <init-param>
      <param-name>ServletRoot</param-name>
      <param-value>com.edocs.samples.umf</param-value>
    </init-param>
    <init-param>
      <param-name>ErrorPage</param-name>
      <param-value>/common/jsp/UserErrorMsg.jsp</param-value>
    </init-param>
    <init-param>
      <param-name>LoginRoot</param-name>
      <param-value>com.edocs.samples.umf</param-value>
    </init-param>
    <init-param>
      <param-name>LoginPage</param-name>
      <param-value>/enrollment/jsp/UserLogin.jsp</param-value>
    </init-param>
```

```
    <init-param>
      <param-name>Account.name</param-name>
      <param-value>edx/umfsample/ejb/SampleAccount</param-
value>
    </init-param>
  </servlet>
```

Even if you do not change the location of the files, you should change the entries to reflect your new J2EE application name. Normally this is done as part of the configuration and deployment of your custom application as described in the *Deploying and Customizing J2EE Applications Guide for Oracle Siebel eStatement Manager*.

After making the changes to the above descriptor files, you must re-jar or re-zip the files you extracted back into new EJB JAR and WAR files to replace the previous ones in your EAR. This is also described in the above guide.

Building Your Version of UMFSample

The User Management Framework SDK module ships with editable Java source files for building your own custom solution. Source code is packaged in a JAR file in the **EDX_HOME/samples/umfsample/src** directory after you install the UMFsample application.

To build custom enrollment EJBs from this source, Oracle uses the free and platform-independent tool **Ant** from Apache. Install Ant to build your customized order capture solution. You can extract example source code, customize it for your Web application, build with Ant, and then package and deploy as usual. This build procedure is the same for Solaris, AIX, and Windows 2000.

To set up the UMFsample source environment:

1. Download and install the JDK 1.3.1 for your system (if it is not already installed).
2. Download and install ANT 1.4.1 from:

   ```
   http://ant.apache.org/index.html
   ```

3. Create a working directory for your new enrollment files, for example **umf_working**.
4. Navigate to **EDX_HOME/samples/umfsample/src** and copy the **umf_src.jar** file to your working directory.
5. Navigate to **EDX_HOME/samples/umfsample/J2EEApps/weblogic** (or **/websphere**) and copy the **ear-umfsample.ear** file to your working directory.
6. In your working directory, extract the source JAR as follow (to use the **jar** command, your PATH environment must have **JAVA_HOME/bin** defined):

   ```
   jar xvf umf_src.jar
   ```

   You see two new directories (**umf_src** and **META-INF**) along with a **build.xml** file that is used to define the build environment. The **META-INF** directory contains a manifest file, while the **umf_src** directory contains source files for the application Web server and enrollment bean. The Web server source JSP files for enrollment are located in **umf_src/web/enrollment/jsp**. The enrollment bean java files are in **umf_src/enrollment**.

After you have finished customizing these Java source files, you can use Ant to compile your class files, package your JARs, and create an EAR file ready to deploy. Make sure to study the **build.xml** script file carefully, and to customize any elements required for your environment. For example, you may decide to change the EAR name from **ear-umfsample.ear** to something like **ear-myapp.ear**. The *Deploying and Customizing J2EE Applications Guide for Oracle Siebel eStatement Manager* describes how to do this for the **ear-umfsample.ear** you copied to your working directory. You also need to change the **build.xml** file to reflect those new names in the following entries:

```
<property name="ear-name" value="ear-umfsample.ear"/>
<property name="war-name" value="war-umfsample.war"/>
<property name="ejb-name" value="ejb-enrollment-
umfsample.jar"/>
```

To build the custom version of UMFsample:

1. In the build.xml file, edit the **app-server-jar** property to the name and location of your app server jar. The following are examples of what you can specify for that property entry; it varies depending on where you install WebLogic or WebSphere.

   On Windows 2000 you can specify:

   ```
   <property name="app-server-jar"
   value="C:\pub\weblogic6\wlserver6.1\lib\weblogic.jar" />
   ```

   On Solaris with WebLogic you can specify:

   ```
   <property name="app-server-jar"
   value="/pub/weblogic6/wlserver6.1/lib/weblogic.jar"/>
   ```

   On Solaris or AIX with WebSphere you can specify:

   ```
   <property name="app-server-jar"
   value="/opt/WebSphere/AppServer/lib/j2ee.jar"/>
   ```

2. In your working directory, run the command:

   ```
   ant build-all
   ```

3. The system places the new custom EAR file in the **deploy** subdirectory of your working directory.

   CAUTION: Move this EAR file to the location where you will deploy it BEFORE proceeding to the next step; cleaning up your working directory will remove the deploy subdirectory and any files it contains.

4. After the EAR file is build, clean up your working directory as follows:

   ```
   ant very-clean
   ```

Additional Reading Sources

LDAP: Lightweight Directory Access Protocol

The Lightweight Directory Access Protocol: X.500 Lite, CITI Technical Report 95-8, Timothy A. Howes, Center for Information Technology Integration, University of Michigan, 1995. http://www.stanford.edu/group/networking/directory/doc/ldap/ldap.html (Requires Login.)

Understanding LDAP – Design and Implementation, SG24-4986-01, Redbooks, June-29-1998, published 16 June 2004, last updated 19 July 2006

http://publib-b.boulder.ibm.com/cgi-bin/searchsite.cgi?query=LDAP

Using LDAP for Directory Integration, SG24-6163-01, published 12 February 2004 http://publib-b.boulder.ibm.com/cgi-bin/searchsite.cgi?query=LDAP

Introduction to Directories and LDAP, Jeff Hodges, June 1997

LDAP FAQ, Mark Wahl. **RFC2251: Lightweight Directory Access Protocol (v3)**, M. Wahl, T. Howes, and S. Kille. December 1997

RFC2252: Lightweight Directory Access Protocol (v3) Attribute Syntax Definitions, M. Wahl, A. Coulbeck, T. Howes, and S. Kille. December 1997

JSP: Java Server Pages

Java Server Pages, http://java.sun.com/products/jsp/index.html

JNDI: Java Naming and Directory Interface

Java Naming and Directory Interface™, http://java.sun.com/products/jndi/

JNDI API Tutorial and Reference: Building Directory-Enabled Java Applications (The Java Series), by Rosanna Lee, Scott Seligman. May 2001. http://java.sun.com/products/jndi/tutorial/

5 Content Access

Planning Your Content Access Interface

About Content Access

The Oracle content access interface retrieves customer content, for example statement summary or detail data, from a database or data stream and presents it in the customer's Web browser. Content access methods also support data post-processing and dynamic presentation of a user's activity on the Web, for example the date and time the user last viewed a statement. The content access interface consists of methods that extend the Java 2 Enterprise Edition (J2EE) language in an Enterprise JavaBeans (EJB) environment.

In deploying an Oracle solution, the content access interface integrates with the *user management framework* to retrieve and present account data for each enrolled customer. For more information, see Chapter 4, User Management.

As the foundation of each deployment, any content access implementation is designed for customization. The customized code of a properly implemented project will integrate seamlessly with Oracle core software.

Goals of Content Access

- Retrieve and present statement summaries and detail data
- Sort statement detail
- Record and present Web-time activity
- Extract and transform data as XML

About XML, XSL, and XSLT

eStatement Manager adds the ability to extract and transform XML data with XSL and XSLT stylesheets. The Extensible Markup Language (XML) is the universal format for structured documents and data on the Web, developed by the WorldWide Web Consortium (W3C). According to the W3C:

"XSL is a language for expressing stylesheets. It consists of three parts:

- XSL Transformations (*XSLT*): a language for transforming XML documents
- XML Path Language (*XPath*), an expression language used by XSLT to access or refer to parts of an XML document. (XPath is also used by the XML Linking specification

- XSL Formatting Objects: an XML vocabulary for specifying formatting semantics.

An *XSL stylesheet* specifies the presentation of a class of XML documents by describing how an instance of the class is transformed into an XML document that uses the formatting vocabulary."

For more information, see W3C, *The Extensible Stylesheet Language (XSL)*, http://www.w3.org/Style/XSL/

Introduction to Oracle Content Access

An Oracle Web application designer uses the content access interface to customize live data retrieval for the eStatement Manager production process. This SDK module requires a thorough understanding of the terms, processes, and workflows presented in the *Administration Guide for Oracle Siebel eStatement Manager*.

More specifically, the Oracle content access interface allows a Web application designer to control how eStatement Manager dynamically retrieves and presents data configured in *version sets* of dynamic Web *views* and *batch jobs*.

XML Views and Jobs since eStatement Manager 3.0

eStatement Manager 2.x could retrieve XML dynamically with the *XML Web view* for processing at the JSP layer. In eStatement Manager 3.0 and later versions, you can now transform XML data directly in the Command Center. You can transform extracted data with a custom XSLT stylesheet in the new XSLT Web view, or present detail, annotation, or dispute data with an XML query document in the new XML Query Web view.

eStatement Manager 2.x could also create static XML with the *XML Output batch job*. Since 3.0, eStatement Manager has a more compact and intuitive DTD structure for Mapping a DDF to XML for the XML Output job. You can also extract recurring data such as detail, disputes, or annotations, as XML with the Detail Extractor job.

About Views, Jobs, and Version Sets

A Web *view* is a set of design files that result in a particular presentation of statement data. A view can dynamically display formatted statements live on the Web or present other account data in various formats.

An eStatement Manager application can have one or more views, customized for an organization's online presentment needs. Multiple views can present different levels of statement information such as a summary page and statement detail pages.

A *version set* is a set of design files which eStatement Manager uses to present a user with an online view of a statement. When you *publish a version set* for the Indexer job, the eStatement Manager *Publisher* tool (in the Command Center) identifies each design file belonging to a particular view and moves the files from the design environment to your application server. You give the view a name, and Publisher creates the version set for further identification.

Publishing a version set of a Web view requires you to specify values for:

- DDN (same as Application)

- View Type (not required for Indexer job)
- View Name

These values must *exactly* match the values passed as parameters **ddn**, **viewName**, and **viewType** of any User methods on any JSP associated with that view.

For more information about views, jobs, and version sets, see the *Administration Guide for Oracle Siebel eStatement Manager*.

TIP: You must index your data by running an Indexer job at least once before you can view online statements in sample Web applications.

New XML Templates for Views and Jobs

eStatement Manager allows you to customize data extraction and presentment with powerful XML tools as well as with APIs. Two new dynamic Web views (XSLT View and XML Query View) and a new batch job (Detail Extractor) accept input from XML templates that can transform content during extraction, execute a SQL query, or format data, for example for download as comma-separated values (CSV).

TIP: For downloading data, use the XSLT View with XSL stylesheets instead of the CSV view with a TOK file. See Transforming Data with XSLT for details.

The following table lists the input file types for each eStatement Manager dynamic view and batch job. Use these files with the data source for the National Wireless application, **Data/NatlWireless.txt**, to practice creating your own views and jobs.

Dynamic Web Views

Job Type	View Name	NatlWireless Sample Files
HTML	User-provided	**NatlWireless.DDF** **NatlWireless.ALF** **NatlWireless.htm** **NW_LocSummary.htm**
CSV	User-provided	**NatlWireless.DDF** **NatlWireless.tok**
XML	User-provided	**NatlWireless.DDF**
CHART	User-provided	None
XSLT	XSLTDetail	**NatlWireless.DDF** **XSLTDownload/summary_info_csv.xsl**
XMLQuery	DetailQuery	**XMLQuery/annot_sql.xml** **XMLQuery/detail_sql.xml** **XMLQuery/dispute_sql.xml**

Batch Jobs

Job Type	View Name	National Wireless Example
Detail Extractor	dtlextr	**NatlWireless.DDF** **DetailExtractor/summary_info.xml** **DetailExtractor/summary_info.xsl**
Email Notification	User-provided	**NatlWireless.DDF** **NW_Email.ALF** **NWEmail.htm** **NWEmailAlternate.htm**
HTML Output	User-provided	**NatlWireless.DDF** **NatlWireless.ALF** **NatlWireless.htm**
Indexer	User-provided	**NatlWireless.DDF**
Purge App	User-provided	None
Purge Logs	User-provided	None
XML Output (XMLDetail	User provided	**NatlWireless.DDF**

For a complete listing of National Wireless sample files, see the *Deploying and Customizing J2EE Applications Guide for Oracle Siebel eStatement Manager*.

For information on creating custom data definition files (DDF) and application logic files (ALF) for your dataset, see the *Data Definition (DefTool) Guide for Oracle Siebel eStatement Manager* and the *Presentation Design (Composer Guide) for Oracle Siebel eStatement Manager*.

For information on creating and configuring each view type, see the *Administration Guide for Oracle Siebel eStatement Manager*.

Command Line Interface (CLI) to Scheduler (PWC)

`com.edocs.pwc.cli.CLIScheduler` is a command line interface to the PWC. You can use it to start PWC jobs from the command line to allow the use of external (third-party) schedulers like cron on UNIX, or CA Unicenter. It can also list all the DDNs, job names and the schedules defined.

To run the job in a particular DDN:

```
java com.edocs.pwc.cli.CLIScheduler -start <DDN> <jobname>
```

To list all the DDNs and the job names defined for each DDN:

```
java com.edocs.pwc.cli.CLIScheduler -list
```

To list all the DDNs, the job names defined for each DDN, and the schedules defined for each job name:

```
java com.edocs.pwc.cli.CLIScheduler –schedules
```

Return Codes	Status
0	The job ran successfully.
1	The job resulted in a NoOp.
2	The job Failed.
3	An instance of this job is in Failed, Processing, Reprocessing, or Reprocess state.
-1	If the DDN, job name are unknown.

Examples of each command are described below.

com.edocs.pwc.cli.CLIScheduler -start <DDN> <jobname>

This command lets you start PWC jobs, where <DDN> is the DDN name and <jobname> is the name of the job.

WebLogic/Solaris Example:

```
EDX_HOME=/opt/estatement

WL_HOME=<BEA Home Directory>

CLASSPATH=$CLASSPATH:$EDX_HOME/lib/edx_client.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/lib/edx_common.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/lib/commons-logging-1.1.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/lib/log4j-1.2.13.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/lib/concurrent-1.3.3.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/lib/spring.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/xma/api-1.1.1.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/xma/configuration-1.1.1.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/xma/platform-1.1.1.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/xma/xma-1.1.1.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/xma/xma-config.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/config

CLASSPATH=$CLASSPATH:$EDX_HOME/lib/weblogic/app-scheduler.jar


LOG_OPTS="-
Dorg.apache.commons.logging.Log=org.apache.commons.logging.imp
l.SimpleLog"

LOG_OPTS="$LOG_OPTS -
Dorg.apache.commons.logging.simplelog.defaultlog=debug"
```

```
LOG_OPTS="$LOG_OPTS -
Dorg.apache.commons.logging.simplelog.log.com.edocs.jndi.cda=d
ebug"


java -classpath $CLASSPATH:$WL_HOME/server/lib/weblogic.jar
$LOG_OPTS -Dedx.home=$EDX_HOME -
Djava.naming.factory.initial=weblogic.jndi.WLInitialContextFac
tory -Djava.naming.provider.url=iiop://<localhost>:7001 -
DUserTransaction.name=javax.transaction.UserTransaction -
DnoTransaction=true -
DPWCDataManager.name=edx/ejb/PWCDataManager -
DLogWriter.name=edx/ejb/LogWriter
com.edocs.pwc.cli.CLIScheduler -start <DDN> <jobname>
```

Where <BEA Home Directory> is the directory where you installed the application server, <DDN> is the name of the DDN, and <jobname> is the job name.

Also provide the correct URL with Application Server IP and bootstrap port for JAVA options:

```
-Djava.naming.provider.url=iiop://<localhost>:7001
```

Note that Windows uses the same WebLogic commands.

Oracle Application Server/Linux Example:

```
EDX_HOME=/opt/eStatement

export
ORACLE_J2EE_HOME=/opt/oracle/product/10.1.3/OracleAS_1/j2ee/ho
me

export JAVA_HOME=/opt/oracle/product/10.1.3/OracleAS_1/jdk

CLASSPATH=$CLASSPATH:$ORACLE_J2EE_HOME/oc4jclient.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/lib/edx_common.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/lib/edx_client.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/lib/commons-logging-1.1.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/lib/log4j-1.2.13.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/lib/spring.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/lib/concurrent-1.3.3.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/xma/api-1.1.1.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/xma/configuration-1.1.1.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/xma/platform-1.1.1.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/xma/xma-1.1.1.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/xma/xma-config.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/lib/oracleAS/app-scheduler.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/config

LOG_OPTS="-
Dorg.apache.commons.logging.Log=org.apache.commons.logging.imp
l.SimpleLog"

LOG_OPTS="$LOG_OPTS -
Dorg.apache.commons.logging.simplelog.defaultlog=info"
```

```
LOG_OPTS="$LOG_OPTS -
Dorg.apache.commons.logging.simplelog.log.com.edocs.jndi.cda=i
nfo"


$JAVA_HOME/bin/java -classpath $CLASSPATH: $LOG_OPTS -
Dedx.home=$EDX_HOME -
Djava.naming.factory.initial=oracle.j2ee.naming.ApplicationCli
entInitialContextFactory -
Djava.naming.security.principal=admin -
Djava.naming.security.credentials=welcome -
Djava.naming.provider.url=opmn:ormi://<localhost>:<oc4j_instan
cename>/<application-name> -
DUserTransaction.name=javax.transaction.UserTransaction -
DnoTransaction=true -
DPWCDataManager.name=edx/ejb/PWCDataManager -
DLogWriter.name=edx/ejb/LogWriter
com.edocs.pwc.cli.CLIScheduler  -start <DDN> <jobname>
```

In the URL, provide the correct Application Server IP for <localhost>, the name of the instance you created in Oracle Application Server for <instancename>, and the name of the application you provided during eStatement.ear deployment for <application-name>:

```
Djava.naming.provider.url=opmn:ormi://<localhost>:<oc4j_instan
cename>/<application-name> -
```

Also replace <DDN> with the name of the DDN and <jobname> with the name of the job.

WebSphere / Solaris Example:

For WebSphere, before running the script you must first extract the *app-scheduler.jar* in the eStatement ear file ($EDX_HOME/J2EEApps/websphere/Deployed_ear-eStatement.ear) to a folder called **websphere** in the lib ($EDX_HOME/lib/websphere).

Sample CLI Scheduler script:

```
export WAS_HOME=/usr/IBM/WebSphere/AppServer/profiles/XMATEST
export EDX_HOME=/edx/eStatement
. $WAS_HOME/bin/setupCmdLine.sh


WAS_INSTALL_HOME=/opt/IBM/WebSphere/AppServer


LOG_OPTS="-
Dorg.apache.commons.logging.Log=org.apache.commons.logging.imp
l.SimpleLog"
```

```
LOG_OPTS="$LOG_OPTS -
Dorg.apache.commons.logging.simplelog.defaultlog=error"

LOG_OPTS="$LOG_OPTS -
Dorg.apache.commons.logging.simplelog.log.com.edocs.jndi.cda=e
rror"


CLASSPATH=$CLASSPATH:$WAS_INSTALL_HOME/runtimes/com.ibm.ws.adm
in.client_6.1.0.jar
CLASSPATH=$CLASSPATH:$WAS_INSTALL_HOME/java/jre/lib/endorsed/i
bmorb.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/lib/edx_common.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/lib/edx_client.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/lib/commons-logging-1.1.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/lib/log4j-1.2.13.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/lib/spring.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/lib/concurrent-1.3.3.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/xma/api-1.1.1.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/xma/configuration-1.1.1.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/xma/platform-1.1.1.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/xma/xma-1.1.1.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/xma/xma-config.jar
CLASSPATH=$CLASSPATH:$EDX_HOME/config

CLASSPATH=$CLASSPATH:"$EDX_HOME/lib/websphere/app-
scheduler.jar":$WAS_CLASSPATH

CLASSPATH=$CLASSPATH:"/usr/IBM/WebSphere/AppServer/profiles/XM
ATEST/installedApps/pandoraNode04Cell/eStatement.ear/ejb-
pwc.jar"

CLASSPATH=$CLASSPATH:"/usr/IBM/WebSphere/AppServer/profiles/XM
ATEST/installedApps/pandoraNode04Cell/eStatement.ear/ejb-
alert-service.jar"


NAMING_FACTORY=com.ibm.websphere.naming.WsnInitialContextFacto
ry

ORB_RAS_MGR=-
Dcom.ibm.CORBA.RasManager=com.ibm.websphere.ras.WsOrbRasManage
r


CMD="$JAVA_HOME/bin/java \

-Xbootclasspath/p:$WAS_BOOTCLASSPATH \

$CLIENTSAS \

$CLIENTSOAP \

$JAASSOAP \

$CLIENTSSL \

$ORB_RAS_MGR \

$USER_INSTALL_PROP \
```

```
$EDX_OPTS \
$LOG_OPTS \
$DEBUG_MODE \
-Dedx.home=$EDX_HOME \
-Dwas.install.root=$WAS_HOME \
-Dws.ext.dirs=$WAS_EXT_DIRS \
-
Djava.security.auth.login.config=$WAS_HOME/properties/wsjaas_c
lient.conf \
-Dcom.ibm.CORBA.BootstrapHost=$DEFAULTSERVERNAME \
-Dcom.ibm.CORBA.BootstrapPort=$SERVERPORTNUMBER \
-
Djava.naming.factory.initial=com.ibm.websphere.naming.WsnIniti
alContextFactory \
-Djava.naming.provider.url=iiop://<localhost>:2812 \
-DUserTransaction.name=jta/usertransaction \
-classpath $CLASSPATH com.ibm.ws.bootstrap.WSLauncher
com.edocs.pwc.cli.CLIScheduler -start <DDN> <jobname>"
$CMD
```

Provide the correct URL with Application Server IP and bootstrap port for JAVA options:

```
-Djava.naming.provider.url=iiop://<localhost>:2812 \
```

Also replace <DDN> with the name of the DDN and <jobname> with the name of the job.

Other variables used in the script:

WAS_HOME = <Your WebSphere Profile Home Directory>
EDX_HOME = <eStatement Installation Home Directory>
WAS_INSTALL_HOME = <WebSphere Application Server Home Directory>
LOG_OPTS = <Simple Log Information>

com.edocs.pwc.cli.CLIScheduler -list

This command lists all the DDNs and the job names defined for each DDN, for example:

```
 TestChart
                    index
 test_app
                   Email
                   index
 test1
                  emailN
                  indexer
 cl_sp9
                  indexer
                  htmlOutput
```

com.edocs.pwc.cli.CLIScheduler -schedules

This command lists all the DDNs, the job names defined for each DDN, and the schedules defined for each job name, for example:

```
  schedulertest8
          indexer
                   StartDate : 06/12/2001
                   StartTime : 7 pm:00
                   Try Every 15mins
                   EndTime   : 2 pm:0
                   Repeat    : on day 5 of the month every
other month
                   EndDate   : Forever
  schedulertest9
          emailN
                   StartDate : 06/11/2001
                   StartTime : 5 pm:00
                   Try Every 5mins
                   EndTime   : 12 pm:15
                   Repeat    : Every
Monday,Tuesday,Wednesday,Thursday,Friday
                   EndDate   : Until   06/30/2001
          indexer
                   No Schedules defined for this jobName
  test
          indexer
                   No Schedules defined for this jobName
```

XML and eStatement Manager

About XML and eStatement Manager

In eStatement Manager, you can use XML throughout the design process: for data definition, composition, extraction, and live presentment. A DDF mapped to an XML DTD can transform data in

almost unlimited ways, replacing many complex customizations and post-processing tasks with automated Command Center views and jobs. By mapping your DDF fields to XML elements, you can apply XML templates that:

- Extract recurring detail data to the eStatement Manager database
- Transform data for download, for example as CSV or QIF
- Wrap a SQL query as XML to extract database data without an API
- Format data with XSL stylesheets
- Use XTags, XPath, and other emerging XML tools

Mapping a DDF to XML

Mapping a DDF to XML is the first step in the process of applying these powerful new tools. You need to create an XML Document Type Definition (DTD) to reflect the structure of your Data Definition File (DDF). To learn about DTDs, which are a core feature of XML, see "Additional Reading about XML, XSL, and XSLT" on page 106.

About XML DTDs for eStatement Manager

The eStatement Manager format for XML output uses DDF names as XML element names. For example, if the DDF contains a FIELD named **`AccountNumber`**, the DTD will have an element name **`AccountNumber`**, with the value of the extracted FIELD appearing in a CDATA section of that XML element.

Therefore, there is no "standard" DTD for XML in eStatement Manager—each DDF defines its own DTD. However, all eStatement Manager DTDs contain a common element, shown in this example fragment:

```
<!ELEMENT doc (view)>
<!ATTLIST doc docid ID #required>
```

The <**`view`**> element contains the complete extracted document content. The required attribute **`docid`** is the standard docid which uniquely identifies the document within the system.

These XML conventions provide a more compact and intuitive reflection of the underlying document structure, improving performance and ease of use.

Standard Elements

- If there is no data extracted for some DDF-defined item, no XML is generated.
- Space characters (' ') in DDF item names are mapped to the dash ('-') character.
- eStatement Manager does not prevent collisions among FIELD, TABLE, and GROUP names. eStatement Manager constrains DDF item names to be unique within the DDF.

The following topics describe the XML representations of common DDF object types in eStatement Manager.

FIELD Elements

A FIELD defined in a DDF is represented as an XML element with the same name as the FIELD. The extracted content is wrapped in a CDATA section of the FIELD element.

Within the DDF, one FIELD is designated as the "primary key" for the document. In the generated XML, this element has an attribute "role" with the value "PRIMARYKEY".

No other attributes are supplied. In particular, no "type" information is presented.

TABLE Elements

A TABLE defined in a DDF is represented as an XML element with the same name as the TABLE. It is a collection of rows, each of which is a collection of the columns. The element name of the columns is the same as the DDF name of the column, and the extracted data is within a CDATA section, just as a FIELD.

However, there is no DDF name for a row, so adding the string "-row" to the TABLE name creates a name for the rows. Thus, a TABLE named Detail with columns Name, Date, and Amount becomes:

```
<Detail>
<Detail-row>
<Name><![CDATA[Joe the Lion]]></Name>
<Date><![CDATA[June 1, 1974]]></Date>
<Amount><![CDATA[801]]></Amount>
</Detail-row>
</Detail>
```

GROUP Elements

A GROUP defined in a DDF is represented as an XML element with the same name as the GROUP.

Additional Reading about XML, XSL, and XSLT

Bradley, Neil, *The XSL Companion*, Addison Wesley, 2000

Burke, Eric M., *Developing, Applying and Optimizing XSLT with Java Servlets,* 12/15/2000
http://www.onjava.com/pub/a/onjava/2000/12/15/xslt_servlets.html

Fung, Khun Yee, *XSLT: Working with XML and HTML*, Addison Wesley, 2001

Holzner, Steven, *Inside XSLT*, New Riders, 201 West 103rd Street, Indiana 46290, July 2001

Sun Microsystems, *Tutorial for the Java™/ API for XML Parsing (JAXP) version 1.1*,
http://java.sun.com/xml/tutorial_intro.html

W3C, *The Extensible Stylesheet Language (XSL)*, http://www.w3.org/Style/XSL/

Extracting Detail Data to the Database

About the Detail Extractor Job

The *Detail Extractor job* extracts recurring data from the data stream and loads it into a database table. This feature supports

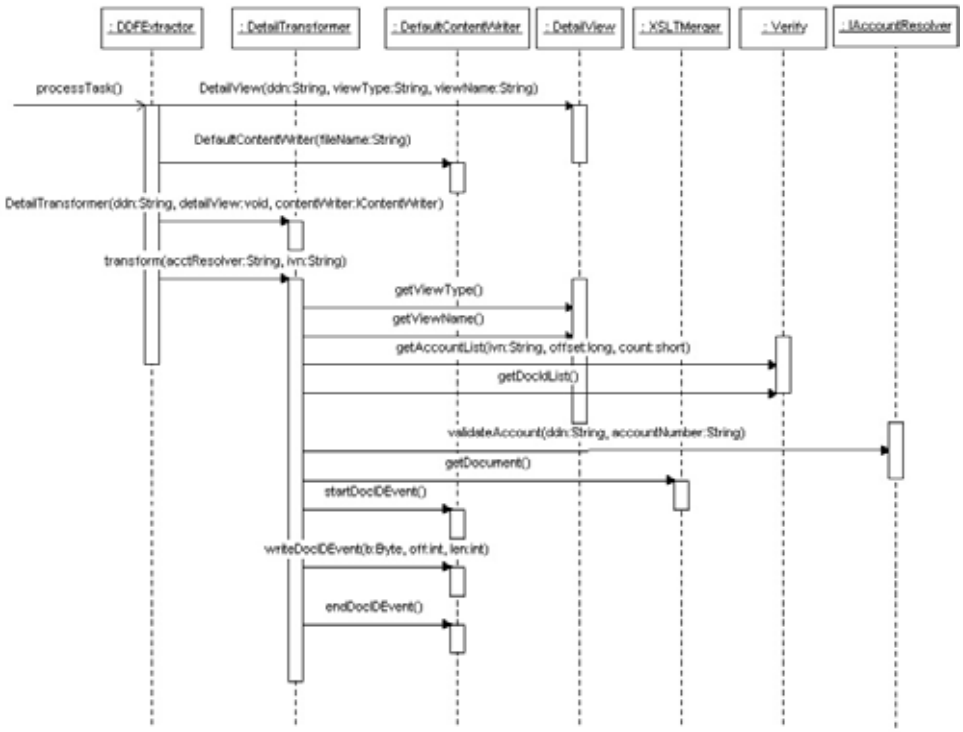
- Retrieving recurring data as XML
- XML transformations with XSLT, for example to WML, CSV, or QIF
- Integrating line item disputes and annotations
- Data mining and analysis

For information on creating and configuring a Detail Extractor job, see the *Administration Guide for Oracle Siebel eStatement Manager.* To use the Detail Extractor with disputes and annotations, see Chapter 6, "Line Item Disputes and Annotations."

This job takes several types of XML input files. Each is discussed in the following sections.

Job Type	Input Files	National Wireless Example
Detail Extractor	DDF	**`NatlWireless.DDF`**
	Database Table XML File	**`DetailExtractor/summary_info.xml`**
	Statement XSLT File	**`DetailExtractor/summary_info.xsl`**

Sequence Diagram for Detail Extractor Job



Detail Extractor Job for National Wireless

The samples provided extract and upload the **`SummaryInfo`** table from **`NatlWireless.ddf`** to the Detail table using the Detail Extractor job, and then access this Detail table using an XMLQuery dynamic view and the method **`User.getDocumentReader`**.

Customizing the Detail Extractor Job

The Detail Extractor feature requires two new tablespaces, EDX_LOAD_DATA and EDX_LOAD_DATA_IDX, which hold detail data and indexes. These are huge tablespaces requiring additional disk space. Each Command Center job adds one LOAD_DATA table to the database. To add multiple tables, you can create additional jobs.

For more information on database tables, see the *Installation Guide for Oracle Siebel eStatement Manager*.

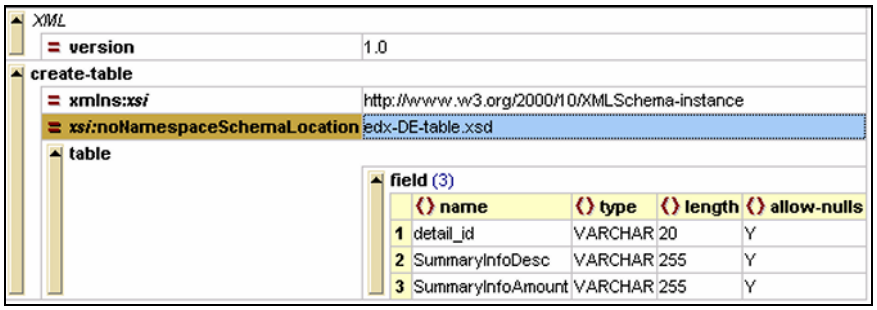
For National Wireless, the XML templates for this view are:

- **`summary_info.xml`** (Database Table XML File)
- **`summary_info.xsl`** (Statement XSLT File)

To customize this job for your own data, you must edit these two XML templates.

Edit summary_info.xml

The **summary_info.xml** template specifies the DDF Table name and the table field names as they appear in your database. Here you can also add columns to the detail table by specifying new XML field elements.



The **summary_info.xml** template also defines the parameters of the **detail_id** field, as shown in this example.

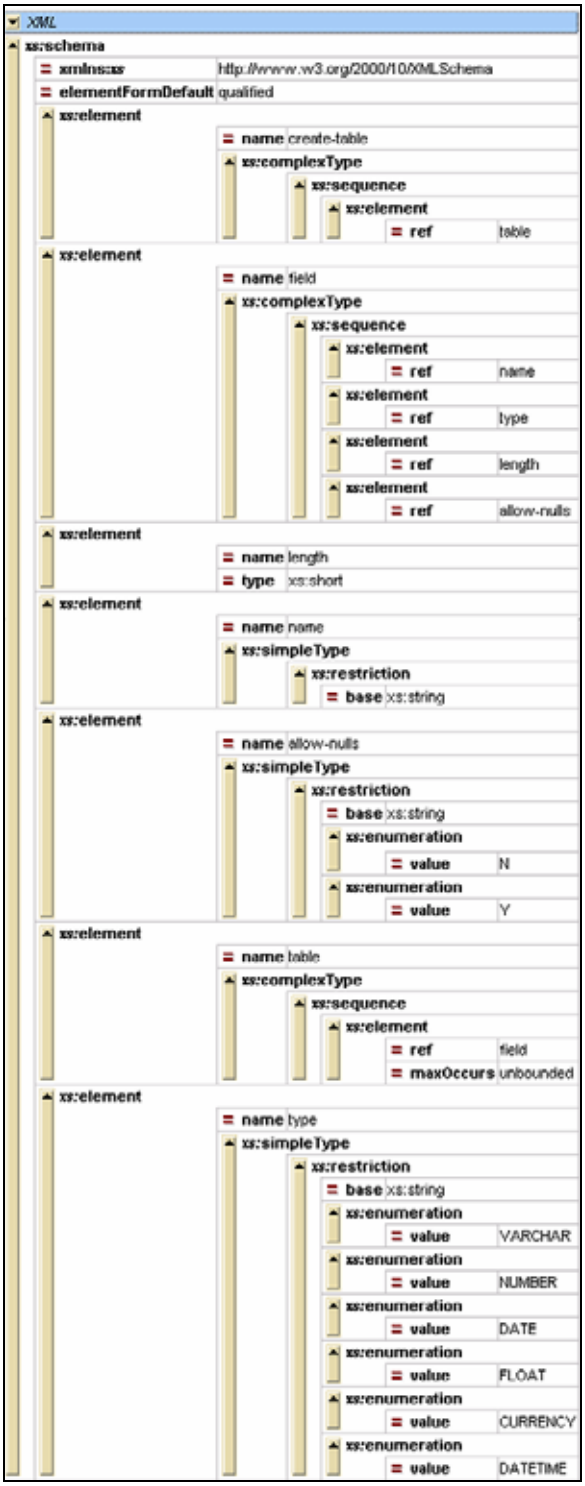
```
<field>
      <name>detail_id</name>
      <type>VARCHAR</type>
      <length>20</length>
      <allow-nulls>Y</allow-nulls>
</field>
```

TIP: The **detail_id** can grow beyond the default length of 20 characters for very large statements. If the Detail Extractor job fails when processing a large statement, it may be because the **detail_id** database field was not large enough. Try increasing the length of **detail_id** to varchar(40) in **summary_info.xml**.

About edx-DE-table.xsd

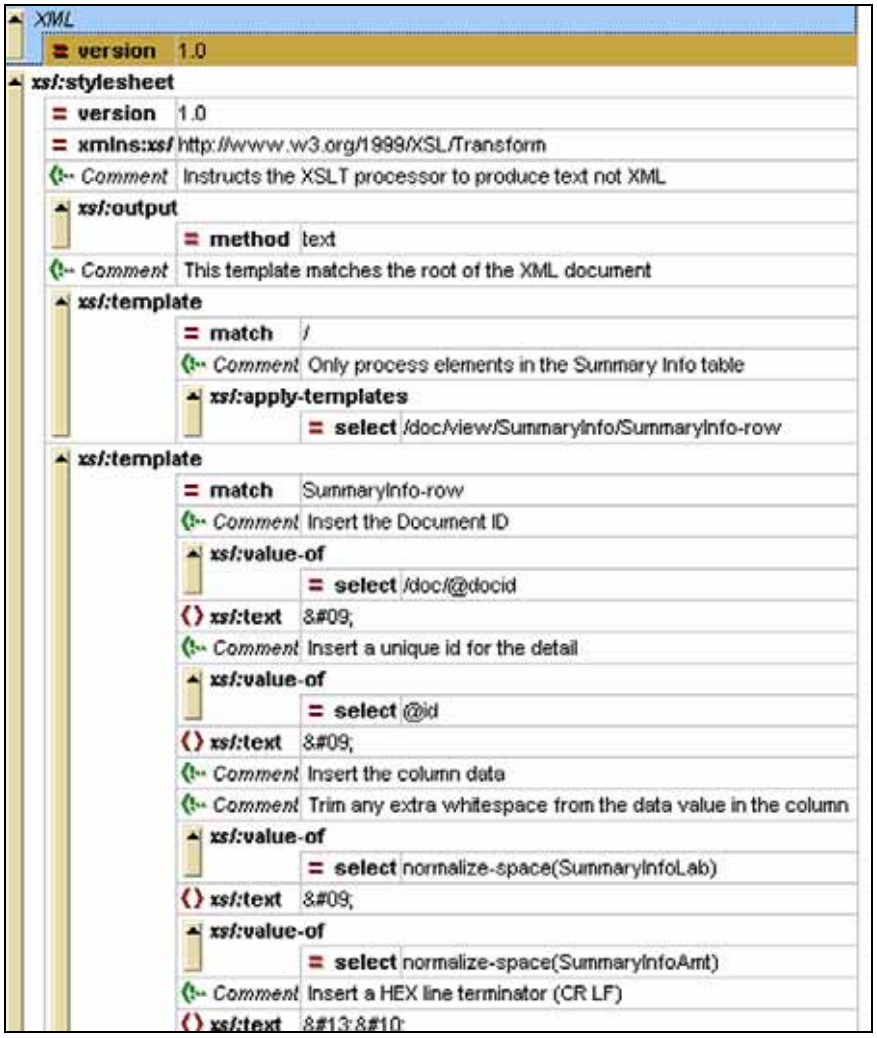
The **summary_info.xml** template uses a schema, **edx-DE-table.xsd**, to define the table data types. This file should *not* be modified, but you should update **summary_info.xml** to point to its correct location in your environment.

CAUTION: **edx-DE-table.xsd** does not use the most current version of the XML Schema declaration: http://www.w3.org/2000/10/XMLSchema is the only version supported by WebLogic 6.1sp2. WebLogic users should *not* update this schema declaration to a newer version, as this version is required by the WebLogic XML parser. WebSphere supports newer schema declarations, but *Oracle will not support implementations using newer schemas*.



Edit summary_info.xsl

The **`summary_info.xsl`** template is a stylesheet that matches the requested **`docID`** and **`detailID`** with the data from the requested database table.



Transforming Data with XSLT

About the XSLT View Type

eStatement Manager uses XML to read, write, and transform data using the universal standard of XSLT. eStatement Manager applications use the XML dynamic Web view and an XSLT stylesheet to transform data into the desired format. For example, an XSLT View could transform one XML format to another, to comma-separated values (CSV) for download, or to a proprietary format such as Quicken QIF (in text or HTML format).

The advantage of using the XSLT View is quick and easy output of different data formats from the same DDF, using the existing functionality of eStatement Manager.

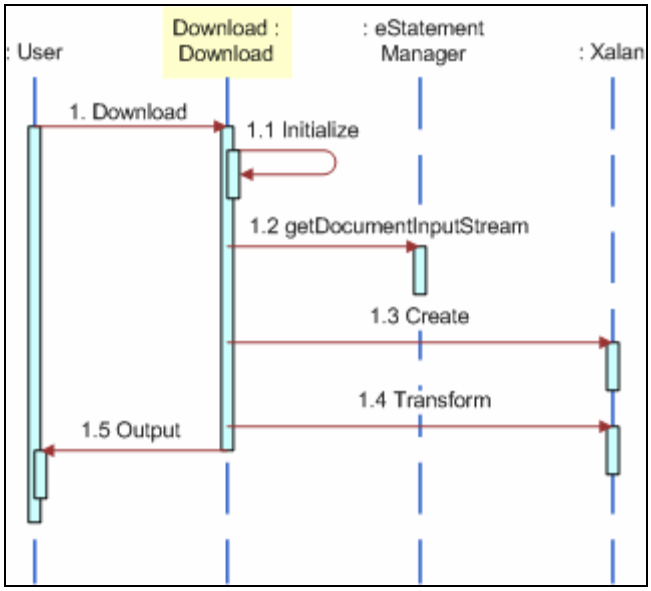
Setting Up Your Environment for XSLT

`dom4j.jar` contains the Document Object Model (DOM) Java interface for dynamic access to HTML documents. For more information, see http://www.w3.org/DOM/.

`xalan.jar` contains Apache's Java implementation of the DOM API as an XSLT stylesheet processor. eStatement Manager uses *xalan-Java-2.2 D12* to process XSLT. For more information, see http://xml.apache.org/.

`xalanj1compat.jar` contains the xalan-Java-1 XSLT processor, which is no longer supported by Apache but is required by the DOM4 API.

Sequence Diagram for XSLT View



Using the extracted details, eStatement Manager renders the XML. The extracted XML is fed into the XSL transformer Apache Xalan, which converts the data into the desired output. The transformed output is then sent to the client browser.

Input Files for XSLT View

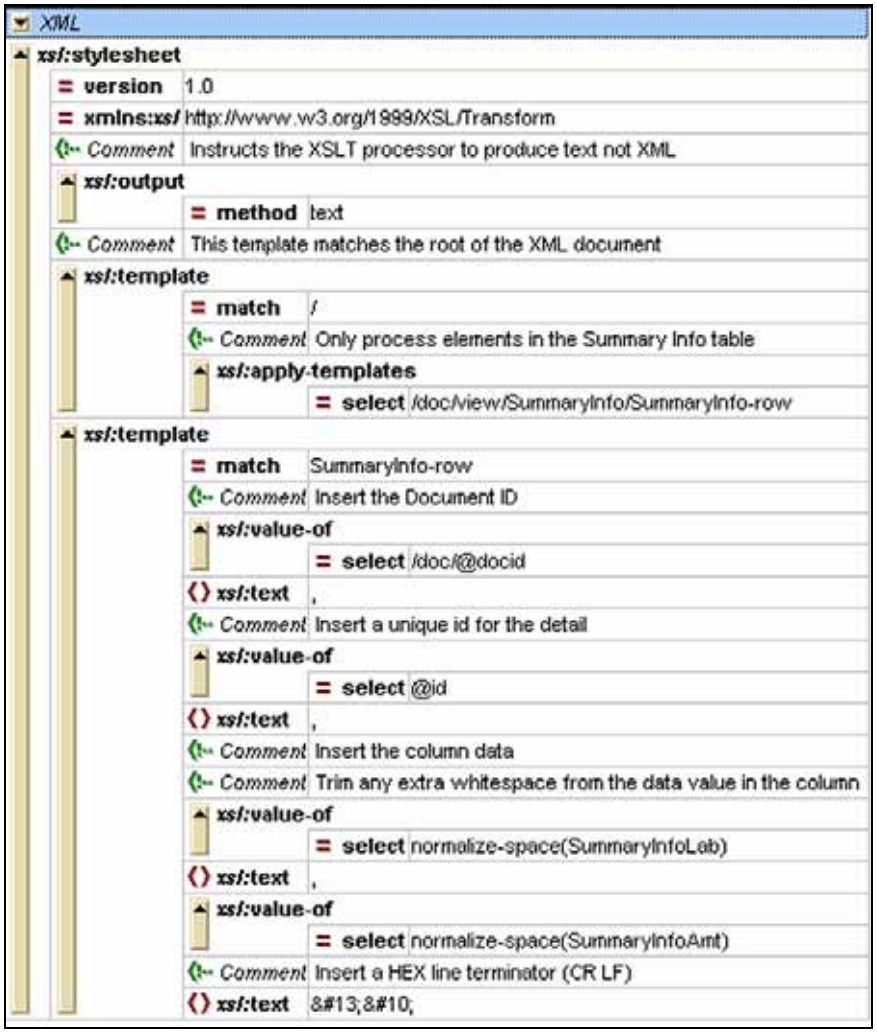
Job Type	Input Files	National Wireless Example
XSLT	DDF	**`NatlWireless.DDF`**
	XSLT	**`XSLTDownload/summary_info_csv.xsl`**

Example: Downloading Data in Comma-Separated Values (CSV) Format

The National Wireless application includes an example of an XSLT template and data to download local call detail in comma-separated values (CSV) format. To view examples, see "XML Templates for National Wireless" on page 132, and "summary_info_csv.xsl" on page 136.

Edit summary_info_csv.xsl

This template processes the elements in the specified table, in this example **`SummaryInfo`**. It selects the **`docID`** specified, inserts a unique **`detailID`**, retrieves the column data and trims any white space, and inserts a comma between values and a line feed between rows.



Sample output: summary_info_csv.txt

