



Developer's Guide for Siebel Billing Manager

Version 5.1

Date Published: 02/28/2006

Copyright © 2005, 2006, Oracle and/or its affiliates

All rights reserved

Printed in the United States of America

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

PRODUCT MODULES AND OPTIONS. This guide contains descriptions of modules that are optional and for which you may not have purchased a license. Siebel's Sample Database also includes data related to these optional modules. As a result, your software implementation may differ from descriptions in this guide. To find out more about the modules your organization has purchased, see your corporate purchasing agent or your Siebel sales representative.

U.S. GOVERNMENT RESTRICTED RIGHTS. Programs, Ancillary Programs and Documentation, delivered subject to the Department of Defense Federal Acquisition Regulation Supplement, are "commercial computer software" as set forth in DFARS 227.7202, Commercial Computer Software and Commercial Computer Software Documentation, and as such, any use, duplication and disclosure of the Programs, Ancillary Programs and Documentation shall be subject to the restrictions contained in the applicable Siebel license agreement. All other use, duplication and disclosure of the Programs, Ancillary Programs and Documentation by the U.S. Government shall be subject to the applicable Siebel license agreement and the restrictions contained in subsection (c) of FAR 52.227-19, Commercial Computer Software - Restricted Rights (June 1987), or FAR 52.227-14, Rights in Data—General, including Alternate III (June 1987), as applicable. Contractor/licensor is Siebel Systems, Inc., 2207 Bridgepointe Parkway, San Mateo, CA 94404.

Proprietary Information

Siebel Systems, Inc. considers information included in this documentation and in Siebel Online Help to be Confidential Information. Your access to and use of this Confidential Information are subject to the terms and conditions of: (1) the applicable Siebel Systems software license agreement, which has been executed and with which you agree to comply; and (2) the proprietary and restricted rights notices included in this documentation.

Contents

1 Preface

About Siebel Billing Manager 13

About This Guide 13

Related Documentation 14

2 Functionality Overview

Billing Manager Features 15

Billing Manager Overview and Architecture 17

Overview 17

Functional Areas 17

Architecture Components and Services 18

Billing Manager Components 20

eStatement Subsystem 20

Payment Subsystem 21

Customer Account Management 22

View Processing 23

Security Access Framework 24

Billing Manager Install Directories 25

Config Directory 26

About Billing Manager Applications 26

Application Directories 27

Application Profiles 28

Application Packaging 28

Views, application logic files, and HTML templates 29

Views 29

Application Logic Files 30

View Functionality 30

View Development 30

3 Presentation Framework

Introducing Billing Manager Application Basics 37

Billing Manager Application Fundamentals 37

Architecture & Structure 38

Standard Page Flows 39

Struts and Tiles and the Siebel View Architecture 39

Billing Manager Application Architecture 40

Billing Manager Struts and Tiles 40

Mapping DDNs 42

Struts Extensions 42

Billing Manager Tiles 42

View Tiles 46

Creating Custom Tabs 47

Development Utilities 51

Billing Manager Tags 51

Working with Properties 53

Working with Logs 54

Developing Custom Actions 55

Customer Account Manager 55

Look and Feel 57

Message Rendering and Localization 58

Style Sheets 58

Images 58

4 Statement Presentation and Content Access

View Architecture Basics 61

View Architecture 61

Display Tiles Run-time Architecture	62
Downloadable View Support	62
Writing and Configuring Post Processors	63
Post Processor lifestyle	64
How to Write a Post Processor	64
Pattern Processing	67
Live Data Extraction	69
About the LDE Manager	69
About LDE Data Streams	70
Developing Live Data Extractors	71

5 Security: Enrollment, Authentication, Authorization

About Role-Based Security	77
What is Security and Authorization Framework (SAF)?	78
Authenticate	78
Security Profile	79
User Profiles	79
Security Management	79
Authorization	79
Restricting URL Access	80
Controlling Access to Resources	81
Using the SAF Sub-system	81
SAF Components	81
SAF Configuration	82
Role Mapping	82
Policy Store	82
Rights Engine	83
Implied Policies	84
Permission Classes	84

Custom Impliers	84
JSP Permission Example	85
Using Programmatic Security	86
Integrating Security Providers	88
BSL Security SPI's	88
BSL Security Concepts	89
Common Implementation Use Cases	89
Authentication	90
Security Manager Profiles	90
User Profiles	91
Configuring BSL Providers	91

6 Customer Service Representative (CSR) Access/Capabilities

Overview	93
CSR Access (Impersonate User)	93
CSR Application	96

7 Payment Cartridges

8 Downloading Views and Converting Advanced AFP to PDF

Downloading CSV, XML, and PDF Views	99
Common Download View Setup	99
Comma Separated Value (CSV) Downloadable views	100
eXtensible Markup Language (XML) views	101
eXtensible Stylesheet Language (XSLT) views	102
Portable Document Format (PDF_FO) views	103
Mapping a DDF to XML	104
Further Reading About XML, XSL, and XSLT	106

9 BSL Hierarchy Connector

Overview 107

How to Create and Manage Hierarchies via XML 108

Example Hierarchy XML File 111

Synchronizing Hierarchies with eStatement Indexer Data 113

Hierarchy APIs 118

Creating Hierarchies Programmatically 118

Adding Entities to a Hierarchy 118

Finding Hierarchies, Nodes, and Folders 119

Updating Nodes 120

Giving User Access to Nodes 120

How to Implement Hierarchy-Based Access Control (HBAC) 121

How to Search for and Find Objects Within Hierarchies 121

Tuning Performance for Large Hierarchies 122

Retrieval Threshold 122

Display Tree Threshold 122

Display List Threshold 123

Dropdown Threshold 123

Large Enterprise User Threshold 123

10 Profile Management

Overview 125

Managing the User Profile 125

Managing the Security Profile 125

Managing the Company and Company User Profiles 125

11 Notifications

Email Overview 127

General Email Subsystem 127

Email Grouping via Composer 128

- Configuring Email Messaging 128
- Email Notifications 129
 - Notification Email Variables 131
- Modifying Email Addresses Programmatically 131
 - Add Email Address Example Action 132

12 Address Books

- Address Book Overview 133
- Personal Address Books 133
- Corporate Address Books 141
- AddressBook Component 142
 - Displaying the Address Book 143

13 Custom Jobs

- About Jobs and the Shell Command Task 147
- Defining a New Job Type 148
 - Create the Job Type Script 148
 - Configuring Your New Job Type 156
 - Another Example of Defining a New Job Type 157

14 Charting

- Introduction and Components 159
 - Charting Use Case Overview Diagram 159
 - Components of Charting 160
- Configuring Charting for Your Server 160
 - About Servers and Charting 160
 - About Fonts 161
 - Configuration Activity Diagram 161
 - Setting Display Devices and xvfb 162
 - Setting Display Permissions and xhost 162
 - Setting Display Awareness 163

Configuring a Headless Server for Charting	164
Composing Charts in Statements	165
About Charting in the Composer	165
Inserting a Chart Tag in the Composer	165
Naming Conventions for Charts	167
About Chart Tags in the ALF	167
About The Chart Properties File	168
About Simulating Charts	168
Customizing Chart Properties	168
About Chart Attributes in the ALF	169
Customizing a Chart in the ALF	170
Customizing the Chart Properties File	171
Chart Type	171
Pie	172
Bar	173
Line	174
Horizontal Bars (HorizBar)	174
Individually Colored Bars (IndBar)	175
Individually Colored Columns (IndColumn)	175
(LabelLine)	175
Polar Chart (Polar)	176
Regression (Regress)	176
Stacked Bars (StackBar)	176
Stack Column Chart (StackColumn)	177
Stick Chart (Stick)	177
StickBar Chart (StickBar)	177
General Properties	178
Background Properties	178

Plot Area Properties	179
Axis Properties	180
Legend Properties	183
Favorite Colors Properties	184
Default Chart Properties	184
Customizing Default Properties	188
Previewing Charts with <code>com.edocs.app.chart.Simulator</code>	188
Publishing Charts	190
Before Publishing Charts	190
Publishing a Chart View	191
Viewing Charts in Statements	192
Designing Custom Charts with the Charting Servlet	192
Customizing <code>Charter.java</code>	192
Troubleshooting Charts	195
Charting Checklist	195
Common Problems and Known Issues	196
Application Programming Interfaces (APIs) for Charting	197
Package <code>com.edocs.app.chart</code> Description	197
Class <code>ChartClient</code>	197
Class <code>ChartData</code>	199
Class <code>Charter</code>	201
Class Constants	202
Class <code>PublisherCommon</code>	202
Class <code>PublisherWrapper</code>	203
Class <code>Simulator</code>	203
Class <code>Util</code>	203
Default Properties and Attributes	204

15 Internationalization (i18n)

About Internationalization in Billing Manager 205

Architecture 205

Message Resources 205

Tile Definitions 205

JSPs 207

Validation XML 207

View 208

Support for New Languages 208

16 Multiple DDN

Overview 211

About Multiple DDN Support 211

Split Stream, Single Account 212

Account Level Stream Mapping 212

Customer Level Stream Mapping 212

High Level Architecture 213

High-Level Description 213

CAM DDN Background 213

Configuration Extensions 214

App-Config DDN Naming Conventions 214

Tiles Definitions 215

Examples 216

Supporting Summary Information for Each Customer Account 217

Mapping Abstract to Realized DDN 218

Index

1 Preface

About Siebel Billing Manager

The objective of the Billing Manager application is to provide online account management and customer self-service for communications provider's business customers.

The main features of Billing Manager are:

- **Administration** – Online management of company profiles and company users, hierarchy management, and managing business structures and corporate address books.
- **Statements** – Customers can view summary and detailed statements on accounts, devices, and usage.
- **Analytics** – Standard account, device, and usage reporting functionality. Advanced reporting and report creation is available in a separate application (TAM).
- **Service Management** – A separate application (Siebel Communications Self-Service Manager) for service management and analysis is available for integration into Billing Manager.
- **Payments** – Companies can set up payment accounts for recurring and one-time payments, which historical payment activity for a configurable period of time.
- **User Profile Management** – Users are able to manage personal profiles and address books, user names and passwords, and notifications.
- **CSR Management** – Siebel's standalone application for CSR management. Includes searching accounts and invoking impersonate company user functionality.