```
ivn-1/po-0/bc-17152/pc-7/dd-20011214,1,PREVIOUS BALANCE,285.12

ivn-1/po-0/bc-17152/pc-7/dd-20011214,2,LESS PAYMENTS APPLIED
THROUGH 03/24/01,158.37CR
```

```
ivn-1/po-0/bc-17152/pc-7/dd-20011214,3,MISCELLANEOUS
CREDIT,19.19CR

ivn-1/po-0/bc-17152/pc-7/dd-20011214,4,BEGINNING
BALANCE,107.56

ivn-1/po-0/bc-17152/pc-7/dd-20011214,5,CURRENT USAGE,.07

ivn-1/po-0/bc-17152/pc-7/dd-20011214,6,PRODUCT MONTHLY
FEES,4.95

ivn-1/po-0/bc-17152/pc-7/dd-20011214,7,CORPORATE CONNECTIONS
WAIVER,4.95CR

ivn-1/po-0/bc-17152/pc-7/dd-20011214,8,LIFELINE ASST/TELE
RELAY,0.80

ivn-1/po-0/bc-17152/pc-7/dd-20011214,9,FEDERAL ACCESS
CHARGE,8.62

ivn-1/po-0/bc-17152/pc-7/dd-20011214,10,LOCAL USAGE
CHARGE,13.68

ivn-1/po-0/bc-17152/pc-7/dd-20011214,11,LOCAL SERVICE
CHARGE,83.75

ivn-1/po-0/bc-17152/pc-7/dd-20011214,12,FEDERAL TAXES - LOCAL
SERVICE,2.62

ivn-1/po-0/bc-17152/pc-7/dd-20011214,13,STATE TAXES - LOCAL
SERVICE,2.15

ivn-1/po-0/bc-17152/pc-7/dd-20011214,14,LATE FEE,1.61

ivn-1/po-0/bc-17152/pc-7/dd-20011214,15,TOTAL CURRENT
AMOUNT,117.17
```

Extracting Data with XML Queries

About the XML Query View

The *XMLQuery* View Type allows you to extract data from the Detail, Dispute, and Annotation tables.

The version set for the XMLQuery View requires a View Name and an XML Query document, which contains the query definition for accessing Detail, Dispute or Annotation content. The National Wireless sample application dataset contains example XML Query documents for retrieving detail, dispute, and annotation data.

Job Type	Input Files	National Wireless Example
XMLQuery	XML Query	**XMLQuery/annot_sql.xml** **XMLQuery/detail_sql.xml** **XMLQuery/dispute_sql.xml**

Query Document Tag Definitions

The column names in the **ResultSet** are the tag names. All date values are converted to a Java type **long** (the number of milliseconds since the epoch).

Tag	Attribute	Required	Description	Valid values
sql-stmt	None	No	The SQL statement	User defined
query	Name	Yes	Query name	User defined
table	Name	Yes	Name of the parameter used by the caller.	User defined
table	Position	Yes	Describes the parameter position in the SQL statement.	1 to N
table	Type	Yes	Determines how to resolve the table name.	Detail, dispute, annotations
table	Viewname	Yes for detail No for D &A	Determines how to resolve the detail table name.	The name of the detail extractor view.
param	Name	Yes	Parameter name used by the caller.	User defined
param	type	Yes	Java type used to validate the input parameters	User defined
param	Position	Yes	Describes the parameter position in the SQL statement.	1 to N

Creating Custom XML Queries

Sample XML Query for Detail: detail_sql.xml

```
<query name="detail_search">
      <sql-stmt>select * from ? where z_doc_id= ?</sql-
stmt>
```

```
         <table name="detail" position="1" type="detail"
viewname="dtlextr"/>
         <param name="docid" type="java.lang.String"
position="1"/>
   </query>
```

There must be a parameter tag that contains the attribute name **docid**.

The view name attribute must be declared and the value must point to the 'detail Extractor' view name.

CAUTION: In the **DBDetail.jsp** for Sample, the XMLQuery view name has been hard coded as 'DetailQuery'. Therefore, the XMLQuery view name must be specified as 'DetailQuery' in the Publisher when creating an XMLQuery dynamic view for Sample.

Method Signature for Detail

```
User.getDocumentReader(userid, account[], ddn, view type, view
name, docid, HttpServletRequest, parameters)
        Map hm = new HashMap();
        hm.put(User.QUERY,"detail_search");
        Reader reader = User.getDocumentReader("userid", new
String[]{"0331734", ddn,          "XMLQuery",
"DetailQuery",docId,null,hm);
```

The Map object is used to hold the parameters for the query as well as the query name. (The query name is used to find the query in the query definition document.)

The User class contains a static String called 'QUERY'. The **docId** parameter is used to map to the parameter tag defined in the query definition document. The 'docid' value from the parameter is mapped to the 'docid' query definition parameter.

Paging in XML Queries

For large result sets, you typically want to retrieve XML data one page at a time rather than in a single large file. You can implement paging in XML query definitions by defining a **page set** element as part of your **query spec**, as shown in this example.

```
<query-spec>
   <paging num_of_pages="4" rows_per_page="5"/>
```

A **page set** is the maximum number of pages to be displayed from a given result set. For example, if your result set contained 20 pages of statement data, but you wanted to display only 4 pages at a time, you would set the **num_of_pages** attribute to **4**. The **rows_per_page** attribute defines the number of rows to display per page.

Next, you must pass a **page** parameter into the **Map** passed into the **User.getDocumentReader** method. The value for this key is the page number. The value type should be a `java.lang.String`.

```
Map hm = new HashMap();
hm.put(User.QUERY,"detail_search");
hm.put(User.PAGE,"3");
```

```
Reader reader = User.getDocumentReader(“userid”, new
String[]{“0331734”, ddn,  "XMLQuery",
"DetailQuery",docId,null,hm);
```

Your XML output now contains an extra section of page information.

```
<meta-view>
   <pageset>
         <current_page><![CDATA[3]]></current_page>
         <first_page><![CDATA[1]]></first_page>
         <last_page><![CDATA[5]]></last_page>
         <next><![CDATA[false]]></next>
   </pageset>
</meta-view>
```

The following table describes the valid values of Tag Definitions for paging.

Tag Name	Description	Value
first_page	The first page of set range.	First page number
current_page	The request page	Requested page number
next	Test for more pages	**true** if there are more pages, **false** if not.
last_page	The last page of set range.	Last page number

Application Programming Interfaces for Content Access

Package com.edocs.app.user Description

Provides classes **User** and **UserMain** for access to statement summary and detail data. The **User** class is the content access interface to the eStatement Manager core. Its methods retrieve and send statement data for a given user account, as well as sorting the retrieved data and updating optional fields at presentment.

TIP: eStatement Manager includes powerful XML tools for extracting, processing, and formatting data for presentment. Instead of, or in addition to, transforming or post-processing data at the JSP layer, consider customizing one of the new XML views discussed in this guide to see if XML can meet your design needs.

- Methods **getDocumentInputStream** (bytes) and **getDocumentReader** (characters) each compose a document. Each method has two signatures.
- Method **getSummary** retrieves statement summary data for a given account. This method has three signatures.
- Method **sendDocument** has four signatures that can retrieve statement data as an output stream of bytes or characters and pass it to a **java.io.writer**, for example to send the data directly to a Web browser.
- Method **updateSummaryInfo** updates live data for an optional field.

Class **UserMain** implements two interfaces. The interface **com.edocs.app.LoginRequired** asks for an account number when a customer logs in. The interface **Servlet** defines basic methods that any servlet must implement. Its methods **DoGet** and **DoPost** support, respectively, HTTP requests for GET and POST.

Using Content Access APIs

Call User Methods in Correct Sequence

User.getSummary returns **docId**, required by all other content access methods. **DocId** is then passed as an URL parameter value to **sendDocument, getDocumentInputStream,** and/or **getDocumentReader**. **User.getSummary** also returns **docDate,** required for **updateSummaryInfo.** A typical implementation invokes **User.getSummary** first and retrieves its results before calling other methods.

Retrieve and Present Statement Summaries

About getSummary

The method **User.getSummary** retrieves statement data from the list of fields defined in the DefTool and indexed when the Indexer job processes a data volume. In the **Sample** Web application (and in typical implementations of eStatement Manager), the Java Server Page **HistoryList.jsp** uses **getSummary** for live retrieval of statement summaries.

getSummary can retrieve data from an indexed volume as soon as the Indexer job completes successfully for that eStatement Manager application. The Indexer job flags the volume **accepted** by default unless the Indexer job has specified **Intercept to Verify**.

CAUTION: You should *only* choose **Intercept to Verify** when the application has implemented the **Verify** interface for auditing data volumes.

For more information on the **Intercept to Verify** option of the **AutoIndexVolAccept** task, see the *Administration Guide for Oracle Siebel eStatement Manager*.

For more information on implementing the **Verify** interface, see Chapter 7, "Auditing Datastreams,"

getSummary Signatures

Four *signatures* of **`getSummary`** support live retrieval of all available statement summary data (default), or allow you to limit retrieval by number of volume(s) and/or by date range. Each signature is discussed in more detail below.

`TIP`: When retrieving composite or compound account numbers, for example when working with subaccounts, you can use the **`javax.naming`** interface to specify the **`account`** parameter as type **`Name`** instead of type **`String`**. For more information about the **`Name`** interface, see the Javadoc.

```
getSummary(String userId, String account, String ddn)

getSummary(String userId, String account, String ddn,
short count)

getSummary(String userId, Name account, String ddn,
short count,long from,long to)

getSummary(String userId, String account, String ddn,
short count,long from,long to)
```

getSummary Parameters

All signatures of **`getSummary`** take the three base parameters in the table below.

Parameter	Description
`account`	The account numbers this customer is authorized to view. Data type may be either **`String`** or **`Name`**.
`ddn`	The DDN (Data Definition Name) of the eStatement Manager application used by a JSP that calls **`getSummary`**
`userId`	The unique user identifier obtained at login by the user management interface.

The signature to return summaries by data volume also takes the parameter **`count`**.

Parameter	Description
`count`	The maximum number of rows to return: one row for each of **`[n]`** indexed volumes of statement data.

The signature to return summaries by date range also takes the parameters **`from`** and **`to`**.

Parameter	Description
`from`	The earliest date for which to retrieve statement summaries (which is the number of milliseconds since "the epoch" beginning January 1, 1970).
`to`	The latest date for which to retrieve statement summaries (which is the number of milliseconds since "the epoch" beginning January 1, 1970).

getSummary Results

getSummary returns a table of statement summaries. The first row of the primary array (row zero) returns the names of the column headings. Each of the remaining rows (rows 1 to [**count**]) contains a secondary array with the column values of summary data for a single indexed volume (typically, one statement cycle).

The secondary array returns the summary column data. For each row, the first column returned is the ID for the indexed volume (**docId**) and the date it was indexed (**docDate**). These two unique identifiers are input parameters for all other content access methods. Each of the remaining columns (columns 3 to [**count**]) contains the data for one indexed field, followed by data for any optional fields specified (Y_1 etc.)

When configuring an Indexer job in the Command Center, the **IXLoader** task allows you to specify the number of optional fields to be included as columns in the summary table. These fields are in addition to the number of indexed fields, and appear at the end of each row. For more information about the **IXLoader** task, see the *Administration Guide for Oracle Siebel eStatement Manager.*

Typically, each data volume corresponds to an account processing cycle, so that each row presents summary data for one cycle. Different signatures of **getSummary** can limit retrieval by data volume with the [**count**] parameter, and/or by date range with the [**from**] and [**to**] parameters, discussed below.

Retrieve All Statement Summaries

This default signature of **getSummary** is provided in the **HistoryList.jsp** page of Sample. It returns summary information for a given user account for all data volumes processed by a DDN.

```
getSummary(String  userId, String account, String ddn)
```

This code snippet shows an example from **HistoryList.jsp.**

```
…//Declare multidimensional string array to hold return of
getSummary
String[][] summary = null;
 //Declare string array to hold column headings
String[] colName = null;
//declare variables to hold the total number of rows and
columns returned
//from the multidimensional array.
       int cols = 1;
        int rows = 0;
…//Call get summary
summary = User.getSummary(userId, accountNumber, ddn);
            colName = summary[0];
            cols = colName.length;
            rows = summary.length;
```

Retrieve Statement Summaries By Data Volume

Real-world applications produce many account processing cycles, and the number of rows returned by **`getSummary`** could grow enormous. An online account application can serve the customer better by retrieving a subset of data and loading it quickly for presentment. The signature

```
getSummary(String userId, String account, String ddn, short count)
```

returns **`[n]`** statements, where **`count=n`**. This signature also takes the parameter **`count`**, described in getSummary Parameters. To return all statements, set **`count=0`**.

Retrieve Statement Summaries by Date Range

A developer may also need to limit the summaries presented by date range, for example to allow a customer, or a customer service representative, to search for statements before or after a certain date. The signature

```
getSummary (String userId, String account, String ddn,
short count,long from,long to)
```

Returns summaries within a given date range specified by **`from`** and **`to`**. This signature also takes the parameters **`from`** and **`to`**, described in getSummary Parameters. This code snippet shows an example of how to retrieve statements by date range.

```
short countNum =0;
            java.util.Date now = new java.util.Date();
            DateFormat myformat = new
SimpleDateFormat("yyyy.MM.dd");
            java.util.Date mydate=
myformat.parse("2002.02.11");
            long from= mydate.getTime();
            long to = now.getTime();
            summary = User.getSummary(userId, accountNumber,
ddn, countNum,  from, to);
            colName = summary[0];
            cols = colName.length;
            rows = summary.length;
```

Retrieve and Present Statement Detail

Once a statement summary is presented, a logical next step is to allow the customer to select that statement and drill down to view statement detail. The content access method **`User.sendDocument`** supports live retrieval of statement detail.

`CAUTION`: **`sendSortedDocument`** is deprecated in 3.0 and later versions.

The eStatement Manager Composer has powerful sorting and subtotaling tools for tables and groups, and the Command Center can retrieve and present line item detail with customizable XML and XSLT templates.

For more information about sorting statement data, see the *Presentation Design (Composer Guide) for Oracle Siebel eStatement Manager.*

For more information about retrieving statement detail, see the XML chapters of this chapter and Chapter 6, "Line Item Dispute and Annotations."

About sendDocument

SendDocument retrieves detail for presentment as it appears in the source file, without sorting or post-processing.

sendDocument Signatures

Eight *signatures* of **sendDocument** can return data as an output stream of bytes or characters and pass it to a **java.io.writer**, for example to send the data directly to a Web browser. Signatures may be used with or without an HTTP servlet request object and with or without map parameters.

TIP: When retrieving composite or compound account numbers, for example when working with subaccounts, you can use the **javax.naming** interface to specify the **accounts** parameter as type **Name** instead of type **String**. For more information about the **Name** interface, see the Javadoc.

```
sendDocument(String userId, Name[] accounts, String ddn,
String viewType, String viewName, String docId,
HttpServletRequest req, OutputStream out, Map parameters)

sendDocument(String userId, Name[] accounts, String ddn,
String viewType, String viewName, String docId,
HttpServletRequest req, Writer out, Map parameters)

sendDocument(String userId, String[] accounts, String ddn,
String viewType, String viewName, String docId,
HttpServletRequest req, OutputStream out)

sendDocument(String userId, String[] accounts, String ddn,
String viewType, String viewName, String docId,
HttpServletRequest req, OutputStream out, Map parameters)

sendDocument(String userId, String[] accounts, String ddn,
String viewType, String viewName, String docId,
HttpServletRequest req, Writer out)

sendDocument(String userId, String[] accounts, String ddn,
String viewType, String viewName, String docId,
HttpServletRequest req, Writer out, Map parameters)

sendDocument(String userId, String[] accounts, String ddn,
String viewType, String viewName, String docId, OutputStream
out)

sendDocument(String userId, String[] accounts, String ddn,
String viewType, String viewName, String docId, Writer out)
```

SendDocument Parameters

Parameter	Description
accounts	The account numbers this customer is authorized to view. Data type may be either **String** or **Name**.
ddn	The DDN (Data Definition Name) of the eStatement Manager application used by a JSP that calls **getSummary**
userId	The unique user identifier obtained at login by the user management interface, for example **IAccount**.
viewType	As specified in the Indexer job for the DDN you are using. Includes: CSV, XML and HTML.
viewName	As specified in the Indexer job for the DDN you are using.
docId	The unique identifier for the document, used to retrieve it from the source data stream.
req	The HTTP Servlet Request object from a servlet or JSP.
map	Map object holding the parameters for an XML query as well as the query name in the XML Query definition document.

DocId is obtained from a **getSummary** call in a separate JSP and passed as an URL parameter value to **sendDocument**. Also, the parameters **ddn**, **viewType**, and **viewName** specified in **sendDocument** must exactly match the values defined in the Publisher when creating the Web view, or the method throws an unspecified exception.

SendDocument Results

This default signature of **sendDocument** is provided in the **Detail.jsp** page of Sample.

```
String viewType = request.getParameter("viewType");
        String viewName = request.getParameter("viewName");
        String docId = request.getParameter("docId");
   if (viewType == null)
       viewType = "HTML";
        if (viewName == null)
       viewName = "HtmlDetail";
        if (viewType.equals("CSV"))
            response.setContentType("application/x-msexcel");
        else if (viewType.equals("XML"))
            response.setContentType("text/xml");
        User.sendDocument(userId,
                          accounts,
```

```
ddn,
viewType,
viewName,
docId,
request,
out);
```

About getDocumentInputStream and getDocumentReader

Two content access methods allow you to retrieve statement data as a byte or character stream that can then be modified before presentment in the JSP. **`getDocumentInputStream`** returns a byte stream, while **`getDocumentReader`** returns a character stream, or reader object. Each method has three signatures, with or without an HTTP servlet request object and with or without map parameters.

TIP: When retrieving composite or compound account numbers, for example when working with subaccounts, you can use the **`javax.naming`** interface to specify the **`account`** parameter as type **`Name`** instead of type **`String`**. For more information about the **`Name`** interface, see the Javadoc.

getDocumentInputStream Signatures

```
getDocumentInputStream(String userId, Name[] accounts, String
ddn, String viewType, String viewName, String docId,
HttpServletRequest req, Map parameters)
```

```
getDocumentInputStream(String userId, String[] accounts,
String ddn, String viewType, String viewName, String docId)
```

```
getDocumentInputStream(String userId, String[] accounts,
String ddn, String viewType, String viewName, String docId,
HttpServletRequest req)
```

```
getDocumentInputStream(String userId, String[] accounts,
String ddn, String viewType, String viewName, String docId,
HttpServletRequest req, Map parameters)
```

Parameter	Description
`accounts`	The account numbers this customer is authorized to view. Data type may be either **`String`** or **`Name`**.
`ddn`	The DDN (Data Definition Name) of the eStatement Manager application used by a JSP that calls **`getSummary`**
`userId`	The unique user identifier obtained at login by the user management interface, for example **`IAccount`**.
`viewType`	As specified in the Indexer job for the DDN you are using. Includes: CSV, XML and HTML.
`viewName`	As specified in the Indexer job for the DDN you are using.
`docId`	The unique identifier for the document, used to retrieve it from the source data stream.

req	The HTTP Servlet Request object from a servlet or JSP.
map	Map object holding the parameters for an XML query as well as the query name in the XML Query definition document.

getDocumentReader Signatures

```
getDocumentReader(String userId, Name[] accounts, String ddn,
String viewType, String viewName, String docId,
HttpServletRequest req, Map parameters)

getDocumentReader(String userId, String[] accounts, String
ddn, String viewType, String viewName, String docId)

getDocumentReader(String userId, String[] accounts, String
ddn, String viewType, String viewName, String docId,
HttpServletRequest req)

getDocumentReader(String userId, String[] accounts, String
ddn, String viewType, String viewName, String docId,
HttpServletRequest req, Map parameters) getDocumentInputStream
Parameters
```

getDocumentReader Parameters

Parameter	Description
accounts	The account numbers this customer is authorized to view. Data type may be either **String** or **Name**.
ddn	The DDN (Data Definition Name) of the eStatement Manager application used by a JSP that calls **getSummary**
userId	The unique user identifier obtained at login by the user management interface, for example **IAccount**.
viewType	As specified in the Indexer job for the DDN you are using. Includes: CSV, XML and HTML.
viewName	As specified in the Indexer job for the DDN you are using.
docId	The unique identifier for the document, used to retrieve it from the source data stream.
req	The HTTP Servlet Request object from a servlet or JSP.
map	Map object holding the parameters for an XML query as well as the query name in the XML Query definition document.

Retrieve and Present XML

eStatement Manager includes powerful XML tools for extracting, processing, and formatting data for presentment. Instead of, or in addition to, modifying or post-processing data at the JSP layer, consider customizing one of the new XML views discussed in this guide to see if XML can meet your design needs.

Method Signatures for XML

The **com.edocs.user.User** class contains a static String called 'QUERY'. The **docId** parameter maps to the parameter tag defined in the XML Query definition document. The **Map** object holds the parameters for the query as well as the query name, used to find the query in the XML Query definition document.

New signatures of three Content Access methods use **Map parameters**:

```
public static Reader getDocumentReader(String userId, String[]
accounts, String ddn, String viewType, String viewName, String
docId, HttpServletRequest req, Map parameters) throws
Exception

public static InputStream getDocumentInputStream(String
userId, String[] accounts, String ddn, String viewType, String
viewName, String docId, HttpServletRequest req, Map
parameters), throws Exception ,

public static void sendDocument(String userId, String[]
accounts, String ddn, String viewType, String viewName, String
docId, HttpServletRequest req, Writer out, Map parameters)
```

user.getDocumentReader Example to Retrieve Detail Data

```
User.getDocumentReader(userid, account[], ddn, view type, view
name, docid, HttpServletRequest, parameters)

    Map hm = new HashMap();

    hm.put(User.QUERY,"detail_search");

    Reader reader = User.getDocumentReader(“userid”, new
String[]{“0331734”, ddn,      "XMLQuery",
"DetailQuery",docId,null,hm);
```

user.getDocumentReader Signature to Retrieve Annotation Data

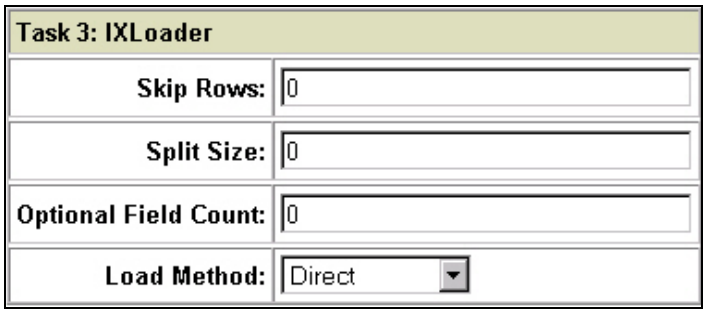
```
    Map hm = new HashMap();

    hm.put(User.QUERY,”annote_search ");

    Reader reader = User.getDocumentReader(“userid”, new
String[]{“0331734”, ddn,      "XMLQuery",
"DetailQuery",docId,null,hm);
```

user.getDocumentReader Example to Retrieve Dispute Data

```
User.getDocumentReader(userid, account[], ddn, view type, view
name, docid, HttpServletRequest, parameters)

    Map hm = new HashMap();

    hm.put(User.QUERY,"dispute_search");

    Reader reader = User.getDocumentReader(“userid”, new
String[]{“0331734”, ddn,      "XMLQuery",
"DetailQuery",docId,null,hm);
```

Record and Present Web-Time Activity

Making a Web view dynamic usually includes more than live retrieval of statement data. eStatement Manager can record and present information captured directly from a customer's activity on the Website, for example the last time the customer viewed a bill. This optional information, which must be a string, is stored in optional fields of the eStatement Manager database, which you can specify in the **IXLoader** task of the Indexer job. For more information about **IXLoader** and the Indexer job, see the *Administration Guide for Oracle Siebel eStatement Manager.*



About UpdateSummaryInfo

updateSummaryInfo provides a user interface to update optional fields through an HTML form in a JSP. By default, these optional fields are named **Y_1, Y_2, …Y_[n],** where **[n]** is the Optional Field Count specified in the **IXLoader** task.

Z_DOC_ID	Z_DOC_DATE	Due Date	Amount	View Status (Y_1)	Y_2
ABD983	01/01/00	01/28/00	$21.86	Viewed	true

This code snippet shows an example of how to update the database Y-1 field to "Updated."

TIP: Y_1 fields cannot be renamed in the database. However, you can (and should) modify a JSP to present the appropriate HTML, such as "Date Viewed," "Date Paid," or "Updated," for the corresponding Y_1 field.

```
/*Make a call to updateSummaryInfo to stamp "Updated" in Y_1
field*/
User.updateSummaryInfo(userId,
 accounts[0],
 DDN,
 docId,
 docDate,
 "Y_1",
 "Updated")
```

Another use of an optional field would be to store user comments from a form.

```
/*Make a call to updateSummaryInfo to update optional field
with user's comments entered into an HTML form and stored in a
string variable called strComments */
```

```
User.updateSummaryInfo(userId,
 accounts[0],
 DDN,
 docId,
 docDate,
 “Y_1”,
 strComments)
```

Both **column_name** and **value** must be string values. You must, for example, convert a date value to a string in order to store it in a Y_1 field.

UpdateSummaryInfo Signature

```
updateSummaryInfo (String userId, String account, String ddn,
String docId, String docDate, String name, String value)
```

updateSummaryInfo takes two parameters, **column_name** and **value**, described below.

UpdateSummaryInfo Parameters

Parameter	Description
column_name	Name of column to be updated. (example: Y_1)
value	The value with which to update **column_name**.

updateSummaryInfo requires the two parameters **DocId** and **DocDate**. **DocId** is returned from **getSummary** and passed by default in the URL. **DocDate** is returned from **getSummary** but is *not* passed by default in the URL. **docId** is the zeroth column element of a row, and **docDate** is the first column element of the row. Y_1 is the first of the optional fields.

Element ID and Composition Hints

About Element ID

eStatement Manager includes a rich layer of metadata called *Composition Hints* to influence the extraction and composition of document data. This XML-based “language” passes context and data through the eStatement Manager composition layer, from the eStatement Manager core classes through a dynamic URL to the composed HTML in a Web browser. One of these Hints is the unique identifier *element ID*. This metadata flag links live data entry to individual elements in a browser page, and retrieves that data again when the page is dynamically composed.

Element ID is a Composition Hints flag that determines whether individual elements of a Data Definition File (DDF), are assigned a unique ID when the data is extracted at runtime. The element ID

of an individual FIELD, TABLE or ROW, or GROUP element can then be correctly identified and reliably retrieved from the data store. Additional data, for example the "note" text of a dispute or annotation, can then be associated with the Element ID of a line item. At runtime, the "note" is retrieved and presented along with the detail. In dynamic database tables, using Element ID as a foreign key between detail elements and their disputes and annotations improves performance by reducing file size in the database, in the XML, and in the composed HTML.

Each Element ID is unique within its Document Definition Name (DDN). When the document is composed, an eStatement Manager application applies ALF application, or business, logic rules (ALF) to a data definition file (DDF), and assigns the resulting data a unique DOCID. This unique combination of DOCID and Element ID ensures, for example, that an annotation on the third line of a March statement will continue to appear on the March statement and not appear on the third line of the April statement, even when eStatement Manager runs again in April.

For each DDF, Generate Element ID is turned on or off in the *Edit Properties* dialog of the DefTool. This assures that the ID remains unique and stable for each DDF, regardless of the view used or the composition tools (paging, sorting, filtering, dynamic pattern matching) used for presentment.

Syntax for Element ID

Element IDs contain metadata that uniquely identifies each element by its location within a document and its element type. Element ID fields are text strings, which may not have leading zeroes, are of varying width, and may contain no embedded blanks. They are composed of up to five possible values:

Value	Description
P	Page number within the document
X	Column number of the element within the page
Y	Row number of the element within the page
Z	Index of element in list of that element type in DDF.
A	Index of row within table. Unique, but not necessarily zero or incremented.

The eStatement Manager core classes maintain a separate list of Fields, Records, and RecordGroups in each DDF. The Z value is the index in that list for each Field, Record, and RecordGroup.

Element	ID Syntax	Example
RecordGroup	P-X-Y-GZ	1-23-4-G4
Record	P-X-Y-TZ	2-2-14-T0
Record Line	P-X-Y-TZ-A	2-2-14-T0-4
Field	P-X-Y-FZ	4-13-4-F1

Note that RecordLine has the same format as Record, with row number appended.

Tag Attributes for Element ID

Tag Attributes in composed HTML

Element	Tag	Example	Since 3.0
RecordGroup	**edx_group**	<edx_group name="Registration" ID="1-23-4-G4">	Added attribute “ID”
Record	**edx_table**	<edx_table name="Registration" ID="2-2-14-T0">	Added attribute “ID”
RecordLine	HTML rows as <tr>	<tr ID="2-2-14-T0-4">	Modified <tr> tag; added attribute “ID”for each row within **edx_table** tag
Field	**FIELD**	<SPAN ID="4-223-36-F195">2000</SPAN>	Added tag with attribute ID (innermost tag for any formatting)

To use HTML Formatting with Tags:

The tag for Element ID becomes the innermost tag, in addition to any other formatting for the field. The following HTML table formatting:

```
<TD width=440><FONT face="" size=2>2000</FONT></TD>
```

becomes, for example:

```
<TD width=440><FONT face="" size=2><SPAN ID="4-3-3-
F1">2000</SPAN></FONT></TD>
```

Tag Attributes in composed XML

Element	Tag	Example	Since 3.0
RecordGroup	GROUP	<GROUP name="Registration" ID="1-23-4-G4">	Added attribute “ID”
Record	TABLE	<TABLE name="Registration" number="0" ID="2-2-14-T0">	Added attribute “ID”
RecordLine	ROW	<ROW ID="2-2-14-T0-4">	Added attribute “ID” for each row within the TABLE tag
Field	FIELD	<FIELD name="Registration" ID="1-23-4-F4">	Added attribute “ID”

Values for Composition Hints Language

Hint	Description
docid	Unique document identifier associated with the document.
ddn	Document Definition Name associated with the data stream.
elink	Value used to compose the email notification link.
pattern	Name/value pairs used to assign values to “variable fields” during extraction for Dynamic Pattern Matching.
env	Name/value pairs used to pass values from the JSP environment to the composition layer of the core classes.
table	Specifies that the indicated table only is to be composed, regardless of ALF or other composition logic. May be null.
delimiter	Specifies a delimiter value to separate columns in a selected table. May be null.
page	Identifies the page to be composed. May be null (defaults to 1).
elementid	Determines whether to generate IDs for detail elements (fields, tables, groups).

DTD for Composition Hints Language

```
<!-- DTD for composition-hints language -->
<!-- Version 1.0 rla 10/3/00 -->
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE composition-hints [
<!ELEMENT composition-hints
(docid,ddn,elink?,page?,subview?,elementid?,pattern*)>
<!ELEMENT docid EMPTY>
<!ATTLIST docid value CDATA #REQUIRED>
<!ELEMENT elink (#PCDATA)>
<!ELEMENT chartpath (#PCDATA)>
<!ELEMENT imagepath (#PCDATA)>
<!ELEMENT env EMPTY>
<!ATTLIST env name ID #REQUIRED>
<!ATTLIST env value CDATA #REQUIRED>
<!ELEMENT ddn EMPTY>
<!ATTLIST ddn value ID #REQUIRED>
<!ELEMENT pattern EMPTY>
```

```
<!ATTLIST pattern name ID #REQUIRED
                  value CDATA #REQUIRED>
<!ELEMENT table (#PCDATA)>
<!ELEMENT delimiter (#PCDATA)>
<!ELEMENT page (#PCDATA)>
<!ELEMENT elementid>
<!ATTLIST elementid value (on|off) #REQUIRED>
]>
```

XML Example of Composition Hint

```
<composition-hints>
  <elink>
<![CDATA[http://www.edocs.com/eBills/stuff.jsp&bing=bang]]></e
link>
  <docid value="ivn-33/po=1034/pc=3/dd-20000429"/>
  <ddn value="beco"/>
  <pattern name="zzz" value="yyy"/>
  <pattern name="abc" value="xyz"/>
  <table value = “CallDetail”/>
  <delimiter value = “”/>
  <page value = “1”/>
  <elementid value="on"></elementid>
</composition-hints>
```

TIP: Note that the Composition Hints DTD does not use a validating parser—the generated hint text must merely be well formed XML. Although in this example the **elink** element is out of order, the Hint is still valid.

XML Templates for National Wireless

Detail Extractor

Files are located in **eStatement/samples/NatlWireless/DetailExtractor.**

summary_info.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
   <!-- Instructs the XSLT processor to produce text not XML -
->
   <xsl:output method="text"/>

   <!-- This template matches the root of the XML document -->
   <xsl:template match="/">
         <!-- Only process elements in the Summary Info table
-->
         <xsl:apply-templates
select="/doc/view/SummaryInfo/SummaryInfo-row"/>
   </xsl:template>

   <xsl:template match="SummaryInfo-row">
         <!-- Inserts the Document ID -->
         <xsl:value-of select="/doc/@docid"/>
         <xsl:text>&#09;</xsl:text>

         <!-- Inserts a unique id for the detail -->
         <xsl:value-of select="@id"/>
         <xsl:text>&#09;</xsl:text>

         <!-- Inserts the column data -->
         <!-- Trims any extra whitespace from the data value
in the column -->
         <xsl:value-of select="normalize-
space(SummaryInfoLab)"/>
         <xsl:text>&#09;</xsl:text>
         <xsl:value-of select="normalize-
space(SummaryInfoAmt)"/>

         <!-- Inserts a HEX line terminator (CR LF)-->
         <xsl:text>&#13;&#10;</xsl:text>
   </xsl:template>
</xsl:stylesheet>
```

summary_info.xml

```
<?xml version="1.0"?>
```

```
<create-table xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="edx-DE-table.xsd">
   <table>
         <field>
               <name>detail_id</name>
               <type>VARCHAR</type>
               <length>20</length>
               <allow-nulls>Y</allow-nulls>
         </field>
         <field>
               <name>SummaryInfoDesc</name>
               <type>VARCHAR</type>
               <length>255</length>
               <allow-nulls>Y</allow-nulls>
         </field>
         <field>
               <name>SummaryInfoAmount</name>
               <type>VARCHAR</type>
               <length>255</length>
               <allow-nulls>Y</allow-nulls>
         </field>
   </table>
</create-table>
```

edx-DE-table.xsd

```
<?xml version="1.0" encoding="UTF-8"?>


<xs:schema xmlns:xs="http://www.w3.org/2000/10/XMLSchema"
elementFormDefault="qualified">
   <xs:element name="create-table">
         <xs:complexType>
               <xs:sequence>
                     <xs:element ref="table"/>
               </xs:sequence>
         </xs:complexType>
   </xs:element>
   <xs:element name="field">
```

```
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="name"/>
                        <xs:element ref="type"/>
                        <xs:element ref="length"/>
                        <xs:element ref="allow-nulls"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="length" type="xs:short"/>
      <xs:element name="name">
            <xs:simpleType>
                  <xs:restriction base="xs:string">
                  </xs:restriction>
            </xs:simpleType>
      </xs:element>
      <xs:element name="allow-nulls">
            <xs:simpleType>
                  <xs:restriction base="xs:string">
                        <xs:enumeration value="N"/>
                        <xs:enumeration value="Y"/>
                  </xs:restriction>
            </xs:simpleType>
      </xs:element>
      <xs:element name="table">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="field"
maxOccurs="unbounded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="type">
            <xs:simpleType>
                  <xs:restriction base="xs:string">
                        <xs:enumeration value="VARCHAR"/>
```

```
                        <xs:enumeration value="NUMBER"/>
                        <xs:enumeration value="DATE"/>
                        <xs:enumeration value="FLOAT"/>
                        <xs:enumeration value="CURRENCY"/>
                        <xs:enumeration value="DATETIME"/>

                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
        </xs:schema>
```

XSLT Download

sampleapplication/NatlWireless/XSLTDownload/summary_info_csv.xsl

summary_info_csv.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">


   <!-- Instructs the XSLT processor to produce text not XML -
->
   <xsl:output method="text"/>


   <!-- This template matches the root of the XML document -->
   <xsl:template match="/">
               <!-- Only process elements in the Summary Info
table -->
               <xsl:apply-templates
select="/doc/view/SummaryInfo/SummaryInfo-row"/>
   </xsl:template>


   <xsl:template match="SummaryInfo-row">
         <!-- Insert the Document ID -->
         <xsl:value-of select="/doc/@docid"/>
         <xsl:text>,</xsl:text>
```

```
        <!-- Insert a unique id for the detail -->
        <xsl:value-of select="@id"/>
        <xsl:text>,</xsl:text>

        <!-- Insert the column data -->
        <!-- Trim any extra whitespace from the data value in
the column -->
        <xsl:value-of select="normalize-
space(SummaryInfoLab)"/>
        <xsl:text>,</xsl:text>
        <xsl:value-of select="normalize-
space(SummaryInfoAmt)"/>

        <!-- Insert a HEX line terminator (CR LF)-->
        <xsl:text>&#13;&#10;</xsl:text>
   </xsl:template>
</xsl:stylesheet>
```

XML Query View

detail_sql.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<query-spec>
   <data_source_type>SQL</data_source_type>
   <paging num_of_pages="4" rows_per_page="5"/>

   <query name="detail_search">
        <sql-stmt>select * from ? where z_doc_id = ?</sql-
stmt>
        <table name="detail" position="1" type="detail"
viewname="dtlextr"/>
      <param name="docid" type="java.lang.String"
position="1"/>
   </query>
</query-spec>
```

Example DDF to XML Mapping

The following example is the XML output format from the National Wireless application.

```
<?xml version="1.0" encoding="UTF-8"?>
<doc docid="ivn-101/po-0/bc-0/pc-27/dd-20000101" >
<view>
<AcctNum role="PRIMARYKEY" id="1-73-2-
F0"><![CDATA[0331734]]></AcctNum>
<CurrentCharges id="2-4-25-F2"
><![CDATA[117.17]]></CurrentCharges>
<PymtTxt id="2-38-26-F3" ><![CDATA[PLEASE PAY THIS
AMOUNT]]></PymtTxt>
<CustName id="1-4-0-F6" ><![CDATA[BILLS BICYCLES]]></CustName>
<LateFee id="3-26-4-F8" ><![CDATA[1.50%]]></LateFee>
<AmountDue id="1-4-28-F10" ><![CDATA[224.73]]></AmountDue>
<EastState id="1-15-2-F13" ><![CDATA[MA]]></EastState>
<CustType id="1-8-48-F20" ><![CDATA[B2C]]></CustType>
<DueDate id="1-37-25-F21" ><![CDATA[04/19/01]]></DueDate>
<CustAddress id="1-0-0-T0" >
<CustAddress-row id="1-0-0-T0-0">
<CustAddressLine><![CDATA[BILLS BICYCLES]]></CustAddressLine>
</CustAddress-row>
<CustAddress-row id="1-0-0-T0-1">
<CustAddressLine><![CDATA[44 HOLLY ST]]></CustAddressLine>
</CustAddress-row>
<CustAddress-row id="1-0-0-T0-2">
<CustAddressLine><![CDATA[WRENTHAM MA
02037]]></CustAddressLine>
</CustAddress-row>
</CustAddress>
<SummaryInfo id="2-33-2-T1" >
<SummaryInfo-row id="2-33-2-T1-3">
<SummaryInfoLab><![CDATA[PREVIOUS BALANCE]]></SummaryInfoLab>
<SummaryInfoAmt><![CDATA[285.12]]></SummaryInfoAmt>
</SummaryInfo-row>
<SummaryInfo-row id="2-33-2-T1-4">
<SummaryInfoLab><![CDATA[LESS PAYMENTS APPLIED THROUGH
03/24/01]]></SummaryInfoLab>
```

```
<SummaryInfoAmt><![CDATA[158.37]]></SummaryInfoAmt>
<SummaryInfoCR><![CDATA[CR]]></SummaryInfoCR>
</SummaryInfo-row>
<SummaryInfo-row id="2-33-2-T1-7">
<SummaryInfoLab><![CDATA[MISCELLANEOUS
CREDIT]]></SummaryInfoLab>
<SummaryInfoAmt><![CDATA[19.19]]></SummaryInfoAmt>
<SummaryInfoCR><![CDATA[CR]]></SummaryInfoCR>
</SummaryInfo-row>
<SummaryInfo-row id="2-33-2-T1-8">
<SummaryInfoLab><![CDATA[BEGINNING BALANCE]]></SummaryInfoLab>
<SummaryInfoAmt><![CDATA[107.56]]></SummaryInfoAmt>
</SummaryInfo-row>
<SummaryInfo-row id="2-33-2-T1-9">
<SummaryInfoLab><![CDATA[CURRENT USAGE]]></SummaryInfoLab>
<SummaryInfoAmt><![CDATA[.07]]></SummaryInfoAmt>
</SummaryInfo-row>
<SummaryInfo-row id="2-33-2-T1-11">
<SummaryInfoLab><![CDATA[PRODUCT MONTHLY
FEES]]></SummaryInfoLab>
<SummaryInfoAmt><![CDATA[4.95]]></SummaryInfoAmt>
</SummaryInfo-row>
<SummaryInfo-row id="2-33-2-T1-12">
<SummaryInfoLab><![CDATA[CORPORATE CONNECTIONS
WAIVER]]></SummaryInfoLab>
<SummaryInfoAmt><![CDATA[4.95]]></SummaryInfoAmt>
<SummaryInfoCR><![CDATA[CR]]></SummaryInfoCR>
</SummaryInfo-row>
<SummaryInfo-row id="2-33-2-T1-14">
<SummaryInfoLab><![CDATA[LIFELINE ASST/TELE
RELAY]]></SummaryInfoLab>
<SummaryInfoAmt><![CDATA[0.80]]></SummaryInfoAmt>
</SummaryInfo-row>
<SummaryInfo-row id="2-33-2-T1-15">
<SummaryInfoLab><![CDATA[FEDERAL ACCESS
CHARGE]]></SummaryInfoLab>
<SummaryInfoAmt><![CDATA[8.62]]></SummaryInfoAmt>
```

```
</SummaryInfo-row>
<SummaryInfo-row id="2-33-2-T1-17">
<SummaryInfoLab><![CDATA[LOCAL USAGE
CHARGE]]></SummaryInfoLab>
<SummaryInfoAmt><![CDATA[13.68]]></SummaryInfoAmt>
</SummaryInfo-row>
<SummaryInfo-row id="2-33-2-T1-18">
<SummaryInfoLab><![CDATA[LOCAL SERVICE
CHARGE]]></SummaryInfoLab>
<SummaryInfoAmt><![CDATA[83.75]]></SummaryInfoAmt>
</SummaryInfo-row>
<SummaryInfo-row id="2-33-2-T1-19">
<SummaryInfoLab><![CDATA[FEDERAL TAXES - LOCAL
SERVICE]]></SummaryInfoLab>
<SummaryInfoAmt><![CDATA[2.62]]></SummaryInfoAmt>
</SummaryInfo-row>
<SummaryInfo-row id="2-33-2-T1-20">
<SummaryInfoLab><![CDATA[STATE TAXES - LOCAL
SERVICE]]></SummaryInfoLab>
<SummaryInfoAmt><![CDATA[2.15]]></SummaryInfoAmt>
</SummaryInfo-row>
<SummaryInfo-row id="2-33-2-T1-21">
<SummaryInfoLab><![CDATA[LATE FEE]]></SummaryInfoLab>
<SummaryInfoAmt><![CDATA[1.61]]></SummaryInfoAmt>
</SummaryInfo-row>
<SummaryInfo-row id="2-33-2-T1-22">
<SummaryInfoLab><![CDATA[TOTAL CURRENT
AMOUNT]]></SummaryInfoLab>
<SummaryInfoAmt><![CDATA[117.17]]></SummaryInfoAmt>
</SummaryInfo-row>
</SummaryInfo>
</view>
</doc>
```

6 Line Item Dispute and Annotations

Introduction

Goals of Line Item Dispute and Annotation

Attach data to line items on a statement

Line item detail is a powerful tool for Web application designers. Using a full suite of transparent J2EE tools, data can be attached directly to a line item and dynamically retrieved with that line item when the statement is refreshed. Dispute or annotation data is stored with the statement detail and can be retrieved whenever the data is dynamically presented.

You can design your Web applications to attach disputes or annotations to a row in a table or a summary record in a dynamic Web view. Disputed or annotated items can be displayed with a wide range of indicators, from check boxes or other standard HTML true/false flags to a GIF image in the disputed column or field. This example shows a credit card statement with multiple disputed items.

Account Number		Billing Date	Payment Due Date	Days in Billing Period	
303 505 755 7		03-16-99	04-11-99	28	
DISPUTE	TRANSACTION DATE	QUANTITY NUMBER	TRANSACTION DESCRIPTION & PURCHASE LOCATION Click here for Description of Codes)	COST	TOTAL AMOUNT
	CREDIT LIMIT	$2000	CREDIT AVAILABLE$1889		
X	02 17	4	01 529-539 KENNEDY BLVD BAYONNE NJ	13.00	52.00
X	02 19	10	01 529-539 KENNEDY BLVD BAYONNE NJ	14.09	140.90
X	02 21	3	01 529-539 KENNEDY BLVD BAYONNE NJ	12.00	36.00
X	02 25	1	01 AVENUE EXXON BAYONNE NJ	14.00	14.00
X	03 02	7	01 AVENUE EXXON BAYONNE NJ	14.01	98.07
X	03 08	3	01 AVENUE EXXON BAYONNE NJ	16.76	50.28
X	03 13	5	01 529-539 KENNEDY BLVD BAYONNE NJ	10.60	53.00
X	03 14	12	01 AVENUE EXXON BAYONNE NJ	17.00	204.00
			THE PERIODIC RATE SHOWN ON THIS STATEMENT MAY VARY		

Disputes and annotations can apply to all or part of a line item, or to an entire statement or group of invoices. In B2B applications, dispute flags may indicate a line item, the invoice for a specific department or sub-account, or the entire invoice for a group of accounts or sub-accounts.

Annotate a statement

Your customers can attach annotations to line items, such as a category or note for future reference. For example, a credit card company could allow customers to mark each line item as a business expense. Annotations are stored with the statement detail and can be retrieved whenever the data is dynamically presented.

Dispute all or part of a line item

In industries like telecommunications, when the customer disputes a line item, the dispute usually means they are challenging an entire call. In other industries like manufacturing, a customer may

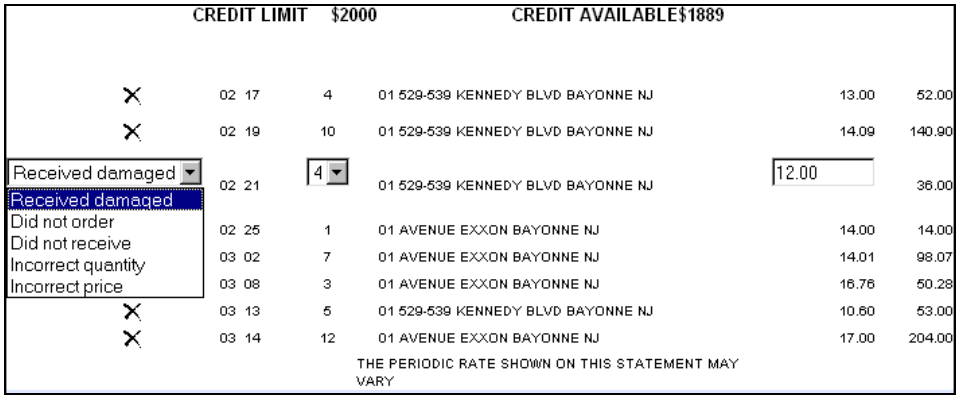
want to dispute only part of a line item, such as either the quantity or price. For example, in manufacturing, partial shipments are common, when the customer orders 60 girders and the manufacturer ships only 40. The customer could dispute the invoice to pay only for the 40 items received.

Delete a dispute or annotation

Once a dispute has been resolved, a customer or CSR will want to delete the dispute data. The system records the deletion in the dispute or annotation table, but the line item reverts to its original display. Deleting a dispute overrides all other states and "freezes" the dispute, which can then be purged from the database as the designer allows. eStatement Manager can support reopening of deleted disputes as long as the dispute data has not been purged from the data store.

Add Reason Codes to explain disputed item(s)

It is common to attach a reason code to a disputed line item as an explanation of why the item is disputed. This is similar to an annotation, for example to categorize expenses in a credit card statement. This example shows reason codes indicating why a customer is disputing a statement line item that was received damaged.



Default reason codes can be presented as a dropdown list on a JSP page, and the examples shipped with eStatement Manager are configurable. Reason codes are stored in a lookup table which may be customized by the application designer.

Disputes and Annotations Compared

Both disputes and annotations use the parameters **`accounts`**, **`ddn`**, **`docId`**, and **`userId`** to store data (**`submit`**) and retrieve data (**`getDocument`**). **`userID`** and **`accounts`** uniquely identify the customer, while **`docID`** and **`ddn`** uniquely identify the statement.

Both **`submit`** methods also record the **`detailID`** of the line item being disputed or annotated, the **`annotationID`** or **`disputeID`** of the record submitted, and the name (**`createdBy`**) of the user submitting the record These ID fields uniquely identify the line item detail to which data is being attached, and permit it to be retrieved when the data is next dynamically composed.

Both **`getDocument`** methods define the data to be retrieved by specifying **`parameters`**, the names of the XML queries (**`queryName`**) and Web views (**`viewName`**) used to extract and publish the detail data, and the HTTP **`req`**.

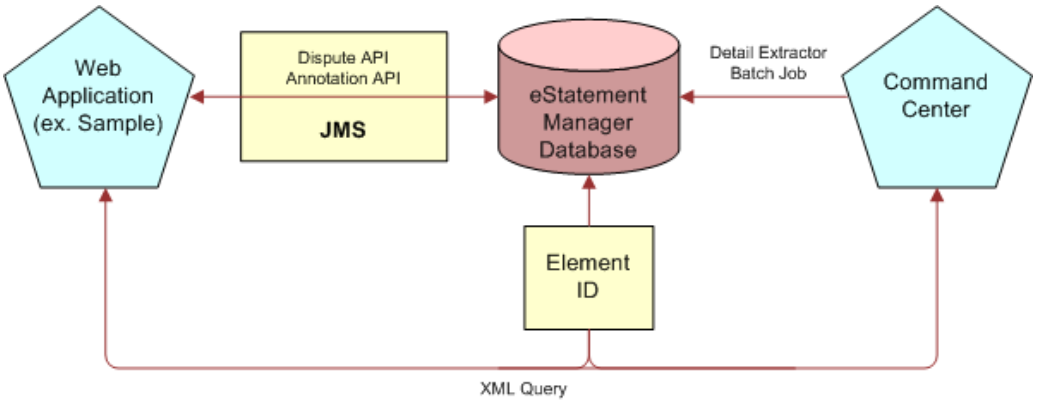
An **annotation** holds **text**, usually for the body of the annotation itself, and a **code**, typically to assign a customizable reason code or other category information.

A **dispute** holds the name of the **disputedColumn** and the **adjustedValue** to which the customer wishes the line item resolved. Disputes include a **state** field, which can be used to resolve a dispute by setting its state to **cancel**. Disputes can also hold text in the **comments** field.

Components of Line Item Dispute and Annotation

Architectural Overview

Two Web applications work together for dispute and annotation: the eStatement Manager Command Center and your client Web application. In the Web and application tiers, APIs leverage the speed and scalability of J2EE through the Java Messaging Service to send disputes and annotations as messages through the system. The database tier enhances the familiar ease of use of the eStatement Manager Command Center with the power and flexibility of XML, using tools like XSLT, XTags, and metadata to capture, store, retrieve, and present data on the fly from your customer's Web browser to your database.



Design is integrated into composition and production with Command Center XML. An application designer composes custom XML and XSL templates to extract and transform data to be uploaded to the database, and formulates SQL queries as XML to retrieve the data once it has been processed. During production, these XML files become the inputs to Command Center jobs and Web views.

Once detail data has been published to the Web, customers or customer service representatives can view line item detail and add disputes or annotations. These transactions travel through the application tier as JMS messages, and are stored by default in the eStatement Manager database.

Implementing dispute and annotation requires that all of these pieces work together. The next sections define the required tasks in terms of an eStatement Manager workflow. Following chapters discuss each topic in detail.

Configuration Tasks

1. **Configure JMS settings** to transport data from Web to database tiers.
2. **Create Database Tables** to store Dispute and Annotation data.

Composition Tasks

1. Enable Element ID in your DDF to identify recurring detail elements.
2. Customize XML and XSLT templates for input to Command Center jobs and views.

Production Tasks

1. **Index/extract detail data** in the Command Center with a Detail Extractor Job.
2. **Publish detail data** to be disputed or annotated with XML Query detail view.
3. **Publish dispute and annotation data** with XML Query dispute and annotation views.

Web Application Tasks

1. **Present dynamic "Manage Statement" JSPs** to allow customers to view recurring detail data and add disputes and annotations.
2. **Customize disputes and annotations** with **`direct.dispute`** and **`direct.annotation`** APIs.
3. **Use XTags** in JSPs to transform and format the XML data retrieved.

Viewing Disputes and Annotations in Sample

Here is a short checklist of how to view the example dispute and annotation solution shipped with eStatement Manager. If you have trouble, consult the required chapter of this guide or the relevant eStatement Manager documentation.

CAUTION: The Sample implementation is intended as a demonstration only. It is not production-ready code and must be customized.

1. Configure JMS settings on your application server.
2. Create an eStatement Manager application for National Wireless and run an Indexer job. Element ID is already enabled in the National Wireless DDF.
3. Create and run a Detail Extractor job named **`dtlxtr`** with the XML templates provided.
4. Publish three XML Query views named **`DetailQuery`** for detail data, **`DisputeQuery`** for dispute data, and **`AnnotationQuery`** for annotation data.
5. In Sample, enroll a user and view statement summaries (**`historyList.jsp`**).
6. From the statement summary page, click **`Manage Statement`.**

TIP: DO NOT click **View Detail**, as **Detail.jsp** does not implement disputes and annotations. These features are implemented in **DBDetail.jsp**, which is linked to **Manage Statement**.

7 Follow the links provided to submit an annotation or dispute.

8 To view data you have entered, refresh the Manage Statement page.

Configuring Dispute and Annotation Services

Configuring JMS Settings

About Java Messaging Services (JMS)

The Java Messaging Service is a messaging service available on J2EE platforms. eStatement Manager uses JMS to transport messages, for example customer inputs of dispute and annotation data, from **producer** applications to **consumer** or **listener** applications. For more information about JMS, see http://java.sun.com/products/jms/.

You must create and configure JMS servers, JMS Stores, JMS topics, and JMS connection factories for both dispute and annotation. Use the parameters listed here to configure JMS services. For details on configuring JMS Services, see the *Installation Guide for Oracle Siebel eStatement Manager.*

TIP: If your eStatement Manager solution does not implement Dispute and Annotation features, you need not configure these JMS settings.

JDBC TX Data Source

Name: **edxCommonDataSource**
JNDI Name: **edx.databasePool**
Pool Name: **edxCommonConnectionPool**

JMS Connection Factories

Name	JNDIName
edxDisputeTCF	edx/tcf/dispute
edxAnnotationTCF	edx/tcf/annotate

JMS Stores

Name	Connection Pool
edxAnnotationStore	edxCommonConnectionPool
edxDisputeStore	edxCommonConnectionPool

JMS Servers

Name	Store
edxAnnotationServer	**edxAnnotationStore**
edxDisputeServer	**edxDisputeStore**

JMS Destinations

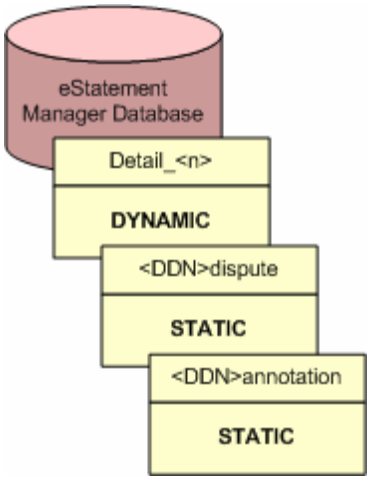
Destination	JNDIName
edxAnnotationTopic	**edx/jms/annotate**
edxDisputeTopic	**edx/jms/dispute**

Further Reading About JMS

"Managing JMS for BEA WebLogic Server 6.1," WebLogic 6.1 Admin Guide, 2001 BEA Systems, Inc. http://e-docs.bea.com/wls/docs61/adminguide/jms.html#jms001

Database Tables for Dispute and Annotation

Data for each disputed or annotated line item is stored in database tables. Detail tables are dynamic. Dispute and annotation tables are static for each DDN, though each Detail Extractor populates them dynamically at runtime.



For detailed information on the eStatement Manager database, see the *Installation Guide for Oracle Siebel eStatement Manager*.

JDBC Drivers for Dispute and Annotation

Line Item Dispute and Annotation requires the database connectivity features of **JDBC 2.0.** eStatement Manager ships with the **i-net OPTA™ 2000** driver for Windows, which replaces the BEA WebLogic JDBC driver for SQL Server. i-net OPTA 2000 is a Type 4 JDBC 2.0 Driver that supports Microsoft SQL Server 2000/7.0/6.5. eStatement Manager also uses the Oracle JDBC driver for Oracle database connectivity. **All necessary drivers are installed with eStatement Manager.**

For more information about the OPTA JDBC driver, see http://www.inetsoftware.de/English/produkte/OPTA/default.htm.

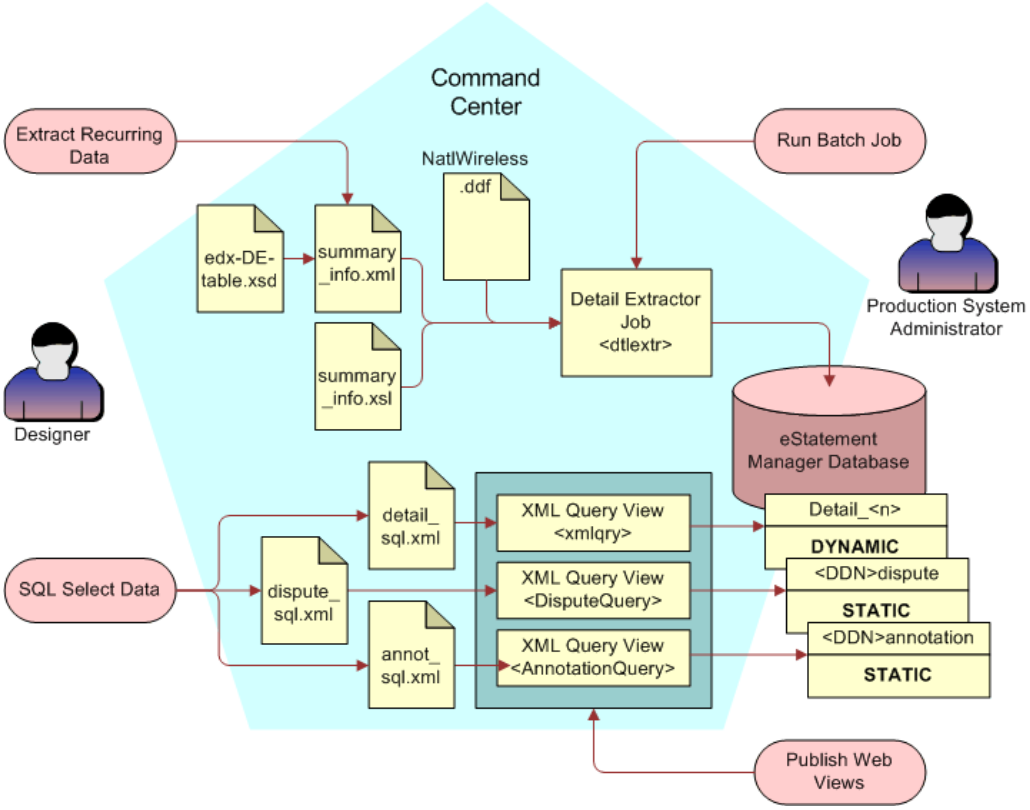
For more information about Oracle JDBC drivers, see http://www.orafaq.com/faqjdbc.htm#JDBC.

For what's new in JDBC 2.0 and a comparison of driver features, see http://java.sun.com/j2se/1.3/docs/guide/jdbc/spec2/jdbc2.1.frame4.html#283844.

Composition and Production for Dispute and Annotation

The Oracle Command Center uses XML to define, extract, upload, and publish recurring detail data. This chapter describes composition and production components and tasks in detail.

Composition and Production Architecture



XML is the core of composition and production for line item detail, dispute, and annotation. Many tasks formerly associated with the DefTool, Composer, or custom JSP development are now handled with XML templates.

1 **Enable Element ID** in your DDF to identify recurring detail elements.

2. **Customize XML and XSLT templates** for input to Command Center jobs and views.
3. **Index/extract detail data** in the Command Center with a Detail Extractor Job.
4. **Publish detail data** to be disputed or annotated with XML Query detail view.
5. **Publish dispute and annotation data** with XML Query dispute and annotation views.

Using Element ID

About Element ID

eStatement Manager includes a rich layer of metadata called **Composition Hints** to influence the extraction and composition of document data. This XML-based "language" passes context and data through the eStatement Manager composition layer, from the eStatement Manager core classes through a dynamic URL to the composed HTML in a Web browser. One of these Hints is the unique identifier **element ID**. This metadata flag links live data entry to individual elements in a browser page, and retrieves that data again when the page is dynamically composed.

For more information on Element ID syntax and the Composition Hints language, see Chapter 5, Content Access.

Enable Element ID in the DefTool

In order to implement dispute or annotation, **you must enable element ID** generation in a new or existing DDF. Also, whenever you turn the element ID feature on or off in an existing DDF, you must republish the version set.

For more information on creating a DDF with the DefTool and other eStatement Manager composition tools, see the *Data Definition (DefTool) Guide for Oracle Siebel eStatement Manager* and the *Presentation Design (Composer Guide) for Oracle Siebel eStatement Manager*.

Generate Element IDs with Detail Extractor

The Detail Extractor job assigns a unique ID number to each element in the DDF tree, and retrieves the location of the element within the document. Element IDs are generated from the DDF number and this location information. Each element ID is stored with its element in the data store.

At composition, each element ID is extracted and composed with the detail data. An Element ID is added as an ID attribute for XML, and as a tag surrounding the element for HTML. To see Element IDs in a composed JSP, select **`View HTML Source`**.

TIP: A given element ID is unique only within a document (indexed volume). Use element ID together with document ID to ensure a globally unique identifier.

For details about jobs, views, and version sets, see the *Administration Guide for Oracle Siebel eStatement Manager*.

Compose XML and XSLT Templates for Detail Extractor

First, the designer decides on the fields to be extracted, for example the summary information for a particular customer, and creates an XML file. Another decision is the XML format of the data, for which the designer creates an XSL stylesheet.

These XML templates, along with the DDF for your application, become the input to the Detail Extractor Job that extracts recurring detail data from your input file and uploads it to the Detail Tables for Disputes and Annotations in the eStatement Manager database.

These XML files are validated with the XML definition file DetailExtractor/edx-DE-table.xsd. This file ensures that eStatement Manager processes the XML correctly for the Detail Extractor tables. Your custom XML files must also validate against **`edx-DE-table.xsd`.**

CAUTION: **`edx-DE-table.xsd`** does not use the most current version of the XML Schema declaration: The schema it uses, http://www.w3.org/2000/10/XMLSchema, is the only version supported by WebLogic 6.1sp2. WebLogic users should **not** update this schema declaration to a newer version, as this version is required by the WebLogic XML parser. WebSphere supports newer schema declarations, but **Oracle will not support implementations using newer schemas**.

Edit summary_info.xml

The **`summary_info.xml`** template specifies the DDF Table name and the table field names as they appear in your database. Here you can also add columns to the detail table by specifying new XML field elements.

Customize **`summary_info.xml`** to define the detail IDs and names of the fields that you wish to extract from the database at runtime. **DO NOT CHANGE** the XML namespace definition in this file.

Edit summary_info.xsl

The **`summary_info.xsl`** template is a stylesheet that matches the requested **`docID`** and **`detailID`** with the data from the requested database table. Customize **`summary_info.xsl`** to define the transformations you want applied to the data at runtime. **DO NOT CHANGE** the XML namespace definition in this file.

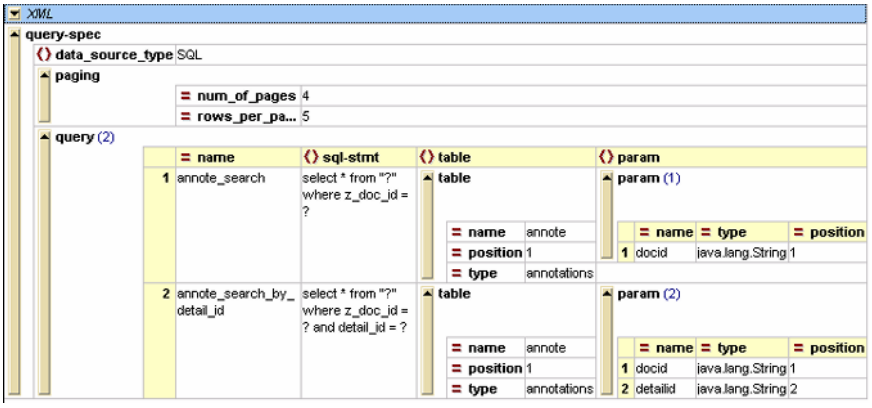
TIP: XML is a purposely open-ended standard for extensible data markup. The examples given with eStatement Manager are intended only as a starting point for building your own flexible custom solutions.

For more information about customizing XML templates for eStatement Manager, see Chapter 5, Content Access. For details about creating and configuring a Detail Extractor job, see Create and Configure a Detail Extractor Job and the *Administration Guide for Oracle Siebel eStatement Manager*.

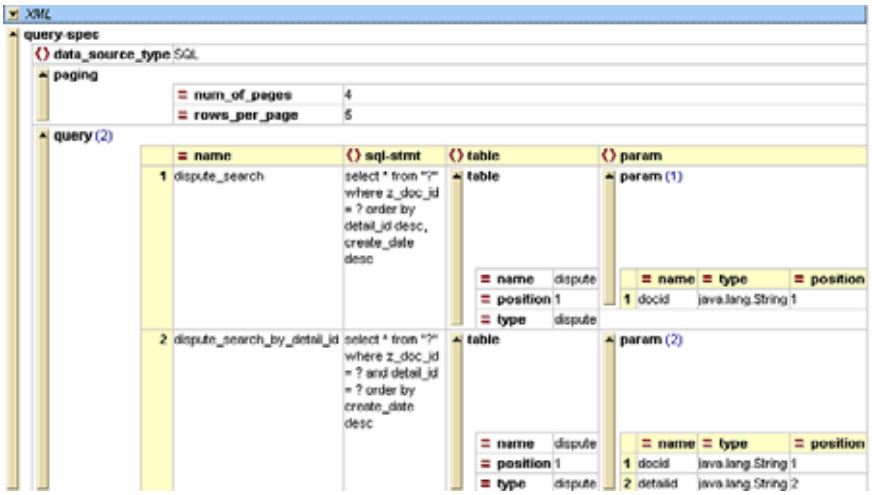
Compose XML Templates for XML Query Views

Once the data is in the database, you can extract it using standard SQL statements embedded in XML templates. See the National Wireless examples XMLQuery/detail_sql.xml for detail, XMLQuery/dispute_sql.xml for dispute, and XMLQuery/annot_sql.xml for annotation. Each of these XML templates becomes the input to the dynamic XML query views that publish your extracted data dynamically to the client Web application.

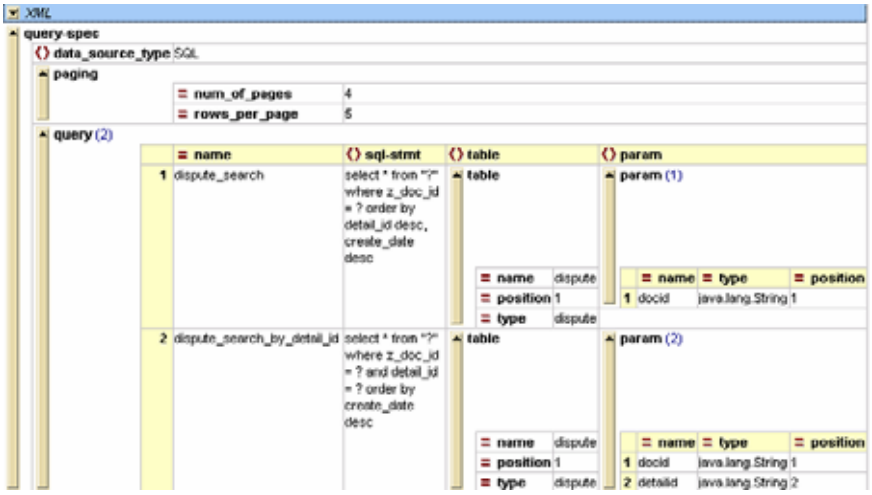
Edit annot_sql.xml



Edit dispute_sql.xml



Edit detail_sql.xml



Create and Configure a Detail Extractor Job

You must configure and run a separate Detail Extractor job for each set of data (table or group of tables) you want to upload to a database table.

For Sample, the name of the Detail Extractor job must be **`dtlextr`**. This name is hard-coded in several of the JSPs for detail, dispute, and annotation.

For details about creating and configuring a Detail Extractor job, see Create and Configure a Detail Extractor Job. For more information on the Detail Extractor job and other eStatement Manager jobs, see the *Administration Guide for Oracle Siebel eStatement Manager.*

TIP: If you have trouble running the Detail Indexer job, check with your installation team to make sure that detail tables and XML inputs are correctly created and configured.

Publish XML Query Dynamic Web Views

After creating and configuring your Detail Extractor job, you must publish **three** XML Query dynamic Web views in your application. Each XML Query view uses SQL statements as XML Templates for Disputes and Annotations to extract recurring data from Detail Tables for Dispute and Annotation and publish it to the Web application through the Manage Statement JSPs. The Sample JSPs integrate these three views for live retrieval and presentment.

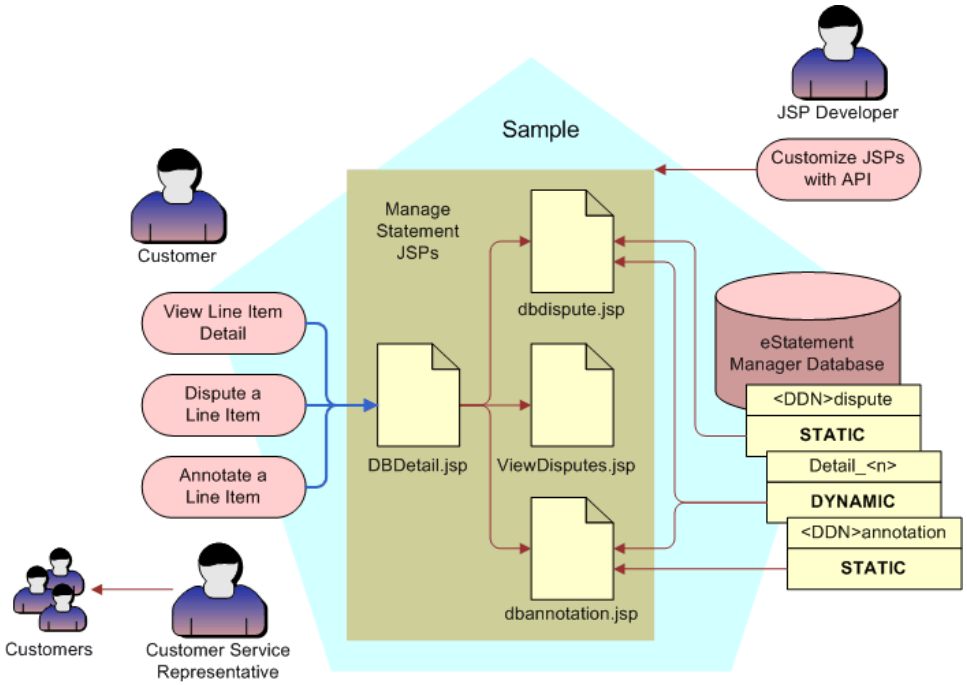
For National Wireless and Sample, these views must be named:

1. **`DetailQuery`** for detail data.
2. **`DisputeQuery`** for dispute data.
3. **`AnnotationQuery`** for annotation data.

For more information about publishing XML Query views, see the *Administration Guide for Oracle Siebel eStatement Manager.*

Web Application Components for Dispute and Annotation

Web Application Component Architecture



Manage Statement JSPs for Detail, Dispute, and Annotation

In Sample, clicking the **Manage Statement** link on a statement summary (**HistoryList.jsp**) brings you to the Line Item Dispute and Annotation page, **DBDetail.jsp**. This page presents line item detail for dispute or annotation by merging the output of **all three** XML Query views for your version set.

From the detail page, a customer or CSR can:

- Click Dispute Item to enter a new dispute in dbdispute.jsp
- Click Annotate Item to enter a new annotation in dbannotation.jsp
- Click View Disputes to view all existing disputes in ViewDisputes.jsp

These JSP pages integrate the content access, user management, and annotation APIs with standard Java and XTags libraries. This example makes extensive use of XTags for data formatting; study the sample code for examples and ideas.

CAUTION: For National Wireless and Sample, the XML Query views must be named **DetailQuery** for detail data, **DisputeQuery** for dispute data, and **AnnotationQuery** for annotation data. These view names are hard-coded into the Manage Statement JSPs.

Use care and attention to detail when customizing these pages. Replace any hard-coded references with variables as needed, test custom JSPs by publishing new version sets, and document changes for your production and presentment team.

DBDetail.jsp

This is the home page in the Sample implementation of dispute and annotation. This JSP specifies the **ReturnTo** value for the page to which the customer returns after the annotation is submitted.

It gets the **ddn**, **docId**, and **detailId** as unique identifiers for this annotation, and uses the **IAccount** object from the session to get the user ID and list of accounts.

The User, Dispute, and Annotation APIs then query the database to retrieve any existing line item detail, disputes, or annotations for this customer, using the three XML Query views published for this application.

```
// User API
  Map detailParameters = new HashMap();
  detailParameters.put(User.QUERY,"detail_search");
  detailReader = User.getDocumentReader(userId, accounts, ddn,
"XMLQuery", "DetailQuery",docId,null,detailParameters);
  // Dispute API
  Map disputeParameters = new HashMap();
  disputeReader =
Dispute.getDocument(userId,accounts,ddn,"DisputeQuery",docId,n
ull,"dispute_search",disputeParameters);
  // Annotation API
  Map annotationParameters = new HashMap();
  annoteReader =
Annotation.getDocument(userId,accounts,ddn,"AnnotationQuery",d
ocId,null,"annote_search",annotationParameters);
```

The JSP formats the retrieved XML data with XTags.

DBAnnotation.jsp

This page gets and validates request parameters, which are specific to the Web application (in this case Sample) and may not be explicitly required by the Annotation API.

The JSP next specifies the page to which the customer returns after the annotation is submitted. A valid **ReturnTo** value is required.

It then requests the field **title** (for display purposes). If the user is adding a new annotation, this value is **Add**. If the user is updating an existing annotation, this value is **Edit**. It also requests the parameter **description**, which is required.

The JSP gets the **ddn**, **docId**, and **detailId** as unique identifiers for this annotation, and uses the **IAccount** object from the session to get the user ID and list of accounts.

It then populates the **detailId** field in the **annote_search_by_detail_id** query defined in the **AnnotationQuery** view.

The Annotation API then retrieves the annotation data for this detail record as an XML document and formats it with XTags.

This JSP also presents an input form for the customer to enter annotation data. This example sets the detail ID to the current **detailID**, sets the doc ID to the current **docID**, sets the **createdBy** field to the current **userID**, and sets the **annotationId** value to zero.