About This Guide

The Siebel Software Developers Kit allows developers to write custom code against Siebel applications. This document assumes an in-depth understanding of and practical experience with:

- Billing 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

This guide also assumes you have:

- Read the Billing Manager product documentation and are familiar with the functionality

- Read the javadoc that comes with the SDK
- Successfully installed Billing Manager in a J2EE development environment
- Knowledge of how to develop J2EE web applications using JSP, Struts, Tiles and XML

Related Documentation

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

Print Document	Description
<i>Siebel Billing Manager Installation Guides</i>	How to install Siebel Billing Manager in a distributed environment.
<i>Siebel Billing Manager Data Definition (DefTool) Guide</i>	How to use DefTool to define the rules for data extraction in a DDF file.
<i>Siebel Billing Manager Presentation (Composer) Design Guide</i>	How to use Composer to define the rules for mapping data to templates for viewing statements.
<i>Siebel Application Guide for Billing Manager (B2B or B2C)</i>	Describes the core functionality of the Billing Manager Foundation Application.
<i>Siebel Billing Manager Administration Guide</i>	How to set up and run a live Billing Manager application in a J2EE environment.
<i>Siebel Billing Manager Troubleshooting Guide</i>	How to troubleshoot general and specific problems with a Billing Manager implementation.

The following online help is also available for the Siebel Command Center and tools (DefTool and Composer).

Online	How to Access
DefTool and Composer Help	In DefTool or Composer, select Help>Help Topics .
Command Center Help	In the Command Center, click the Help button on the menu. You can also click the Help button on Command Center screens for context-sensitive help.

2 Functionality Overview

Billing Manager Features

The following table provides a complete list of the use cases that specify the functionality of the Billing Manager application:

Requirement Category	Description	Use Cases
Enrollment and Login	Enroll users and manage the login user session. The <i>Log In To Application</i> use cases is a pre-condition for all customer use cases	Login to Application Logout of Application Enroll Company Users Enroll As User Forgot Password
Administration	Manage the billing hierarchy.	Manage Company Profile Manage Billing Accounts (3 use cases) Manage Billing Account Users Manage Business Structures (4 use cases) Manage Business Structure Users Manage Business Structure Devices Edit Business Structure Devices or Accounts Manage Folder Nodes (4 use cases) Manage Company Users (2 use cases) Manage Corporate Address Book
Statement Presentment	Provide a place to perform functions related to presenting and accessing customer statements	View Business Dashboard View Account Summary View Device List View Device Summary View Record Details for Device View Unbilled Activity View Bill Messages Print Current View Print Invoice Download Account Data
Payment	Handle user payments	Make One-Time Payment Setup Recurring Payment

Requirement Category	Description	Use Cases
		Manage Payment Accounts View Payment Activity
Analytics	Summarize data in a variety of reports	View Account Level Report View Device Report
Profile Management	Manage user account and profile information Send e-mail notifications as appropriate to customer	Manage Personal Profile Manage Notifications Manage Personal Address Book (PAB) Manage PAB Entry Manage Password Manage Username
Customer Service	Customer Service Application that enables critical account management access and capabilities to customer service representatives.	Create Organization Create Organization Summary Create Administrator User Create Administrator Summary Edit Administrator Edit Administrator Details Summary Search Organization Search Organization Results View Accounts for Organization View Administrators for Organization Search for an Account Search Account Results View Last Statement for Account Add Billing Account Billing Account Added Search for Internal User Search Internal User Results Edit Internal Users Edit Internal Users Summary Add Internal Users Add Internal User Summary Edit Internal User Profile
Help		Siebel Billing Manager-specific FAQs, Contact, Terms, and Conditions

Billing Manager Overview and Architecture

Overview

Billing Manager is the leading Electronic Bill Presentment and Payment (EBPP) solution for communications service providers. Billing Manager provides the mission-critical application platform required for securely managing customer account information such as bills, statements, and other data. With Billing Manager, carriers can provide business and consumer customers with highly personalized online account information and self-service capabilities that can be served across multiple channels (Web, PDF, IVR, hand-held devices and e-mail). It serves as the foundation for managing recurring customer relationships.

Billing Manager is specifically designed for organizations with large numbers of customers, high data volumes and extensive integration with systems and business processes across the enterprise. With its sophisticated data access layer, platform services and data stores, Billing Manager is uniquely capable of powering the most complex EBPP and Customer Self-Service solutions.

Billing Manager is available in a consumer edition and business edition with specific features and functionality designed for each user base.

Functional Areas

Data Access

Billing Manager provides a core set of features in support of defining file formats, defining presentation formats and accessing both indexed (previous prepared) or live (returned at runtime) data. The DefTool and Composer tools provide easy to use, yet sophisticated, mechanisms for defining record layouts and combining those layouts into presentable content templates known as views.

Command Center

The Siebel Command Center is a sophisticated web interface for managing all the aspects of a Billing Manager install. Siebel Command Center provides all the required tools for managing batch processes, defining payment interactions, determining run time status and configuring Siebel applications. Using command center you can quickly and easily add a new application, configure application services or determine the state of a running system.

Enrollment

Siebel provides a complete user management framework (UMF). Siebel out-of-the-box enrollment supports a user management framework known as Common Directory Access (CDA), an LDAP light implementation perfect for the needs of emerging and smaller businesses. Siebel also provides a pluggable interface providing the interfaces required to connect the Siebel enrollment services with an external user management framework such as LDAP.

Role Based Access Control

Siebel provides a complete role based access control subsystem (RBAC). RBAC provides all the require services to define roles, define permissions, define resources and to grant or deny access to resources based on these definitions.

Hierarchy Based Access Control

Siebel provides a complete Hierarchy based access control system allowing for the definition and display of content based on relative position in a corporate hierarchy. Using Siebel Hierarchy clients can specify their particular set of hierarchy elements, for example geographies, divisions within geographies, managers in divisions, accounts associated with managers etc and then grant or deny access to accounts and other data based on a users location within the defined corporate hierarchy.

Payment Integration

Siebel provides interfaces and tools for managing connections to external payment providers while also including the ability to schedule and manage personal payment data. Using the Siebel Payment subsystems clients can define the relationship and connection data required to interface with external payment providers such as the Check Free, Verisign, or a custom payment-clearing house.

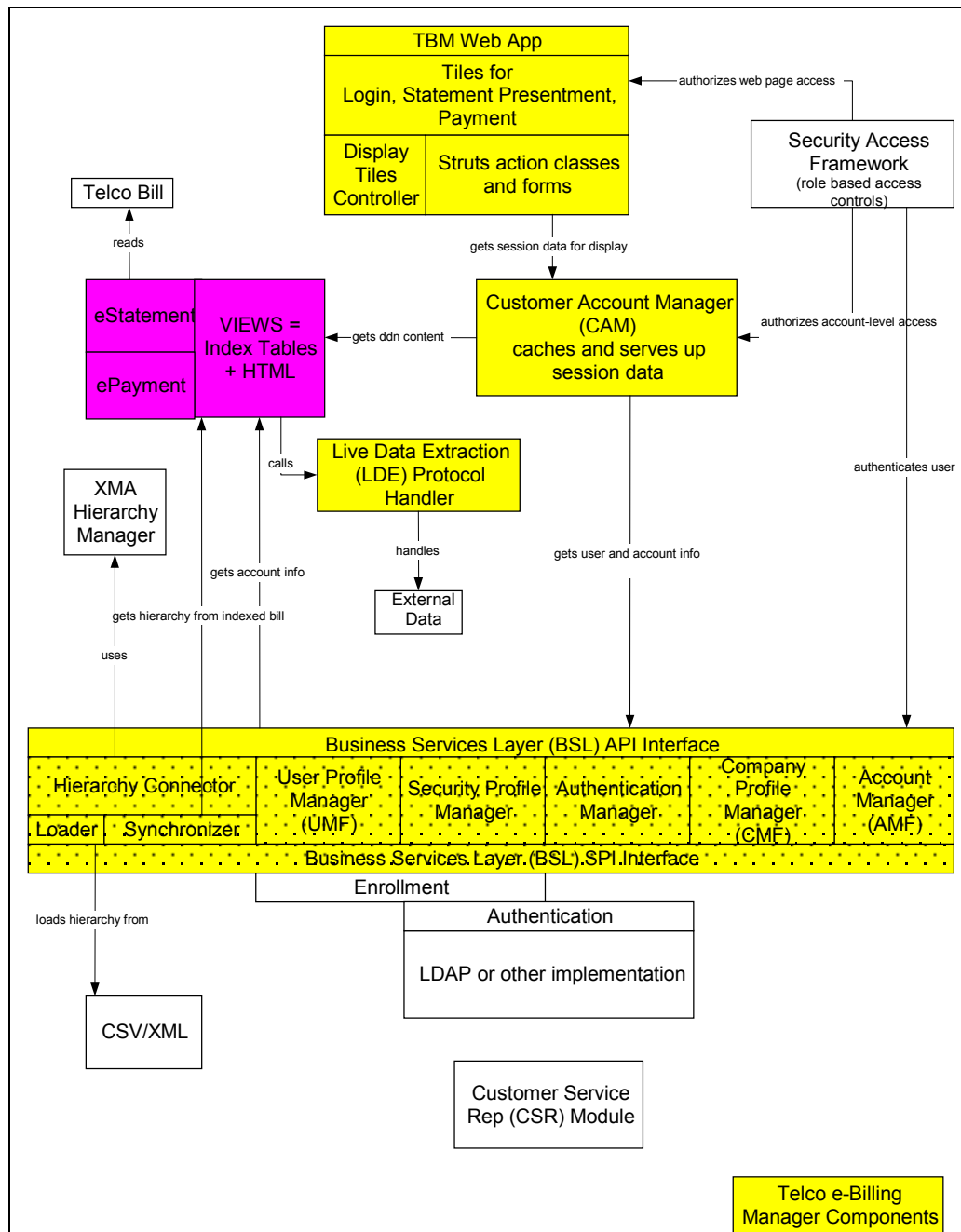
Architecture Components and Services

The Billing Manager Architecture is based on a set of core concepts and sub-systems that allow developers and external service providers the ability to interaction with and provide services to the Billing Manager Core. Conceptually Billing Manager is composed of:

- Action Classes – the set of actions used by client applications to extract data from Billing Manager for presentation and to perform application navigation. Billing Manager provides support for developing and extending existing functionality around action classes via `EdocsAction`, `EdocsActionForm` and similar classes.
- Customer Account Management (CAM) – the set of API classes providing mechanisms to access subscriber data and current subscriber session data. The CAM API provides support for obtaining subscriber data and all the underlying subscriber data such as Accounts, Sub Accounts, User profile, etc. For version 5.1, CAM has been extended to support Multiple-DDN.
- View Processing – The view processing system combines the areas obtaining data and combining with a predefined view; Live Data Extraction (LDE), obtaining data from a non-traditional data source (DDN) and View Post Processing, massaging view content before presentation.
- Security Access Framework (SAF) – the SAF framework provides support for rules based authentication and access control. SAF provides a complete framework for defining resources, and granting or denying access to resources based on well-defined sets of users and rights granting uses access to resources.