It presents the description as read-only, provides a choice of Personal or Business category **code**, and provides a **text** field to enter the annotation body. The form posts this data to the database.

DBDispute.jsp

This page gets and validates request parameters, which are specific to the Web application (in this case Sample) and may not be explicitly required by the Annotation API.

The JSP next specifies the page to which the customer returns after the annotation is submitted.

The example page presents the current amount being disputed; this is only one possible implementation and is not required by the API.

The JSP gets the **ddn**, **docId**, **detailId,** and **disputedColumn** as unique identifiers for this dispute, and uses the **IAccount** object from the session to get the user ID (unlike Annotation, this page does not request the list of accounts).

The JSP presents an input form for the customer to enter dispute data. This example sets the detail ID to the current **detailID**, sets the **createdBy** field to the current **userID**, and sets the **disputeId** value to zero. It sets the dispute state to Open, presents the current amount to be disputed as read-only, and provides two text fields to enter the adjusted amount and any comments. The form posts this data to the database.

ViewDisputes.jsp

This JSP sets the **ReturnTo** value of the page to which the customer returns after the annotation is submitted. In this example, the **ReturnTo** page is **DBDetail.jsp**.

The example page presents the current amount being disputed; this is only one possible implementation and is not required by the API.

It then retrieves the **docId**, **detailId**, and **disputedColumn**, and uses the **IAccount** object from the session to get the user ID and list of accounts.

The JSP then populates the **detailId** field in the **dispute_search_by_detail_id** query defined in the **DisputeQuery** view.

The Dispute API then retrieves the dispute data for this detail record as an XML document and formats it with XTags.

Using XTags with Dispute and Annotation JSPs

About XTags

XTags is a JSP custom tag library for working with XML. XTags implements an XSLT-like language that allows you to style and process XML directly within a JSP page, using familiar XSLT and XPath techniques. For more information about XTags, see http://jakarta.apache.org/taglibs/doc/xtags-doc/intro.html.

XTags Example

The JSPs for annotation and dispute make extensive use of XTags. This example from **viewDisputes.jsp** uses an XTags stylesheet to format dispute data for presentment in the JSP. See the Sample code for more ideas on using XTags.

```
<xtags:stylesheet>
   <xtags:template match="/">
    <TABLE cellSpacing="0" cellPadding="5" align="center"
border="1">
     <tr>
      <td class="tableHead">Disputed Item</td>
      <td class="tableHead">Adjusted Amount</td>
      <td class="tableHead">Comments</td>
     </tr>
     <xtags:applyTemplates
select="/doc/view"></xtags:applyTemplates>
    </table>
   </xtags:template>
      <xtags:template match="row">
       <tr>
        <td class="text">
         <xtags:valueOf select="disputed_column"/>
        </td>
        <td class="text" align="right">
         <xtags:valueOf select="adjusted_value"/>
        </td>
        <td class="text">
         <xtags:valueOf select="comments"/>
        </td>
       </tr>
      </xtags:template>
  </xtags:stylesheet>
```

Application Programming Interfaces (API) for Dispute and Annotation

Data Flow for Annotation and Dispute Services

The annotation and dispute APIs are responsible for collecting all the data attached to a specific line item element and transporting it from the Web browser to the back end, which is typically, **but not necessarily**, a RDBMS data store.

The use of JMS and JDBC creates a flexible J2EE framework that allows client and partner designers great freedom to customize the storage and retrieval of dispute and annotation data. This section describes the sequence of data flow for annotation and dispute services.

The process for Dispute, shown below and described in the following sections, is identical for Annotation.

API Data Flow

1. The static Dispute class creates a Dispute object. This object is one of the parameters of the **`submit`** method call.
2. When a JSP calls **`submit`** with the Dispute object as the argument, the eStatement Manager core classes transform the Dispute object into a JMS message and broadcast (publish) the message with a preset message type.
3. The JMS service then broadcasts the JMS message.

   CAUTION: This step requires that JMS be correctly configured. See Java Messaging Service for Disputes and Annotations.
4. When the J2EE application server is started, the JMS service starts any subscribers. These subscribers create a JMS listener, whose class name is given in the property file.
5. When the JMS listener receives a message, it calls the 'onMessage' method of the listener for DBAccess.
6. The default listener for eStatement Manager uploads the details of Dispute into a database table called **`<DDN>Dispute`**. All Disputes for a single DDN are stored in this table.

TIP: Advanced developers familiar with EJB customization and JMS can write a custom JMS listener to change the final destination of the message. For details of EJB components, see Javadoc.

Package com.edocs.direct.annotation Description

About the Annotation API

The Annotation API allows users to submit, update, or cancel an annotation to line item detail. Typically, this data is stored in a relational database, usually the eStatement Manager database, as in the Sample implementation shipped with eStatement Manager. However, annotation data may also be

stored in an external data source, usually customized by Oracle Professional Services or client developers.

TIP: This EJB component architecture provides an industry standard framework for expanding the Annotation framework, for example to post-process incoming data to a file system.

To submit an annotation, the **submit** method of the abstract **Annotation** class collects required information about the annotation as parameters, and submits the annotation and its metadata to the data store through the **IAnnotationBean** EJB interface.

To retrieve an annotation, the **getDocument** method of the abstract **Annotation** class requests the specified information about the annotation as parameters, and retrieves the annotation and its metadata from the data store through the **IAnnotationBean** EJB interface.

For information on related EJB classes and interfaces, see the Javadoc.

submit signature

```
public static submit( String ddn, String[] accounts, String
docId, String detailID, String userID, String code, String
text, String createdBy )
```

TIP: When retrieving composite or compound account numbers, for example when working with subaccounts, you can use the **javax.naming** interface to specify the **accounts** parameter as type **Name** instead of type **String**. For more information about the **Name** interface, see the Javadoc.

submit Parameters

Name	Description
accounts	The account numbers this customer is authorized to view. Data type may be either **String** or **Name**.
annotationID	ID that uniquely identifies the annotation, to be created by the statement provider. Incrementing this ID inserts a new record; supplying an existing ID updates the record. Required.
code	Customizable reason codes, to be created by the statement provider. Customer selects from allowable values. May be NULL.
createdBy	Name of the user submitting the annotation.
ddn	The Document Definition Name (DDN) of the parent document containing the detail line item.
detailID	An ID that uniquely identifies the detail line item. Used with HTML composed from Detail Extractor tables. Required.
docId	The DOC_ID of the parent document containing the detail im. Required.
text	Text body of the annotation.
userID	The user ID of the customer submitting the annotation.

getDocument Signature

```
public static Reader getDocument( String ddn, String docId,
String [] accounts, String userID, String queryName, HashMap
params, HttpServletRequest req )
```

TIP: When retrieving composite or compound account numbers, for example when working with subaccounts, you can use the **javax.naming** interface to specify the **accounts** parameter as type **Name** instead of type **String**. For more information about the **Name** interface, see the Javadoc.

getDocument Parameters

Name	Description
accounts	The account numbers this customer is authorized to view. Data type may be either **String** or **Name**.
ddn	The Document Definition Name (DDN) of the parent document containing the detail line item.
docId	The DOC_ID of the parent document containing the detail item. **Required**.
parameters	Hashmap containing name-value pairs of parameters for the XML query.
queryName	The name of the XML Query used in the Detail Extractor job to extract the detail data.
req	HTTP Servlet Request object from the Web tier, used to implement paging. May be NULL.
userId	The user ID of the customer submitting the annotation.

GetDocument Results

getDocument returns the requested annotation(s) as a reader object which streams XML (but not an actual XML reader object).

Package com.edocs.direct.dispute Description

The Dispute API allows users to submit, update, or cancel a dispute to line item detail. Typically, this data is stored in a relational database, usually the eStatement Manager database, as in the Sample implementation shipped with eStatement Manager. It may also be stored in an external data source, usually customized by Oracle Professional Services or client developers.

TIP: This EJB component architecture provides an industry standard framework for expanding the Dispute framework, for example to post-process incoming data to a file system.

To submit a dispute, the **submit** method of the abstract **Dispute** class collects required information about the dispute as parameters, and submits the dispute and its metadata to the data store through the **IDisputeBean** EJB interface.

To retrieve a dispute, the **getDocument** method of the abstract **Dispute** class requests the specified information about the dispute as parameters, and retrieves the dispute and its metadata from the data store through the **IDisputeBean** EJB interface.

For information on related EJB classes and interfaces, see the Javadoc.

submit Signature

```
public static void submit(String ddn, String[] accounts,
String docId, String detailID, String userID, String state,
String comments, String adjustedValue, String disputedColumn,
String createdBy )
```

TIP: When retrieving composite or compound account numbers, for example when working with subaccounts, you can use the **javax.naming** interface to specify the **accounts** parameter as type **Name** instead of type **String**. For more information about the **Name** interface, see the Javadoc.

submit Parameters

Name	Description
accounts	The account numbers this customer is authorized to view. Data type may be either **String** or **Name**.
adjustedValue	The new value to which the disputed line item should be adjusted.
comments	Text body of the dispute.
createdBy	Name of the user submitting the dispute.
ddn	The Document Definition Name (DDN) of the parent document containing the detail line item.
detailID	An ID that uniquely identifies the detail line item. Used with HTML composed from Detail Extractor tables. Required.
disputedColumn	Name of the column being disputed.
disputeID	ID that uniquely identifies the dispute, to be created by the statement provider. Incrementing this ID inserts a new record; supplying an existing ID updates the record. Required.
docId	The DOC_ID of the parent document containing the detail item. Required.
state	State of the dispute
userID	The user ID of the customer submitting the annotation.

getDocument Signature

```
getDocument(String userId, String[] accounts, String ddn,
String viewName, String docId, HttpServletRequest req, String
queryName, Map parameters)
```

TIP: When retrieving composite or compound account numbers, for example when working with subaccounts, you can use the **javax.naming** interface to specify the **accounts** parameter as type **Name** instead of type **String**. For more information about the **Name** interface, see the Javadoc.

getDocument Parameters

Name	Description
accounts	The account numbers this customer is authorized to view. Data type may be either **String** or **Name**.
ddn	The Document Definition Name (DDN) of the parent document containing the detail line item.
docId	DOC_ID of the parent document containing the detail item. Required.
parameters	Hashmap containing name-value pairs of parameters for the XML query.
queryName	The name of the XML Query used in the Detail Extractor job to extract the detail data.
req	HTTP Servlet Request object from the Web tier, used to implement paging. May be NULL.
state	The state of the dispute.
userId	The user ID of the customer submitting the annotation.

GetDocument Results

getDocument returns the requested dispute(s) as a reader object which streams XML (but not an actual XML reader object).

Using the Dispute and Annotation APIs

Submit a dispute

To submit a new dispute, use the dispute.submit Signature with required dispute.submit Parameters and increment the **disputeID** by 1.

Submit an annotation

To submit a new annotation, use the annotation.submit signature with required annotation.submit Parameters and increment the **annotationID** by 1.

Update an existing dispute

To update an existing dispute, submit a new entry for that **disputeID** with new data for any of the parameters **userID, State, Comments, AdjustedValue, disputedColumn, createdBy**.

Update an existing annotation

To update an existing annotation, submit a new entry for that **annotationID** with new data for any of the parameters **code, text, createdBy**.

Cancel a dispute

To cancel a dispute, submit a new entry for that **disputeID** with the state set to **cancel**.

CAUTION: The dispute and annotation APIs throws an exception on failure of any step, but these exceptions are not written to the log file for eStatement Manager.

Retrieving Detail, Disputes, and Annotations with the Content Access API (com.edocs.app.user)

Sometimes it may be more convenient to retrieve detail, dispute, or annotation data along with other content instead of using the Detail and Annotation APIs. Special signatures of the Content Access API, **com.edocs.user,** use a **Map** object **parameters** to retrieve line item detail, dispute, or annotation data.

Calling one of these methods returns the data as XML in a **java.io.Reader** reference. The column names in the **ResultSet** are the tag names. All date values are converted to a Java type **long**.

For example implementations of these methods and signatures, see the sample files in the next section.

For more information on XML methods in **com.edocs.app.user**, see Chapter 5, Content Access.

Sample Files for Dispute and Annotation

Sample JSPs

dbannotation.jsp

```
<%@ page session="false"
     import="java.io.*,
        java.util.*,
         javax.naming.*,
         javax.naming.directory.*,
         com.edocs.app.user.*,
         com.edocs.enrollment.user.IAccount,

com.edocs.enrollment.user.jndi.JNDIAccountAttributes,
     com.edocs.direct.annotation.Annotation"
```

```
%>
<%@ taglib uri="http://jakarta.apache.org/taglibs/xtags-1.0"
prefix="xtags" %>
<%
 // Get request parameters and validate that they are all
here.


 // The following request parameters are used in this sample
and
 // are not required by the Annotation API.



 // The returnTo parameter is JSP page that the user should
return
 // to once the annotation is submitted.
 String returnTo = request.getParameter("returnTo");
 if(returnTo == null)
   throw new NullPointerException("returnTo is null");


 // If the user is adding a new annotation, this value is Add.
 // If the user is updating an existing annotation, this value
is Edit.
 String title = request.getParameter("title");


 // The name of the field to annotate. This is only used for
display purposes.
 String description = request.getParameter("description");
 if(description == null)
   throw new NullPointerException("description is null");


 // This information is necesasry to submit an annotation


 // The ddn is the application that the user is submitting the
dispute for.
 String ddn = request.getParameter("ddn");
 if(ddn == null)
   throw new NullPointerException("ddn is null");
```

```
 // The docId uniquely identifies a statment
 String docId = request.getParameter("docId");
 if(docId == null)
   throw new NullPointerException("docId is null");


 // The detailId uniquely identifies a detail row (record) in
a statment
 String detailId = request.getParameter("detailId");
 if(detailId == null)
   throw new NullPointerException("detailId is null");


 // Get the IAccount object from the users session to access
user attributes
 IAccount account =
(IAccount)request.getAttribute("com.edocs.enrollment.user.IAcc
ount");


 if(account == null) {
   throw new Exception("You are not logged in.");
 }


 // Get user id and the list of accounts from the IAccount
object
 String userId = getUserId(account);
 String[] accounts = getAccounts(account);


 // Populate the value of the detailid for the
annote_search_by_detail_id query defined in the
AnnotationQuery view.
 // select * from annotation table where docId='docid' and
detailId =value
 Map annotationParameters = new HashMap();
 annotationParameters.put("detailid",detailId);


 // Use the Annotation API to retreive the annotations for
this detail record
 // The reader will contain an XML document
```

```
 Reader annoteReader =
Annotation.getDocument(userId,accounts,ddn,"AnnotationQuery",d
ocId,request,"annote_search_by_detail_id",annotationParameters
);


%>


<xtags:parse id ="annoteDocument" reader="<%=annoteReader%>">
</xtags:parse>


<html>


<%@ include file="/enrollment/html/userHead.htm" %>


    <!-- Input Form -->
    <form name="inputForm"
action="UserAnnotation?app=AnnotationApp&returnTo=<%=returnTo%
>&ddn=<%=ddn%>&docId=<%=docId%>" method="POST">


      <input type=hidden name="detailId"
value="<%=detailId%>">
      <input type=hidden name="docId" value="<%=docId%>">
      <input type=hidden name="annotationId" value="<%=0%>">
      <input type=hidden name="createdBy" value="<%=userId%>">



                                  <center>
         <table width="320" border="1" cellspacing="0"
cellpadding="3">
          <tr>
           <td colspan="2" class="tableHead"><%= title %>
note:</td>
          </tr>
          <tr>
           <td align="right" class="label">Description:</td>
           <td class="readonly"><%= description %></td>
          </tr>
          <tr>
```

```
          <td align="right" class="label">Category:</td>

          <td class="text">
         <select name="code">

    <%
     String code = "//doc/view/row/code='Personal'";
     boolean isBusiness = true;
    %>
    <xtags:if context="<%= annoteDocument %>" test="<%= code
%>">
      <% isBusiness = false; %>
    </xtags:if>

       <% if(isBusiness) { %>
         <option selected>Business</option>
         <option>Personal</option>
       <% } else { %>
        <option>Business</option>
        <option selected>Personal</option>
       <% } %>
          </select>
          </td>

          </tr>
          <tr>
           <td align="right" class="label">Comment:</td>
           <td><textarea rows="5" cols="25" name="text"
maxlength="255"><xtags:valueOf context="<%= annoteDocument %>"
select="//doc/view/row/text"/></textarea></td>
          </tr>
         </table>

         <table width="320" border="0" cellspacing="0"
cellpadding="3">
          <tr>
            <td align="center">
```

```
              <input type="submit" value="Submit"
name="submit">
              <input type="reset" value="Reset" name="reset">
            </td>
          </tr>
         </table>
      </center>
    </form>


<%@ include file="/enrollment/html/userFoot.htm" %>
</html>


<%!
 private Attributes getAttributes(IAccount account) throws
Exception {
  return account.getAttributes("", new String[] {
        JNDIAccountAttributes.JNDI_UID,
        JNDIAccountAttributes.JNDI_ACCOUNT_NUMBER});
 }
 private String getUserId(IAccount account) throws Exception {
   Attributes attrs = getAttributes(account);
   Attribute a =
(Attribute)attrs.get(JNDIAccountAttributes.JNDI_UID);
   String uid = null;
   if(a.size() > 0)
     uid = (String)a.get(0);
   return uid;
 }


  private String[] getAccounts(IAccount account) throws
Exception {
    Attributes attrs = getAttributes(account);
    Attribute a =
(Attribute)attrs.get(JNDIAccountAttributes.JNDI_ACCOUNT_NUMBER
);
    ArrayList values = new ArrayList(a.size());
    String[] accts = null;
    for (NamingEnumeration an = a.getAll(); an.hasMore(); )
```

```
      values.add((String)an.next());
    accts = (String [])values.toArray(new String [] {});
    return accts;
  }
%>
```

DBDetail.jsp

```
<html>

<%@ page session="false"
     import="java.io.*,
         java.util.*,
         javax.naming.*,
         javax.naming.directory.*,
         java.net.URLEncoder,
         com.edocs.app.AppConstants,
         com.edocs.enrollment.user.IAccount,
         com.edocs.jndi.cda.CDANameParser,
         com.edocs.enrollment.user.jndi.JNDIAccountAttributes,
         com.edocs.direct.dispute.Dispute,
         com.edocs.direct.annotation.Annotation,
         com.edocs.services.merger.MergerDataAccessException,
         com.edocs.app.user.User"

     contentType="text/html"
%>

<%@ taglib uri="http://jakarta.apache.org/taglibs/xtags-1.0"
prefix="xtags" %>

<%@ include file="/enrollment/html/userHead.htm" %>

<%
 String returnTo = "/user/jsp/DBDetail.jsp" ;

 String ddn = request.getParameter("ddn");
```

```
 if (ddn == null)
   throw new IllegalArgumentException("ddn");


 String docId = request.getParameter("docId");
 if (docId == null)
   throw new IllegalArgumentException("docId");


 // Get the IAccount object from the users session to access
user attributes
 IAccount account =
(IAccount)request.getAttribute("com.edocs.enrollment.user.IAcc
ount");


 if(account == null) {
  throw new Exception("You are not logged in.");
 }


 // Get user id and the list of accounts from the IAccount
object
 String userId = getUserId(account);
 String[] accounts = getAccounts(account);



 Reader detailReader = null;
 Reader disputeReader = null;
 Reader annoteReader = null;


 try {
  // User API
  Map detailParameters = new HashMap();
  detailParameters.put(User.QUERY,"detail_search");
  detailReader = User.getDocumentReader(userId, accounts, ddn,
"XMLQuery", "DetailQuery",docId,null,detailParameters);


  // Dispute API
  Map disputeParameters = new HashMap();
```

```
  disputeReader =
Dispute.getDocument(userId,accounts,ddn,"DisputeQuery",docId,n
ull,"dispute_search",disputeParameters);


  // Annotation API

  Map annotationParameters = new HashMap();

  annoteReader =
Annotation.getDocument(userId,accounts,ddn,"AnnotationQuery",d
ocId,null,"annote_search",annotationParameters);


 } catch (MergerDataAccessException mde) {

  request.setAttribute("UserMsg", "This statement is currently
unavailable!"
   + " <br>Please try again later or "
   + " contact technical support for assistance.");


  throw mde;


 } catch (Exception e) {


  String returnInfo = "appRoot2=User&ddn=" + ddn
    + "&app2=UserMain&jsp2=/user/jsp/HistoryList.jsp";


  request.setAttribute("UserMsg",
       "There has been an error processing your request. "
       + "<br>Please try your request again or contact
technical"
       + " support for assistance. <br> Please <a href="
       +"/"UserEnrollment?app=Logout&"
       + "forwardto=/enrollment/jsp/UserLogout.jsp&"
       + returnInfo + "/">logout </a> and Retry");
  throw e;
 }


%>

<%-- Get Documents --%>

<xtags:parse id ="disputeDocument"
reader="<%=disputeReader%>">
```

```
</xtags:parse>


<xtags:parse id ="annoteDocument" reader="<%=annoteReader%>">
</xtags:parse>


<xtags:parse id ="detailDocument" reader="<%=detailReader%>">
</xtags:parse>


<%
    String reqURI = (String)
request.getAttribute("FORWARDURL");
    String returnInfo = "appRoot2=" + reqURI + "&ddn=" + ddn +
"&app2=UserMain&jsp2=/user/jsp/HistoryList.jsp";


    // get eaPay's Servlet context
         ServletContext payContext =
getServletConfig().getServletContext().getContext("/eaPay/Paym
ent");
%>


<table border="0" cellpadding="0" cellspacing="0" width="625"
align="center">
 <tr align="center" class="text">
    <td><a
href="User?app=UserMain&jsp=/user/jsp/HistoryList.jsp&ddn=<%=
ddn %>"><img border=0
src="enrollment/images/his_summary.gif"></a></td>
 <%if (payContext != null) {%>
  <td><a href="/eaPay/Payment?app=Payment&ddn="<%=ddn %>"><img
border=0 src="enrollment/images/his_pay.gif"></a></td>
 <% } else { %>
  <td><img border=0 src="enrollment/images/his_pay.gif"></td>
 <% }%>
  <td><a
href="User?app=UserMain&jsp=/samples/oc/telco/home.jsp&ddn=<%=
ddn %>"><img border=0
src="enrollment/images/his_order.gif"></a></td>
```

```
  <td><a
href="UserEnrollment?app=UpdateApp&jsp=/enrollment/jsp/user_ge
t_subscribe.jsp&forwardto=<%= reqURI %>&<%= returnInfo
%>"><img border=0
src="enrollment/images/his_profile.gif"></a></td>

  <td><a
href="User?app=UserMain&jsp=/user/jsp/DBDetail.jsp&ddn=<%= ddn
%>&docId=<%= docId %>"><img border=0
src="enrollment/images/his_refresh.gif"></a></td>

  <td><a
href="UserEnrollment?app=Logout&forwardto=/enrollment/jsp/User
Logout.jsp&<%= returnInfo %>"><img border=0
src="enrollment/images/his_logout.gif"></a></td>

 </tr>

</table>



<br>


<xtags:stylesheet>

  <xtags:template match="/">

    <TABLE cellSpacing="0" cellPadding="5" align="center"
border="1">

      <tr>

        <td class="tableHead">Comments</td>

        <td class="tableHead">Summary Info Description</td>

        <td class="tableHead">Summary Info Amount</td>


        <% // Display Dispute Amount Header if there is at
least 1 dispute %>

        <xtags:if context="<%= disputeDocument %>"
test="count(/doc/view/row) > 0">

          <td class="tableHead">Disputed Amount</td>

        </xtags:if>

        <td class="tableHead">Dispute</td>

      </tr>


      <xtags:applyTemplates
select="/doc/view"></xtags:applyTemplates>

    </table>

  </xtags:template>
```

```
  <xtags:template match="row">
    <xtags:variable id="Z_DOC_ID" select="z_doc_id"/>
    <xtags:variable id="detail_id" select="detail_id"/>
    <xtags:variable id="SummaryInfoAmount"
select="summaryinfoamount"/>
    <xtags:variable id="SummaryInfoDesc"
select="summaryinfodesc"/>


    <tr>
    <td align="center" class="text">
      <%

/*************************************************************
********

          Annotation lookup.

*************************************************************
*******/
        String annoteLookup = "//doc/view/row/detail_id='" +
detail_id + "'";
        boolean annoteExists = false;
      %>


      <xtags:if context="<%= annoteDocument %>" test="<%=
annoteLookup %>">
        <%
          annoteExists = true;
        %>
      </xtags:if>


      <% // If annotation exists then display 'view
annotation'
        if(annoteExists) { %>
          <a
href="User?app=UserMain&jsp=/user/jsp/dbannotation.jsp&ddn=<%=
ddn%>&title=Edit&returnTo=<%= returnTo %>&docId=<%=
java.net.URLEncoder.encode( Z_DOC_ID ) %>&detailId=<%=
java.net.URLEncoder.encode( detail_id ) %>&description=<%=
java.net.URLEncoder.encode( SummaryInfoDesc) %>"><img border=0
src="enrollment/images/annotate_black.gif"/></a>
```

```
      <% } else {%>
          <a
href="User?app=UserMain&jsp=/user/jsp/dbannotation.jsp&ddn=<%=
ddn%>&title=Add&returnTo=<%= returnTo %>&docId=<%=
java.net.URLEncoder.encode( Z_DOC_ID ) %>&detailId=<%=
java.net.URLEncoder.encode( detail_id ) %>&description=<%=
java.net.URLEncoder.encode( SummaryInfoDesc) %>"><img border=0
src="enrollment/images/annotate_white.gif"/></a>
      <% }%>


    </td>
    <td class="text">
      <%= SummaryInfoDesc %>
    </td>
    <td class="text" align="right">
      <xtags:valueOf select="summaryinfoamount"/>
    </td>
    <%

/*************************************************************
********
        Dispute lookup.

*************************************************************
*******/
      String disputeLookup = "//doc/view/row/detail_id='" +
detail_id + "'";
      boolean disputeExists = false;
    %>
    <xtags:if context="<%= disputeDocument %>" test="<%=
disputeLookup %>">
      <% disputeExists = true; %>
    </xtags:if>
    <% // If there is a dispute, display either the value (if
any) or just an empty cell. %>
    <xtags:if context="<%= disputeDocument %>"
test="count(/doc/view/row) > 0">
      <td class="text" align="right">
       <% // If dispute exists then display 'view dispute'
         if(disputeExists) { %>
          <%
```

```
            String findAdjValue =
"//doc/view/row[detail_id='" + detail_id +
"']/adjusted_value";
          %>
          <xtags:valueOf context="<%= disputeDocument %>"
select="<%=findAdjValue%>"/>
        <% } else {
           out.println(" ");
         }%>
      </td>
    </xtags:if>
    <td align="center" class="text">
      <% // If dispute exists then display 'view dispute'
        if(disputeExists) { %>
          <a
href="User?app=UserMain&jsp=/user/jsp/ViewDisputes.jsp&ddn=<%=
ddn%>&returnTo=<%= returnTo %>&docId=<%=
java.net.URLEncoder.encode( Z_DOC_ID ) %>&detailId=<%=
java.net.URLEncoder.encode( detail_id ) %>&disputedColumn=<%=
java.net.URLEncoder.encode( "Summary Info Amount"
)%>&currentAmount=<%= java.net.URLEncoder.encode(
SummaryInfoAmount )%>"><img border=0
src="enrollment/images/dispute_red.gif"/></a>
      <% } else {%>
          <a
href="User?app=UserMain&jsp=/user/jsp/dbdispute.jsp&ddn=<%=ddn
%>&returnTo=<%= returnTo %>&docId=<%=
java.net.URLEncoder.encode( Z_DOC_ID ) %>&detailId=<%=
java.net.URLEncoder.encode( detail_id ) %>&disputedColumn=<%=
java.net.URLEncoder.encode( "Summary Info Amount"
)%>&currentAmount=<%= java.net.URLEncoder.encode(
SummaryInfoAmount )%>"><img border=0
src="enrollment/images/dispute_green.gif"/></a>
      <% }%>
    </td>
    </tr>
  </xtags:template>
</xtags:stylesheet>


<%@ include file="/enrollment/html/userFoot.htm" %>


</html>
```

```
<%!
 private Attributes getAttributes(IAccount account) throws
Exception {
  return account.getAttributes("", new String[] {
        JNDIAccountAttributes.JNDI_UID,
        JNDIAccountAttributes.JNDI_ACCOUNT_NUMBER});
 }
 private String getUserId(IAccount account) throws Exception {
   Attributes attrs = getAttributes(account);
   Attribute a =
(Attribute)attrs.get(JNDIAccountAttributes.JNDI_UID);
   String uid = null;
   if(a.size() > 0)
     uid = (String)a.get(0);
   return uid;
 }


  private String[] getAccounts(IAccount account) throws
Exception {
    Attributes attrs = getAttributes(account);
    Attribute a =
(Attribute)attrs.get(JNDIAccountAttributes.JNDI_ACCOUNT_NUMBER
);
    ArrayList values = new ArrayList(a.size());
    String[] accts = null;
    for (NamingEnumeration an = a.getAll(); an.hasMore(); )
      values.add((String)an.next());
    accts = (String [])values.toArray(new String [] {});
    return accts;
  }
%>
```

dbdispute.jsp

```
<%@ page session="false"
     import="java.io.*,
        java.util.*,
         javax.naming.*,
```

```
            javax.naming.directory.*,
            com.edocs.app.user.*,
            com.edocs.enrollment.user.IAccount,
            com.edocs.enrollment.user.jndi.JNDIAccountAttributes"
%>

<html>

<%@ include file="/enrollment/html/userHead.htm" %>
<SCRIPT language=javascript
src="user/scripts/script.js"></SCRIPT>

<%
 // Get request parameters and validate that they are all
here.

 // The following request parameters are used in this sample
and
 // are not required by the Dispute API.



 // The returnTo parameter is JSP page that the user should
return
 // to once the dispute is submitted.
 String returnTo = request.getParameter("returnTo");
 if(returnTo == null)
   throw new NullPointerException("returnTo is null");

 // The currentAmount is the current amount the user is
disputing.
 String currentAmount = request.getParameter("currentAmount");
 if(currentAmount == null)
   throw new NullPointerException("currentAmount is null");

 // This information is necesasry to submit a dispute

 // The ddn is the application that the user is submitting the
dispute for.
```

```
 String ddn = request.getParameter ("ddn");
 if(ddn == null)
   throw new NullPointerException("ddn is null");

 // The docId uniquely identifies a statment
 String docId = request.getParameter("docId");
 if(docId == null)
   throw new NullPointerException("docId is null");

 // The detailId uniquely identifies a detail row (record) in
a statment
 String detailId = request.getParameter("detailId");
 if(detailId == null)
   throw new NullPointerException("detailId is null");

 // The disputedColumn identifies the field the user wishes to
dispute.
 // Examples of a disputedColumn would be amount, finance
charge, or phone number.
 String disputedColumn =
request.getParameter("disputedColumn");
 if(disputedColumn == null)
   throw new NullPointerException("disputedColumn is null");

 // Get the IAccount object from the users session to access
user attributes
 IAccount account =
(IAccount)request.getAttribute("com.edocs.enrollment.user.IAcc
ount");
 if(account == null) {
  throw new Exception("You are not logged in.");
 }

 // Get user id from the IAccount object
 String userId = getUserId(account);
%>
```

```
    <!-- Input Form -->
    <form name="inputForm" onSubmit="return
processForm(this);"
action="UserDispute?app=DisputeApp&returnTo=<%=returnTo%>&ddn=
<%=ddn%>&docId=<%=docId%>" method="POST">


      <input type=hidden name="detailId"
value="<%=detailId%>">
      <input type=hidden name="createdBy" value="<%=userId%>">
      <input type=hidden name="disputeId" value="0">


      <!-- State of the dispute -->
      <input type=hidden name="state" size="20" value="Open">
                                <center>
         <table width="500" border="1" cellspacing="0"
cellpadding="3" align="center">
          <tr>
           <td colspan="2" class="tableHead">Dispute your
statement:</td>
          </tr>
          <tr>
           <td align="right" class="label">Disputed Item:</td>
           <td class="readonly"><%=disputedColumn%></td>
           <input type=hidden name="disputedColumn"
value="<%=disputedColumn%>">
          </tr>
          <tr>
           <td align="right" class="label">Current
Amount:</td>
           <td class="readonly"><%=currentAmount%></td>
          </tr>
          <tr>
           <td align="right" class="label">Adjusted
Amount:</td>
           <td class="text"><input type="text"
name="adjustedAmount" size="25" maxLength="10"></td>
          </tr>
          <tr>
           <td align="right" class="label">Comments:</td>
```

```
           <td class="text"><textarea rows="5" name="comment"
size="25" maxlength="255"></textarea></td>
          </tr>
         </table>

         <table width="500" cellspacing="0" cellpadding="3">
          <tr>
            <td align="center">
              <input type="submit" value="Submit"
name="submit">
              <input type="reset" value="Reset"
name="submit2">
            </td>
          </tr>
         </table>
      </center>
    </form>
<%!
 private Attributes getAttributes(IAccount account) throws
Exception {
  return account.getAttributes("", new String[] {
        JNDIAccountAttributes.JNDI_UID,
        JNDIAccountAttributes.JNDI_ACCOUNT_NUMBER});
 }
 private String getUserId(IAccount account) throws Exception {
   Attributes attrs = getAttributes(account);
   Attribute a =
(Attribute)attrs.get(JNDIAccountAttributes.JNDI_UID);
   String uid = null;
   if(a.size() > 0)
     uid = (String)a.get(0);
   return uid;
 }

%>
<%@ include file="/enrollment/html/userFoot.htm" %>
</html>
```

viewdisputes.jsp

```
<html>

<%@ page session="false"
     import="java.io.*,
         java.util.*,
         javax.naming.*,
         javax.naming.directory.*,
         java.net.URLEncoder,
         com.edocs.app.user.*,
         com.edocs.app.AppConstants,
         com.edocs.enrollment.user.IAccount,
         com.edocs.jndi.cda.CDANameParser,
         com.edocs.enrollment.user.jndi.JNDIAccountAttributes,
         com.edocs.direct.dispute.Dispute"

     contentType="text/html"
%>

<%@ taglib uri="http://jakarta.apache.org/taglibs/xtags-1.0"
prefix="xtags" %>

<%@ include file="/enrollment/html/userHead.htm" %>
<%
 // Get request parameters and validate that they are all
here.

 // The following request parameters are used in this sample
and
 // are not required by the Dispute API.



 // The returnTo parameter is JSP page that the user should
return
 // to once the dispute is submitted.
 String returnTo = "/user/jsp/DBDetail.jsp" ;
```

```
 // The currentAmount is the current amount the user is
disputing.
 String currentAmount = request.getParameter("currentAmount");
 if(currentAmount == null) {
   throw new NullPointerException("currentAmount is null");
 }


 // This information is necesasry to submit a dispute


 // The ddn is the application that the user is submitting the
dispute for.
 String ddn = request.getParameter ("ddn");
 if(ddn == null)
   throw new NullPointerException("ddn is null");


 // The docId uniquely identifies a statment
 String docId = request.getParameter("docId");
 if(docId == null)
   throw new NullPointerException("docId is null");


 // The detailId uniquely identifies a detail row (record) in
a statment
 String detailId = request.getParameter("detailId");
 if(detailId == null)
   throw new NullPointerException("detailId is null");


 // The disputedColumn identifies the field the user wishes to
dispute.
 // Examples of a disputedColumn would be amount, finance
charge, or phone number.
 String disputedColumn =
request.getParameter("disputedColumn");
 if(disputedColumn == null)
   throw new NullPointerException("disputedColumn is null");


 // Get the IAccount object from the users session to access
user attributes
```

```
 IAccount account = (IAccount)
request.getAttribute("com.edocs.enrollment.user.IAccount");


 // Get user id and the list of accounts from the IAccount
object
 String userId = getUserId(account);
 String [] accounts = getAccounts(account);


 // Populate the detailId for the dispute_search_by_detail_id
query defined in the DisputeQuery view.
 Map disputeParameters = new HashMap();
 disputeParameters.put("detailid",detailId);


 // Use the Dispute API to retreive the disputes for this
detail record
 // The reader will contain an XML document
 Reader disputeReader =
Dispute.getDocument(userId,accounts,ddn,"DisputeQuery",docId,r
equest,"dispute_search_by_detail_id",disputeParameters);


%>


<xtags:parse id ="disputeDocument"
reader="<%=disputeReader%>">
</xtags:parse>
<%
    String reqURI = (String)
request.getAttribute("FORWARDURL");
    String returnInfo = "appRoot2=" + reqURI + "&ddn=" + ddn +
"&app2=UserMain&jsp2=/user/jsp/HistoryList.jsp";
    // get eaPay's Servlet context
    ServletContext payContext =
getServletConfig().getServletContext().getContext("/eaPay/Paym
ent");
%>
<html>


<table border="0" cellpadding="0" cellspacing="0" width="625"
align="center">
 <tr align="center" class="text">
```

```
    <td><a
href="User?app=UserMain&jsp=/user/jsp/HistoryList.jsp&ddn=<%=
ddn %>"><img border=0
src="enrollment/images/his_summary.gif"></a></td>

 <%if (payContext != null) {%>

  <td><a href="/eaPay/Payment?app=Payment&ddn="<%=ddn %>"><img
border=0 src="enrollment/images/his_pay.gif"></a></td>

 <% } else { %>

  <td><img border=0 src="enrollment/images/his_pay.gif"></td>

 <% }%>

  <td><a
href="User?app=UserMain&jsp=/samples/oc/telco/home.jsp&ddn=<%=
ddn %>"><img border=0
src="enrollment/images/his_order.gif"></a></td>

  <td><a
href="UserEnrollment?app=UpdateApp&jsp=/enrollment/jsp/user_ge
t_subscribe.jsp&forwardto=<%= reqURI %>&<%= returnInfo
%>"><img border=0
src="enrollment/images/his_profile.gif"></a></td>

  <td><a
href="UserEnrollment?app=Logout&forwardto=/enrollment/jsp/User
Logout.jsp&<%= returnInfo %>"><img border=0
src="enrollment/images/his_logout.gif"></a></td>

 </tr>

</table>


  <br>

  <body>

    <table align="center" border="0">

      <tr>

        <td>

          <a
href="User?app=UserMain&jsp=/user/jsp/dbdispute.jsp&returnTo=<
%= returnTo %>&ddn=<%= ddn %>&disputedColumn=<%=
java.net.URLEncoder.encode(disputedColumn)
%>&currentAmount=<%= java.net.URLEncoder.encode(currentAmount)
%>&docId=<%= docId %>&detailId=<%= detailId%>">Add Dispute</a>

        </td>

      </tr>

    </table>


    <xtags:stylesheet>
```

```
      <xtags:template match="/">
        <TABLE cellSpacing="0" cellPadding="5" align="center"
border="1">
          <tr>
            <td class="tableHead">Disputed Item</td>
            <td class="tableHead">Adjusted Amount</td>
            <td class="tableHead">Comments</td>
          </tr>
          <xtags:applyTemplates
select="/doc/view"></xtags:applyTemplates>
        </table>
      </xtags:template>

      <xtags:template match="row">
        <tr>
          <td class="text">
            <xtags:valueOf select="disputed_column"/>
          </td>
          <td class="text" align="right">
            <xtags:valueOf select="adjusted_value"/>
          </td>
          <td class="text">
            <xtags:valueOf select="comments"/>
          </td>
        </tr>
      </xtags:template>

    </xtags:stylesheet>
<%!
 private Attributes getAttributes(IAccount account) throws
Exception {
  return account.getAttributes("", new String[] {
        JNDIAccountAttributes.JNDI_UID,
        JNDIAccountAttributes.JNDI_ACCOUNT_NUMBER});
 }
 private String getUserId(IAccount account) throws Exception {
   Attributes attrs = getAttributes(account);
```

```
   Attribute a =
(Attribute)attrs.get(JNDIAccountAttributes.JNDI_UID);
   String uid = null;
   if(a.size() > 0)
     uid = (String)a.get(0);
   return uid;
 }


  private String[] getAccounts(IAccount account) throws
Exception {
    Attributes attrs = getAttributes(account);
    Attribute a =
(Attribute)attrs.get(JNDIAccountAttributes.JNDI_ACCOUNT_NUMBER
);
    ArrayList values = new ArrayList(a.size());
    String[] accts = null;
    for (NamingEnumeration an = a.getAll(); an.hasMore(); )
      values.add((String)an.next());
    accts = (String [])values.toArray(new String [] {});
    return accts;
  }
%>


<%@ include file="/enrollment/html/userFoot.htm" %>


</html>
```