- Business Services Layer (BSL) – BSL supports both exposing underlying system content and accessing that content for the purposes of user authentication, user, user profile, company, company profile and other entity management. BSL provides the core services to access current use state as well as provides the underlying SPI interfaces for integration user management sub-systems.
- Payment Manager – The Payment sub-subsystems provide both the ability to integrate back end payment processors, via payment gateways, and the UI services to manage payment data such as registering a bank account or setting up a recurring payment.
- eStatement Manager – The eStatement sub-subsystems provide the abilities to select, index and present billing data via custom data definitions (DDFs) and presentment definitions (views) rendered via Struts and Tiles.
- Hierarchy – The hierarchy sub-system provides the required services to define and load hierarchy data; associate a subscriber with a location in a hierarchy; and return resources viewable at a location in a hierarchy.

Billing Manager Components



eStatement Subsystem

Content Retrieval

Content Retrieval is a domain-independent API set that provides runtime access to client data elements and presentation of data. Key features include:

- Well defined API for accessing statement and client data.
- Core object definitions such as `Customer`, `Account`, `Statement`, etc.
- Definition of data content via DDF files, mapping traditional “bill” content to field definitions
- Definition of presentation via views, describing the placement, output format, sort, filtering and other characteristics of data presentment at run time.
- Flexible Struts/Tiles based application development, supporting easily definition of applications
- Definition of “statement/bill” structure for easy data access in client application

Data Indexing

Data Indexing is a domain-independent component that provides support for fast loading and return of client data. Key features include:

- Easily customizable indexing batch jobs, managing client data for ultra-fast retrieval
- Easily customizable data definition files (DDF) for defining which fields within a file should be indexed for fast access

Payment Subsystem

Payment Utilities

Payment Utilities is a domain-independent API set providing run time access to payment account data. Key features include:

- Well-defined API for accessing payment data.
- Core object definitions such as `Check`, `CreditCard`, `PaymentAccount`, `Invoice` etc, fully modeling traditional payment requirements.

Payment Gateway

Payment Gateways are domain-independent components that provide development-time and run-time support for interacting with payment services providers such as ACH, Checkfree, Verisign and others. Key features include:

- Fully integrated into the Siebel Command Center for ease of gateway definition and management.
- Comprehensive set of gateway attributes covering areas of connectivity, auditing, Siebel security etc.
- Comprehensive API/SPI set for developing custom payment gateways
- Comprehensive set of payment batch jobs for managing the day-to-day process of interacting with payment service providers.
- Payment Plugins – a complete set of payment job lifecycle interfaces providing client developers the support required to interact with each and every payment job during its run time lifecycle.

Customer Account Management

Customer Account Management (CAM)

CAM is a Telco-domain specific API set that provides access to subscriber data. Key features include:

- Well-defined API for accessing Customer/Subscriber data.
- Ability to access current account information, including both default and accounts selected by account id.
- Ability to access current and specific sets of summary account data.

Using the CAM Layer developers can read, write and create customer specific data including profile, account and individual attributes.

In addition the CAM API interfaces can be extended to return custom extensions to core objects. CAM interfaces are extended by modifying properties to specify custom classes extending core Siebel implementations. The custom implementations are then returned via traditional API calls.

Examples of extendable interfaces in the package `com.edocs.application.tbm.cam.api.*` are:

- `Customer` – Implementation of `ICustomer`, may be extended to add custom functionality for the specific vertical domain.
- `CustomerAccount` – Implementation of `ICustomerAccount`, may be extended to add custom functionality specific to customer accounts.

Business Services Layer

Business Services Layer API (BSL)

BSL API is a domain-independent component that provides run-time access to core Billing Manager services required by API based applications. Key features of BSL specific to API applications include:

- Ability to read, write and create Billing Accounts.
- Ability to read, write and create user objects, representing a logged in user.
- Ability to read, write and create Customer objects and associated profile objects.
- Ability to read, write and create Company objects, and associated profile objects.
- Ability to search for and return instances of a variety of entities such as users, companies, company profiles etc.

Often BSL is used as an integration point with various entity managers, however BSL can be used directly to manage various entities, typically in support of CSR functionality.

Business Services Layer Service Provider Interfaces

BSL SPI is a domain-independent component focused on providing key interfaces linking Billing Manager with external entities managers such as LDAP. Key interfaces and features including:

- `IUserProfileManager` – Interface between BSL and external user management systems for the purposes of Authentication
- `ISecurityProfileManager` – Interface between BSL and external user management systems for the purposes of user management operations such as create, update and delete.
- `ICompanyProfileManager` – Interface between BSL and external company management systems for the purpose of managing Company and Company profile instances.

The BSL package contains a complete set of core implementations based on the Siebel Database user management sub-system any of which can be replaced as required. Additionally the BSL package comes with an LDAP implementation that can be used as a starting point for custom LDAP integration.

View Processing

Display Tiles Controller

The Display Tiles Controller is a domain-independent component that extends Struts/Tiles framework to provide key support for Siebel Communications Self-Service Manager features. Features include:

- View Loading Support – The ability to load a specific view using a provided JSP page and a Billing Manager Tag.
- Live Data Extraction Support – the ability to use the Billing Manager view merge engine with non-traditional data sources, for example merging data from a database with view defined specifically for that data
- View Post Processing Support – the ability to take a view and change its contents programmatically *after* the content has been merged with the view but *before* the content is presented to the user.

Live Data Extraction (LDE)

LDE is a domain-independent component of view processing that allows for the integration of external data content with the view merge engine. Key features include:

- Seamlessly integrated into the Siebel Communications Self-Service Manager product, and can take advantage of all view specific features such as sorting, filtering etc
- The ability to apply DDF templates to external data for use with Views
- The ability to take advantage of client context, from session data obtained via an action class, to extract data only appropriate for current user.

View Post Processing

View Post Processing is a domain-independent component of view processing that provides mechanisms for interacting with the view processing engine to perform run time changes to data pre-display. Key features include:

- Custom view processing, with complete access to user context and raw data content.
- Complete access to Billing Manager features and functions, in support of data transformations
- Seamlessly integrated into the Siebel Communications Self-Service Manager applications
- Can be configured per tile, per view or for all views
- Can be chained together to perform complex processing

Security Access Framework

Role Based Access Control (RBAC)

RBCA is a domain-independent component that provides role-based access control to Billing Manager applications. Key features include:

- XML based rights repository, defining role to resource accessibility mappings, often referred to as rights, privileges or permissions.
- Role-based mapping engine supporting mapping client roles to a simpler set of roles for use in web applications.
- Out of the box permission implementations for protecting resources such as Struts actions, JSP buttons, JSP Pages, sections of pages, menu items on pages etc.
- Java based Permission definitions, allowing for sophisticated permission definitions that can be used to create complex "implies" permissions where permissions imply or leads to other permission.
- Complete API for accessing the permission engine directly, supporting custom access development.

Hierarchy Based Access Control

Hierarchy Bulk Load

Hierarchy Bulk Load is a domain-independent component that provides mechanisms for limited statement content display based on user position in a hierarchy. Key features include:

- Well-defined XML based bulk loading and synchronization services, provided as standard Billing Manager jobs for easy integration of client hierarchy data.
- Custom integration with BSL for integration with external providers of hierarchy data such as Companies, company profiles etc
- Well-define developer API for use in custom hierarchy application implementation

Billing Manager Install Directories

Folder	Contents
Root folder	Root Directory (c:\Siebel or /opt/Siebel)
Common	SDK Support default files
Config	Configuration common all of Billing Manager
DB	Schema files for Oracle, DB2 and MSSQL
estatement	Statement specific support
J2EE apps	Command Center and Billing Manager default applications
Jre	JRE Support
Lib	Common libraries
payment	Payment specific support
Release notes	Release notes
Uninstall	Uninstall Support
views	Prepackaged views

The J2EEApps directory contains:

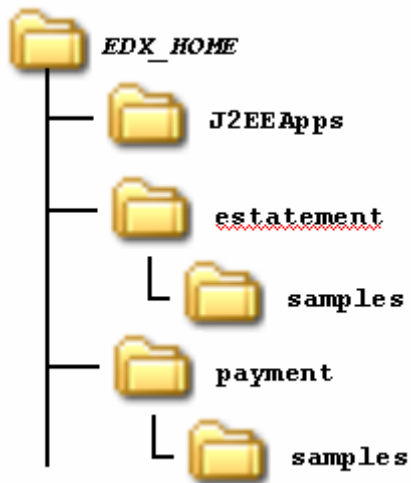
- eStatement - The Siebel command center application
- tbm-b2b (Business Edition) - Communications eBilling Manager B2B out of the box application
- tbm-csr (Business Edition) - Billing Manager CSR Application
- tbm-b2c (Consumer Edition) - Billing Manager B2C out of the box application

The estatement directory contains:

- Support, samples and documentation core to statement based applications

The payment directory contains:

- support, samples and documentation core to payment based applications



Config Directory

The Config directory contains:

- app-config.properties - Properties specific to all Billing Manager applications
- common-logging.properties - Apache logging properties
- hibernate.properties - Hibernate properties for Oracle,DB2 and MS-SQL
- hierarchy.cfg.xml - Support for hierarchy
- hierarchy.hibernate.properties - Hibernate properties specific to hierarchy
- log4j.xml - Logging properties

About Billing Manager Applications

Billing Manager Applications are combination of:

- A set of artifacts created and managed by the Siebel Command Center
- J2EE Applications
- A set of application DDN and other properties associated with an application (run time configuration via app-config.properties)

You create and name a Billing Manager application in the Command Center to match the APP_DD entry in app-config.properties. This DDN becomes the parent or primary DDN associated with the application.

By default Billing Manager applications contain two DDN entries:

- DDN.Summary - which specifies the source of the summary data
- DDN.TBM - which specifies where core Billing Manager data will be obtained

While these two DDNs are often the same, both referring to the actual DDN Billing Manager, they can use different data sources for content.