XML Templates for National Wireless

XMLQuery/annot_sql.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<query-spec>
   <data_source_type>SQL</data_source_type>
   <paging num_of_pages="4" rows_per_page="5"/>


   <query name="annote_search">
```

```
         <sql-stmt>select * from "?" where z_doc_id = ?</sql-
stmt>
         <table name="annote" position="1"
type="annotations"/>
     <param name="docid" type="java.lang.String"
position="1"/>
   </query>


   <query name="annote_search_by_detail_id">
         <sql-stmt>select * from "?" where z_doc_id = ? and
detail_id = ?</sql-stmt>
         <table name="annote" position="1"
type="annotations"/>
     <param name="docid" type="java.lang.String"
position="1"/>
     <param name="detailid" type="java.lang.String"
position="2"/>
   </query>
</query-spec>
```

XMLQuery/detail_sql.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<query-spec>
   <data_source_type>SQL</data_source_type>
   <paging num_of_pages="4" rows_per_page="5"/>


   <query name="detail_search">
         <sql-stmt>select * from ? where z_doc_id = ?</sql-
stmt>
         <table name="detail" position="1" type="detail"
viewname="dtlextr"/>
     <param name="docid" type="java.lang.String"
position="1"/>
   </query>
</query-spec>
```

XMLQuery/dispute_sql.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<query-spec>
   <data_source_type>SQL</data_source_type>
```

```
   <paging num_of_pages="4" rows_per_page="5"/>


   <query name="dispute_search">
         <sql-stmt>select * from "?" where z_doc_id = ? order
by detail_id desc, create_date desc</sql-stmt>
         <table name="dispute" position="1" type="dispute"/>
     <param name="docid" type="java.lang.String"
position="1"/>
   </query>
   <query name="dispute_search_by_detail_id">
         <sql-stmt>select * from "?" where z_doc_id = ? and
detail_id = ? order by create_date desc</sql-stmt>
         <table name="dispute" position="1" type="dispute"/>
     <param name="docid" type="java.lang.String"
position="1"/>
     <param name="detailid" type="java.lang.String"
position="2"/>
   </query>
</query-spec>
```

.

7 Auditing Datastreams

Introduction to Auditing Data Streams

This module describes the Verify API, made available to help administrators verify whether a data stream has been correctly processed in eStatement Manager. Code examples are provided to illustrate the use of the various methods. (Note: this document uses the terms data stream, data stream, and volume interchangeably.)

About Auditing Data for Presentment

Before online statements are released for public access, it may be necessary to validate the input data stream. Various verification criteria may be defined depending on the information available about the data stream.

The methods in the Verify API only provide the means to manipulate an input stream as a whole, including the ability to mark the volume as accepted or rejected for online presentment.

Alternatively, the Content Access API operates at the statement level. If Verify is enabled, the Content Access API can only access statements in a volume that has been **accepted**.

Typical uses include:

- Add an audit level so administrators can ensure the print files are error free before making statements available to end-users.
- Confirm that the data delivered to eStatement Manager is compliant with the data definition specified.
- Define various verification criteria depending on information available about the data stream, such as number of accounts present in the data stream or account numbers known to be present in the data stream.
- Compare the number of extracted accounts to the number supposed to be present.
- Check to see if the account numbers known to be present in the data stream have actually been extracted (dummy accounts can be inserted in a data stream to facilitate such tests).
- Examine a random set of accounts more closely to determine whether line item values total up correctly.
- Test whether totals tally across all accounts to match some data stream total.

Verify API methods allow you to:

- Query which applications (DDNs) are deployed on an eStatement Manager server
- Obtain a list of accounts extracted from a single data stream
- Examine account summary information (via the Content Access API)

- Examine detailed account information (via the Content Access API)
- Accept or reject a processed data stream for online presentment

  CAUTION: Once an indexed volume has been rejected, it cannot later be accepted. The same applies for one that has been accepted; it cannot later be rejected.

APIs for Auditing Data Streams

Package com.edocs.app.verify Description

About the Verify API

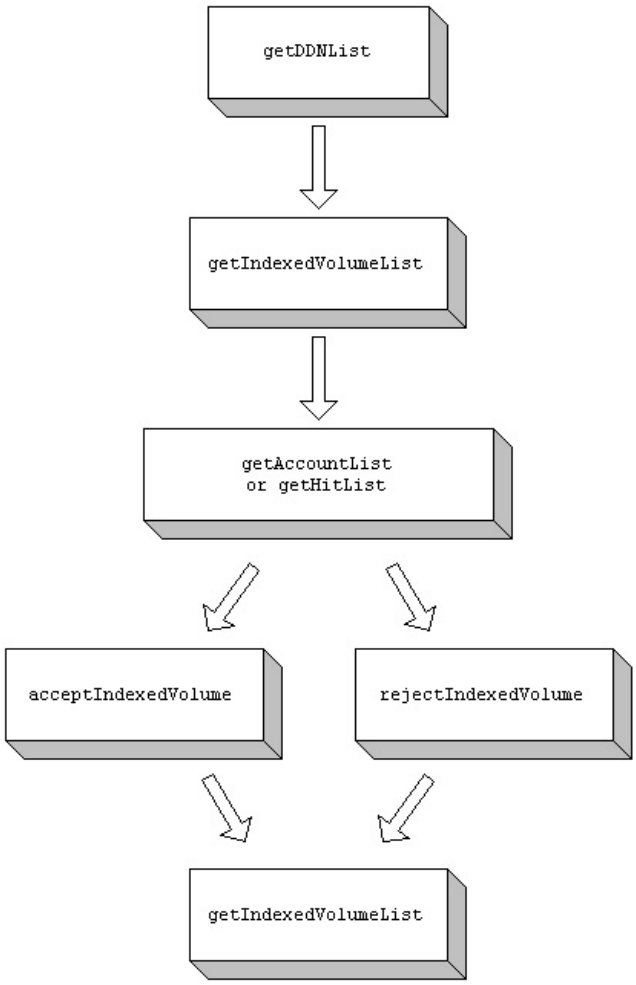
Provides the *`Verify`* class and methods for auditing indexed volumes of data before releasing them for presentment. Verify method *`getIndexedVolumeList`* retrieves a list of indexed volumes available for audit, while *`getAccountList`* retrieves all the account numbers in a volume. *`GetDDNList`* retrieves all DDNs. Two signatures of *`GetHitList`* retrieve all Description items either for a given volume, or for a given account. *`acceptIndexedVolume`* or *`rejectIndexedVolume`* respectively accept or reject a volume for presentment to customers. Finally, *`updateDescriptionInfo`* supports updates to the optional information field (Y_#) on a statement page.

Process Flow for Verify Methods

The following table summarizes the Verify methods; the Javadoc file provided with this module contains the reference information to provide detailed programming usage.

Method Name	Description
`acceptIndexedVolume`	Mark a processed data stream as valid.
`getAccountList`	Returns a list of account numbers associated with a processed print (data) file.
`getDDNList`	Returns a list of the DDNs defined.
`getHitList`	Returns account summary information.
`getIndexedVolumeList`	Returns a list of the processed data streams, identified by their indexed volume number (IVN).
`rejectIndexedVolume`	Marks as rejected an indexed volume
`updateSummaryInfo`	Enables updating of optional data fields

While the above list is in alphabetical order, Verify methods are typically called more or less in a fixed order, because the results of one call are often times necessary as input for another. The following diagram shows a possible process flow from one set of calls to the next. Methods listed in a single box or on the same level may be called in any order.



By default, when one runs an indexer job on a data stream in eStatement Manager, one has the option of requesting automatic verification (by selecting **`AutoIndexVolAccept`**) or intercepting to apply customized or manual verification. Interception requires that one select the **`Intercept to Verify`** option in the indexer job specification as shown below.

This task is initially set to **Auto Accept**, which automatically verifies each volume processed by the Indexer job. If you set this value to **Intercept to Verify**, it does not return summary items for that print file until it is marked as accepted. Using the Verify API, you can define the parameters to accept or reject a volume based on the analysis results. If they are marked as accepted, **`getSummary`** returns the summary items in that print file and end users may have access to them.

Auditing Data Streams with the Verify API

The following sections discuss the individual methods of the Verify API. It begins with **`getDDNList`**, which allows you to determine which applications have been defined on an eStatement Manager server.

These methods require the import of the **`com.edocs.app.verify.*`** into the JSP to access them (see the example below).

Retrieve a List of All Applications

About getDDNList

To obtain the list of applications (DDNs) defined on an eStatement Manager server, the Verify API provides the **`getDDNList`** method that takes no arguments. Often one may already know the application one is interested in.

`getDDNList` is provided to facilitate the building of tools or user interfaces that enable dynamic selection of the application name. This method returns a string array (**`String []`**).

Example

The following example returns a list of DDNs deployed on the server you execute it on. You could use the return to populate a drop down list box or an HTML table to enable the user to pick which application he/she wishes to verify print files from.

```
<HTML>
<HEAD>
<TITLE>Applications List Example</TITLE>
</HEAD>
<%@
        page import ="
java.util.Properties,
java.text.*,
com.edocs.app.verify.*,
com.edocs.app.util.*" %>
<%
try
{
    String[] DDNList = Verify.getDDNList();
    out.print("<BR>" + "DDN List on this Server: " + "<BR>")
    for (int i = 0; i < DDNList.length ; i++)
    out.print(DDNList[i] + "<BR>")
```

```
}
catch (Exception e)
{
}
%>
</TABLE>
</HTML>
```

Retrieve a List of Indexed Volumes

About getIndexedVolumeList

Each time a data stream is processed, it returns an **`indexed volume`** identified by a unique number, the index volume number (IVN). The method **`getIndexedVolumeList`** returns information about all the data streams processed. The information returned contains particulars such as date of processing and number of accounts extracted.

Parameters

`getIndexedVolumeList` accepts the following input parameters:

Name	Description
`count`	Used to indicate the maximum number of rows to be returned. Zero returns all. Data type is **`short`**.
`ddn`	The application name (DDN). Data type is **`string`**.

Results

`getIndexedVolumeList` returns a table. Row zero contains the column headings. Row one and onwards contain information about each of the indexed volumes. The number of rows retrieved is controlled using the "count" argument. Information about the last "count" number of indexed volumes processed is returned. If count is specified to be zero, information about all indexed volumes is returned. The column values in row one and beyond capture the value specified by the column heading (column value of the row zero element).

CAUTION: Note the zero-based counting for rows and columns. Row [0] contains the column headings; data starts at row [1].

Row Name	Description
Row[0]	Column headings
Row[1], Row [2]... Row[count]	Information about an indexed volume.

Column #	Column Name	Description
0	Z_IVN	Index volume number
1	Z_FILE_NAME	Original file name
2	Z_FILE_PATH	Current file path
3	Z_DATE_CREATED	Date file was used as input. (long as string)
4	Z_DATE_ACCEPTED	Date file was accepted. Empty if file not accepted.
5	Z_DATE_REJECTED	Date file was rejected. Empty if file not rejected.
6	Z_DATE_EXPIRED	Date the file expired. Empty if it has not expired.
7	Z_DOC_COUNT	Number of documents in file

Date values returned as a string are really a single long value indicating the number of milliseconds since the *epoch* (January 1, 1970, 00:00:00 GMT).

When the indexer job runs with the selection `Intercept to Verify`, the *Z_DATE_ACCEPTED* and *Z_DATE_REJECTED* fields are left empty; it is neither accepted nor rejected. By contrast, the **`AutoIndexVolAccept`** task is used to automatically mark the IVN as accepted. Using the other Verify API methods, you can analyze the indexed volume before marking it accepted or rejected.

Example

The code sample below demonstrates one use of **`getIndexedVolumeList`**:

```
<%
     String ddn = "Training"; // any deployed application
     short count = 5;
     String[][] ivnList = getIndexVolumeList(ddn, count);
     String[] columnHeadings = invList[0];
     int numCols = columnHeadings.length;
     int numRows = ivnList.length;
<table>
<tr>
<%  // output column headings in BOLD
      for (int  i = 0; i< numCols; i++) {  %>
          <td> <b> <%= columnHeadings[i] %> </b> </td>
<%    } %>
</tr>
```

```
<%    for (int  i = 1; i< numRows;  i++) {
%>
<tr>
 <%      for (int  j = 0; j< numCols; j++) {    %>
             <td>  <%= ivnList[i][j] %>  </td>
 <%      }
     }  %>
</tr>
<%   }
</table>
```

The other Verify API methods deal for the most part with a single IVN. They allow a more detailed examination of the processing results and finally enable one to accept or reject such processing.

Example

The following example retrieves all files processed for the DDN **`training`**.

```
<HTML>
<HEAD>
<TITLE>IVN List Example</TITLE>
</HEAD>
<%@ page import ="
java.util.Properties,
java.text.*,
com.edocs.app.verify.*,
com.edocs.app.util.*" %>
<% short count = 0
String DDN = "training"
/*Declare multidimensional array to hold
return of getIndexVolumeList*/
String[][] ivlArr
//Declare a string called ivn and initialize to null
String ivn = null
/*Make a call to getIndexVolumeList and set
return equal to ivlArr*/
ivlArr = Verify.getIndexedVolumeList(DDN, count)
int rows = ivlArr.length
out.print("<B>" + "ivns for " + DDN + "
application"+ "</B>" + "<br>")
for (int i = 1; i < rows; i++)
```

```
{
ivn = ivlArr[i][0]
out.print( "ivn= " + ivn + "<br>")
}
%>
</HTML>
```

Retrieve a List of Account Numbers

About getAccountList

You have selected an indexed volume identified by its IVN (indexed volume number). What's next? You might be interested in finding out the account (or subaccount) numbers for which information was extracted. Three signatures of **`getAccountList`** return a list of account numbers:

`getAccountList (string ivn, long offset, short count)`

`getAccountList (string ivn, name context, long offset, short count)`

`getAccountList(string ivn, long offset, short count, String pattern)`

The length of the list returned can be tuned by selecting values for the input arguments **`offset`**, **`count`** and **`pattern`**.

When retrieving composite or compound account numbers, for example when working with subaccounts, use the **`javax.naming`** interface to specify the **`context`** parameter as type **`Name`**. For more information about the Name interface, see the Javadoc.

Parameters

Name	Description
`ivn`	The index volume number (IVN). It is unique to a processed data stream. Data type is **`string`**.
`context`	The account number for which to retrieve subaccount data. Data type is **`name`**. See the Javadoc.
`offset`	Determines the starting point to process account numbers in the data stream. Data type is **`long`**.
`count`	Used to indicate the maximum number of rows to be returned. Zero returns all. Data type is **`short`**.
`pattern`	A string used to filter the account numbers with a SQL command in a LIKE clause. Data type is **`string`**.

Using the offset parameter

Consider the data stream as a long list of statement records. **`getAccountList`** allows you to specify beyond which point in that data stream to look and retrieve account numbers. The offset acts as a bookmark, letting you home in on where you last left off. You may seek to examine the processing

results chunk by chunk, sliding your view window perhaps by some fixed amount each time. This start position for viewing is the "offset." Often all account information may be accurately retrieved up to some point either because the data stream from that point on got corrupted or because the parsing rules failed to handle some situation at that point. By remembering the offset where the problem lay, the next time you could home in to see if the problem was resolved by selecting the same offset.

Example

```
String pattern = "2300%";
String ivn = request.getParameter("ivn");
// Z_IVN from a previous call to getIndexedVolumeList
short count = 10;
long offset = 0;
String[] accountNumbers  = null;
    accountNumbers = Verify.getAccountList(ivn,
                                           offset,
                                           count,
                                           pattern);
   // do some further processing of these accounts
```

The account number in conjunction with the IVN number can be used to retrieve statement summary and detail information. By examining them closely, one may determine whether the account information extraction from the data stream is accurate.

Example

The following example returns all accounts for the IVN with a Z_IVN of "2" which maps back to the training application as depicted in the prior JSP example.

```
<HTML>
<HEAD>
<TITLE>Account List Example</TITLE>
</HEAD>
<%@ page import =" java.util.Properties, java.text.*,
com.edocs.app.verify.*, com.edocs.app.util.*" %>
<%
long l = 0; short s = 0
/*Start at the beginning of the file and return all accounts for ivn
passed in using getAccountList*/
String ivn = "2";
String[] acctList =
Verify.getAccountList(ivn, l, s);
out.print( "<BR>" + "<B>" + "Account List" + "</B>" + "<BR>")
    for (int i = 0; i < acctList.length ; i++)
        out.print( acctList[i] + "<BR>")
%>
```

```
</HTML>
```

Retrieve Account Summary Information

About getHitList

Two signatures of the **`getHitList`** method retrieve account summary information. It is similar to the Content Access API method **`getSummary`**, but it also returns summary information for statements that have not been accepted. One signature retrieves summary information for all accounts within a single indexed volume. The other signature operates across all indexed volumes for a single account number. The sections that follow discuss each in detail.

getHitList Signature For All Accounts In A Single IVN

Typically, use **`getHitList(String ivn, long offset, short count)`** to present an administrator with account summaries for accounts extracted from the processing of a single data stream. The purpose is to facilitate a closer inspection of one or more accounts to ensure accuracy.

Parameters

This signature of **`getHitList`** accepts the following input parameters:

Name	Description
`offset`	Determines the starting point to process account numbers in the data stream. Data type is **`long`**.
`count`	Used to indicate the maximum number of rows to be returned. Zero returns all. Data type is **`short`**.

The offset argument determines the point beyond which account numbers in the data stream are returned. Typically, one marches down the list of account numbers, retrieving a set of them, examines them carefully, and then moves on to the next set. The count argument determines the set size. The number of account summaries retrieved is the smaller of the two values: number of account numbers available beyond "offset," and the set size.

Results

The signature **`getHitList(String ivn, long offset, short count)`** returns a table. The first row of the table contains the column headings and the remainder of the rows, one per account summary retrieved, contains column values. The tables below summarize the result set structure and column contents.

Row Name	Description
Row[0]	Column headings
Row[1]...Row[count]	Summary information for accounts
n	Number of columns
k	Number of optional fields as defined in indexer job.

Column #	Heading	Description
0	Z_PRIMARY_KEY	Account number
1	Z_DOC_ID	Document identifier
2	Z_DOC_DATE	Document date
3	Z_IVN	Indexed volume number
4 .. (n-k –1)	Indexed field names	Value of indexed fields
(n-k)...(n-1)	Optional field names (Y_1 to Y_k)	Values of optional fields

This method is also useful for customer service representative applications in navigating to an account in question.

getHitList Signature For One Account Across All IVNs

The signature **`getHitList (String account, String ddn, short count, long from, long to)`** provides summary information across all processed data streams for a given application (DDN) for a given account number. The "from" and "to" values indicate the range of processing dates from which to retrieve IVNs.

Typically, this method is used to test changes in parsing rules for the data stream. In particular, one tests that an account summary originally extracted correctly remains so and that problematic accounts cease to be so.

TIP: When retrieving composite or compound account numbers, for example when working with subaccounts, you can use the **`javax.naming`** interface to specify the account parameter as type **`Name`** instead of type **`String`**. For more information about the **`Name`** interface, see the Javadoc.

Parameters

This signature of **`getHitList`** accepts the following input parameters:

Name	Description
`account`	The list of accounts this user is allowed to view. Data type may be either String or Name.
`ddn`	The application name (DDN). Data type is string.
`count`	Used to indicate the maximum number of rows of summary information to be returned. Zero returns all. Data type is short.
`from`	The "from" date determines from how far back in time to retrieve information. "from" is used in conjunction with the "to" date to control the amount of account summary information to retrieve. Data type is long.
`to`	The upper bound date used in retrieving information. Data type is long.

Results

This signature of **`getHitList`** returns the following table information:

Row Name	Description
Row[0]	Column headings
Row[1] .. Row[count]	Summary information for accounts
n	Number of columns
k	Number of optional fields

Column #	Column Heading	Description
0	Z_DOC_ID	Document identifier
1	Z_DOC_DATE	Document date
2	Z_IVN	Indexed volume number
3 .. (n-k –1)	Indexed field names	Value of indexed fields
n-k .. n-1	Optional field names (Y_1 to Y_n)	Value of optional fields

This method is very similar to the Content Access API method **`getSummary`**. The only difference is that it provides summary information for an account regardless of whether the IVN was accepted or rejected, while the **`app.user`** method restricts itself to accepted IVNs only. The **`app.verify`** method allows comparison of the information extracted, which is handy in determining the effect of changes in data stream parsing rules.

The above methods work together to help make a decision whether to accept or reject the processing results of an indexed volume. The next step is to actually accept or reject an indexed volume.

Accept or Reject an Indexed Volume

About acceptIndexedVolume and rejectIndexedVolume

After careful inspection, one makes a decision to accept or reject an indexed volume. Accordingly, one invokes the **`acceptIndexedVolume`** or **`rejectIndexedVolume`** method. Both methods take as their sole input the IVN number. Only accepted indexed volumes become available to the end-user. The code sample below illustrates one usage:

```
String ivn = request.getParameter(“ivn”);
     Verify.acceptIndexedVolume(ivn);
     System.out.println(“Accepted indexed volume: “ + ivn);
```

To ascertain whether you have accepted or rejected an indexed volume, invoke the **`getIndexedVolumeList`** method again and examine the *Z_DATE_ACCEPTED* and/or *Z_DATE_REJECTED* fields for the IVN of interest. For an accepted IVN, the *Z_DATE*_ACCEPTED field

reflects the date on which you accepted it (the date itself is represented as a java long integer) and the *Z_DATE_REJECTED* field would be empty.

For example, the following fields would contain values for a DDN that has been accepted (the date values shown in these examples are only a "visual" representation of the actual values, which are number values):

Z_DATE_ACCEPTED	Z_DATE_REJECTED	Z_DATE_CREATED
2001-03-22		2001-03-21

If the method **`rejectIndexedVolume`** were subsequently called for this IVN then the fields would contain the following values:

Z_DATE_ACCEPTED	Z_DATE_REJECTED	Z_DATE_CREATED
	2001-03-22	2001-03-21

The reverse is true as well. Calling **`acceptIndexedVolume`** causes eStatement Manager to clear the *Z_DATE_REJECTED* value and add a *Z_DATE_ACCEPTED* value.

TIP: The above calls might used with an HTML form submission if a user interface implements the Verify API methods.

Example

In the following example, one IVN is rejected, stamping the *Z_DATE_REJECTED* field with the current date and rendering the IVN or data stream unavailable for customer viewing:

```
<HTML>

<HEAD>

<TITLE>Reject IVN Example</TITLE>

</HEAD>

<%@ page import ="java.util.Properties, java.text.*,
com.edocs.app.verify.*, com.edocs.app.util.*" %>

<%

String ivn = "2";    Verify.rejectIndexedVolume(ivn); out.print("<B>" +
"ivn: " + "</B>"+ ivn + " was marked as Rejected!" + "<br>")

%>

</HTML>
```

Example

The following example accepts the IVN passed in, stamping the *Z_DATE_ACCEPTED* field with the current date and thus enabling end users to retrieve information from the billing cycle that IVN represents:

```
<HTML>

<HEAD>

<TITLE>Accept IVN Example</TITLE> </HEAD>
```

```
<%@ page import ="java.util.Properties, java.text.*,
com.edocs.app.verify.*, com.edocs.app.util.*" %>

<%

String ivn = "2";    Verify.acceptIndexedVolume(ivn); out.print("<B>" +
"ivn: " + "</B>"+ ivn + " was marked as accepted and is ready for end
users!" + "<br>")

%>

</HTML>
```

Update Summary Information

About updateSummaryInfo

An account may have information that is not extracted from the data stream. Such pieces of information, saved in optional fields in the eStatement Manager database, can be accessed using the methods **`user.getSummary`** and **`verify.getHitList.`**

These optional fields are defined in the Command Center as part of the **`IXLoader`** task of a job that allows you to enter a value for the Optional Field Count parameter. For example, if you want five optional fields you would enter 5 in the Optional Field Count parameter.

An optional field may be **`DueDate`**. The application business rules may stipulate that the statement is payable twenty days from the date it is posted. That is a fixed length of time, and perhaps not one necessarily captured in the data stream. The due date in this case would be a function of the date the indexed volume is accepted.

Optional fields of this nature, and for that matter all of the optional fields, can be updated using the Verify API **`updateSummaryInfo`** method. (The Content Access API method **`updateSummaryInfo`** is similar.)

Parameters

Name	Description
`ivn`	The index volume number (IVN). It is unique to a processed data stream.
`docId`	The document identifier of the IVN. Obtain **`docId`** by calling **`getHitList`**.
`name`	The name of an optional field column, such as Y_1 or Y_2. The number selected during the **`IXLoader`** task determines the number of optional fields available. If you specify five optional fields, the names would range from Y_1 to Y_5.
`value`	The string value to assign to an optional field.

`Verify.updateSummaryInfo(String ivn, String docId, String name, String value)` requires the index volume number and the **`docId`**. Obtain these with one of the **`getHitList`** signatures. It also requires the name of the optional field and the value that must be assigned.

Example: Update Optional Field "Due Date"

The code sample below illustrates how it can be used in the context of an IVN that has the optional field "Y_1" that is associated with the semantics of a **`DueDate`**.

```
String ivn = "IVN_of_interest";
Date today = Date();
long 20DaysForward = 20*24*60*60*1000; // in  milliseconds
String dueDate = "" + (new Date(today.getTime() +
20DaysForward)).getTime();
String docId = null; Verify.acceptIndexVolume(ivn);
// lets accept it  today!
  String[][] acctSummrys = Verify.getHitList(ivn, 0L, 0S);
// count == 0 => all accounts
  for (int i = 1; i< acctSummrys.length; i++) {
    docId =  acctSummrys [i][0];
    Verify.updateSummaryInfo(ivn,
                             docId,
                             "Y_1", // optional field
                             dueDate);
    }
```

Example: Update Optional Field to Accept Accounts

Another use for **`updateSummaryInfo`** is to update an optional field **`AcceptAccount`**. Its default value could be **`true`**, and if closer inspection of the account summary or detail indicates otherwise, it could be marked **`false`**. This can give you fine-grained control over processing results: acceptance or rejection at the account level. Application business logic may be specified that programmatically totals the number of account level rejects, determines whether it is within some acceptable threshold, and if so, accepts the whole IVN.

.

8 Custom Jobs

About Custom Job Types

This guide describes how to create custom job types that include the Shell Command Task. This task allows you to run an external command script to process the output files from other tasks within the job. You can use this guide to:

- Define a custom job type for the Command Center and create a SQL script, containing job type and task information, to add the new job type.
- View and configure the new job type in the Command Center.

About Jobs and the Shell Command Task

eStatement Manager has several predefined job types available in its Command Center. Each job is made up of one or more tasks. For complete listing of jobs and tasks, see the *Administration Guide for Oracle Siebel eStatement Manager*.

However, there may be times when you want to expand these predefined Jobs to fit your needs. For cases like this eStatement Manager has the ability to define your own **custom Job** Type that you can make up from a combination of the predefined tasks that come with eStatement Manager and/or your own custom task by defining what is referred to in eStatement Manager as a **Shell Command Task**.

A Shell Command Task is a way of invoking a shell script, executable, or other program that was written to perform a task specific to your requirements. It enables you to run custom scripts or programs, such as pre- or post- processors as part of a user-defined job. You can create your own Job Type by creating a SQL script that updates the database. Once the database is updated this Job Type becomes available to you via the Command Center. The new Job Type can then be configured, scheduled, and run from within the Command Center.

For example, you could create a new custom job called **`Preprocess`** to run a pre-processor on the input file in an Indexer job. At runtime, the **`Preprocess`** task would be inserted between the Scanner and the Indexer tasks. Another use would be to create a job to run a validation engine (sum all amount due, for example) on the output of the Indexer task. At runtime, the **`SumAllDue`** task would be inserted between the **`Indexer`** and the **`IXLoader`** tasks.

Defining a New Job Type

This chapter includes information about:

- Creating the job type script
- Configuring the new job type

- Examples of the job type script

Create the Job Type Script

To create a job type you create a single SQL script to run in the eStatement Manager database using the Oracle utility **`sqlplus`**. Within this SQL script you define:

- The job name
- The tasks and the order in which they run
- The input arguments

The following sections provide a detailed topic description of each part. Each section uses the example of specifying a new job type that is similar to the current Indexer job, except that between scanning for an input file (Scanner Task) and actually indexing the file (Indexer Task) you need to invoke a preprocessor to modify the input file. This is the situation where you need to insert the **`ShellCmdTask`** between the other tasks.

Example *sqlplus* script for Oracle

```
DECLARE jtid NUMBER;
BEGIN

  -- Define the job name
  jtid := pwc_job_types.create_job_type ('myIndexer');

  -- Specify the job tasks and their order

  pwc_job_types.create_job_type_task(jtid,'Scanner', 1);

  pwc_job_types.create_job_type_task(jtid,'ShellCmdTask', 2);

  pwc_job_types.create_job_type_task(jtid,'Indexer', 3);

  pwc_job_types.create_job_type_task(jtid,'IXLoader', 4);

  pwc_job_types.create_job_type_task(jtid,
'AutoIndexVolAccept', 5);

  -- Define the tasks input arguments

  pwc_job_types.create_job_type_io(jtid,'ShellCmdTask', 'input
params', 'INPUT', 2,'Scanner', 'output file name', 'OUTPUT',
1);

  pwc_job_types.create_job_type_io(jtid, 'Indexer','data file
name', 'INPUT', 3, 'ShellCmdTask','shell output', 'OUTPUT',
2);

  pwc_job_types.create_job_type_io(jtid, 'Indexer', 'ddn
volume number', 'INPUT', 3,'Scanner', 'ddn volume number',
'OUTPUT', 1);

  pwc_job_types.create_job_type_io(jtid, 'IXLoader', 'index
volume number', 'INPUT', 4,'Scanner', 'ddn volume number',
'OUTPUT', 1);
```

```
  pwc_job_types. create_job_type_io (jtid, 'IXLoader', 'ir
file name', 'INPUT', 4, 'Indexer', 'ir file name', 'OUTPUT',
3);
  pwc_job_types.create_job_type_io(jtid, 'AutoIndexVolAccept',
'index volume number', 'INPUT', 5, 'Scanner', 'ddn volume
number, 'OUTPUT', 1);

END;
```

Example script for AIX/DB2

To create a DB2 shell command for a custom job in AIX, run the following command:

```
db2 -td@ -vf customjob.sh
```

Where **customjob.sh** is the name of a shell script customized for your job, platform, and environment. See the example below for a sample script to customize.

```
DROP PROCEDURE db2inst1.tmp_pwc_jtt_sp() @
CREATE PROCEDURE db2inst1.tmp_pwc_jtt_sp()
   LANGUAGE SQL
BEGIN
   DECLARE jtid             INTEGER;
   DECLARE l_job_type_name  VARCHAR(32);
   DECLARE l_task_name      VARCHAR(32);
   DECLARE l_task_order     INTEGER;
   DECLARE l_i_task_name    VARCHAR(32);
   DECLARE l_i_task_io_name VARCHAR(32);
   DECLARE l_i_task_io_type VARCHAR(32);
   DECLARE l_i_task_order   INTEGER;
   DECLARE l_o_task_name    VARCHAR(32);
   DECLARE l_o_task_io_name VARCHAR(32);
   DECLARE l_o_task_io_type VARCHAR(32);
   DECLARE l_o_task_order   INTEGER;


    -- job type with
'Scanner':'ShellCmdTask':'Indexer':'IXLoader':'AutoIndexVolAcc
ept'
   SET l_job_type_name = 'Custom_Indexer';
   CALL pwc_job_types.create_job_type(jtid, l_job_type_name);


   SET l_task_name = 'Scanner';
```

```
   SET l_task_order = 1;

   CALL pwc_job_types.create_job_type_task(jtid, l_task_name,
l_task_order);

   SET l_task_name = 'ShellCmdTask';

   SET l_task_order = 2;

   CALL pwc_job_types.create_job_type_task(jtid, l_task_name,
l_task_order);

   SET l_task_name = 'Indexer';

   SET l_task_order = 3;

   CALL pwc_job_types.create_job_type_task(jtid, l_task_name,
l_task_order);

   SET l_task_name = 'IXLoader';

   SET l_task_order = 4;

   CALL pwc_job_types.create_job_type_task(jtid, l_task_name,
l_task_order);

   SET l_task_name = 'AutoIndexVolAccept';

   SET l_task_order = 5;

   CALL pwc_job_types.create_job_type_task(jtid, l_task_name,
l_task_order);


   SET l_i_task_name    = 'ShellCmdTask';

   SET l_i_task_io_name = 'input params';

   SET l_i_task_io_type = 'INPUT';

   SET l_i_task_order   = 2;

   SET l_o_task_name    = 'Scanner';

   SET l_o_task_io_name = 'output file name';

   SET l_o_task_io_type = 'OUTPUT';

   SET l_o_task_order   = 1;

CALL pwc_job_types.create_job_type_io(jtid, l_i_task_name,
l_i_task_io_name,

l_i_task_io_type, l_i_task_order, l_o_task_name,
l_o_task_io_name, l_o_task_io_type,

l_o_task_order);

   SET l_i_task_name    = 'Indexer';

   SET l_i_task_io_name = 'data file name';

   SET l_i_task_io_type = 'INPUT';

   SET l_i_task_order   = 3;

   SET l_o_task_name    = 'ShellCmdTask';
```

```
   SET l_o_task_io_name = 'shell output';
   SET l_o_task_io_type = 'OUTPUT';
   SET l_o_task_order   = 2;
CALL pwc_job_types.create_job_type_io(jtid, l_i_task_name,
l_i_task_io_name,
l_i_task_io_type, l_i_task_order, l_o_task_name,
l_o_task_io_name, l_o_task_io_type,
l_o_task_order);
   SET l_i_task_name    = 'Indexer';
   SET l_i_task_io_name = 'ddn volume number';
   SET l_i_task_io_type = 'INPUT';
   SET l_i_task_order   = 3;
   SET l_o_task_name    = 'Scanner';
   SET l_o_task_io_name = 'ddn volume number';
   SET l_o_task_io_type = 'OUTPUT';
   SET l_o_task_order   = 1;
CALL pwc_job_types.create_job_type_io(jtid, l_i_task_name,
l_i_task_io_name,
l_i_task_io_type, l_i_task_order, l_o_task_name,
l_o_task_io_name, l_o_task_io_type,
l_o_task_order);
   SET l_i_task_name    = 'IXLoader';
   SET l_i_task_io_name = 'index volume number';
   SET l_i_task_io_type = 'INPUT';
   SET l_i_task_order   = 4;
   SET l_o_task_name    = 'Scanner';
   SET l_o_task_io_name = 'ddn volume number';
   SET l_o_task_io_type = 'OUTPUT';
   SET l_o_task_order   = 1;
CALL pwc_job_types.create_job_type_io(jtid, l_i_task_name,
l_i_task_io_name,
l_i_task_io_type, l_i_task_order, l_o_task_name,
l_o_task_io_name, l_o_task_io_type,
l_o_task_order);
   SET l_i_task_name    = 'IXLoader';
   SET l_i_task_io_name = 'ir file name';
   SET l_i_task_io_type = 'INPUT';
   SET l_i_task_order   = 4;
```

```
   SET l_o_task_name    = 'Indexer';
   SET l_o_task_io_name = 'ir file name';
   SET l_o_task_io_type = 'OUTPUT';
   SET l_o_task_order   = 3;
CALL pwc_job_types.create_job_type_io(jtid, l_i_task_name,
l_i_task_io_name,
l_i_task_io_type, l_i_task_order, l_o_task_name,
l_o_task_io_name, l_o_task_io_type,
l_o_task_order);
   SET l_i_task_name    = 'AutoIndexVolAccept';
   SET l_i_task_io_name = 'index volume number';
   SET l_i_task_io_type = 'INPUT';
   SET l_i_task_order   = 5;
   SET l_o_task_name    = 'Scanner';
   SET l_o_task_io_name = 'ddn volume number';
   SET l_o_task_io_type = 'OUTPUT';
   SET l_o_task_order   = 1;
CALL pwc_job_types.create_job_type_io(jtid, l_i_task_name,
l_i_task_io_name,
l_i_task_io_type, l_i_task_order, l_o_task_name,
l_o_task_io_name, l_o_task_io_type,
l_o_task_order);


END @
CALL db2inst1.tmp_pwc_jtt_sp() @
DROP PROCEDURE db2inst1.tmp_pwc_jtt_sp() @
```

Name the Job

The first part of the script is to give your new task a name. The syntax to do this is:

```
jtid := pwc_job_types.create_job_type ('<new_job_name>');
```

In the script, the **`create_job_type`** call defines a unique job type ID (**`jtid`**) for the new *Indexer1* job type.

So if your new job name is **`myIndexer`**, then the code script would be:

```
jtid := pwc_job_types.create_job_type ('myIndexer');
```

Specify Job Tasks

The next step is to specify what tasks are part of the new job, and in what order they execute. The syntax is:

```
pwc_job_types.create_job_type_task(jtid,'<task_name>', n);
```

where *n* equals the order number of the task and *jtid* is the job type id – created with **`pwc_job_types.create_job_type()`** function. The **`create_job_type_task`** call defines the order of the tasks in the job.

In the above example, the plan is to create a new job type based on the current Indexer job type. The tasks included in the Indexer Job are (in their order of execution):

Job	**Tasks**
Indexer	**`Scanner`**
	`Indexer`
	`IXLoader`
	`AutoIndexVolAccept`

If you insert the **`ShellCmdTask`** after the Scanner Task, it will become task 2, and the others will be incremented by one. The code example is:

```
pwc_job_types.create_job_type_task(jtid,'Scanner', 1);
pwc_job_types.create_job_type_task(jtid,'ShellCmdTask', 2);
pwc_job_types.create_job_type_task(jtid,'Indexer', 3);
pwc_job_types.create_job_type_task(jtid,'IXLoader', 4);
pwc_job_types.create_job_type_task(jtid, 'AutoIndexVolAccept',
5);
```

Define Input Arguments

Each task has input and output arguments, and a particular task may require the output arguments from a previous task to function properly. For example, in the default Indexer job, its Indexer task takes two input arguments from the Scanner Task. In the SQL Script you define which specific input arguments for a task are used from the specific output arguments from another task.

For a list of arguments, see the Javadoc.

To define the input and output parameters, the following is the syntax of the function of the call that uses nine arguments:

```
pwc_job_types.create_job_type_io(jtid,
                                 '<input_task_name>',
                                 '<input_argument>',
                                 'INPUT',
                                 x,
                                '<output_task_name>',
                                '<output_argument>',
                                'OUTPUT',
                                y);
```

where *x* is the order number of the input task and *y* is the order number of the output task. The **`create_job_type_io`** calls define the input values for each job task. It accepts the following parameter values:

- The job type ID (**jtid**)
- The task name receiving the input value
- The input parameter name
- The I/O type (**INPUT**)
- The order number for the task receiving the input value (defined earlier in the script)
- The previous task name dispensing the output to be used for input
- The output parameter name from the previous task
- The I/O type (**OUTPUT**)
- The order number of the task dispensing the output value (defined earlier in the script)

The following breaks down the input arguments used in the above example script:

```
pwc_job_types.create_job_type_io(jtid,
                                 'ShellCmdTask',
                                 'input params',
                                 'INPUT',
                                 2,
                                 'Scanner',
                                 'output file name',
                                 'OUTPUT',
                                 1);
```

The input argument *input params* for the **ShellCmdTask** uses the output argument *output file name* from the Scanner task.

```
pwc_job_types.create_job_type_io(jtid,
                                 'Indexer',
                                 'data file name',
                                 'INPUT',
                                 3,
                                 'ShellCmdTask',
                                 'shell output',
                                 'OUTPUT',
                                 2);
pwc_job_types.create_job_type_io(jtid,
                                 'Indexer',
                                 'ddn volume number',
                                 'INPUT',
                                 3,
                                 'Scanner',
                                 'ddn volume number',
                                 'OUTPUT',
                                 1);
```

The input arguments *data file name* and *ddn volume number* for the Indexer task uses the output arguments *shell output* from the **ShellCmdTask** and *ddn volume number* from the Scanner task respectively.

```
pwc_job_types.create_job_type_io(jtid,
                                 'IXLoader',
                                 'index volume number',
                                 'INPUT',
                                 4,
                                 'Scanner',
                                 'ddn volume number',
                                 'OUTPUT',
                                 1);

pwc_job_types. create_job_type_io (jtid,
                                   'IXLoader',
                                   'ir file name',
                                   'INPUT',
                                   4,
                                   'Indexer',
                                   'ir file name',
                                   'OUTPUT',
                                   3);
```

The input arguments *index volume number* and *ir file name* for the **IXLoader** task uses the output arguments *ddn volume number* from the Scanner and *ir file name* from the Indexer respectively.

```
pwc_job_types.create_job_type_io(jtid,
                                 'AutoIndexVolAccept',
                                 'index volume number',
                                 'INPUT',
                                 5,
                                 'Scanner',
                                 'ddn volume number,
                                 'OUTPUT',
                                 1);
```

The input argument *index volume number* for the **AutoIndexVolAccept** task uses the output argument *ddn volume number* from the Scanner task.

Configuring Your New Job Type

After creating the script, you run it against the Oracle database used by eStatement Manager (as described in the *Installation Guide for Oracle Siebel eStatement Manager*). For example, if the script is named **myindexer.sql** and placed in **/opt/eStatement/db** (the default database location for eStatement Manager), you could run the following in SQL*Plus:

```
$ sqlplus -s edx_dba/edx@edx.db @ /opt/eStatement/db/myindexer.sql
```

The above command presumes you are using the default names for the eStatement Manager database (**edx0**) and database administrator/password (**edx_dba/edx**).

TIP: Before the new job type is available in the Command Center, you have to stop and start your application server after running the script.

Once the new job type is available to you from the Command Center, you can define the new job using that new job type.

Define the Shell Command Task

If you have included the **`ShellCmdTask`** with your new job type, it has 2 input fields to define:

- Shell Command
- Environment variables

The Shell Command field defines the location of the shell script to execute on your system. Note that the user that starts the application server must have read/execute permissions for that location.

The shell command must output, on its standard output, the name of its output file that is the input file to be processed by the next task in the job. If the shell command doesn't output any file name, the job stops as a no-op. If it is successful, the shell command must set its exit code to 0.

If the shell command fails, it must set its exit code to a non-zero value. Additionally, it may output, on its standard error, a message describing the failure. eStatement Manager adds the error message to the log file. However, eStatement Manager does not log any errors that occur within the shell command; these must be handled separately.

For example, the following shell command would be useful after the Scanner task to ensure Windows files have the correct format for UNIX:

```
#!/bin/csh
# Preprocessor to run dos2unix on the input file

dos2unix $SHELL_INPUT $SHELL_INPUT.ux >& /dev/null
if ($status != 0) exit $status          # failure
echo $SHELL_INPUT.ux                    # new input file
exit 0                                  # success
```

The Environment variables field specifies the environment variables for the shell command. By default, the external command is passed the following environment variables:

- DDN - the name of the application to which the job belongs
- JOB_NAME - the name of the job to which the task is a part of.
- STATUS - the status of the job (has it been started, did it succeed/fail, etc).
- PREVIOUS_STATUS
- SHELL_INPUT - any input from a previous task. The SHELL_INPUT variable is only set if the shell command task is linked with another task in the context of a job. Otherwise it is `null`.

If your shell command requires any other environment variables, you'll need to specify them in this field.

Another Example of Defining a New Job Type

The following is another example that defines an index job called Indexer2 with the following tasks:

Job	Tasks
Indexer2	**Scanner**
	Indexer
	ShellCmdTask
	IXLoader
	AutoIndexVolAccept

As mentioned in the previous chapter, a reason for this new job type could be to run a validation engine (sum all amount due for example) on the output of the Indexer task. If the amount due exceeds a certain amount, it may require a careful verification of the input data stream as described in the SDK Module: Auditing Data Streams with the Verify API.

For this case you can create the following SQL script:

```
DECLARE jtid NUMBER;
BEGIN
 jtid := pwc_job_types.create_job_type('Indexer2');
 pwc_job_types.create_job_type_task(jtid, 'Scanner', 1);
 pwc_job_types.create_job_type_task(jtid, 'Indexer', 2);
 pwc_job_types.create_job_type_task(jtid, 'ShellCmdTask', 3);
 pwc_job_types.create_job_type_task(jtid, 'IXLoader', 4);
 pwc_job_types.create_job_type_task(jtid,
'AutoIndexVolAccept', 5);
 pwc_job_types.create_job_type_io(jtid, 'Indexer', 'data file
name', 'INPUT', 2, 'Scanner', 'output file name', 'OUTPUT',
1);
 pwc_job_types.create_job_type_io(jtid, 'Indexer', 'ddn volume
number', 'INPUT', 2, 'Scanner', 'ddn volume number', 'OUTPUT',
1);
 pwc_job_types.create_job_type_io(jtid, 'ShellCmdTask', 'input
params', 'INPUT', 3, 'Indexer', 'ir file name', 'OUTPUT', 2);
 pwc_job_types.create_job_type_io(jtid, 'IXLoader', 'index
volume number', 'INPUT', 4, 'Scanner', 'ddn volume number',
'OUTPUT', 1);
 pwc_job_types.create_job_type_io(jtid, 'IXLoader', 'ir file
name', 'INPUT', 4, 'Indexer', 'ir file name', 'OUTPUT', 2);
 pwc_job_types.create_job_type_io(jtid, 'AutoIndexVolAccept',
'index volume number', 'INPUT', 5, 'Scanner', 'ddn volume
number', 'OUTPUT', 1);
END;
```

.

9 Charting

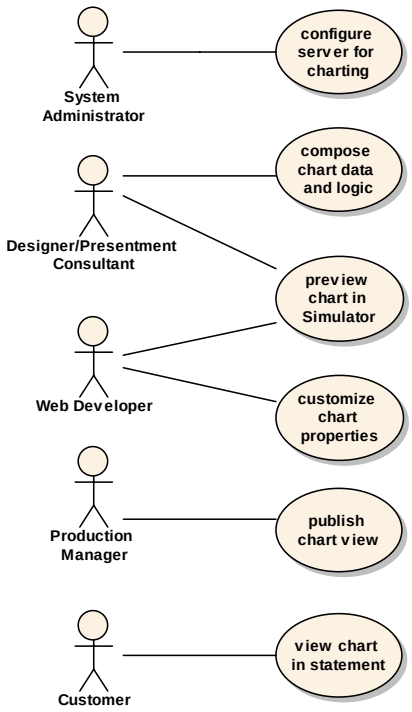
Introduction to Charting

About Charting in eStatement Manager

eStatement Manager can format statement data as a graphical chart in a dynamic HTML page. Charts consist of the **chart data**, which must be a table or group with at least two data rows, and the **chart properties**, which specify the type, design, and layout of the chart graphic.

Charting can involve most of the actors in a typical eStatement Manager workflow. The following overview diagram highlights the main tasks in the charting process.

Charting Use Case Overview Diagram



To present charts in online statements:

1 The **system administrator** follows the steps in Configuring Charting for Your Server to set up the display device, permissions, and awareness on the application server rendering the charts, and to install any specified fonts.

2. The eStatement Manager application **designer** follows the steps in Composing Charts in Statements to insert a chart placeholder in the **Application Logic File** (ALF) with the Composer.
3. The Web application **developer** or the **designer** follows the steps in Customizing Chart Properties to fine-tune the design, layout, and data presentation of the chart in the ALF and the **chart properties file.** Advanced designers and developers may extend the available chart properties for Designing Custom Charts with the Charting Servlet.
4. During the design process, the **designer** or **developer** can follow the steps in Simulating Charts to preview the chart in a simulated online statement with the Simulator API.
5. When the chart data and design are finalized, the **production manager** follows the steps in Publishing Charts to include the chart(s) in any dynamic online statements processed through the eaSuite Command Center.
6. Once the chart view and any associated views are published, the **customer** can view the chart as part of an online statement, so that charts refresh dynamically with each new version set of statement data.

Components of Charting

- Indexed data source (DDN and Indexer job)
- Application Logic File (ALF)
- Chart Properties File (*.properties)
- Simulator API
- HTML Web View for a Charting ALF
- Chart View for each Chart

Configuring Charting for Your Server

About Servers and Charting

The server rendering the charts, not the machine viewing the statement, determines font sizes and styles in charts. The server displaying charts must also have access **permissions** set to display charts, and requires **awareness** of an actual or virtual **display** device. This chapter discusses:

Fonts

Configuring a Headless Server for Charting

Display Devices and xvfb

Display Permissions and xhost

Display Awareness

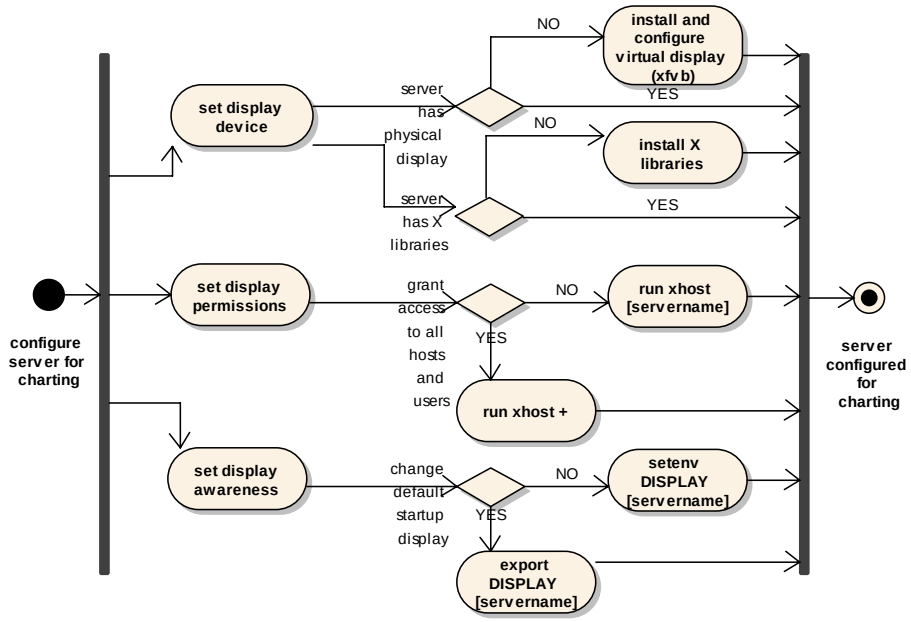
CAUTION: The configuration steps in this chapter apply primarily to **deployment** servers. Servers in a **production** environment often have physical display devices with graphics support, so that configuration may not be an issue. **Always test charts** (with the rest of your Web application) **on your deployment platform**, and make any needed configurations for your charts to display properly with the correct fonts and styles.

About Fonts

Charts require graphics utilities and fonts that vary across platforms. **Windows NT/2000** has rich support for both graphics and fonts. UNIX systems like **Solaris** and **AIX** support graphics with an **X server**, or by using a virtual display, for example **xvfb**. Either option can offer rich font and style support, depending on fonts installed.

Any fonts you reference in your chart properties must be available *on your deployment server*, not on the machine where your browser views the charts. If you receive "font not found" or similar error messages when charting, check the fonts and styles available on your X server against those in your chart properties file.

Configuration Activity Diagram



Setting Display Devices and xvfb

Like other Java graphics packages, Oracle charting extends the java.awt interface, which contains all of the classes for creating user interfaces and for painting graphics and images. These classes in turn require X libraries and access to an X display. To display charts properly, the Web server rendering the charts must have a real or virtual X display device and the necessary X libraries.

In a development environment, the Web server may have a real physical display device attached and have X Libraries loaded. However, at a typical server host site, few if any of the racks of server machines are connected to a display, and system administrators may hesitate to load X libraries if they are not installed.

If your deployment environment does not have a physical display **and** X libraries, your "headless" server needs a virtual X display like **xvfb**. The X Virtual Frame Buffer (**xvfb**) is an X server that can run on machines without display hardware or input devices. It emulates a dumb framebuffer using virtual memory.

xvfb may already be installed on your UNIX system, in **/usr/X11R6**. If not, you need to obtain and install a copy.

TIP: Documentation for **xvfb** (**man xvfb**) is hard to find. Many versions of UNIX have no manual entry for **xvfb** or have it in the wrong place. The University of Texas has posted **man xvfb** version 1 at http://dell5.ma.utexas.edu/cgi-bin/man-cgi?**xvfb**+1. NOAA also has an excellent README.**xvfb** and a binary of **xvfb** at ftp://ferret.wrc.noaa.gov/special_request/**xvfb**/solaris.

Setting Display Permissions and xhost

You can control access to your X server with the UNIX program **xhost**. This access control program can add and delete host names or user names to the list permitted to connect to the X server.

TIP: The privacy and security controls in **xhost** are generally sufficient for a single-user workstation environment. You may prefer to use a custom authentication system for stronger access control.

xhost is located in different places on different systems. Look in **/usr/openwin/bin** or **/usr/local/share** to start. Oracle recommends that you add **xhost** to your environment PATH variable.

To grant X server display access to all available hosts and users, type:

```
xhost +
```

xhost Syntax and Parameters

Security requires that **xhost** be run only from the controlling host. For workstations, this is the server machine. For X terminals, it is the login host. The command syntax is:

```
xhost [[+-]name ...]
```

Parameter	Description
[+]name	Adds the given host name or user name to the list allowed to connect to the X server. The plus sign is optional.
-name	Removes the given host name or user name from the list allowed to connect to the X server. Existing connections are not broken, but new connection attempts are denied.
+	Turns off access control; grants access to all host names and user names, even if not on the X server list.

Parameter	Description
-	Turns on access control; restricts access to only those host names and user names on the X server list.
nothing	Typing **xhost** without arguments prints a message indicating whether access control is enabled and listing those allowed to connect. This is the only option available to machines other than the controlling host.

CAUTION: Use care in removing hosts and users. **xhost** allows you to remove the current machine, but then will not permit further connections, including attempts to add it back. You must then reset the server in order to allow local connections again.

Setting Display Awareness

When you use X Windows tools, you must assign the environment variable DISPLAY to point to your local workstation, or wherever you would like the windows from the X Windows application displayed. When you run an application or Web server from the command line, your server uses the current DISPLAY environment variable.

TIP: If you are running an X server on a remote machine, and displaying the windows on your local machine, you may also have to run **xhost** on your local machine to allow windows to be opened there: **xhost +remote_machine**.

UNIX users can change where windows are displayed with the shell commands **setenv DISPLAY** or **export DISPLAY**.

To change the default display awareness and permissions:

1. Advanced users can modify the startup script for your application server.

   For WebLogic, the startup script is located at:

   ```
   <WL_HOME>/config/mydomain/startWebLogic.sh
   ```

   For WebSphere, the startup script is located at:

   ```
   <WS_HOME>/bin/startupServer.sh
   ```

2. Insert the following lines in your startup script, where *MyServer:2.0* is the name of your display:

   ```
   DISPLAY=MyServer:2.0
   export DISPLAY
   /usr/openwin/bin/xhost + webservername
   ```

3. Specifying the Web server name limits the X DISPLAY 2.0 to connections from the specified server. If the Web server name is omitted (**xhost +**), then any host machine can connect to X on the server.

For more information on working with application server scripts, see the *Installation Guide for Oracle Siebel eStatement Manager*.

Configuring a Headless Server for Charting

If your deployment environment does not have a physical display **and** X libraries, your "headless" server needs a virtual X display like **xvfb**. For more information on display devices, permissions, and awareness, see the previous sections.

TIP: The X Windows client for AIX systems requires the X11 package, which comes with the O/S but is not installed by default. To check whether X11 is installed, run **smit** and check the installed packages option for AIX Windows X11 libraries, or look in the default directory **/usr/lpp/X11**.

To enable charting on a "headless" server (Solaris):

Download **xvfb** from `ftp://www.ferret.noaa.gov/special_request/xvfb/solaris/`

Install to **/usr/X11R6. xvfb** will be installed in the /bin directory.

Enable X display permission on your Web server with the command **xhost +.**

To set the current display to use the frame buffer for graphics display, set your DISPLAY variable, for example:

```
DISPLAY=ella:1; export DISPLAY
```

This sends any graphics output going to display 1 to shared memory.

Run **xvfb** as a background process.

```
/usr/X11R6/bin/xvfb :1 -screen 0 800x600x24 &
```

The "&" closes the command window and leave the task running in the background.

This procedure creates a virtual display at :1.0 with a size of 800x600 pixels and a color depth of 24 bits. To ensure that your Java environment draws to this display, you must set the DISPLAY environment variable to :1.0 **before** invoking Java. If you receive an environment exception, try changing the color depth or screen size.

CAUTION: **xvfb** must be installed in the directory **/usr/X11R6**, as it looks in this directory for needed fonts. If these fonts are not found under **/usr/X11R6**, **xvfb** will fail.

Composing Charts in Statements

About Charting in the Composer

Web designers and developers can use the eStatement Manager **Composer** tool to define data objects and custom tags in HTML templates for eStatement Manager **applications**. Defining a **chart tag** for a **table** or **group** displays that data object as a graphical chart in the online statement.

The Composer GUI allows you to define only a few basic chart properties: a **chart type** of Pie, the X and Y-axes for data, and the width and height of the chart. Once you have created this "placeholder chart" in the Composer, you customize the look and feel of the chart by customizing chart properties in the **ALF** or the **chart properties file.**

For more information on working with the Composer, see the *Presentation Design (Composer Guide) for Oracle Siebel eStatement Manager*.

Inserting a Chart Tag in the Composer

To chart data for any table in the DDF, you can drag and drop tables into the HTML template using the WYSIWYG or the HTML editor. You can represent a table as either a text table or as a chart.

TIP: Tables to be charted must have **at least two fields**, one of which must contain **numeric values**. The Composer converts any values in non-numeric field types to numbers.

To insert a chart tag in the Composer:

1. Open an ALF in the Composer. For this example, open **NatlWireless\NW_LocSummary.alf.**
2. Click the **WYSIWYG** tab or **HTML** tab.
3. Click the Definition tab in the Tree.
4. Click to open Tables in the Tree.
5. Drag and drop the table  definitions into the HTML template. The table assumes the properties of the area in which it is placed. (Drag and drop the **LocalChargeSummary** table to the HTML template.)
6. Select **Add Chart.**
7. Select a field for the X-axis of the chart. (Select **LocalChargeAmount**.)
8. Select a field for the Y-axis of the chart. (Select **LocalChargeDesc**.)
9. Select the type of chart. (Select **Pie**.)

   CAUTION: Pie charts are the only chart type available through the Composer Selecting Bar or Line still generates a Chart Type of Pie in the ALF and the chart properties file. For how to create chart types other than pie charts, see "Customizing Chart Properties" on page 225.
10. Select the width and height settings for the chart. (Leave at 400 and 300 respectively.)
11. Enter the URL path to your Web application root, for example \Sample.
12. Click **OK**. (The tag **[E]LocalChargeSummary_0,U[/E]** appears.) This chart tag adds a placeholder for the chart to the HTML template.

    TIP: Make a note of the name of the table you are charting, which appears in the chart tag. When you publish a chart view, you must name the view with this table name, in this example LocalChargeSummary_0. This name will also match the name of the chart properties file created by the Composer.
13. Delete any temporary placeholders in the HTML template, for example "XX."
14. Click the **Save Template** icon.
15. Save the ALF by clicking the **Save ALF** icon.

TIP: When mapping a table to a template in the Composer, it is not necessary to encapsulate the table with HTML table row <*TR*> and table data <*TD*> (cell) tags. The table rows and table data (cells) are generated when the data is dynamically pulled from the data file and passed to the Oracle WebComposer object. This object formats the table rows, cells, and font characteristics of the data based on the settings defined in the Composer.

Naming Conventions for Charts

The Composer names each chart tag and properties file with the name of the table being charted, plus an incremental counter. For example, the first chart for the table **`LocalChargeSummary`** would generate the chart tag **`[E]LocalChargeSummary_0,U[/E]`** and the properties file **`LocalChargeSummary_0.properties.`**

If you created a second chart for the same table, the Composer would generate the chart tag **`[E]LocalChargeSummary_1,U[/E]`** and the properties file **`LocalChargeSummary_1.properties.`**

When you publish an HTML view, you select the application name (**`NatlWireless`**) and specify a view name (**`LocSummary`**). For each chart in your HTML view, you must give the matching view name (**`LocSummary`**) and name the Chart view name with the chart tag (**`LocalChargeSummary_0).`** This name allows eStatement Manager to match each published chart properties view with the correct chart tag in the ALF.

CAUTION: The chart properties file overrides ALF attributes. Do not rename charts in the Composer, the ALF, or the HTML template. Use the chart properties **`X.Axis.Title.String`** and **`X.Axis.Title.String`** to define more user-friendly names for chart titles and legends.

About Chart Tags in the ALF

The Composer writes the chart tag and properties into the ALF, which is an XML file.

The Composer creates many more default chart properties in the ALF than those you edit in the Chart dialog window. You can edit these properties directly in the ALF, or override them by editing the chart properties file. For tables of available properties and values in the ALF and in the chart properties file, see "Customizing Chart Properties" on page 225.

About The Chart Properties File

The Composer also stores your chart definition in a **chart properties file**, for example *LocalLineSummary_0.properties*. This file has the same name as the table data being charted, with a counter appended. The Composer creates the properties file in the same folder as the ALF and HTML template files. You must edit these properties directly in the chart properties file. For tables of available properties and values in the ALF and in the chart properties file, see "Customizing Chart Properties" on page 225.

TIP: You can chart the same data table in two different charts. This increments the counter in the chart tag and properties files, for example **`LocalLineSummary_0.properties`** and **`LocalLineSummary_1.properties`**.

About Simulating Charts

The Composer has a Simulator tool that allows you to view your sample data as published with the current HTML template. However, Composer simulation does **not** render charts. You use the Chart Simulator API, which is a command-line Java tool.

Before you simulate your chart, you probably want to edit the ALF and properties files to get a closer first approximation of your desired chart look and feel. You can then simulate, edit, and simulate again until you are satisfied with the final design and layout.

For more information, see "Customizing Chart Properties" on page 225; this includes procedures for using com.edocs.app.chart.Simulator.

Customizing Chart Properties

About Customizing Charts

The Composer allows you to set only a few chart properties directly. To customize the format and design of your charts, edit chart properties in the ALF file itself; in the chart properties file; and/or in the HTML template. Any of these files can be edited with the text editor of your choice.

CAUTION: When you make any manual edits to ALF files, make sure to validate the XML and check to see that it is well formed.

This chapter discusses how to customize chart properties in the ALF and in the chart properties file. It also describes how to simulate, or preview, charts.

About Chart Attributes in the ALF

The ALF, or Application Logic File, is an XML document that defines business logic and formatting for presenting statement data. An element of type ALF must contain certain required sub-elements:

```
<!ELEMENT ALF (VERSION, DATA_GROUP, DDF, SWITCH, HOME,
TEMPLATES, CONTENTS, CONDITIONS, PROFILES, BUSINESSCONDITIONS,
RECORDS, PAGE_ELEMENTS, composition-specs)>
```

Charts are defined as a subelement of the TEMPLATES element.

```
<!ELEMENT TEMPLATES (Template)+>

   <!ELEMENT Template (SECTIONS, CHARTS, GROUPS,
GroupTemplate*)>

   <!ATTLIST Template
```

The CHART element in its turn defines a list of chart attributes, listed in the Table of ALF Chart Attributes.

```
   <!ELEMENT CHARTS (Chart*)>

   <!ELEMENT Chart EMPTY>
```

```
<!ATTLIST Chart
```

Table of ALF Chart Attributes

Attribute Name	Description	Example
Name	Table name in Composer	LocalLineSummary_0
XField	X axis of chart	LocalLinePhNo
YField	Y axis of chart	LocalLineAmt
Type	Chart Type (pie, bar, &c)	Pie
HidePieLegend	Set to 1 only if Type=5 (pie)	0
AddValueToLegend	Displays the percentage in the chart legend 1=yes, 0=no	0
Height	Total height of the canvas in pixels	300
Width	Total width of the canvas in pixels	400
HidePieLegend	Toggles the display of legends for Pie charts	
BaseURL	Points to the Web application associated with the chart data. This property writes only to an existing directory, and does not create one if none exists.	/Sample
UnixChart	By default, the Composer sets UNIXChart=Pie. To create other types of charts, set the chart type in the properties file.	Pie

You may notice other attributes listed in the ALF. These attributes are placed in the ALF for backward compatibility with previous versions of eStatement Manager and have no effect on the current version of eStatement Manager.

The following XML example shows the default chart properties written to **`NatlWireless.alf`** after creating a chart tag for the **`LocalLineSummary`** table.

XML Example of Chart Attributes

```
<TEMPLATES>
        <Template Name="Default_Template">
            <SECTIONS/>
            <CHARTS>
```

```
                    <Chart Name=" _0"
RecordName="LocalLineSummary" TopTitle="Top Lable"
BottomTitle="Bottom Lable" LeftTitle="" RightTitle=""
XField="LocalLinePhNo" YField="LocalLineAmt" Key="0"
StackedStyle="0" ColorScheme="0" GridLines="0" Full3D="0"
AngleX="0" AngleY="0" Atribute="0" MarkerVolume="0" Shadow="0"
MultiShape="0" Dimension_3D="0" View3DDepth="0" Type="1"
CGITimeSpan="" BackgroundColor="White" ForgroundColor="Black"
Height="300" Width="400" LegendShow="1" LegendToolSize="100"
LegendToolStyle="167116800" HidePieLegend="0" SeriesColor=""
LeftGap="40" RightGap="40" ImgQuality="75" ImgSmooth="0"
AddValueToLegend="0" BaseURL="/Chart" UNIXChart="Pie"/>
            </CHARTS>
            <GROUPS/>
        </Template>
    </TEMPLATES>
```

Customizing a Chart in the ALF

Adding Percentages or Values to Labels

To display the chart with data values as labels, set **`AddValueToLegend=1.`**

To display the chart without data values, set **`AddValueToLegend=0.`**

Changing Axis Titles

By default, the Composer allows you to select from the names of table rows as titles for the X- and Y-axes. Usually, these titles are not suitable for presentment to end customers. Modify the text of axis titles in the chart properties file. *Do not modify titles in the ALF* as your changes will not stick.

CAUTION: Title values defined in the chart properties file (**`X.Axis.TitleString`** and **`Y.Axis.TitleString`**) take precedence over those defined in the ALF (**`XField`** and **`YField`**.

Suppressing Percentage Values in Pie Charts

Pie charts (type=5) have percentage values for each slice set as the default. To suppress these values, set the URL T/F flag in the ALF for HidePieLegend attribute to 1. This applies to pie charts **only**.

Customizing the Chart Properties File

The chart properties file is a list of name-value pairs that control the graphic look and feel of the chart: its type, legend, labels, axes, et cetera.

- The first and most important property is Type. This determines whether the data appears as a pie, line, bar, or other type of chart. Note that *this property name and its value are both case sensitive*. All the remaining property names and their values are case insensitive.

- The naming convention of a chart property indicates its scope. For example, properties **`Chart.*`** affect the entire chart, while properties **`X.Axis.*`** affect the X-axis only. The final element of the property name indicates the property being set.
- Color and font properties have three sub-properties each. To define a color, specify individual RGB values between 0 and 255. To define a font, specify its name, style, and size.
- Most display properties are Boolean (true/false); for example, whether to display axis title or gridlines, or to display the legend vertically.
- Do not set properties that are not applicable to a chart type. For example, do not set Axis properties when requesting a Pie chart. Do not set Bar properties while rendering a Pie chart.
- For charts created using the Composer tool, the chart types: **`HiLoBar`**, **`HorizHiLoBar`**, and **Speedo** are not available, as these charts typically require additional data.

Chart Type

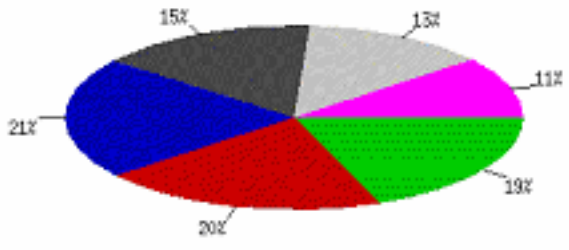
The primary chart property is **`Type`**, which defines the visual representation of the data. To create a pie chart, set **`Type=Pie.`** To create a bar chart, set **`Type=Bar`**.

CAUTION: Both the Type property and its value are case-sensitive, unlike other chart properties in the properties file.

This section illustrates each available chart type for this example dataset.