The "primary" application created with Command Center must match the APP_DNN entry in `app-config.properties`:

```
FILE_PROP_VERSION=TIME_STAMP

APP_DDN=TBM
# DDN
DDN.Summary = TBM
DDN.TBM=TBM
# DDN for Unbilled Activity
DDN.TBM_UA_Detail=TBM_UA
DDN.TBM_UA=TBM_UA

# Views
DDN_NAME_AccountList = Account_Summary

#LDE_MAX_CACHE_SIZE=52428800
. . .
```

Application Directories

Artifacts of the Command Center create application operation are:

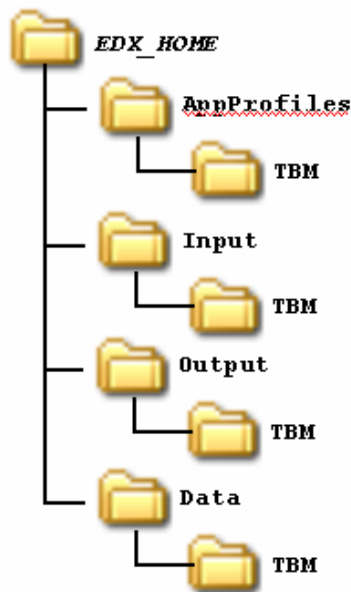
- Registration of the application
- Creation of AppProfile, input, data and output directories

When you use the Command Center to create Billing Manager application, it creates a number of directory artifacts. These artifacts represent Input, Data and Output processing directories, each named based on the application. Indexer and other jobs use these directories to manage files.

As files are processed time stamps are prepended to the file name to allow for intelligent management of multiple instances of the same file name.

Indexer jobs:

- Process files found in Input directories
- Move results to Data directories

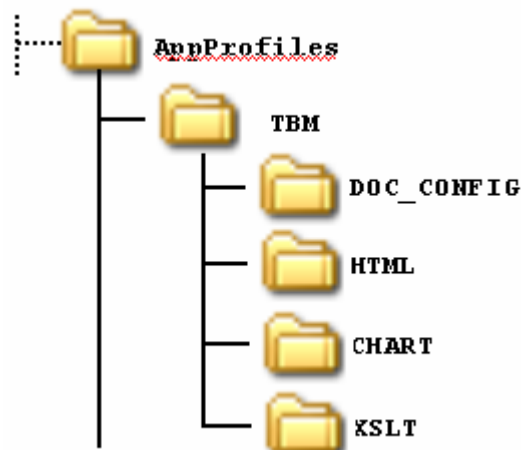


Application Profiles

Application profiles:

- Are created when you add a new application in Command Center
- Contain all the underlying artifacts of publishing
- Contain subdirectories by functional area (HTML, Indexer support etc)

Functional areas contain time-stamped directories containing "version sets"



Application Packaging

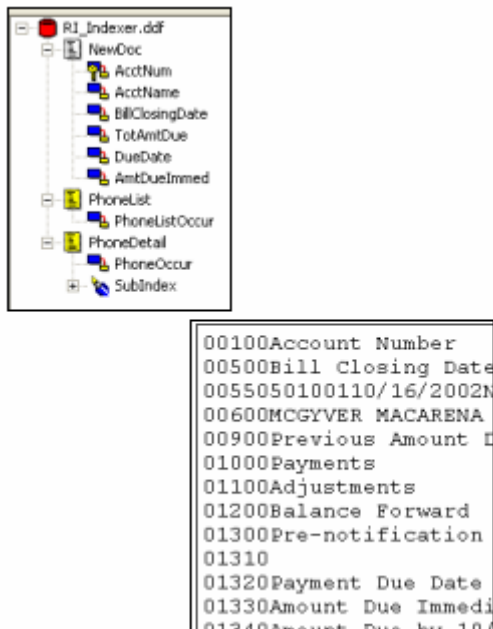
Billing Manager applications:

- Are packaged as an Enterprise Application Archives (.EARS)
- Contain a set of supporting EJBs
- Contain one or more Web Applications (.WARS)
- Must have an associated a Data Definition Name (DDN) created and configured using the Command Center, including all views, application logic files, html templates and data definitions required for presentment

Data Definition Files:

- End with a .ddf extension
- Contain the rules for finding and extracting customer data with a "paper" bill
- Are created using representative business data and Siebel tools

Specialized DDF files are used for a core indexer batch job run regularly to prepare data for presentation:



Views, application logic files, and HTML templates

Views

- Are combinations of data definitions, HTML templates and data created with Siebel tools
- Are used to present client data

- Can be formatted to generate HTML, XML, Charts or other formats such as CVS
- Are versioned

Application Logic Files

- Contain the rules to present extracted data
- Define formatting, sorting, filtering, business logic and output formatting.
- May use HTML templates

A view is a set of design files what result in particular presentation of statement data. View files enable a user to dynamically display formatted statements, review notifications, emails or other account data.

A typical HTML based statement view includes a pair of DDF and ALF files as well as one or more template HTML files.

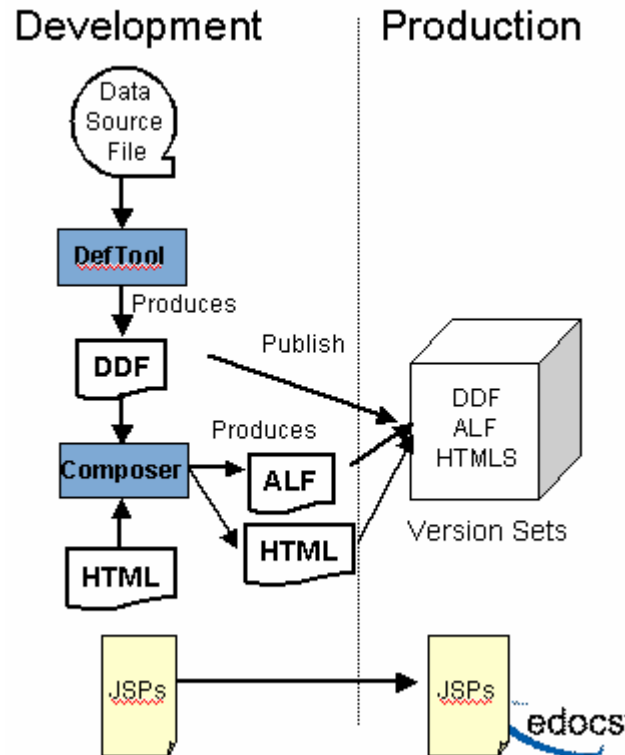
View Functionality

- Views provide the ability to sort, filter, page, and total fields easily
- View types include charts, drill-downs, downloads, and printer friendly
- Views can also be used as front ends to messaging & personal address book style functionality

View Development

Design Elements

- . Version Sets
 - DDF - Provides data extraction rules
 - ALF - Provides formatting and personalization rules
 - HTML with embedded Siebel tags - Provides presentation templates



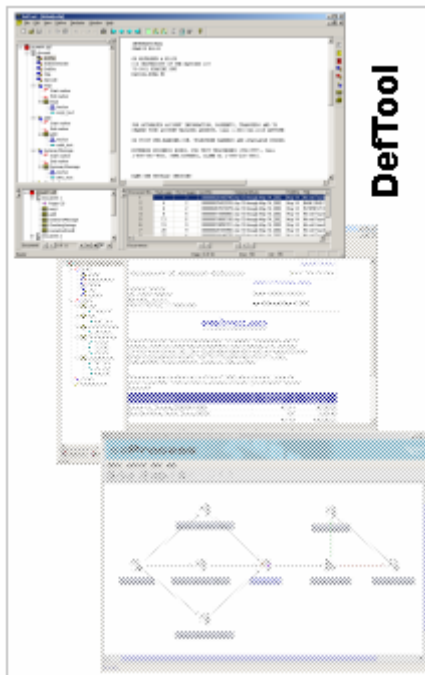
View Types

- CSV - Downloadable comma-separated values based on core data
- CHART - Displayable chart generated by Kavachart. Customizable via properties.
- HTML - Standard view type using generated or core data, an ALF and a HTML Template.
- XSLT - Downloadable XML, HTML or anything else generatable by XSLT and XML.
- XML, XML Query - Specialized view run of a detail extractor job to load the statement detail into the database, then generated by an SQL statement off stored data.

DefTool – Data extraction and mapping

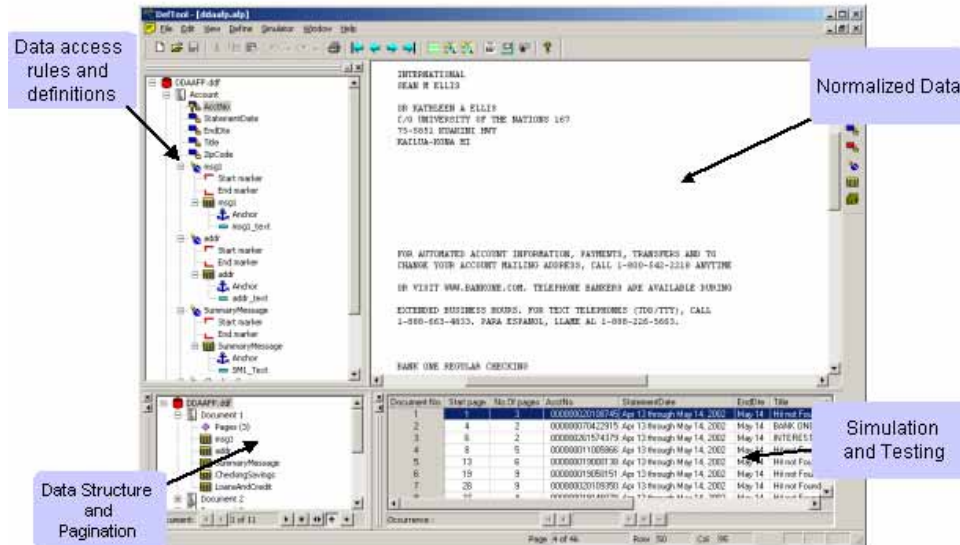
DefTool:

- Is a GUI tool for graphically defining data formats and outputs .DDF files
- Requires no programming or scripting experience
- Supports multi-format data-parsing to allow building connectors into multiple format data sources
- Include powerful post-processing formatting
- Includes a simulator for real-time rules testing