```
X-axis label = {"Jan-Feb", "Mar-Apr", "May-Jun",
          "Jul-Aug", "Sep-Oct", "Nov-Dec"};
DataSet for 1999 = {1000.0, 1200.0, 1400, 1900.0, 1800.0,
1700.0};
DataSet for 2000 = {900.0, 1100.0, 1300, 1800.0, 1700.0,
1600.0};
X-Axis = Months
Y-Axis = Fuel Consumption;
```

Pie

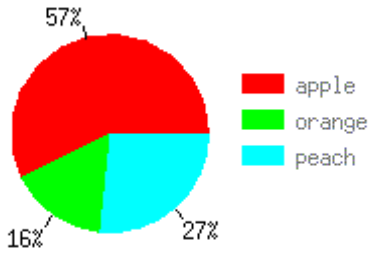


Pie chart with one slice per data point.

To define pie properties:

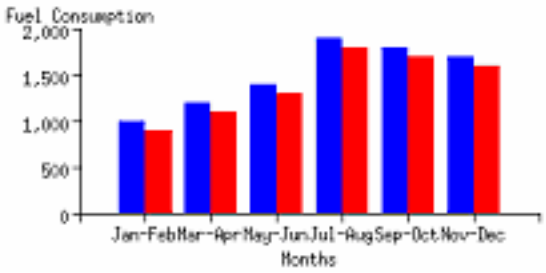
`Pie.*` properties should be defined only when **`Type=Pie.`** These properties control the aspect ratio (height and width) of the pie; the angle, size, and colors of the slices; and the labels and legends. For a round pie, set **`Pie.Height`** and **`Pie.Radius`** based on the height and width of the chart in pixels.

Property	Default	Description
PieLabelPosition	=2	Defines the position of the pie slice labels.
PieStartDegrees	=0	Defines the angle of the first pie slice.
PieTextLabelsOn	=false	**true** displays pie slice name, for example *College Fund*
PieValueLabelsOn	=false	**true** displays the numeric data value of each pie slice, for example 30.5
PiePercentLabelsOn	=true	**true** displays percent of total for each pie slice, for example 30.5%
PieLabelColor.Blue	=0	RGB value of blue (0-255).
PieLabelColor.Green	=0	RGB value of green (0-255).
PieLabelColor.Red	=0	RGB value of red (0-255).
PieLabelFont.Name	=Times New Roman	same as java.awt.font
PieLabelFont.Style	=plain	same as java.awt.font
PieLabelFont.Size	=12	same as java.awt.font
Pie.Height	=0.5	Vertical dimension of the pie, as a percentage of plot area height. Default value produces a circle.
PieWidth	=0.33	Horizontal dimension of the pie, as a percentage of plot area width. Default value produces a circle.
PieXLoc	=0.5	Horizontal center of the pie, as a percentage of plot area height.
Pie.YLoc	=0.5	Vertical center of the pie, as a percentage of plot area height.

Sample Properties	Sample Chart
height=200 pixels width=300 pixels Pie.Height=0.5 (0.5 * 200=100) Pie.Width=0.33 (0.33 * 300=100) Chart.LegendVisible=true Legend.IconGap=0.02 Legend.IconHeight=0.05 Legend.IconWidth=0.07 Legend.LabelColor.Red=123 Legend.LabelColor.Blue=126 Legend.LabelColor.Green=129 Legend.LlX=0.7 Legend.LlY=0.4 Legend.VerticalLayout=false Pie.LabelPosition=2 Pie.PercentLabelsOn=true Pie.StartDegrees=0 Pie.TextLabelsOn=false Pie.ValueLabelsOn=false Pie.XLoc=0.5 Pie.YLoc=0.5	

TIP: If labels appear too crowded, you can use a legend instead. Set **Chart.Legend.Visible**=true and specify values for legend height, width, and color. Turn off pie labels by setting **TextLabelsOn**, **ValueLabelsOn**, and **PercentLabelsOn** properties to *false*.

Bar

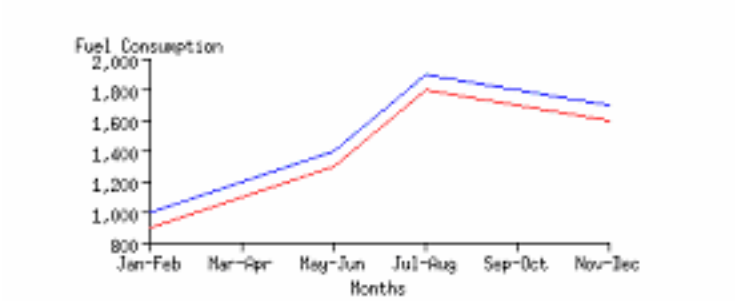


Displays each data series **vertically** in a single color (sometimes called a column chart). To display horizontally, use **HorizBar**.To display different colors for each bar in a series, use **IndBar** (horizontal) or **IndColumn** (vertical).

To define bar properties:

Property	Default	Description
Bar.Baseline	=0.0	Value from which bars ascend or descend. Default is X-axis.
Bar.ClusterWidth	=0.8	Width of a cluster of bars, as a percentage of the available space (1.0 means that clusters touch; 0.5 means that clusters are as wide as the space separating clusters).
Bar.DoClip	=false	**true** clips bar values to the outer edge of the plot area (off by default).

Line



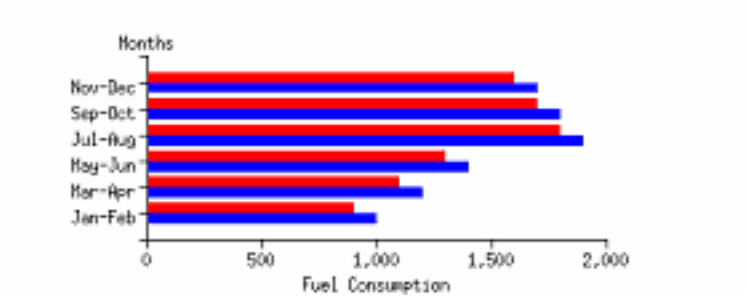
Displays data values as lines on a graph, without value labels for each data point.

To define LineClip property:

To clip line values at the boundary of the plot area, set **`LineClip=true`**. Default is **`Clip=false`**.

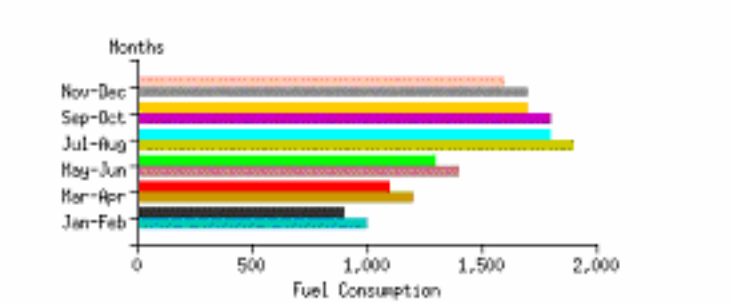
Vertical bar chart with High and Low values indicated.

Horizontal Bars (HorizBar)



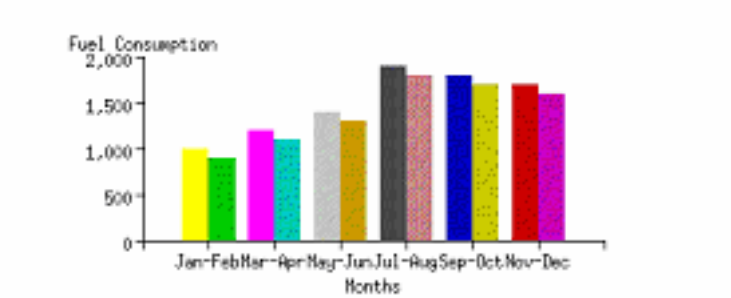
Displays each data series **horizontally** in a single color. To display vertically, (sometimes called a column chart). To display different colors for each bar in a series, use **`IndBar`** (horizontal) or **`IndColumn`** (vertical).

Individually Colored Bars (IndBar)



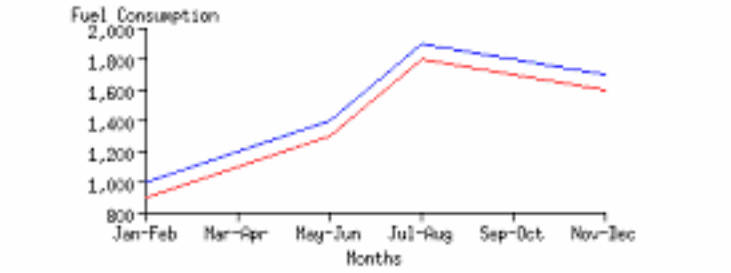
Horizontal bar chart that uses a separate color for each bar.

Individually Colored Columns (IndColumn)



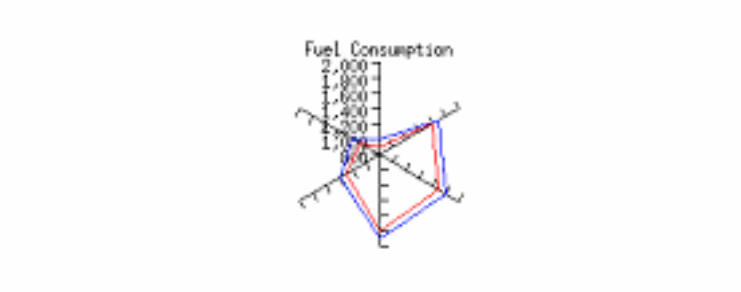
Vertical bar chart that uses a separate color for each bar.

(LabelLine)



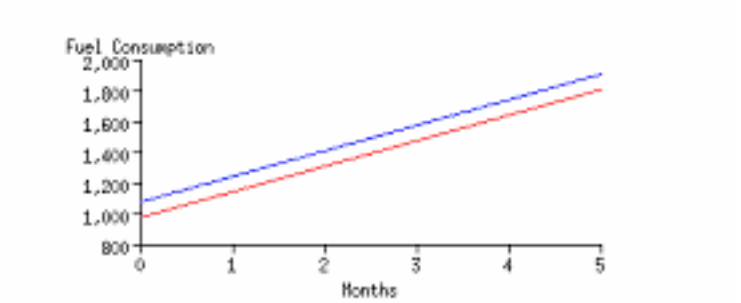
Displays data values as lines on a graph, with user-defined labels on the X-axis.

Polar Chart (Polar**)**



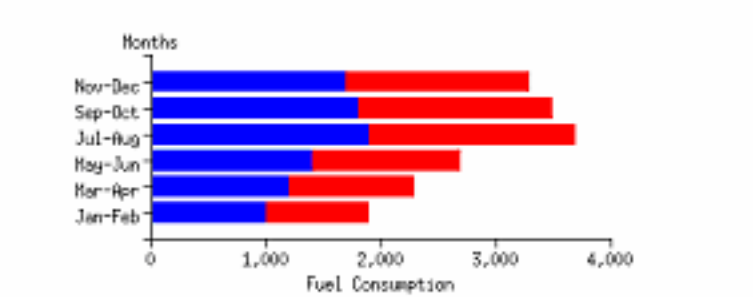
A chart that looks like a radar screen. Plots only one data value, but the scale is determined by all the data.

Regression (Regress**)**



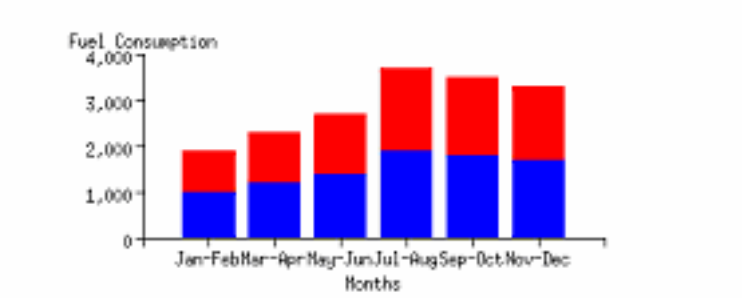
A chart that plots OLS regression for data values.

Stacked Bars (StackBar**)**



A chart that stacks data values horizontally.

Stack Column Chart (StackColumn)



A chart that stacks data values vertically.

Stick Chart (Stick)

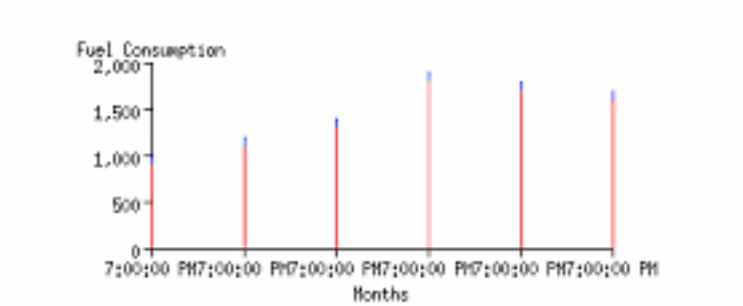


Chart that draws a vertical line to the Y-axis height of each data value.

StickBar Chart (StickBar)

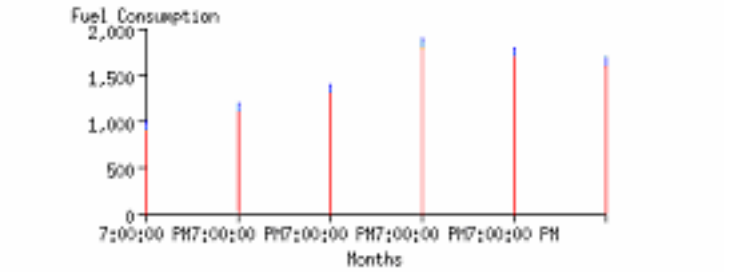


Chart that draws a vertical line to the Y-axis height of each data value.

Other Chart Properties

General Properties

Property	Default	Description
Chart.LegendVisible	FALSE	true sets legend visible. Invisible by default.
Chart.Name	MyChart	User-defined string for chart title.
Chart.ThreeD	FALSE	true displays chart with 3D drop shadows.
Chart.XAxisVisible	TRUE	true sets X-axis visible (default).
Chart.XOffset	0	number of pixels of offset in X direction for 3D effect (default 0)
Chart.YAxisVisible	TRUE	true sets Y-axis visible (default).
Chart.YOffset	0	number of pixels of offset in Y direction for 3D effect (default 0)
ChartQuality	=1	Set to 1 for highest quality (larger) image, 0 for lower quality (smaller) image.

Background Properties

Titles and sub-titles are elements of the chart background. Their color, font and string value are controlled by the following properties:

Property	Default	Description
Background.Gc.FillColor.Blue	0	RGB value of blue (0-255).
Background.Gc.FillColor.Green	0	RGB value of green (0-255).
Background.Gc.FillColor.Red	0	RGB value of red (0-255).
Background.Gc.Image	<unimplemented>	UNIMPLEMENTED. Sets a background image for the chart. **Do not use.**
Background.Gc.LineColor.Blue	0	RGB value of blue (0-255).
Background.Gc.LineColor.Green	0	RGB value of green (0-255).
Background.Gc.LineColor.Red	0	RGB value of red (0-255).
Background.Gc.LineWidth	1	Sets line width in pixels.
Background.Gc.MarkerColor.Blue	0	RGB value of blue (0-255).
Background.Gc.MarkerColor.Green	0	RGB value of green (0-255).
Background.Gc.MarkerColor.Red	0	RGB value of red (0-255).

Property	Default	Description
Background.SubTitleColor.Blue	0	RGB value of blue (0-255).
Background.SubTitleColor.Green	0	RGB value of green (0-255).
Background.SubTitleColor.Red	0	RGB value of red (0-255).
Background.SubTitleFont.name	Times New Roman	Uses available values from **`java.awt.font.`**
Background.SubTitleFont.size	12	Uses available values from **`java.awt.font.`**
Background.SubTitleFont.style	plain	Uses available values from **`java.awt.font.`**
Background.SubTitleString	null	User-defined string for the background subtitle.
Background.TitleColor.Blue	0	RGB value of blue (0-255).
Background.TitleColor.Green	0	RGB value of green (0-255).
Background.TitleColor.Red	0	RGB value of red (0-255).
Background.TitleFont.name	Times New Roman	Uses available values from java.awt.font.
Background.TitleFont.size	12	Uses available values from java.awt.font.
Background.TitleFont.style	plain	Uses available values from java.awt.font.
Background.TitleString	null	User-defined string for the background title.

Plot Area Properties

The plot area is the region bounded by the axes; where the data are plotted. These properties specify the fill color for this region, and marker and grid line settings.

Property	Default	Description
Plotarea.Gc.FillColor.Blue	0	RGB value of blue (0-255).
Plotarea.Gc.FillColor.Green	0	RGB value of green (0-255).
Plotarea.Gc.FillColor.Red	0	RGB value of red (0-255).
Plotarea.Gc.LineColor.Blue	0	RGB value of blue (0-255).
Plotarea.Gc.LineColor.Green	0	RGB value of green (0-255).
Plotarea.Gc.LineColor.Red	0	RGB value of red (0-255).
Plotarea.Gc.LineWidth	1	Sets line width in pixels.
Plotarea.Gc.MarkerColor.Blue	0	RGB value of blue (0-255).
Plotarea.Gc.MarkerColor.Green	0	RGB value of green (0-255).

Property	Default	Description
Plotarea.Gc.MarkerColor.Red	0	RGB value of red (0-255).

Axis Properties

The axis properties control the location of the axis on the canvas, its major and minor ticks, title, grid, and labels.

- Properties listed here are for the X-axis. An identical set of properties exists for the Y-axis, **`Y.Axis.*`**.
- **`Axis.Start`**: By default, axes automatically determine a starting and ending value. By setting this value, you can give the axis a default minimum value. If the Axis is set to **`noAutoScale`**, this value is used directly. Otherwise, this value may be adjusted slightly to yield better-looking labels. For example, if you set **`X.AxisStart`** to 0.01, the chart may decide to round the value down to 0.0 to create even axis increments.

Property	Default	Description
X.Axis.AutoScale	TRUE	Automatically creates X axis scale based on data values (default).
X.Axis.AxisEnd	6	Ending value of X-axis. Set to greater than or equal to number of data points anticipated.
X.Axis.AxisStart	0	Starting value on X-axis.
X.Axis.BarScaling	TRUE	Scales bars to axis length. Set true for bar charts (default).
X.Axis.GridGc.FillColor.Blue	0	RGB value of blue (0-255).
X.Axis.GridGc.FillColor.Green	0	RGB value of green (0-255).
X.Axis.GridGc.FillColor.Red	0	RGB value of red (0-255).
X.Axis.GridGc.LineColor.Blue	0	RGB value of blue (0-255).
X.Axis.GridGc.LineColor.Green	0	RGB value of green (0-255).
X.Axis.GridGc.LineColor.Red	0	RGB value of red (0-255).
X.Axis.GridGc.LineWidth	1	Sets line width in pixels.
X.Axis.GridGc.MarkerColor.Blue	0	RGB value of blue (0-255).
X.Axis.GridGc.MarkerColor.Green	0	RGB value of green (0-255).
X.Axis.GridGc.MarkerColor.Red	0	RGB value of red (0-255).
X.Axis.GridVis	FALSE	True sets X-axis grid lines visible (invisible by default).
X.Axis.LabelAngle	0	Sets the number of degrees to rotate X axis labels.
X.Axis.LabelColor.Blue	0	RGB value of blue (0-255).
X.Axis.LabelColor.Green	0	RGB value of green (0-255).

Property	Default	Description
X.Axis.LabelColor.Red	0	RGB value of red (0-255).
X.Axis.LabelFont.Name	Times Roman	Uses available values from java.awt.font.
X.Axis.LabelFont.Size	12	Uses available values from java.awt.font.
X.Axis.LabelFont.Style	plain	Uses available values from java.awt.font.
X.Axis.LabelFormat	null	Defines data format for labels, for example first three letters of month name.
X.Axis.LabelPrecision	2	Sets the number of digits past the decimal point to display X axis labels.
X.Axis.LabelVis	TRUE	True sets X axis labels visible (default).
X.Axis.LineGc.FillColor.Blue	0	RGB value of blue (0-255).
X.Axis.LineGc.FillColor.Green	0	RGB value of green (0-255).
X.Axis.LineGc.FillColor.Red	0	RGB value of red (0-255).
X.Axis.LineGc.LineColor.Blue	0	RGB value of blue (0-255).
X.Axis.LineGc.LineColor.Green	0	RGB value of green (0-255).
X.Axis.LineGc.LineColor.Red	0	RGB value of red (0-255).
X.Axis.LineGc.LineWidth	1	Sets line width in pixels.
X.Axis.LineGc.MarkerColor.Blue	0	RGB value of blue (0-255).
X.Axis.LineGc.MarkerColor.Green	0	RGB value of green (0-255).
X.Axis.LineGc.MarkerColor.Red	0	RGB value of red (0-255).
X.Axis.LineVis	TRUE	True sets X axis lines visible (default).
X.Axis.LogScaling	FALSE	True sets X axis to use log scaling; linear by default.
X.Axis.MajTickLength	5	Sets length of X axis major ticks in pixels.
X.Axis.MajTickVis	TRUE	Sets X axis major ticks visible (default).
X.Axis.MinTickLength	2	Sets length of X axis minor ticks in pixels.
X.Axis.MinTickVis	FALSE	Sets X axis minor ticks visible (default).
X.Axis.NumGrids	5	Sets the number of grid lines on the X axis to set to **noAutoScale**.
X.Axis.NumLabels	5	Sets the number of labels on the X axis to set to **noAutoScale**.
X.Axis.NumMajTicks	5	Sets the number of major ticks on the X axis to set to **noAutoScale**.
X.Axis.NumMinTicks	10	Sets the number of minor ticks on the X axis to set to **noAutoScale**.

Property	Default	Description
X.Axis.Plotarea.LlX	0.2	Shifts the horizontal position of the axis start on the canvas. Negative values shift left, positive shift right.
X.Axis.Plotarea.LlY	0.2	Shifts the vertical position of the axis start on the canvas. Negative values shift down, positive shift up.
X.Axis.Plotarea.UrX	0.8	Sets the upper right X location of the plot area as a double ranging from 0 to 1.
X.Axis.Plotarea.UrY	0.8	Sets the upper right X location of the plot area as a double ranging from 0 to 1.
X.Axis.Side	1	
X.Axis.TickGc.FillColor.Blue	0	RGB value of blue (0-255).
X.Axis.TickGc.FillColor.Green	0	RGB value of green (0-255).
X.Axis.TickGc.FillColor.Red	0	RGB value of red (0-255).
X.Axis.TickGc.LineColor.Blue	0	RGB value of blue (0-255).
X.Axis.TickGc.LineColor.Green	0	RGB value of green (0-255).
X.Axis.TickGc.LineColor.Red	0	RGB value of red (0-255).
X.Axis.TickGc.LineWidth	1	Sets line width in pixels.
X.Axis.TickGc.MarkerColor.Blue	0	RGB value of blue (0-255).
X.Axis.TickGc.MarkerColor.Green	0	RGB value of green (0-255).
X.Axis.TickGc.MarkerColor.Red	0	RGB value of red (0-255).
X.Axis.TitleColor.Blue	0	RGB value of blue (0-255).
X.Axis.TitleColor.Green	0	RGB value of green (0-255).
X.Axis.TitleColor.Red	0	RGB value of red (0-255).
X.Axis.TitleFont.Family	Times New Roman	Uses available values from **java.awt.font**.
X.Axis.TitleFont.Name	Times New Roman	Uses available values from **java.awt.font**.
X.Axis.TitleFont.Size	12	Uses available values from **java.awt.font**.
X.Axis.TitleFont.Style	plain	Uses available values from **java.awt.font**.
X.Axis.TitleString		User-defined string for X axis title.
X.Axis.UseDisplayList	FALSE	UNIMPLEMENTED. Retrieves objects using mouse click events. Do not use.

Legend Properties

Property	Default	Description
Legend.BackgroundGC.Gc.FillColor.Blue	0	RGB value of blue (0-255).
Legend.BackgroundGC.Gc.FillColor.Green	0	RGB value of green (0-255).
Legend.BackgroundGC.Gc.FillColor.Red	0	RGB value of red (0-255).
Legend.BackgroundGC.Gc.Image	<unimplemented>	Sets a background image for the legend. Do not use.
Legend.BackgroundGC.Gc.LineColor.Blue	0	RGB value of blue (0-255).
Legend.BackgroundGC.Gc.LineColor.Green	0	RGB value of green (0-255).
Legend.BackgroundGC.Gc.LineColor.Red	0	RGB value of red (0-255).
Legend.BackgroundGC.Gc.LineWidth	1	Sets line width in pixels.
Legend.BackgroundGC.Gc.MarkerColor.Blue	0	RGB value of blue (0-255).
Legend.BackgroundGC.Gc.MarkerColor.Green	0	RGB value of green (0-255).
Legend.BackgroundGC.Gc.MarkerColor.Red	0	RGB value of red (0-255).
Legend.BackgroundVisible	TRUE	Set false to avoid displaying background in chart legend.
Legend.IconGap	0.02	Separation between rows of the legend.
Legend.IconHeight	0.05	Legend icon height 0 < k < 1.0, where 1.0 = full height of canvas
Legend.IconWidth	0.07	Legend icon width 0 < k < 1.0, where 1.0 = full width of canvas
Legend.LabelColor.Blue	0	RGB value of blue (0-255).
Legend.LabelColor.Green	0	RGB value of green (0-255).
Legend.LabelColor.Red	0	RGB value of red (0-255).
Legend.LabelFont.Name	Times New Roman	Uses available values from **java.awt.font.**
Legend.LabelFont.Size	12	Uses available values from **java.awt.font.**
Legend.LabelFont.Style	plain	Uses available values from **java.awt.font.**
Legend.LlX	0	lower x left corner 0 < y < 1.0, 1.0 = full width
Legend.LlY	0	lower y left corner 0 < y < 1.0, 1.0 = full height

Property	Default	Description
Legend.UrX	0	upper x right corner 0 < y < 1.0, 1.0 = full width
Legend.UrY	0	upper y right corner 0 < y < 1.0, 1.0 = full height
Legend.VerticalLayout	FALSE	Set true to display legend at side of chart, false to display below chart.

Favorite Colors Properties

You can specify **favorite colors** as RGB values, to fill pie slices, bars, and other data values. For example, you may want to match the chart colors to the color scheme of the embedding page. Favorite colors are specified using the prefix **`Favorite.1.Color`**, where 1 is the first color in the series.

- **Number:** Favorite colors become active when the number of colors defined is greater than or equal to the number of data points displayed (unless for individual colors). For example, if a Pie has six slices but only five favorite colors specified, the favorite colors are not used. This is because there is no way to guess which colors would go well with those already specified.
- **Order**: The favorite colors are used in the order specified. Define each color to be distinguishable from adjacent colors for contrast and readability.

This example defines two favorite colors:

```
Favorite.1.Color.Red=201
Favorite.1.Color.Blue=92
Favorite.1.Color.Green=132

Favorite.2.Color.Red=51
Favorite.2.Color.Blue=52
Favorite.2.Color.Green=53
```

Default Chart Properties

The following properties are the default values created in the chart properties file. For a full listing, see "Default Properties and Attributes" on page 261.

```
Type=Pie
Legend.BackgroundVisible=true
Legend.IconGap=0.02
Legend.IconHeight=0.05
Legend.IconWidth=0.07
Legend.LabelColor.Red=0
Legend.LabelColor.Blue=0
Legend.LabelColor.Green=0
Legend.LabelFont.Name=Times New Roman
```

```
Legend.LabelFont.Style=plain
Legend.LabelFont.Size=12
Legend.LlX=0.0
Legend.LlY=0.0
Legend.UrX=0.0
Legend.UrY=0.0
Legend.VerticalLayout=false
X.Axis.CullingLabel=false;
X.Axis.AutoScale=true
X.Axis.AxisEnd=6.0
X.Axis.AxisStart=0.0
X.Axis.BarScaling=true
X.Axis.GridVis=false
X.Axis.LabelAngle=0
X.Axis.LabelColor.Red=0
X.Axis.LabelColor.Blue=0
X.Axis.LabelColor.Green=0
X.Axis.LabelFont.Name=Times New Roman
X.Axis.LabelFont.Style=plain
X.Axis.LabelFont.Size=12
X.Axis.LabelFormat=null
X.Axis.LabelPrecision=2
X.Axis.LabelVis=true
X.Axis.LineVis=true
X.Axis.LogScaling=false
X.Axis.MajTickLength=5
X.Axis.MajTickVis=true
X.Axis.MinTickLength=2
X.Axis.MinTickVis=false
X.Axis.NumGrids=5
X.Axis.NumLabels=5
X.Axis.NumMajTicks=5
X.Axis.NumMinTicks=10
X.Axis.Plotarea.LlX=0.2
X.Axis.Plotarea.LlY=0.2
X.Axis.Plotarea.UrX=0.8
```

```
X.Axis.Plotarea.UrY=0.8
X.Axis.TitleRotated=true
X.Axis.TitleColor.Red=0
X.Axis.TitleColor.Blue=0
X.Axis.TitleColor.Green=0
X.Axis.TitleFont.Family=TimesNewRoman
X.Axis.TitleFont.Name=Times New Roman
X.Axis.TitleFont.Style=plain
X.Axis.TitleFont.Size=12
X.Axis.TitleString=RemembertosetXaxistitle!
X.Axis.UseDisplayList=false
Y.Axis.CullingLabel=false;
Y.Axis.AutoScale=true
Y.Axis.AxisEnd=6.0
Y.Axis.AxisStart=0.0
Y.Axis.BarScaling=true
Y.Axis.GridVis=false
Y.Axis.LabelAngle=0
Y.Axis.LabelColor.Red=0
Y.Axis.LabelColor.Blue=0
Y.Axis.LabelColor.Green=0
Y.Axis.LabelFont.Name=Times New Roman
Y.Axis.LabelFont.Style=plain
Y.Axis.LabelFont.Size=12
Y.Axis.LabelFormat=null
Y.Axis.LabelPrecision=2
Y.Axis.LabelVis=true
Y.Axis.LineVis=true
Y.Axis.LogScaling=false
Y.Axis.MajTickLength=5
Y.Axis.MajTickVis=true
Y.Axis.MinTickLength=2
Y.Axis.MinTickVis=false
Y.Axis.NumGrids=5
Y.Axis.NumLabels=5
Y.Axis.NumMajTicks=5
```

```
Y.Axis.NumMinTicks=10
Y.Axis.Plotarea.LlX=0.2
Y.Axis.Plotarea.LlY=0.2
Y.Axis.Plotarea.UrX=0.8
Y.Axis.Plotarea.UrY=0.8
X.Axis.TitleRotated=true
Y.Axis.TitleColor.Red=0
Y.Axis.TitleColor.Blue=0
Y.Axis.TitleColor.Green=0
Y.Axis.TitleFont.Family=TimesNewRoman
Y.Axis.TitleFont.Name=Times New Roman
Y.Axis.TitleFont.Style=plain
Y.Axis.TitleFont.Size=12
Y.Axis.TitleString=RemembertosetXaxistitle!
Y.Axis.UseDisplayList=false
Pie.LabelColor.Red=0
Pie.LabelColor.Blue=0
Pie.LabelColor.Green=0
Pie.LabelFont.Name=Times New Roman
Pie.LabelFont.Style=plain
PieLabelFont.Size=12
Pie.LabelPosition=2
Pie.PercentLabelsOn=true
Pie.StartDegrees=0
Pie.TextLabelsOn=false
Pie.ValueLabelsOn=false
Pie.Height=0.6
Pie.Width=0.6
Pie.XLoc=0.5
Pie.YLoc=0.5
Chart.LegendVisible=false
Chart.Name=MyChart
Chart.ThreeD=false
Chart.XOffset=0
Chart.YOffset=0
Chart.YAxisVisible=true
```

```
Chart.XAxisVisible=true
Chart.Quality=1.0
```

Customizing Default Properties

The default settings for the chart properties file are stored in the text file **ChartDefaults.properties** inside the **com.edocs.app.chart** directory of **edx_servlet.jar**. If you are creating a series if similar charts, you may find it convenient to modify the default properties as a base template.

The **Sample** Web application contains two instances of **edx_servlet.jar:** one in the **WEB-INF/lib** directory of the EAR file **ear-Sample-ear** and another in the **WEB-INF/lib** directory of WAR file **war-Sample.war**. You can edit either instance of **ChartDefaults.properties** as long as you add the edited version of **edx_servlet.jar** to your classpath.

Unjar the EAR and WAR archive files, and then unjar **edx_servlet.jar,** to find **ChartDefaults.properties** in each archive. For example, the path to the default properties file in the WAR file for a default installation of eStatement Manager (WebLogic for Windows 2000) is:

```
C:/eStatement/samples/Sample/J2EEApps/weblogic/ear-sample/war-sample/WEB-
INF/lib/edx_servlet/ com/edocs/app/chart/ChartDefaults.properties
```

Open **ChartDefaults.properties** with a text editor and make any desired changes to the default properties; for example, change the default if you are creating a series of bar charts. Jar up the servlet, WAR, and EAR files, and add your modified **edx_servlet.jar** to your classpath. Now, when you create a chart in the Composer, the default chart properties should reflect your new settings.

Previewing Charts with com.edocs.app.chart.Simulator

The Chart Simulator API is a simple command line interface for **com.edocs.app.chart.ChartClient.java.** The API **chart.Simulator** requires a Java environment with **javachart.jar, edx_servlet.jar** and **edx_client.jar** in the default classpath. These JAR files are installed with eStatement Manager, but *you must add them to your classpath* to use the Simulator.

Setting the Display Environment for Simulation

You can control where your charts are simulated by setting your display awareness to either your production or deployment server. For simple previews to check if data is being correctly retrieved, a local simulation on your production workstation may be fine. However, remember that the appearance of your chart is controlled by the **deployment server**. This server may have different fonts available or be running a virtual display with different resolution or other graphics settings. Always preview your charts in a deployment environment before finalizing your chart properties and ALF.

For more information on display environments, including display devices, permissions, and awareness, see "Configuring Charting for Your Server" on page 218.

Formatting Data Strings for Chart Data

The Charting API passes chart data directly as one or more encoded data strings, for example:

```
"Series1*F*30*apple*56.8*orange*12.5*banana"
```

Enclose the data string in quotes, and separate each value with an asterisk (*). The first value in the string must be the data series name. The second is a T/F value that determines whether to label data values with the pie slice percentage (T=labels, F=no labels).

All remaining values in the string must be value/name pairs, where the first item in each pair is the numeric data value and the second item is its label name.

TIP: Remember to put values first, then labels (the opposite of a standard name/value pair).

The Simulator also takes parameters for the height and width of the chart canvas. If you find that labels or legends are clipped or cut off, adjust your chart canvas and properties file settings to accommodate the maximum length of legends and other objects.

To simulate a chart:

1. Edit your chart properties file.
2. Add **javachart.jar, edx_servlet.jar** and **edx_client.jar** to your classpath.
3. (optional) Create a text file with your formatted data strings, which you can then copy and paste into the command.
4. Run the Simulator from the command line.

```
java com.edocs.app.chart.Simulator propsFileName
imgOutputFilename.jpg width height encodedDataString1
encodedDataString2
```

5. View the generated image file in your browser. By default, the generated image is saved in the directory containing the ALF and properties file.

The Composer can simulate only charts containing a single data string. To display multiple datasets, you must customize the charting servlet to extract and present data as multiple strings. For more information, see your Oracle Professional Services representative.

Parameters

Parameter	Description
propsFileName	Chart properties file
imgOutputFilename	File name for image output. JPEG format required
Width	Width of chart canvas in pixels.
Height	Height of chart canvas in pixels.
encodedDataString	A single data string enclosed in quotes and separated by asterisks. See “Formatting Data Strings for Chart Data” on page 245.

Publishing Charts

About Publishing Charts

The eStatement Manager application server compiles charts dynamically at Web time. When an HTML template contains a chart tag, the Web application requests the table from the eStatement Manager database and posts an HTTP request to the charting servlet. The servlet uses the chart properties file published for that version set to format the data from the table, and renders the chart as a JPEG embedded in the dynamic HTML.

Before Publishing Charts

To prepare your application for chart viewing:

1. Create an application, for example, **`NatlWireless`**.
2. Create, configure, and run an Indexer job to index your dataset.
3. Publish an HTML Web view.

Publishing a Chart View

You must publish a CHART view for *each individual chart* in an online statement. Chart views merge the DDF and ALF information from the specified HTML Web view with the properties you defined for the chart, and embed the chart data in the Chart URL published by the HTML Web view.

CAUTION: If you publish an HTML Web view for an ALF with a chart tag, you must publish a CHART view for that chart before it will display. If there is no chart view available, the HTML Web view displays only a placeholder for the chart graphic.

To publish a Chart view:

1. In the Command Center, select Publisher from the navigation bar. A new Publisher window appears.
2. Select Create from the navigation bar. The Create a version set for CHART window appears.
3. Select the name of your application from the dropdown list, for example, NatlWireless. The view type is set to CHART.
4. Enter the view name associated with this chart. This view name must be the name of the chart, for example LocChargeSummary_0.
5. Enter the path to the chart properties file. This file must have an extension of **`*.properties`**, for example:

   ```
   C:/eStatement/samples/NatlWireless/LocChargeSummary_0.properties
   ```

6 If you have modified the view name since creating your ALF, update the ALF to point to the correct Web view for your application,

7 Click **Submit** to publish your chart view.

Viewing Charts in Statements

To view charts in statements:

1 Index your statement data with an Indexer job.

2 Publish at least one HTML Web view with an ALF containing chart tag(s).

3 Publish a Chart view **for each chart**.

4 Browse your Web application, for example Sample, and enroll one or more customers.

5 Log in as the customer whose statement you wish to view and browse the statement with the chart(s).

Designing Custom Charts with the Charting Servlet

About The Charting Servlet

The previous chapters describe how to use the Oracle charting servlet to compose and publish charts in online statements. The **`com.edocs.app.chart`** API allows you to create your own charting servlets to generate customized charts. Your servlet will create an instance of the **`ChartClient`** class.

CAUTION: Ensure that you set servlet response type appropriately before sending any output to the servlet. The response object of the servlet is a required input to the **`generateChart`** method of the **`ChartClient`** class, which streams the chart as a jpeg. Always set **`response.setContentType("image/jpeg")`** in custom servlets.

Browse to the charting servlet with a URL of this syntax for your application:

http://<hostname>:<port>/Sample/Chart?app=Charter&ddn=Payment&viewType=CHART&viewName=MyView&H=300&W=400&T=L&XT=Xvalue&YT=Yvalue&data=encodeddata ….

ChartData is a constructor that takes an encoded data string. A chart may have one or more such encoded data sets.

The width and height parameters are not present in the properties file, since the dimensions of the canvas are not actual properties of the chart.

TIP: Remember to register any custom servlets you create in the web.xml file for your Web application.

Customizing Charter.java

This topic presents the complete code for the default charting servlet that ships with eStatement, with comments on where and how to begin customization.

To customize your servlet, import the following packages, as well as any other packages you intend to use. Package **chart** is the Charting API. Class **App** is the base class for all eStatement application servlets, and class **LoginRequired** is the interface which signals that an account is required before access should be granted.

```
package com.edocs.app.chart;
import java.io.*;
import java.util.*;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.Frame;
import javax.servlet.*;
import javax.servlet.http.*;
import com.edocs.app.App;
import com.edocs.app.LoginRequired;
```

Your custom servlet can extend the base servlet class with a new instance of the **Charter** class, which contains the **getDataSets** method that requests the **ChartData** object. This class takes the response from the client browser and sends it to the application server, which in turn fulfills the servlet request and passes a chart URL back to the browser. *You may customize a chart servlet to obtain its data from another source, or in a different format.*

```
public class Charter extends App implements LoginRequired {
  static private boolean DEBUG = Constants.DEBUG;
  public void doPost(HttpServletRequest request,
           HttpServletResponse response) {
```

The charting engine returns images in JPEG format. You **must** set the content type of your servlet to JPEG for the output stream.

```
    try {
response.setContentType("image/jpeg");
ServletOutputStream out = response.getOutputStream();
```

Then request the parameters from the DDN (data source) and the CHART view name (ALF for presentation logic and properties from the chart properties file). Together, these parameters identify the file to retrieve from the versioning system and determine the chart properties. If you have multiple charts in a single statement, create a

```
String ddn = request.getParameter("ddn");
```

```
String viewName = request.getParameter("viewName");
```

Requesting the width and height parameters determines the canvas size of your chart. You set these parameters in the chart properties file.

```
int width = Integer.parseInt(request.getParameter("W"));
int height = Integer.parseInt(request.getParameter("H"));
```

You then call a Java Properties object that loads the specified chart properties, DDN, and view name with the **`getChartPropsStream`** method of the **`PublisherWrapper`** class. This links the data source and graphic elements of the chart.

```
Properties chartProps = new Properties();
chartProps.load(PublisherWrapper.getChartPropsStream(ddn,viewN
ame));
```

The **`ChartData`** class constructs the datasets for the chart from the encoded data passed in the chart URL. For details of this constructor, which has five signatures, see Class ChartData in Application Programming Interfaces (APIs) for Charting.

```
ChartData[] dataSets = getDataSets(request);
```

TIP: You can insert a custom error message here, for example to advise of too much data in the chart URL; see the Custom Error Message example.

Your servlet now creates a new **`ChartClient`** to hold the chart properties and the dimensions of the canvas, and generates the chart as an **`out`** object for the servlet response, catching any exceptions.

```
      ChartClient cl = new ChartClient(chartProps, width,
height);
      cl.generateChart(out, dataSets);
    } catch (Exception e) {
      e.printStackTrace();
      doForwardException(request, response, e);
    }
  }
  public void doGet(HttpServletRequest request,
HttpServletResponse response)
  {
    doPost(request, response);
  }
```

ChartData gets the encoded data string from the chart object in the ALF. **Any data properties specified in the chart properties file will override these ALF attributes.**

```
  static private ChartData[] getDataSets(HttpServletRequest
request)
```

```
   throws ChartException {
   String[] dataStrs = request.getParameterValues("data");
   int num = dataStrs.length;
   ChartData[] dataSets = new ChartData[num];
   for (int i = 0; i < num; i++) {
     dataSets[i] = new ChartData(dataStrs[i]);
     if (DEBUG) {
         System.out.println("DataSet(" + i + ") : " +
dataSets[i]);
     }
   }
   return dataSets;
  }
}
```

Example: Custom Error Message

```
chartProps.load(PublisherCommon.getChartPropsStream(ddn,viewNa
me));
try {
ChartData[] dataSets = getDataSets(request);
} catch (Exception be) {
String msg = e.getMessage(); if (msg.indexOf("data format
error")) { // then perhaps our URL is too long // get the
sorry too much charting data // open the gif tooMuch.gif //
write it to the output stream return; } else { throw be; }
}
ChartClient cl = new ChartClient(chartProps, width, height);
cl.generateChart(out, dataSets);
 } catch (Exception e) { e.printStackTrace();
doForwardException(request, response, e); } }
```

Troubleshooting Charts

Charting Checklist

- Are xwindow displays enabled on the Web-server machine? In an x-term window or a terminal that knows a notion of "DISPLAY" enter "echo $DISPLAY." If you get a non-null string, run xhost + on the machine indicated in the display variable.

- Does your machine have a physical display device or is it headless? A machine without a physical display requires an x-virtual frame buffer, such as xvfb.
- Does your Web/application server know where to send its x-displays? Make sure the DISPLAY environment variable is correctly set, either in the start script for your Web/application server or in the xterm for your start command.
- Is xhost running? Ensure you have not closed the xterm which issued "xhost + " unless you have "xhost +" as part of your server startup script.
- Have you published a CHART view in the eStatement Manager Publisher? A Chart view requires a chart properties file; make sure you have published the one associated with your eStatement Manager application.
- Does the BaseURL charting attribute in the ALF file match your Web application name? This attribute points the servlet to the correct CHART view. Make sure they match.
- Can you see charts in statements? If not, repeat the steps above. If you are still having trouble:
- How long is your chart URL? For large datasets, you may need to customize the charting servlet.
- Does your data contain special characters? The chart servlet may not handle these characters correctly. You need to customize the charting servlet.

Common Problems and Known Issues

AIX Does Not Display Charts

The X Windows client for AIX systems requires the X11 package, which comes with the O/S but is not installed by default. To check whether X11 is installed, run **`smit`** and check the installed packages option for AIX Windows X11 libraries, or look in the default directory **`/usr/lpp/X11`**.

Pie Chart Displays When another Chart Type Is Selected

The Composer creates only a Pie chart by default (though it displays other chart types on the dialog). To change the chart type and display an example, edit the chart properties file (not the ALF) and simulate the chart with the Simulator API.

Small Segments Collide In Pie Charts

Remove the % values that appear close to the pie, by setting chart properties

```
Pie.PercentLabelsOn=false
```

```
Pie.TextLabelsOn=false
```

UNDO Button in Composer Does Not Affect ALF

Inserting a chart modifies both the ALF and the HTML template, enabling the UNDO button. If you then click UNDO, only the HTML template changes are reversed—not those in the ALF. This can cause the Composer to fail when processing a section of a statement. Use caution in using the UNDO button to reverse changes.

Chart Quality Is Poor

The default chart property is **`Chart.Quality=.75`**. For highest quality charts, set chart property **`Chart.Quality=1`**.

ALF Axis Titles Overwritten By Properties File

Title values defined in the chart properties file (**`X.Axis.TitleString`** and **`Y.Axis.TitleString`**) take precedence over those defined in the ALF (**`XField`** and **`YField`**).

Changing Addtolegend In ALF Does Not Change Chart URL

Changing the value in the ALF for the property **`AddValuetoLegend`** does not change the URL "T"/"F" property. Instead, it actually passes the value as part of the legend. In order to change the URL "T"/"F" property through the ALF file, set the **`HidePieLegend`** property to 1. This works only if the Type property is set to 5 for **`pie`**.

Title Fonts Do Not Appear Bold

Setting font properties, for example **`Y.Axis.TitleFont.Style=Bold,`** on a headless server requires that fonts be available **`and`** requires a virtual display, or virtual frame buffer, such as **`xvfb`**. To display fonts and styles correctly, see "Setting Display Devices and xvfb" on page 219 and "Configuring a Headless Server for Charting" on page 222.

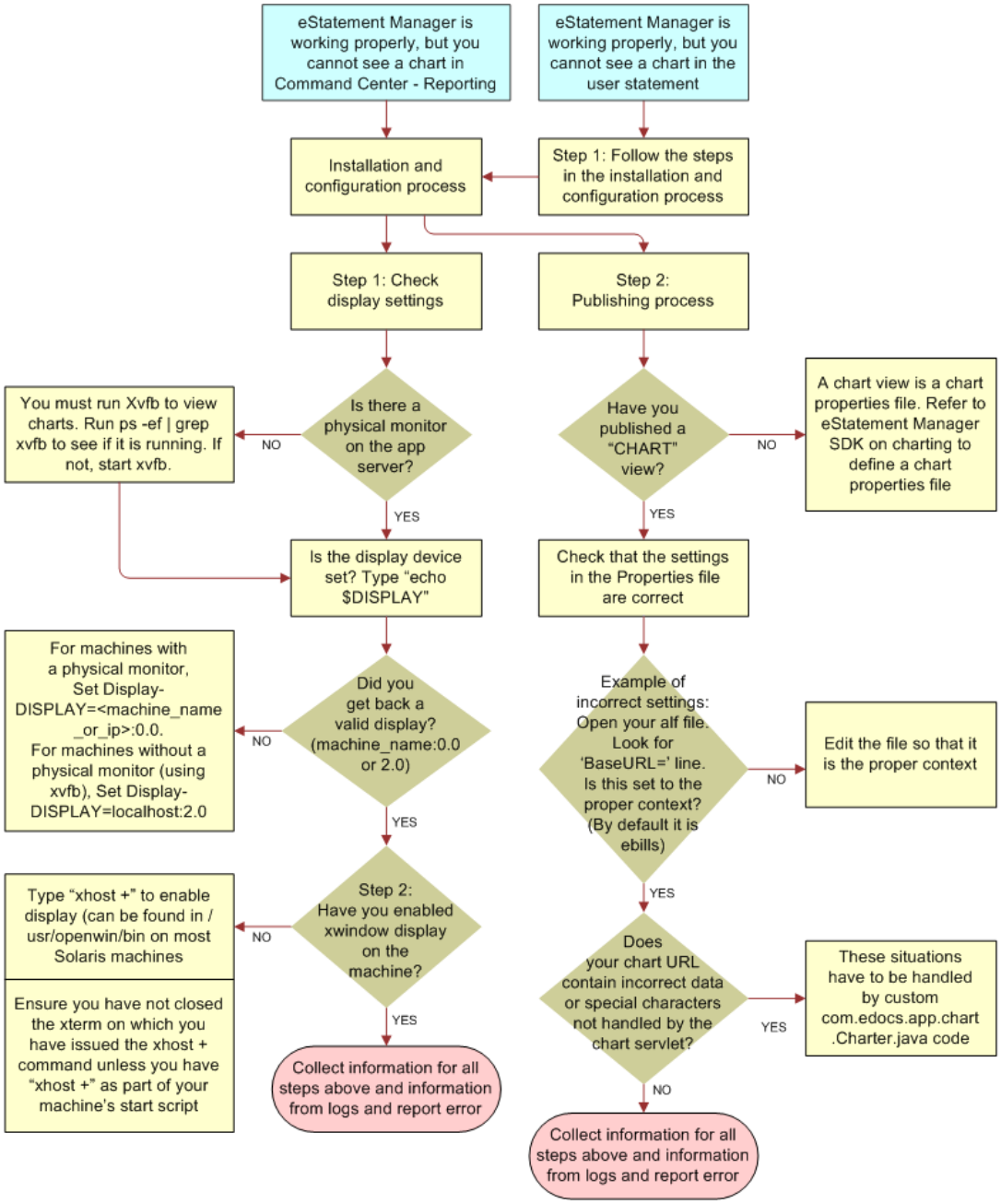
Bold Italic Does Not Display Correctly

Setting fonts to both bold and italic in the chart properties file may cause text to display as a bitmap. Charting implements fonts through **`java.awt.font,`** and the bold italic combination is handled as a bitmap of **`java.awt.Font.BOLD`** and **`java.awt.ITALIC`**.

Chart Servlet Suppresses Commas and Spaces

In a legend label, the charting servlet interprets "July 25, 2002" as "July252002." The workaround is to reformat the data at the JSP layer, but this does not work for Web views.

Troubleshooting Flowchart



Application Programming Interfaces (APIs) for Charting

Package com.edocs.app.chart Description

Contains classes, constructors, and methods to render and publish charts as JPEG graphics and to extend the charting servlet (an instance of the Charter class).

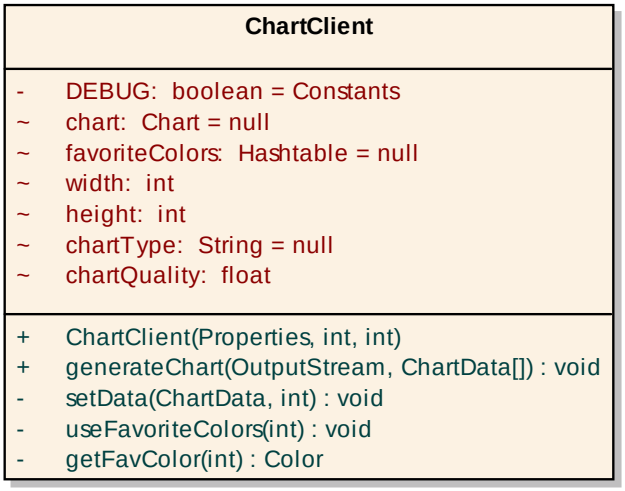
Class ChartClient

Contains a constructor and methods to draw a chart as a JPEG graphic.

Available Chart Types

Type	Description
Pie	Pie chart with one slice per data point.
Bar	Displays each data series **vertically** in a single color (sometimes called a column chart). To display horizontally, use **`HorizBar`** To display different colors for each bar use **`IndBar`** (horizontal) or **`IndColumn`** (vertical).
Line	Displays data values as lines on a graph, without value labels for each data point.
HiLoBar	Vertical bar chart with High and Low values indicated.
HorizBar	Displays each data series **horizontally** in a single color. To display vertically, use **`Bar`** (sometimes called a column chart). To display different colors for each bar, use **`IndBar`** (horizontal) or **`IndColumn`** (vertical).
HorizHiLoBar	Horizontal bar chart with High and Low values indicated.
IndBar	Horizontal bar chart that uses separate color for each bar.
IndColumn	Vertical bar chart that uses a separate color for each bar.
LabelLine	Displays data values as lines on a graph, with user-defined labels on the X-axis.
Polar	A chart that looks like a radar screen. Plots only one data value, but the scale is determined by all the data.
Regress	Subclass of Line chart that plots OLS regression for data values.
Speedo	A chart that looks like a gauge or speedometer, similar to Polar.
StackBar	Bar chart that stacks data values horizontally.
StackColumn	Bar chart that stacks data values vertically.
Stick	Chart that draws a vertical line to the Y-axis height of each data value.
StickBar	Chart that draws a vertical line to the Y-axis height of each data value.

Class Diagram



Constructors

```
chartClient(java.util.Properties properties, int canvasWidth, int
canvasHeight) throws ChartException
```

The ChartClient constructor takes a java properties object that specifies the default properties for the chart, and integers that specify the dimensions of the canvas in pixels.

Parameters

Parameter	Description
properties	Default property list. See **`java.util.Properties`**.
canvasWidth	Integer specifying the chart width in pixels.
canvasHeight	Integer specifying the chart height in pixels.

Methods

```
generateChart(java.io.OutputStream out, ChartData[] dataSets) throws
ChartException
```

`generateChart` is invoked to render the chart as a graphic.

Parameters

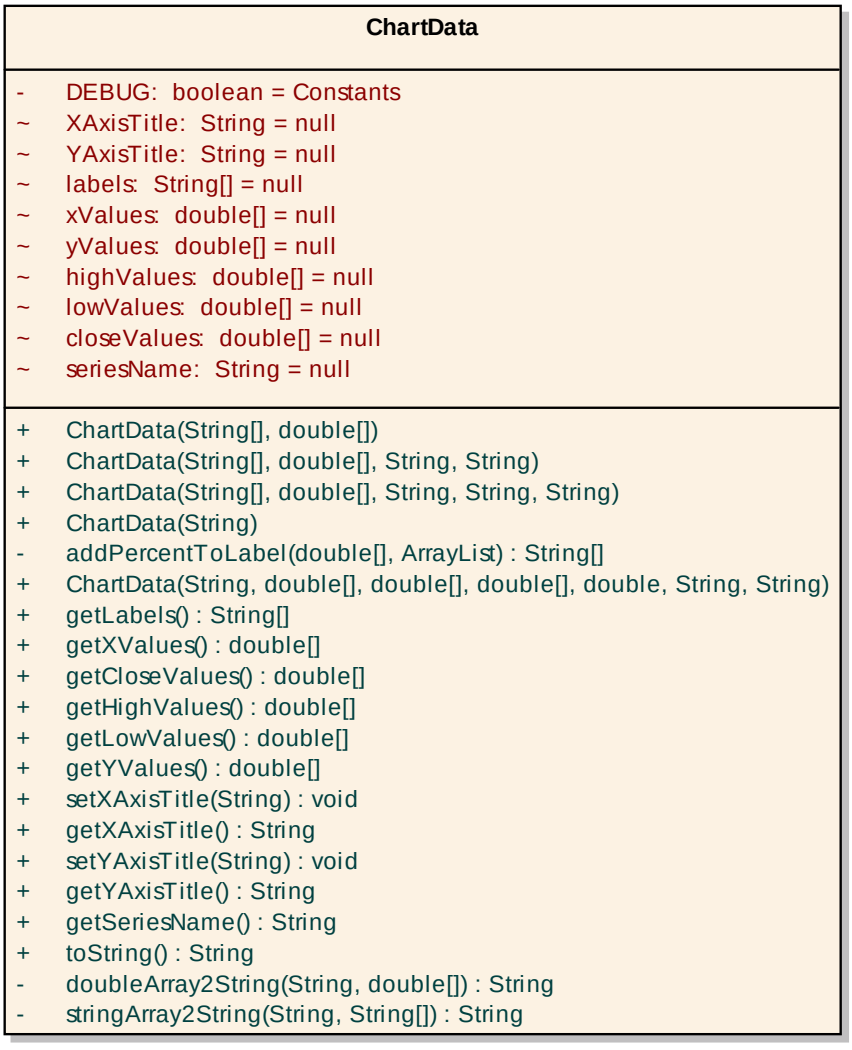
Parameter	Description
out	Defines an output stream, for example to generate the output for a servlet response object.

Parameter	Description
data	String specifying the data to be charted.

Class ChartData

Contains a constructor and methods to create the **`ChartData`** object required by the **`ChartClient`**. Also contains **`get`** and **`set`** methods for constructor parameters. For details, see the Javadoc.

Class Diagram



Constructors

Five signatures of **`ChartData`** construct the **`ChartData`** object to pass to the **`ChartClient`**.