DefTool

DefTool - Interface

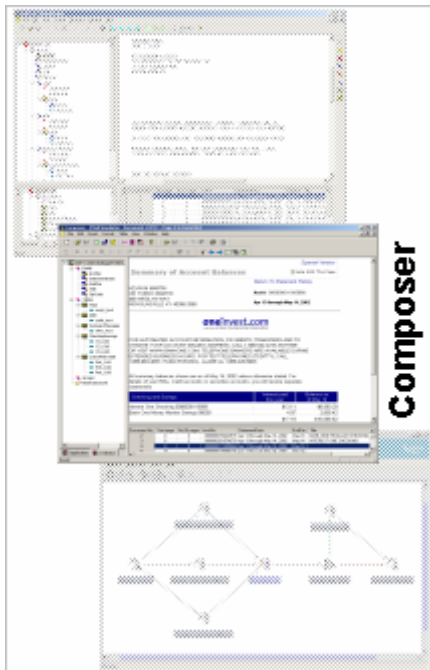


Composer – Business rules and presentation

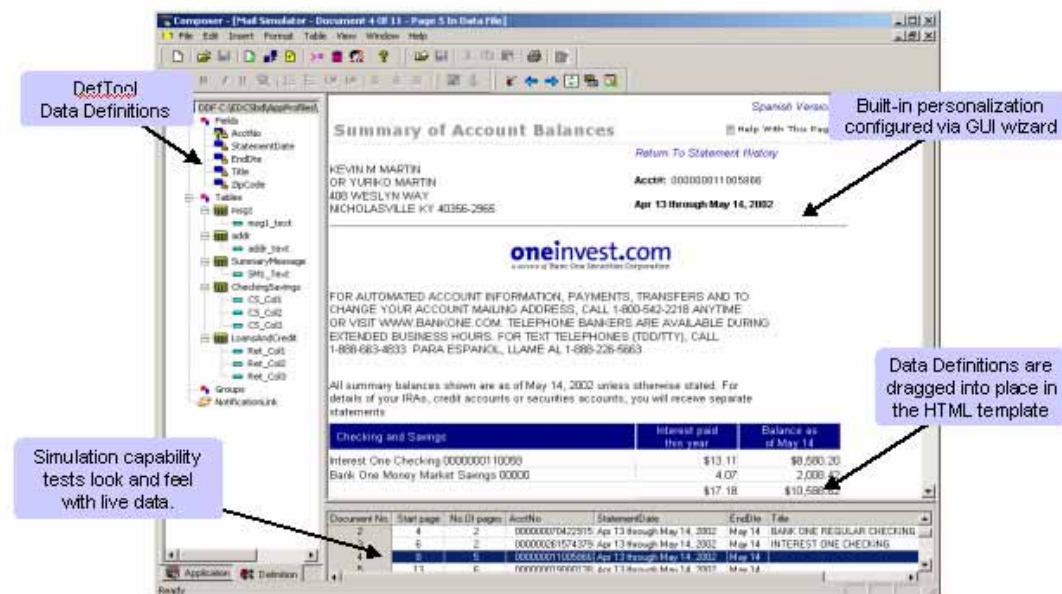
Composer:

- Is a GUI tool for defining how data and HTML are merged, including WYSIWYG HTML editing
- Requires no programming or scripting experience

- Includes powerful business logic and personalization engine as well as simulator for testing views



Composer Interface



The Command Center

The Command Center provides easy to use, integrated:

- Application configuration

- Run time reporting and statistics
- View and batch process management

Command center includes:

- Main Console - For creating and managing applications and job execution
- Publisher - For managing views

Batch Processes

Billing Manager supports standard and custom batch processing.

Batch processes:

- Are known as jobs
- Have one or more steps known as tasks
- Can be scheduled to run periodically

All installed applications have at least one job known as the Indexer Job.

Billing Manager comes with two types of batch jobs:

- The Indexer job, which:
 - Scans for and processes input billing statements
- Specialized jobs that:
 - Generate email notification
 - Extract specialized details for charts and reports
 - Generate HTML output
 - Generate XML output

Such specialized jobs are often run in preparation of large reports, etc.

Common Jobs

- **Indexer** - Standard job to prepare statement data for presentment. Required for dynamic presentment
- **Email Notification** - Standard job to generate an email notification when an online statement is complete and ready to review.
- **Purge App** - System maintenance job that removes index, email, reporting, detail, and other data from the database
- **Purge Logs** - System maintenance job that removes historical log information.
- **HTML & XML Output** - Statement presentment job which creates either HTML or XML output for each extracted primary key. Not required if live data retrieval is used.
- **Detail Extractor** - Specialized job to generate up-loadable XML based on XSLT style sheets.

Creating Applications

Billing Manager Applications are combinations of:

- Application name
- Data source
- Indexer job

Creating an application is a multi-step process composed of:

- Deploy the Billing Manager J2EE application ear
- Create the Billing Manager application
- Publish the indexer files
 - Create the indexer job

Creating applications, adding jobs

See the *Siebel Billing Manager Administration Guide* for information on setting up applications and jobs.

Publishing

Publishing is the process of uploading design time artifacts to the Siebel system such that applications can use them. You can publish individual sets of files or entire groups of files (bulk).

Version sets are:

- Dated combinations of files
- Are applied based on the current data

Live retrieval of historical bills & statement is handled by:

- Extract data from statement
- Compare against version set dates
- Select version set and apply

Batch Processing Vs. Dynamic Version Set

Batch processing:

- Uses most recent version set
- Applies to: Indexing, Email Notification, HTML Output, XML Output, Detail Extractor

Web view version sets are

Determined based on data of presentation data

- Apply to: HTML, XML, CSV, XSLT, XMLQuery

Scheduler

The scheduler:

- Must be running to use scheduling & jobs
- Is found in: %EDX_HOME/%estatement/bin
- Is started with: `wl_scheduler OR ws_scheduler -start -url t3://<server>:<port>`

3 Presentation Framework

This chapter describes:

- Core features of the Siebel struts & tiles extensions
- Billing Manager application development architecture
- Extending Billing Manager applications to include new menus and pages
- The `TBMTags` tag library used by Billing Manager, and Siebel libraries
- How to configure Billing Manager applications, including resources, properties and internationalization
- How to modify the application look and feel

Introducing Billing Manager Application Basics

This section describes:

- Billing Manager application fundamentals
- The Billing Manager directory architecture & structure
- Standard pageflows

Billing Manager Application Fundamentals

Billing Manager applications:

- Are packaged as an Enterprise Application Archives (`.EARS`)
- Contain a set of supporting EJBs
- Contain one or more Web Applications (`.WARS`)
- Must have an associated a Data Definition Name (DDN) created and configured using the Command Center
- Are configured by `app-config.properties`

Billing Manager -based applications:

- Are J2EE applications
- Are based on Struts and Tiles
- Support a set of core functionality including:
 - Login/logout - Authenticate a user

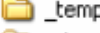
- Statement presentation - Present summary and statement details
- Payment - Make payments
- Profile management - Manage uid/pwd, personal address book etc.
- Hierarchy - Present and manage company and business based bill presentation hierarchy

Architecture & Structure

Billing Manager application architecture features:

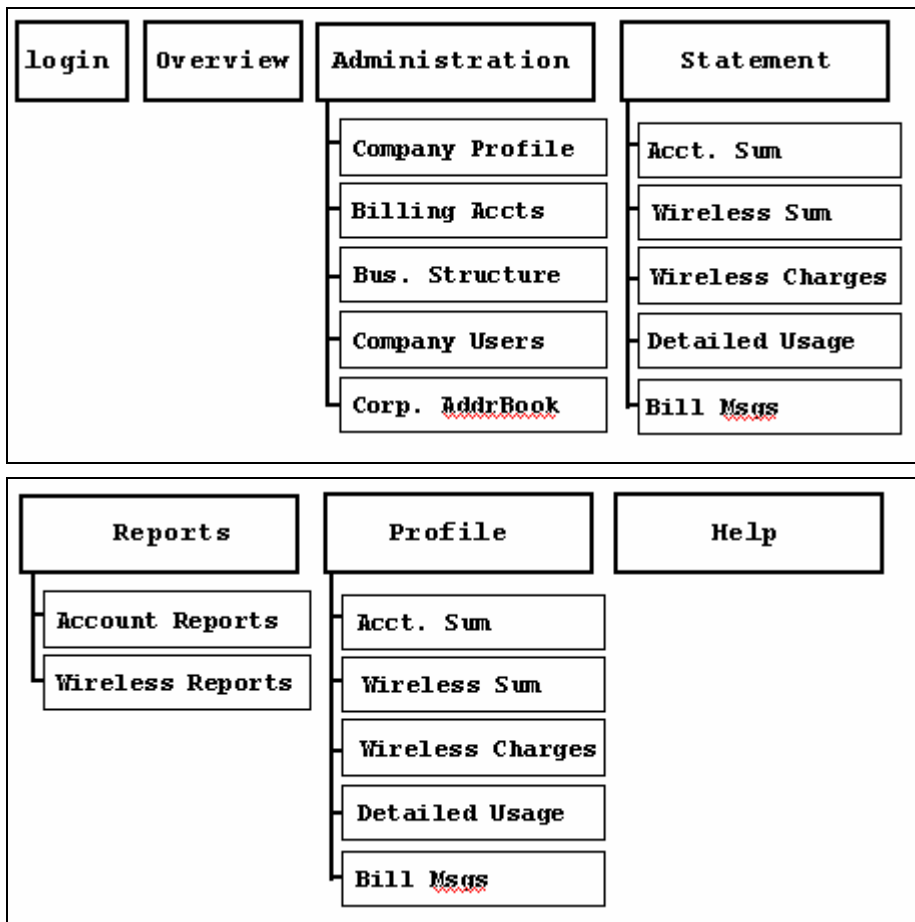
- Clear separation between UI, business logic and controller
 - Struts MVC framework developed by the Apache Software Foundation
 - Separation of roles in a project
- Customer-ready, flexible UI's
 - Use of the Tiles templating engine
 - Internationalization and multi-language support
- Best Practices
 - Standardized validation and error handling
 - No Java (business logic) code in JSP

Application structure

 tbm-b2b	Root
 lib	Support jars for TBM as well as TBM Application .war
 META-INF	Application.xml, defining web application as thmb..
 tbm-b2b.war	thm web application root
 _assets	Contains all application assets such as icons, style sheets etc.
 _includes	.jsp include files used by the tiles framework
 _templates	.jsp templates used by the tiles framework
 administration	.jsp specific to company administration
 enroll	.jsp specific enrollment support
 hierarchy	.jsp specific to hierarchy support
 images	immutable image files
 login	.jsp specific to login/log out support
 META-INF	
 profile	.jsp specific to profile support support
 WEB-INF	Struts, Tiles, application specific libraries and classes

Note: Every tab normally has a set of JSP

Standard Page Flows

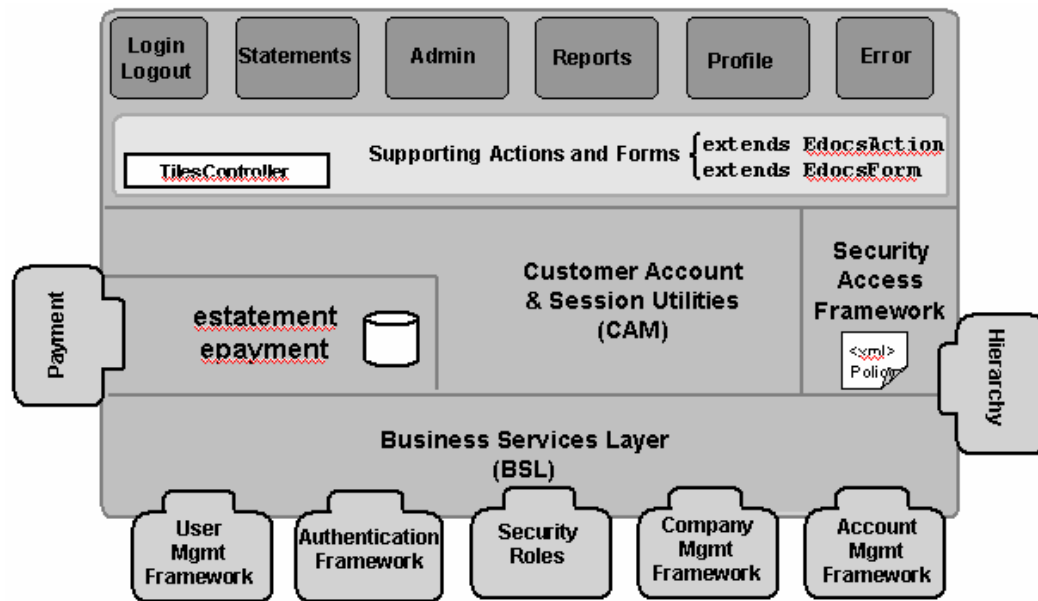


Struts and Tiles and the Siebel View Architecture

This section describes:

- Out of the box architecture components
- Modifying Existing Views
- Billing Manager Tags
- Configuring Billing Manager Applications

Billing Manager Application Architecture



- UserManagement: IUser, IUserProfile, IUserProfileManager
- Security Roles: ISecurityProfile, ISecurityProfileManager
- Authentication: IAuthenticationManager
- Company Management Framework: ICompany, ICompanyProfile, ICompanyManager, ICompanyManagerProfile
- Account Management: IBillingAccount, IBilling

Billing Manager Struts and Tiles

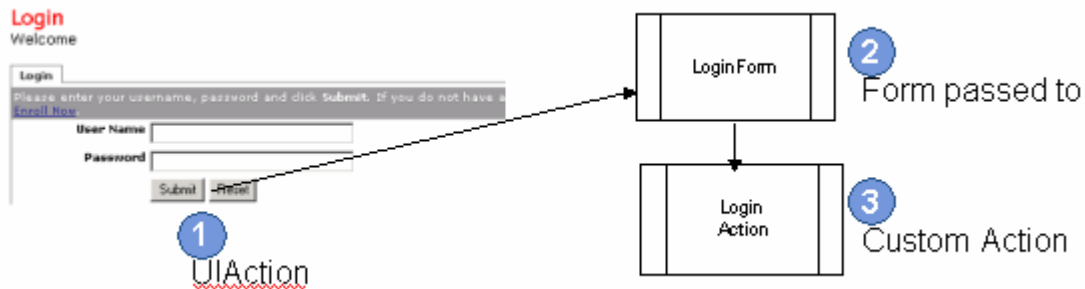
Billing Manager extends the Struts/Tiles framework to:

- Provide specialized struts support infrastructure
- Define core page flow and navigation common to all Billing Manager applications
- Define core page layout, used by navigation, common to all applications
- Provide internationalization and look & feel support

Overall the Billing Manager Struts/files framework provides an extensible environment that can be easily extended to meet customer needs

Typical flow

Pages can include forms and invoke Siebel defined actions.



Configuration example

Form:

```
<form-bean name="loginForm"
           type="com.edocs.application.tbm.core.forms.LoginForm"/>
```

Action:

```
<action path="/login/perform"
        type="com.edocs.application.tbm.actions.LoginAction"
        name="loginForm"
        scope="request"
        validate="true"
        input="login">
  <forward name="success"
           path="/hierarchy/main/dashboard/show.do"
           redirect="true"/>
</action>
```

Typical app-config.properties

app-config.properties Example:

```
FILE_PROP_VERSION=TIME_STAMP
```

```
APP_DDNAME=TBM your application name here
```

```
# DDN
```

```
DDN.Summary = TBM
```

```
DDN.TBM=TBM
```

```
DDN.Unbilled=Unbilled
```

```
# Views
```

```
DDN_NAME_AccountList = Account_Summary
```

```
#LDE_MAX_CACHE_SIZE=52428800
```

```
...
```

```
%EDX_HOME%/tbm/config/app-config.properties
```

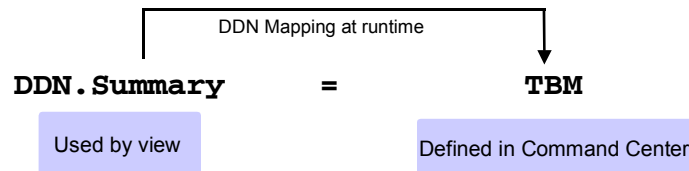
Name of application
in Command Center

DDNs
Summary typically
matches APP_DDNAME

Other properties
as required

Mapping DDNs

- *Actual* DDNs represent available sources of data
- Views specify data via *abstract* DDNs
- Abstract DDNs are mapped at run time to actual DDNs via configuration entries.



DDN mapping is important because it allows the developer and system administrator to quickly and easily associate content with data without application coding changes. Applications normally have a default DDN, which also is specified by the Siebel application name. The default DDN is used in a variety of places within the struts/tiles hierarchy as well as other locations and should rename TBN. Under normal circumstances the Summary view is mapped to the application DDN although it can be changed to map to any appropriate DDN associated with an indexer job.

You can create other DDNs that map to whatever data is required. Typically a detail extractor job is run, associated with a details job, for the purpose of quick access to account level billing details. Other DDNs typically used are unbilled data, or custom charting or reporting.

Struts Extensions

Billing Manager extends Struts in several areas

- `EdocsActionServlet` extends `ActionServlet` to handle Billing Manager session initialization
- `EdocsAction` extends `Action` to:
 - Check and force login as required
 - Check for and handle multiple submits
 - Forward to developer implemented `doAction`
- `EdocsActionForm` extends `ActionForm` to handle reset and auto-population of fields. In addition `EdocsActionForm` adds a base hashmap variable which can be used to add new content to the form without the need for adding additional class scope variables.

Billing Manager Tiles

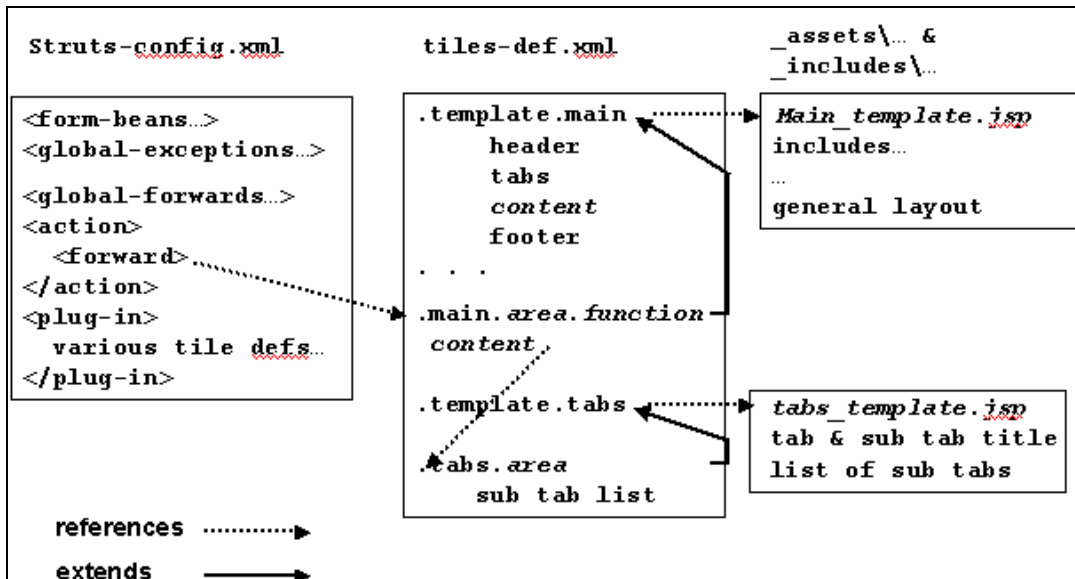
Billing Manager uses a tiles structure to define the look and feel of applications. The default structure includes a set of template tiles definitions that define the main functional areas of applications.

- Billing Manager uses a set of core tiles definitions to define the general Billing Manager structure and actions.
- Tiles *definitions* are used to create templates for each UI functional area.
- Templates are further specified by tab and sub tabs.

Some of the Billing Manager template tiles definitions included are:

- `.template.main` - Defines the overall structure of a presented page. Header, footer etc sections are defined here. tab menus etc.
- `.template.tabs` - Defines a core tab layout for applications. Pages normally have a set of tabs and sub tabs. The template tabs definition defines the page and other attributes of this list such as selected tab and sub tab list.
- `.template.module` - Defines a number of top level pages which do NOT include content. Such pages include, login, error, etc.
- `.view.setup` - Defines the default attributes for a presentable view (core defaults). Common attributes of a view include its name, type and DDN.