```
ChartData(java.lang.String URLencodedDataStr)
```

Constructs an object containing an encoded URL in an HTTP Get request, or any raw data string.

```
ChartData(java.lang.String[] theLabels, double[] vals)
```

Constructs an object containing chart labels and name-value pairs of data values, as from a properties file.

```
ChartData(java.lang.String[] theLabels, double[] vals, java.lang.String
xTitle, java.lang.String yTitle)
```

Constructs an object containing chart labels, name-value pairs of data values, and axis titles, as from a properties file. Use when charting a single data series.

```
ChartData(java.lang.String[] theLabels, double[] vals,
java.lang.String xTitle, java.lang.String yTitle,
java.lang.String dataSeriesName)
```

Constructs an object containing chart labels, name-value pairs of data values, axis titles, and the name of each data series, as from a properties file. Use when charting multiple data series, as for stacked lines or bars.

```
ChartData(java.lang.String dataSeriesName, double[] xVals,
double[] hiVals, double[] loVals, double[] closeVals,
java.lang.String xTitle, java.lang.String yTitle)
```

Constructs an object containing parameters for high-low bar charts.

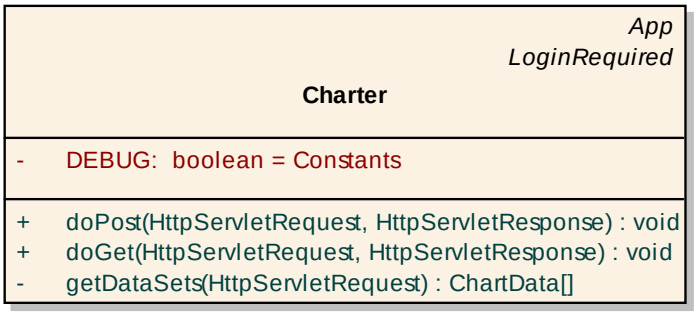
Parameters

Parameter	Description
closeVals[]	Double parameter for closing values in a high-low bar chart.
dataSeriesName	Display name for the data series being charted. Use when displaying multiple data series in a single chart.
hiVals	Double parameter for high values in a high-low bar chart.
loVals	Double parameter for low values in a high-low bar chart.
theLabels	String array containing values for data labels.
URLencodedDataStr	Chart data passed as a string. For syntax, see “Formatting Data Strings for Chart Data” on page 245.
vals	Array of doubles, the value to chart
xTitle	Display name for the X-Axis.
xVals	double[]
yTitle	Display name for the Y-Axis.

Class Charter

Servlet class for the charting servlet. Contains the **`getDataSets`** method that requests the **`ChartData`** object. Implements **`com.edocs.app.LoginRequired, javax.servlet.Servlet`**, and extends **`com.edocs.app.App.`** Also contains **`doPost`** and **`doGet`** methods that override those in class **`com.edocs.app.App.`** For details, see the Javadoc.

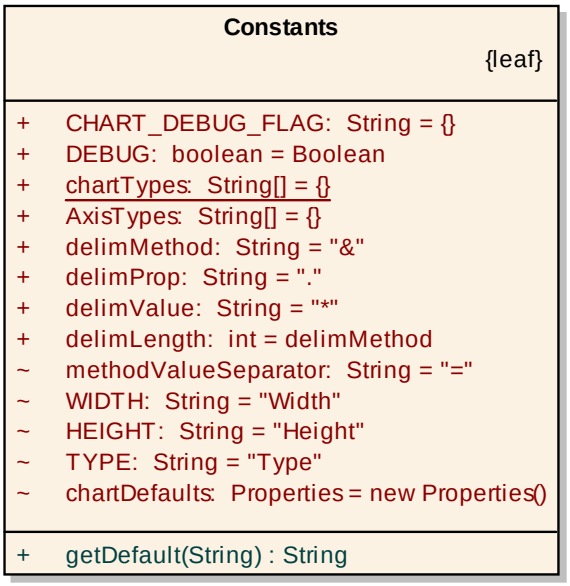
Class Diagram



Class Constants

Contains a constructor and default fields for defining custom chart parameters. For details, see the Javadoc.

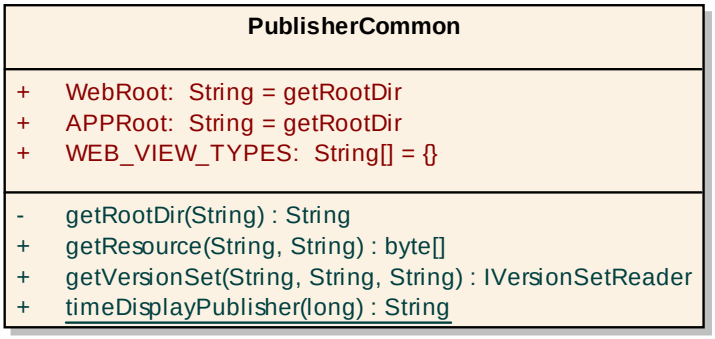
Class Diagram



Class PublisherCommon

Contains a constructor, fields, and methods to retrieve the root directory and Web views for the chart from the Publisher. For details, see the Javadoc.

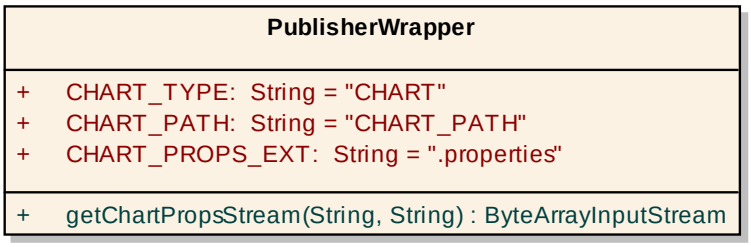
Class Diagram



Class PublisherWrapper

Contains a constructor, fields, and methods to retrieve chart properties as an input stream. For details, see the Javadoc.

Class Diagram



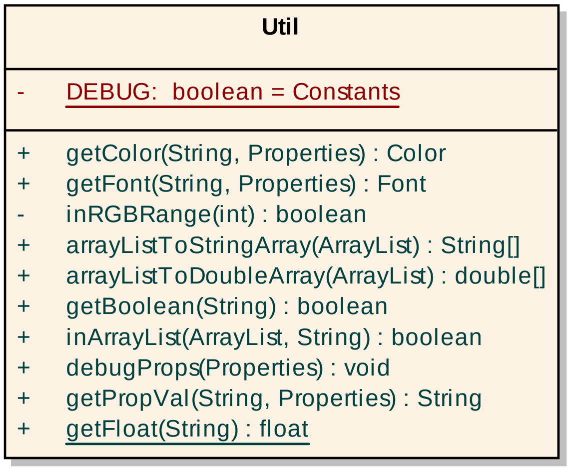
Class Simulator

Constructor and methods that behave like a shell command to create a JPEG image of the specified data and chart properties. Extends class **`java.awt.Frame.`** For usage, see "Previewing Charts with com.edocs.app.chart.Simulator" on page 245. Also see the Javadoc.

Class Util

Utility class to support the Charting API. For details, see the Javadoc.

Class Diagram



Default Properties and Attributes

ChartDefaults.properties

```
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
        # Kavachart3.2 properties influencing chart presentation # initially
        3.1, with addition of CullingLabel, its 3.2

        #

        # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
        # # # #


        # -------------------------------------------------------

        # Primary Properties

        # -------------------------------------------------------

        # Chart type: Pie



        # DateLine

        # FinCom

        # HiLoBar

        # HiLoClose
```

```
# HorizBar == to generate horizontal bar chart
#
# **** Please note, the property name, "Type" IS case
sensitive, and its
# **** value is case sensitive.
Type=Pie


# ------------------------------------------------------
# Legend related
# ------------------------------------------------------


 Legend.BackgroundVisible=true
## -- separation between rows of the legend
 Legend.IconGap=0.02
## -- Legend icon height 0 < k < 1.0, where 1.0 = full height
of canvas
 Legend.IconHeight=0.05
## -- Legend icon width 0 < k < 1.0, where 1.0 = full width of
canvas
 Legend.IconWidth=0.07
## -- R G B values
 Legend.LabelColor.Red=0
 Legend.LabelColor.Blue=0
 Legend.LabelColor.Green=0
## -- this needs to be broken up into its components
 Legend.LabelFont.Name=Times New Roman
 Legend.LabelFont.Style=plain
 Legend.LabelFont.Size=12


## -- lower x left corner 0 < y < 1.0, 1.0 = full width
 Legend.LlX=0.0
## -- lower y left corner 0 < y < 1.0, 1.0 = full height
 Legend.LlY=0.0
## -- upper x right corner 0 < y < 1.0, 1.0 = full width
 Legend.UrX=0.0
## --upper y right corner 0 < y < 1.0, 1.0 = full height
 Legend.UrY=0.0
```

```
## -- legend below chart
 Legend.VerticalLayout=false

## -- To change the Legend's Graphic component
## Legend.BackgroundGC.Gc.FillColor.Red=0
## Legend.BackgroundGC.Gc.FillColor.Blue=0
## Legend.BackgroundGC.Gc.FillColor.Green=0
## Legend.BackgroundGC.Gc.LineColor.Red=0
## Legend.BackgroundGC.Gc.LineColor.Blue=0
## Legend.BackgroundGC.Gc.LineColor.Green=0
## Legend.BackgroundGC.Gc.LineWidth=1
## Legend.BackgroundGC.Gc.MarkerColor.Red=0
## Legend.BackgroundGC.Gc.MarkerColor.Blue=0
## Legend.BackgroundGC.Gc.MarkerColor.Green=0
## Legend.BackgroundGC.Gc.Image -- unimplemented

# ------------------------------------------------------
# Class name = javachart.chart.Axis
# Specific to X axis
# ------------------------------------------------------
## to skip labels that collide
## meaningful if the user can guess the missed labels
 X.Axis.CullingLabel=false;
 X.Axis.AutoScale=true
## -- Determines end of an axis for a default axis.
## -- For log-scale its a power of ten.
 X.Axis.AxisEnd=6.0
## -- Determines start of axis.
## -- For AUTO_SCALE, selection of axis start is automatic,
## -- for log scale its a pwoer of ten.
 X.Axis.AxisStart=0.0
## -- placed bar within axis, set true for bar charts
 X.Axis.BarScaling=true
 X.Axis.GridVis=false
 X.Axis.LabelAngle=0
#### -- Axis label color
```

```
 X.Axis.LabelColor.Red=0
 X.Axis.LabelColor.Blue=0
 X.Axis.LabelColor.Green=0


 X.Axis.LabelFont.Name=Times New Roman
 X.Axis.LabelFont.Style=plain
 X.Axis.LabelFont.Size=12
#### -- Label Format determines how the label must be
redenered,
#### -- eg. first three letters of month name, basically the
defined method
#### -- is applied to the actual label
 X.Axis.LabelFormat=null
## For double quantities such as currency
 X.Axis.LabelPrecision=2
## Determines whether the label is visible
 X.Axis.LabelVis=true
## Determines whether the axis line is visible
 X.Axis.LineVis=true
## Determines whether the scale is log based
 X.Axis.LogScaling=false
 X.Axis.MajTickLength=5
 X.Axis.MajTickVis=true
 X.Axis.MinTickLength=2
 X.Axis.MinTickVis=false
 X.Axis.NumGrids=5
 X.Axis.NumLabels=5
 X.Axis.NumMajTicks=5
 X.Axis.NumMinTicks=10


## reduce LlX to left shift axis position on canvas
## increase for right shift, that is along the X direction
 X.Axis.Plotarea.LlX=0.2
#### -- reduce LlX to left shift on the canvas the axis start
#### -- increase LlX to right shift axis start on canvas
 X.Axis.Plotarea.LlY=0.2
```

```
## similar to LlX and LlY, but controls upper right corner
 X.Axis.Plotarea.UrX=0.8
 X.Axis.Plotarea.UrY=0.8
## true sets the title perpendicular to the axis, in the
middle
 X.Axis.TitleRotated=true
 X.Axis.TitleColor.Red=0
 X.Axis.TitleColor.Blue=0
 X.Axis.TitleColor.Green=0
 X.Axis.TitleFont.Family=Times New Roman
 X.Axis.TitleFont.Name=Times New Roman
 X.Axis.TitleFont.Style=plain
 X.Axis.TitleFont.Size=12

 X.Axis.TitleString=Remember to set X axis title!
#### -- ignore this, not planning to retrieve objects using
#### -- mouse click events
 X.Axis.UseDisplayList=false



## X.Axis.GridGc.FillColor.Red=0
## X.Axis.GridGc.FillColor.Blue=0
## X.Axis.GridGc.FillColor.Green=0
## X.Axis.GridGc.Image -- unimplemented
## X.Axis.GridGc.LineColor.Red=0
## X.Axis.GridGc.LineColor.Blue=0
## X.Axis.GridGc.LineColor.Green=0
## X.Axis.GridGc.LineWidth=1
## X.Axis.GridGc.MarkerColor.Red=0
## X.Axis.GridGc.MarkerColor.Blue=0
## X.Axis.GridGc.MarkerColor.Green=0

## valid values = Left, Right, Top, Bottom
## X.Axis.Side=Left

## X.Axis.TickGc.FillColor.Red=0
```

```
## X.Axis.TickGc.FillColor.Blue=0
## X.Axis.TickGc.FillColor.Green=0
## X.Axis.TickGc.Image -- unimplemented
## X.Axis.TickGc.LineColor.Red=0
## X.Axis.TickGc.LineColor.Blue=0
## X.Axis.TickGc.LineColor.Green=0
## X.Axis.TickGc.LineWidth=1
## X.Axis.TickGc.MarkerColor.Red=0
## X.Axis.TickGc.MarkerColor.Blue=0
## X.Axis.TickGc.MarkerColor.Green=0

## X.Axis.LineGc.FillColor.Red=0
## X.Axis.LineGc.FillColor.Blue=0
## X.Axis.LineGc.FillColor.Green=0
## X.Axis.LineGc.Image -- unimplemented
## X.Axis.LineGc.LineColor.Red=0
## X.Axis.LineGc.LineColor.Blue=0
## X.Axis.LineGc.LineColor.Green=0
## X.Axis.LineGc.LineWidth=1
## X.Axis.LineGc.MarkerColor.Red=0
## X.Axis.LineGc.MarkerColor.Blue=0
## X.Axis.LineGc.MarkerColor.Green=0


# ------------------------------------------------------
# Class name = javachart.chart.Axis
# Specific to Y axis
# ------------------------------------------------------

## to skip labels that collide
## meaningful if the user can guess the missed labels
 Y.Axis.CullingLabel=false;
 Y.Axis.AutoScale=true
## -- Determines end of an axis for a default axis.
## -- For log-scale its a power of ten.
 Y.Axis.AxisEnd=6.0
```

```
## -- Determines start of axis.
## -- For AUTO_SCALE, selection of axis start is automatic,
## -- for log scale its a pwoer of ten.
 Y.Axis.AxisStart=0.0
## -- placed bar within axis, set true for bar charts
 Y.Axis.BarScaling=true
 Y.Axis.GridVis=false
 Y.Axis.LabelAngle=0
#### -- Axis label color
 Y.Axis.LabelColor.Red=0
 Y.Axis.LabelColor.Blue=0
 Y.Axis.LabelColor.Green=0


 Y.Axis.LabelFont.Name=Times New Roman
 Y.Axis.LabelFont.Style=plain
 Y.Axis.LabelFont.Size=12
#### -- Label Format determines how the label must be
redenered,
#### -- eg. first three letters of month name, basically the
defined method
#### -- is applied to the actual label
 Y.Axis.LabelFormat=null
## For double quantities such as currency
 Y.Axis.LabelPrecision=2
## Determines whether the label is visible
 Y.Axis.LabelVis=true
## Determines whether the axis line is visible
 Y.Axis.LineVis=true
## Determines whether the scale is log based
 Y.Axis.LogScaling=false
 Y.Axis.MajTickLength=5
 Y.Axis.MajTickVis=true
 Y.Axis.MinTickLength=2
 Y.Axis.MinTickVis=false
 Y.Axis.NumGrids=5
 Y.Axis.NumLabels=5
```

```
 Y.Axis.NumMajTicks=5
 Y.Axis.NumMinTicks=10

## reduce LlX to left shift axis position on canvas
## increase for right shift, that is along the X direction
 Y.Axis.Plotarea.LlX=0.2
#### -- reduce LlX to left shift on the canvas the axis start
#### -- increase LlX to right shift axis start on canvas
 Y.Axis.Plotarea.LlY=0.2
## similar to LlX and LlY, but controls upper right corner
 Y.Axis.Plotarea.UrX=0.8
 Y.Axis.Plotarea.UrY=0.8

## true sets the title perpendicular to the axis, in the
middle
 X.Axis.TitleRotated=true
 Y.Axis.TitleColor.Red=0
 Y.Axis.TitleColor.Blue=0
 Y.Axis.TitleColor.Green=0
 Y.Axis.TitleFont.Family=Times New Roman
 Y.Axis.TitleFont.Name=Times New Roman
 Y.Axis.TitleFont.Style=plain
 Y.Axis.TitleFont.Size=12

 Y.Axis.TitleString=Remember to set X axis title!
#### -- ignore this, not planning to retrieve objects using
#### -- mouse click events
 Y.Axis.UseDisplayList=false


## Y.Axis.GridGc.FillColor.Red=0
## Y.Axis.GridGc.FillColor.Blue=0
## Y.Axis.GridGc.FillColor.Green=0
## Y.Axis.GridGc.Image -- unimplemented
## Y.Axis.GridGc.LineColor.Red=0
## Y.Axis.GridGc.LineColor.Blue=0
```

```
## Y.Axis.GridGc.LineColor.Green=0
## Y.Axis.GridGc.LineWidth=1
## Y.Axis.GridGc.MarkerColor.Red=0
## Y.Axis.GridGc.MarkerColor.Blue=0
## Y.Axis.GridGc.MarkerColor.Green=0

## valid values = Left, Right, Top, Bottom
## Y.Axis.Side=Left

## Y.Axis.TickGc.FillColor.Red=0
## Y.Axis.TickGc.FillColor.Blue=0
## Y.Axis.TickGc.FillColor.Green=0
## Y.Axis.TickGc.Image -- unimplemented
## Y.Axis.TickGc.LineColor.Red=0
## Y.Axis.TickGc.LineColor.Blue=0
## Y.Axis.TickGc.LineColor.Green=0
## Y.Axis.TickGc.LineWidth=1
## Y.Axis.TickGc.MarkerColor.Red=0
## Y.Axis.TickGc.MarkerColor.Blue=0
## Y.Axis.TickGc.MarkerColor.Green=0

## Y.Axis.LineGc.FillColor.Red=0
## Y.Axis.LineGc.FillColor.Blue=0
## Y.Axis.LineGc.FillColor.Green=0
## Y.Axis.LineGc.Image -- unimplemented
## Y.Axis.LineGc.LineColor.Red=0
## Y.Axis.LineGc.LineColor.Blue=0
## Y.Axis.LineGc.LineColor.Green=0
## Y.Axis.LineGc.LineWidth=1
## Y.Axis.LineGc.MarkerColor.Red=0
## Y.Axis.LineGc.MarkerColor.Blue=0
## Y.Axis.LineGc.MarkerColor.Green=0
```

```
# ----------------------------------------------------
#  Optional Properties:
# ----------------------------------------------------

## Set the Bar baseline.
##Bar.Baseline=0.0
## sets the cluster width
##Bar.ClusterWidth=0.8
## Set to true to clip bars at Plotarea boundaries.
## Bar.DoClip=false

# ----------------------------------------------------
#  Optional Properties: Line
# ----------------------------------------------------
## true clips lines at the plot area boundary
## Line.Clip=false
# ----------------------------------------------------
#  Optional Properties: Pie
# ----------------------------------------------------

 Pie.LabelColor.Red=0
 Pie.LabelColor.Blue=0
 Pie.LabelColor.Green=0
 Pie.LabelFont.Name=Times New Roman
 Pie.LabelFont.Style=plain
 Pie.LabelFont.Size=12

 Pie.LabelPosition=2
 Pie.PercentLabelsOn=true
 Pie.StartDegrees=0
 Pie.TextLabelsOn=false
 Pie.ValueLabelsOn=false
#### -- if you want a circular pie, scale Height and Width to
#### -- be equal in terms of pixels after you've taken into
#### -- consideration true canvas height and width
 Pie.Height=0.6
```

```
 Pie.Width=0.6
#### -- this is the center of the pie, do you want it in the
center
#### -- of the canvas or to one side. Elect side if the labels
to be
#### -- rendered on the legend are long.
#### -- choose up or down, if you have more vertical real
estate on the
#### -- html page
 Pie.XLoc=0.5
 Pie.YLoc=0.5


# ------------------------------------------------------
#  General chart properties
# ------------------------------------------------------
 Chart.LegendVisible=false
 Chart.Name=MyChart
## x and y offset determine the three dimensional effect
 Chart.ThreeD=false
 Chart.XOffset=0
 Chart.YOffset=0
 Chart.YAxisVisible=true
 Chart.XAxisVisible=true
## -- The chart quality has a default value of 1. It can take
values
## -- from 0 to 1, where 0 is the poorest quality, while 1 is
the best
## -- 0.75 is a good balance between image size and quality
 Chart.Quality=1.0


# ------------------------------------------------------
#  Plotarea Graphic Component properties
# ------------------------------------------------------


## Plotarea.Gc.FillColor.Red=0
## Plotarea.Gc.FillColor.Blue=0
## Plotarea.Gc.FillColor.Green=0
```

```
## Plotarea.Gc.Image -- unimplemented
## Plotarea.Gc.LineColor.Red=0
## Plotarea.Gc.LineColor.Blue=0
## Plotarea.Gc.LineColor.Green=0
## Plotarea.Gc.LineWidth=1
## Plotarea.Gc.MarkerColor.Red=0
## Plotarea.Gc.MarkerColor.Blue=0
## Plotarea.Gc.MarkerColor.Green=0

# -----------------------------------------------------
#  Background properties
# -----------------------------------------------------

## Background.Gc.FillColor.Red=0
## Background.Gc.FillColor.Blue=0
## Background.Gc.FillColor.Green=0
## Background.Gc.Image -- unimplemented
## Background.Gc.LineColor.Red=0
## Background.Gc.LineColor.Blue=0
## Background.Gc.LineColor.Green=0
## Background.Gc.LineWidth=1
## Background.Gc.MarkerColor.Red=0
## Background.Gc.MarkerColor.Blue=0
## Background.Gc.MarkerColor.Green=0

## Background.SubTitleColor.Red=0
## Background.SubTitleColor.Blue=0
## Background.SubTitleColor.Green=0
## Background.SubTitleFont.name=Times New Roman
## Background.SubTitleFont.style=plain
## Background.SubTitleFont.size=12
## Background.SubTitleString=null

## Background.TitleColor.Red=0
## Background.TitleColor.Blue=0
## Background.TitleColor.Green=0
```

```
## Background.TitleFont.Name=Times New Roman
## Background.TitleFont.Style=plain
## Background.TitleFont.Size=12
## Background.TitleString=null

## Favorite.1.Color.Red=0
## Favorite.1.Color.Blue=0
## Favorite.1.Color.Green=0

## Favorite.2.Color.Red=0
## Favorite.2.Color.Blue=0
## Favorite.2.Color.Green=0
## Favorite.3.Color.Red=0
## Favorite.3.Color.Blue=0
## Favorite.3.Color.Green=0
## Favorite.4.Color.Red=0
## Favorite.4.Color.Blue=0
## Favorite.4.Color.Green=0

## Favorite.5.Color.Red=0
## Favorite.5.Color.Blue=0
## Favorite.5.Color.Green=0
```

NW_LocSummary.ALF

```
<?xml version="1.0"?>
<!DOCTYPE ALF [
   <!-- An element of type ALF must contain following
subelements -->
   <!ELEMENT ALF (VERSION, DATA_GROUP, DDF, SWITCH, HOME,
TEMPLATES, CONTENTS, CONDITIONS, PROFILES, BUSINESSCONDITIONS,
RECORDS, PAGE_ELEMENTS, composition-specs)>
   <!-- An element of type VERSION contains a mixture of
character data -->
   <!ELEMENT VERSION (#PCDATA)>
   <!-- An element of type DATA_GROUP contains a mixture of
character data -->
```

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

   <!-- An element of type DDF contains a mixture of character
data -->

   <!ELEMENT DDF (#PCDATA)>

   <!-- An element of type SWITCH consist of Optional
Statement element -->

   <!ELEMENT SWITCH (Statement?)>

   <!-- An element of type Statement can contain three
subelements. Firstly it must

have Condition element and Action1 element. This is Optionaly
followed by the Action2 element -->

   <!ELEMENT Statement (Condition, Action1, Action2?)>

   <!ATTLIST Statement

   Profile (Y | N) #IMPLIED

>

   <!-- An element of type Condition contains a mixture of
character data -->

   <!ELEMENT Condition (#PCDATA)>

   <!-- An element of type Action1 contains a mixture of
character data -->

   <!ELEMENT Action1 (#PCDATA)>

   <!-- An element of type Action2 contains a mixture of
character data or Statement elements in any order-->

   <!ELEMENT Action2 (#PCDATA | Statement)*>

   <!ELEMENT HOME (DefaultTemplate, Statement?)>

   <!ELEMENT DefaultTemplate (#PCDATA)>

   <!ELEMENT TEMPLATES (Template)+>

   <!ELEMENT Template (SECTIONS, CHARTS, GROUPS,
GroupTemplate*)>

   <!ATTLIST Template

   Name CDATA #REQUIRED

>

   <!ELEMENT FormatSpecification (#PCDATA)>

   <!ELEMENT Action (#PCDATA)>

   <!ELEMENT SECTIONS (Section*)>

   <!ELEMENT Section (Statement+ | (FormatSpecification,
Action)+)>

   <!ATTLIST Section

   Name CDATA #REQUIRED
```

```
            Promotional CDATA #IMPLIED
>
   <!ELEMENT CHARTS (Chart*)>
   <!ELEMENT Chart EMPTY>
   <!ATTLIST Chart
   Name CDATA #REQUIRED
            RecordName CDATA #REQUIRED
            TopTitle CDATA #REQUIRED
            BottomTitle CDATA #REQUIRED
            LeftTitle CDATA #REQUIRED
            RightTitle CDATA #REQUIRED
            XField CDATA #REQUIRED
            YField CDATA #REQUIRED
            Key CDATA #REQUIRED
            StackedStyle CDATA #REQUIRED
            ColorScheme CDATA #REQUIRED
            GridLines CDATA #REQUIRED
            Full3D CDATA #REQUIRED
            AngleX CDATA #REQUIRED
            AngleY CDATA #REQUIRED
            Atribute CDATA #REQUIRED
            MarkerVolume CDATA #REQUIRED
            Shadow CDATA #REQUIRED
            MultiShape CDATA #REQUIRED
            Dimension_3D CDATA #REQUIRED
            View3DDepth CDATA #REQUIRED
            Type CDATA #REQUIRED
            CGITimeSpan CDATA #REQUIRED
            BackgroundColor CDATA #REQUIRED
            ForgroundColor CDATA #REQUIRED
            Height CDATA #REQUIRED
            Width CDATA #REQUIRED
            LegendShow CDATA #REQUIRED
            LegendToolSize CDATA #REQUIRED
            LegendToolStyle CDATA #REQUIRED
            HidePieLegend CDATA #REQUIRED
```

```
            SeriesColor CDATA #REQUIRED
            LeftGap CDATA #REQUIRED
            RightGap CDATA #REQUIRED
            ImgQuality CDATA #REQUIRED
            ImgSmooth CDATA #REQUIRED
            AddValueToLegend CDATA #REQUIRED
            BaseURL CDATA #REQUIRED
            UNIXChart CDATA #REQUIRED
>
   <!ELEMENT GROUPS (Group*)>
   <!ELEMENT Group (Statement)>
   <!ATTLIST Group
   Name CDATA #REQUIRED
>
   <!ELEMENT GroupTemplate (SECTIONS, CHARTS, GROUPS,
GroupTemplate*)>
   <!ATTLIST GroupTemplate
   Name CDATA #REQUIRED
>
   <!ELEMENT CONTENTS (Content)+>
   <!ELEMENT Content (#PCDATA)>
   <!ATTLIST Content
   Name CDATA #REQUIRED
         Type (MainTemplate | RGTemplate1 | RGTemplate2 |
SectionTemplate | ALF | Image | Text | Active | GlobalAction)
"MainTemplate"
         Parent CDATA #REQUIRED
         ParentTemplate CDATA #REQUIRED
>
   <!ELEMENT CONDITIONS (SavedCondition)*>
   <!ELEMENT SavedCondition (#PCDATA)>
   <!ATTLIST SavedCondition
   Name CDATA #REQUIRED
         SavedConditionProfile (Y | N) #REQUIRED
>
   <!ELEMENT PROFILES (Profile)*>
   <!ELEMENT Profile (#PCDATA)>
```

```
   <!ATTLIST Profile
   Name CDATA #REQUIRED
>
   <!ELEMENT BUSINESSCONDITIONS (BusinessCondition)*>
   <!ELEMENT BusinessCondition (#PCDATA)>
   <!ATTLIST BusinessCondition
   Name CDATA #REQUIRED
>
   <!ELEMENT RECORDS (Record)*>
   <!ELEMENT Record (#PCDATA)>
   <!ATTLIST Record
   Name CDATA #REQUIRED
         ApplyAll (Y | N) #REQUIRED
         PresentationTable CDATA #IMPLIED
>
   <!ELEMENT PAGE_ELEMENTS (PageElement)*>
   <!ELEMENT PageElement (#PCDATA)>
   <!ATTLIST PageElement
   Name CDATA #REQUIRED
         Type (Table | Group) #REQUIRED
         Enable (yes | no) #REQUIRED
         Mode (line | occurrence) #REQUIRED
         SetSize CDATA #REQUIRED
         Occurrences CDATA #REQUIRED
>
   <!ELEMENT composition-specs ((sort-spec | filter-spec |
select-spec | arithmetic-spec)*, combine-spec)>
   <!ELEMENT sort-spec (sorted-element, sort-by-element)+>
   <!ATTLIST sort-spec
   name CDATA #REQUIRED
         mode (Table | Group) #REQUIRED
>
   <!ELEMENT sorted-element (#PCDATA)>
   <!ELEMENT sort-by-element (#PCDATA)>
   <!ATTLIST sort-by-element
   data-type CDATA #REQUIRED
```

```
            format-string CDATA #REQUIRED
            direction (a | d) #REQUIRED
>
   <!ELEMENT filter-spec (filtered-element, filtered-by-
element, filter-expression)+>
   <!ATTLIST filter-spec
   name CDATA #REQUIRED
            mode (Table | Group) #REQUIRED
>
   <!ELEMENT filtered-element (#PCDATA)>
   <!ELEMENT filtered-by-element (#PCDATA)>
   <!ELEMENT filter-expression (#PCDATA)>
   <!ELEMENT select-spec (selected-element, selected-by-
element)+>
   <!ATTLIST select-spec
   name CDATA #REQUIRED
            mode (Table | Group) #REQUIRED
>
   <!ELEMENT selected-element (#PCDATA)>
   <!ELEMENT selected-by-element (#PCDATA)>
   <!ATTLIST selected-by-element
   data-type CDATA #REQUIRED
            format-string CDATA #REQUIRED
            direction (Top | Bottom) #REQUIRED
            default-count CDATA #REQUIRED
>
   <!ELEMENT combine-spec (combine-element)*>
   <!ELEMENT combine-element (#PCDATA)>
   <!ELEMENT arithmetic-spec (arithmetic-element, arithmetic-
by-element)>
   <!ATTLIST arithmetic-spec
   name CDATA #REQUIRED
>
   <!ELEMENT arithmetic-element (#PCDATA)>
   <!ELEMENT arithmetic-by-element (#PCDATA)>
   <!ATTLIST arithmetic-by-element
   data-type CDATA #REQUIRED
```

```
        format-string CDATA #REQUIRED
        mode (total | count) #REQUIRED
        output-format-string CDATA #REQUIRED
>
]>
<ALF>
   <VERSION>3.0</VERSION>
   <DATA_GROUP>Local_Summary</DATA_GROUP>
   <DDF>C:/eStatement/samples/NatlWireless/NW_LocSummary.ddf</
DDF>
   <SWITCH/>
   <HOME>
        <DefaultTemplate>Default_Template</DefaultTemplate>
   </HOME>
   <TEMPLATES>
        <Template Name="Default_Template">
             <SECTIONS/>
             <CHARTS>
                  <Chart Name="LocalLineSummary_0"
RecordName="LocalLineSummary" TopTitle="Top Lable"
BottomTitle="Bottom Lable" LeftTitle="" RightTitle=""
XField="LocalLinePhNo" YField="LocalLineAmt" Key="0"
StackedStyle="0" ColorScheme="0" GridLines="0" Full3D="0"
AngleX="0" AngleY="0" Atribute="0" MarkerVolume="0" Shadow="0"
MultiShape="0" Dimension_3D="0" View3DDepth="0" Type="1"
CGITimeSpan="" BackgroundColor="White" ForgroundColor="Black"
Height="300" Width="400" LegendShow="1" LegendToolSize="100"
LegendToolStyle="167116800" HidePieLegend="0" SeriesColor=""
LeftGap="40" RightGap="40" ImgQuality="75" ImgSmooth="0"
AddValueToLegend="0" BaseURL="/Chart" UNIXChart="Pie"/>
                  <Chart Name="LocalChargeSummary_1"
RecordName="LocalChargeSummary" TopTitle="Top Lable"
BottomTitle="Bottom Lable" LeftTitle="" RightTitle=""
XField="LocalChargeAmt" YField="LocalChargeDesc" Key="1"
StackedStyle="0" ColorScheme="0" GridLines="0" Full3D="0"
AngleX="0" AngleY="0" Atribute="0" MarkerVolume="0" Shadow="0"
MultiShape="0" Dimension_3D="0" View3DDepth="0" Type="1"
CGITimeSpan="" BackgroundColor="White" ForgroundColor="Black"
Height="300" Width="400" LegendShow="1" LegendToolSize="100"
LegendToolStyle="167116800" HidePieLegend="0" SeriesColor=""
LeftGap="40" RightGap="40" ImgQuality="75" ImgSmooth="0"
AddValueToLegend="0" BaseURL="/Sample" UNIXChart="Pie"/>
             </CHARTS>
             <GROUPS/>
```

```
        </Template>
   </TEMPLATES>
   <CONTENTS>
        <Content Name="Default_Template" Type="MainTemplate"
Parent=""
ParentTemplate=""><![CDATA[C:/eStatement/samples/NatlWireless/
NW_LocSummary.htm]]></Content>
   </CONTENTS>
   <CONDITIONS/>
   <PROFILES/>
   <BUSINESSCONDITIONS/>
   <RECORDS>
        <Record Name="CustAddress"
ApplyAll="Y"><![CDATA[<table border=1 width="100%">
 <TBODY>
 <tr>
  <td height=% width=%><font color=#5c00d9 face=Arial

size=2><STRONG>[E]CustAddressLine[/E]</STRONG></font></td></tr
></TBODY></table>]]></Record>
        <Record Name="LocalChargeSummary"
ApplyAll="Y"><![CDATA[<TABLE border=1 width="100%">
 <TBODY>
 <TR>
  <TD height=% width=%><FONT color=#000000 face=Arial
   size=2>[E]LocalChargeDesc[/E]</FONT></TD>
  <TD align=right height=% width=%><FONT color=#000000
face=Arial

size=2>[E]LocalChargeAmt[/E]</FONT></TD></TR></TBODY></TABLE>]
]></Record>
        <Record Name="LocalLineSummary"
ApplyAll="Y"><![CDATA[<table border=1 width="100%">
 <TBODY>
 <tr>
  <td height=% width=%><font color=#000000 face=Arial
   size=2>[E]LocalLinePhNo[/E]</font></td>
  <td align=right height=% width=%><font color=#000000
face=Arial
```

```
size=2>[E]LocalLineAmt[/E]</font></td></tr></TBODY></table>]]>
</Record>
   </RECORDS>
   <PAGE_ELEMENTS/>
   <composition-specs>
         <combine-spec/>
   </composition-specs>
</ALF>
```