General tiles structure



.template.main**.template.main snippet Example:**

based on main template

```

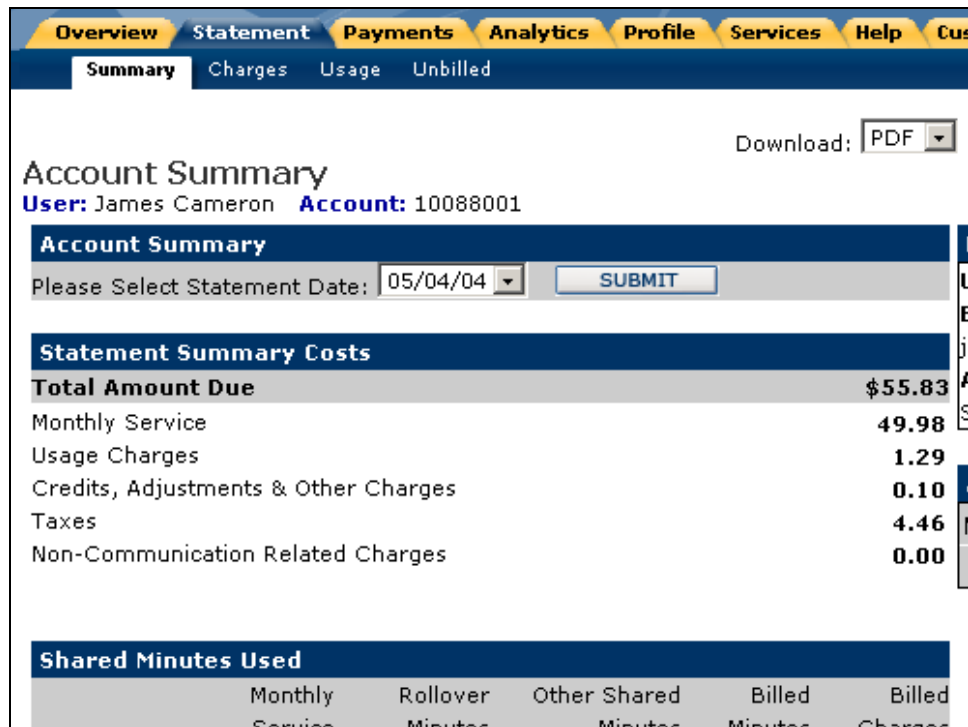
<definition name=".template.main" path="/_assets/templates/main_template.jsp">
  <put name="title" value="pagetitle" type="string"/>
  . . .
  <put name="tabs" value=".template.tabs" type="definition"/>
  <put name="tab" value="AccountSummary"/>
  <put name="subtab" value="" type="string"/>
  . . .
  <put name="action" value="/_assets/includes/blank_page.jsp" type="page"/>
  <put name="lnav" value="/_assets/_includes/blank_page.jsp" type="page"/>
  <put name="summary" value="/_includes/blank_page.jsp" type="page"/>
  <put name="module1" value="/_includes/blank_page.jsp" type="page"/>
  . . .
  <put name="analytics" value="/_includes/blank_page.jsp" type="page"/>
  <put name="view" value="/_includes/blank_page.jsp" type="page"/>
  <put name="sidebar" value="sidebar.simple" type="definition"/>
  <put name="footer" value="/_includes/footer.jsp" type="page"/>
</definition>

```

includes various navigation tiles

various content tiles

ear-tbm-xxx.ear/war-tbm-xxx.war/WEB-INF/tiles-def.xml

Example Main Summary Screen


Overview Statement Payments Analytics Profile Services Help Customer

Summary Charges Usage Unbilled

Download: PDF

Account Summary
 User: James Cameron Account: 10088001

Please Select Statement Date: 05/04/04 SUBMIT

Statement Summary Costs

Total Amount Due	\$55.83
Monthly Service	49.98
Usage Charges	1.29
Credits, Adjustments & Other Charges	0.10
Taxes	4.46
Non-Communication Related Charges	0.00

Shared Minutes Used

Monthly Service	Rollover Minutes	Other Shared Minutes	Billed Minutes	Billed Charges

.main.statement.account.summary**Example:**

includes various navigation tiles

```

<tiles-definitions>
  <definition name=".main.account.summary" extends=".template.main">
    <put name="page_name" value="com.edocs.page.name.bill.summary"/>
    <put name="page_description"
        value="com.edocs.page.description.bill.summary"/>
    <put name="page_title" value="com.edocs.page.name.account.summary"/>
    <put name="tabs" value=".tabs.statements" type="definition"/>
    <put name="tab" value="Statements"/>
    <put name="subtab" value="AccountSummary"/>
    <put name="action" value="/_assets/includes/action.jsp" type="page"/>
    <put name="lnav" value="/_assets/includes/blank_page.jsp" type="page"/>
    <put name="module1" value="/_assets/includes/account_info.jsp" type="page"/>
    . . .
  </definition>

```

ear-tbm.ear/war-tbm.war/WEB-INF/tiles-defs-statement.xml

The statement account summary definition defines a tabbed page that presents an account summary under the statements general tab.

Examining the definition reveals that `.main.account.summary`:

- Defines page name, description and title properties, loaded from `ApplicationResources.properties` via `com.edocs.page.description.bill.summary` and `name.bill.summary`.
- Specifies the sub tab set that should be displayed via the **tab** tile, which in this example uses `.tabs.statements`.
- Specifies itself as the within the "Statements" menu and as the "AccountSummary", defined in `applicationresources.properties` via properties of the form `com.edocs.tabname.Tab` and `com.edocs.tabname.AccountSummary`.
- Includes a navigation panel, shown on the left side of the page and module content pages.

.template.tabs**.template.tabs:**

```

<definition name=".template.tabs" path="/_assets/templates/tab_template.jsp">
  <putList name="tabList">
    <item value="Overview" link="/hierarchy/main/dashboard/show.do"
        classtype="com.edocs. . .EdocsBaseMenuItem"/>
    <item value="Administration" . . . />
    <item value="Help" link="/view/toDo.do"
        classtype="com.edocs. . .EdocsBaseMenuItem"/>
  </putList>
  <putList name="subtabList"/>
</definition>

```

ear-tbm-xxx.ear/war-tbm-xxx.war/WEB-INF/tiles-def.xml

`.template.tabs` defines the top level tab menus. Additionally top level menu items are added by inserting a new `<item...>` element specifying the name of the tab and the action to execute when it is selected.

.tabs.statements

.tabs.statements:

includes various navigation tiles

```
<!-- Statements Sub Tab List -->
<definition name=".tabs.statements" extends=".template.tabs">
  <putList name="subtabList">
    <item value="AccountSummary" link="/view/account_summary.do"
      classtype="com.edocs.application.tbm.core.menus.EdocsBaseMenuItem"/>
    <item value="DeviceSummary" link="/view/device_summary.do"
      classtype="com.edocs.application.tbm.core.menus.EdocsBaseMenuItem"/>
    <item value="DeviceUsage" link="/view/device_details.do"
      classtype="com.edocs.application.tbm.core.menus.EdocsBaseMenuItem"/>
    <item value="UnbilledActivity" link="/view/unbilled_activity.do"
      classtype="com.edocs.application.tbm.core.menus.EdocsBaseMenuItem"/>
  </putList>
</definition>
```

various content tiles

ear-tbm-xxx.ear/war-tbm-xxx.war/WEB-INF/tiles-def.xml

`.tab.statements` represents the set of sub tabs for a specific functional area. Sub tab lists always extend `.template.tabs`, inheriting the top level tab definitions, and add a set of items under the `subtabList` element representing each of the sub tabs, their associated. Note that the value attribute of each item element is used as a look up property in `applicationresources.properties` to find the corresponding tab text.

View Tiles

The `.view.setup` tiles definition defines the default values for view tiles. At a minimum tiles inheriting from view setup set the `viewName` tile to the view that should be displayed. Additionally you may specify the view type and `ddn`. Note that the DDN is a virtual or abstract DDN that is mapped by the `app-config.properties` file to the actual DDN at run time. The `viewName` attribute is used by the `display_view.jsp` to define which view should be rendered at runtime.

.view.setup Example:

```
<definition name=".view.setup" path="/_templates/display_view.jsp"
controllerClass="com.edocs.application.thm.displayview.DisplayTiledView">
  <put name="viewName" value="DefaultView" />
  <put name="viewType" value="HTML" />
  <put name="ddn" value="TBM" />
</definition>
```

Default view values

.statement.account.summary.view Example:

```
<definition name=".statement.account.summary.view" extends=".view.setup">
  <put name="viewName" value="B2B_AcctSum" />
</definition>
```

Override the view name

ear-thm-xxx.ear/war-thm-xxx.war/WEB-INF/tiles-def.xml

View JSP:

```
<%@ include file="/_includes/taglibraries.jsp" %>
<tiles:useAttribute id="viewName"
  name="viewName"
  classname="java.lang.String" />

<thmtags:getView name="<%=viewName%"/>" />
```

Obtain view name from
tiles definition

.../war-thm-xxx.war/_templates/display_view.jsp

Creating Custom Tabs

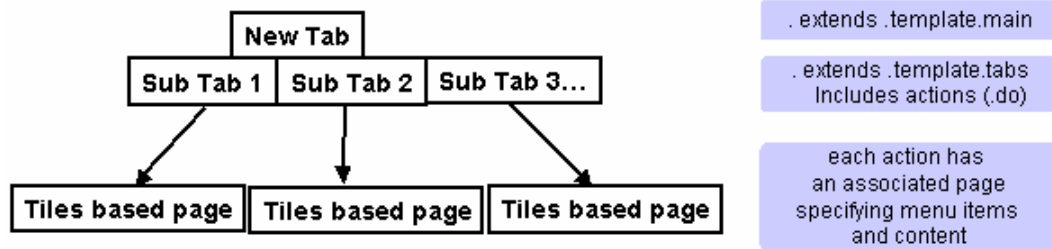
You can extend Billing Manager using custom tabs. Custom tabs include:

- Tab and sub tab page definitions
- Struts actions defining page flows
- Various properties files containing localizable text

To create custom tabs:

- 1 Define the tab structure and flows.
- 2 Create a tab, sub tabs and tab menu elements
- 3 Create action mapping.

Tab structure and flows



To create a Tabs & Subtab sets:

- 1 Add the new top menu tab element to `.template.tabs`
- 2 Define a set of sub tabs extending `.template.tabs`
- 3 Define page definition(s) that extends `.template.main`
- 4 Add properties to `Application.Properties` for various properties

Add `.template.tabs` item example

- 1 Define tabs using the `.template.tabs` tile.
- 2 Create sub tabs lists by extending this tile and adding a sub tab list

.template.tabs:

```

<definition name=".template.tabs" path="/_templates/tab_template.jsp">
  <putList name="tablist">
    <item value="Overview" link="/hierarchy/main/dashboard/show.do"
      classtype="com.edocs. . .EdocsBaseMenuItem" />
    <item value="Administration" . . . />
    <item value="MyTab" link="/MyTab/action.do"
      classtype="com.edocs.application.thm.core.menus.EdocsBaseMenuItem" />
  </putList>
  <putList name="subtabList" />
</definition>

```

Add new tab element and associated action

ear-thm-xxx.ear/war-thm-xxx.war/WEB-INF/tiles-def.xml

Define sub tabs

Define Sub Tab List:

```

<definition name=".tabs.test" extends=".template.tabs">
  <putList name="subtabList">
    <item value="Sub Tab 1"
          link="action"
          classtype="com..EdocsBaseMenuItem"/>
    <!--2nd and subsequent items... -->
    <item value="Sub Tab 2"
          link="/test/subtab2.do"
          classtype=""/>
  </putList>
</definition>

```

tab sets extend .template.tabs

must include a subtabList

contain a list of sub tab elements

Sub tabs contain:
Tab Display Text
Action
definition of base item class

Sub tab lists contain the list of sub tab element to action mappings displayed for a sub tab set. Each element in a sub tab list consists of three things:

- value - Containing property id of the sub tab, always prefixed with com.edocs.tabname.
- link - The action to perform when the sub tab is selected
- classtype - Either com.edocs.application.tbm.core.menus.EdocsBaseMenuItem or a class that extends EdocsBaseMenuItem.

Extending EdocsBaseMenuItem

You can extend EdocsBaseMenuItem to support custom visible/invisibility code for a menu element as shown below:

EdocsBaseMenuItem extension:

```

package com.mypackagestructure.menus;
import org.apache.struts.tiles.beans.SimpleMenuItem;
import javax.servlet.http.HttpServletRequest;
import com.edocs.application.tbm.core.menus.EdocsBaseMenuItem;
public class MyMenuItem extends EdocsBaseMenuItem {
  public MyMenuItem () { }
  public boolean displayItem(HttpServletRequest request) {
    boolean visible=//sometest
    return visible;
  }
}

```

Perform custom visibility test

Define Tab Pages

Every tab page extends `.template.main` and specifies a number of elements. The properties, `page_description`, `page_name` and `page_title` are rendered based on potentially localized version of properties found in `ApplicationResources.Properties` or a locale specific version. The `tabs` element defines the tab and sub tab lists to be defined in the page as well as the tab and sub tab properties

The `module1-n` tiles defines the page content. Modules 1-7 are defined and over-writable from `.template.main`. Normally module definitions specify another tiles definition containing a view show below the actual `.template.main` example show.

Defining pages:

```
<definition name=".main.area.subarea" extends=".template.main">
  <put name="selectDisplayMenu" value="_includes/select_display_type.jsp"
      type="page" />

  . . .
  <put name="page_name" value="page name property" />
  <put name="page_description" value="page desc property" />
  <put name="tabs" value=".tabs.name" type="definition">
    <put name="tab" value="tabtitlepropertyid" />
    <put name="subtab" value="subtabtitlepropertyid" />
    <put name="action" value="_includes/action.jsp" type="page" />
    <put name="leftSideNav" value="_includes/side_navbar.jsp" type="page" />
    <put name="module1" value=".area.subarea...view" type="definition" />
  </definition>

  <definition name=".area.subarea...view" extends=".view.setup">
    <put name="viewName" value="AssociatedViewName" />
  </definition>
```

Annotations:

- extend .template.main**: Points to the `extends=".template.main"` attribute.
- Set Titles**: Points to the `page_name` and `page_description` properties.
- Set properties values for name & description**: Points to the `page_name` and `page_description` properties.
- Set content**: Points to the `viewName` property in the nested definition.

Create Action Mapping

Create Action Mappings in `struts-config.xml`. For each sub tab in the tabs list create an action:

```
<definition name=".tabs.test" ...
  <item value="Sub tabx"
    link="/test/stab2.do" ... />
</definition>

. . .

<definition name=".main.test.XXX"
  extends=".template.main">
  ...
</definition>
```

Annotations:

- tiles-def.xml**: Points to the `<definition name=".tabs.test" ...` block.
- struts-config.xml**: Points to the `<action path="/test/stab2" forward=".main.test.XXX" />` block.

© 2004 edocs, Inc. All rights reserved. 

Development Utilities

Billing Manager includes support for:

- Custom Billing Manager tags
- Custom configuration files
- Custom log4j

Developers can use either custom properties files, or layer properties within the packaged configuration files

Billing Manager includes all required jars and configuration for log4j support.

Billing Manager Tags

Billing Manager comes with a set of custom tags that can:

- Force login or return login/user or account data
- Display a view, view data, or control view display
- Render content in a JSP page (buttons and other content) based on security settings

Billing Manager Tags are specified in JSP include but you can specify them directly.

TBM Tags Tag Library:

```
<%@ include file="/_includes/taglibraries.jsp" %> ← Specified in
or
<%@ taglib uri="/WEB-INF/tbmtags.tld" prefix="tbmtags" %> ↑
```

getView Tag

The `getView` tag displays a named view or the summary view by default.

Syntax:

```
<tbmtags:getView [name="viewName"]/>
```

Example:

```
<%@ include file="/_includes/taglibraries.jsp" %>
<tiles:useAttribute id="viewName" name="viewName"
    classname="java.lang.String" />
<tbmtags:getView name="<%=viewName%>" />
```

Typically a tiles-def defined page specifies a view by name

summaryDates Tag

The `summaryDates` tag outputs the list of summary dates as a section name. It returns a `SELECT=` statement populated with the current set of selectable summary dates.

Syntax:

```
<tbmtags:summaryDates property="fieldname in form"
                      [styleclass="optional style class"]
                      [onchange="optional on change method"]
                      [sortorder="optinal sort order=asc|desc|none"]/>
```

Example:

```
<%@ include file="/_includes/taglibraries.jsp" %>

<html:form action="myaction.do">
  <tbmtags:summaryDates property="date"
                        styleclass="formlabel" sortorder="desc"/>
</html:form>
```

summaryInfo Tag

The `summaryInfo` tag returns any summary field based on the currently selected statement. Common summary fields are:

- AcctNum
- CurrentCharges
- PymtType
- StatementDate
- AmountDue
- LateFee

Syntax:

```
<tbmtags:summaryInfo name="summary field"/>
```

Example:

```
<%@ include file="/_includes/taglibraries.jsp" %>

The amount due for Account <tbmtags:summaryInfo name="AcctNum"/> is
<tbmtags:summaryInfo name="AmountDue"/> for statement
<tbmtags:summaryInfo name="StatementDate"/>.
```

checkLogin Tag

The `checkLogin` tag forces the application to the login page if no valid user is in the session. Use this tag on any page requiring a login.

Example Check Login: If user

```
<%@ include file="/_includes/taglibraries.jsp" %> ← Specified in
or
<%@ taglib uri="/WEB-INF/tbmtags.tld" prefix="tbmtags" %> ↑
<tbmtags:checkLogin/>
```

getGeneralAccountInfo tag

getGeneralAccountInfo returns a specified account or billing account level property

Example Check Login: If user

```
<%@ include file="/_includes/taglibraries.jsp" %>
or
<%@ taglib uri="/WEB-INF/tbmtags.tld" prefix="tbmtags" %>
```

```
<tbmtags:getGeneralAccountInfo
  [billingAccount="billing account"]

  property="property to retrieve"

  [accountLevelRequest=true|false]/>
```

Optional Billing Account Number
Assumed to be a billing request if set
and all information will be
from the billing account given

Property to return

Assumed billing request
if set. Defaults to false

Here are some common Properties used by getGeneralAccountInfo. See the IAccount documentation for a complete list.

- UserId - Currently logged in user
- Associated Email Address - EmailAddress";
- First name - FirstName
- Last Name - LastName
- Default – DDN
- Associated Account Number

Working with Properties

Billing Manager applications can access properties via Configuration objects.

Configuration objects:

- Can load and store string, integer and sets of configuration values
- Auto update based on changes in backing store
- Auto store based on changes to in memory instances

Configuration Example:

```
app-config.properties:  
usesConfig.myProperty=value  
userConfig.timeout=1000  
userConfig.list=value1,value2,value3
```

Java Class:

```
import com.edocs.application.tbm.config.Configuration;  
Configuration c = Configuration.getConfiguration();  
String result = c.getValue("usesConfig.myProperty ");  
  
Int timeout = c.getIntValue("userConfig.timeout",10);  
  
String[] listof = c.getValues("userConfig.list");  
  
c.setValue(String key, String value);
```

May specify a
properties file name
on the system CLASSPATH

Returns value
or default

Returns array
representing values

Persist a value

Working with Logs

Billing Manager Applications can perform custom logging via log4J. Log4J supports:

- Controlling logging levels at runtime
- Configuring custom loggers
- Defining log message format

Classes can instantiate loggers, write logging code and then control log output via configuration files.

Log Example:config\log4j.xml

```
<category name="com.myco">
  <priority value="info|debug|warn|fatal|error">
</category>
```

Select msg level to
log for this category

Java Class:

```
package com.myco;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
public class usesLogging {
private Log m_log = LogFactory.getLog(usesLogging.class);
  public void logsMsg() {
    if ( m_log.isDebugEnabled())
      m_log.debug("in logsMsg");
  }
}
```

Create a logger &
log to it!

Developing Custom Actions

Billing Manager custom struts actions:

- Extend `EdocsAction`
- May use forms which extend `EdocsActionForm`
- Can access environment using the CAM/SessionUtils layer.

Customer Account Manager

See the CAM javadocs delivered in this SDK for implementation details about the Customer Account Manager (CAM) components of Billing Manager.

The Customer Account Manager (CAM):

- Provides information required to manage that state of a customer session.
- Includes a hierarchy of objects including customer, customer accounts, billing information etc.
- Uses a factory metaphor to access underlying customer data.

The Customer Account Manager, or CAM, is a subsystem designed for the purpose of managing access to underlying data. In the context of an action class CAM is typically used to access customer state information. The `CAMClassFactory` is a factory object which accesses both the current session and the underlying BSL subsystem to return requested data.

Customer account numbers are accessed via the `ICustomerAccount` interface which exposes a `getAccountNumber` method.

```

Accessing a Customers from within an action
import com.edocs.application.thm.cam.api.*;
...
ICustomer cust = CAMClassFactory.getCustomer(httpServletRequest);
ICustomerAccount acct = cust.getCurrentAccount();
String acctNumber = acct.getAccountNumber();

```

Obtain the current customer

Use ICustomer to obtain accounts

Common CAM interfaces

Interface	Description and Use
<code>CAMClassFactory</code>	Create or access customer information. Common use: <code>ICustomer getCustomer(HttpServletRequest);</code>
<code>ICustomer</code>	Return attributes of Customers. Common use: <code>ICustomerAccount getCurrentAccount();</code> <code>String getAttribute(String name);</code> <code>String getUserID();</code>
<code>ICustomerAccount</code>	Returns attributes of customers such as account #'s and summary data Common use: <code>String getAccountNumber();</code> <code>IAccountSummary getSummaryInfoDefaultDDN();</code>
<code>IAccountSummary</code>	Returns or sets data about the associated summary such as valid dates, current date, amount due, and parameters Common use: <code>String[] getStatementDates();</code> <code>String[] getAmountDue();</code>

CAM Example

Obtain the range of summary dates for customers current account:

```

import com.edocs.application.thm.cam.api.CAMClassFactory;
import com.edocs.application.thm.cam.api.IAccountSummary;
import com.edocs.application.thm.cam.api.ICustomer;
...
ICustomer cust = CAMClassFactory.getCustomer(request);
ICustomerAccount acct = cust.getCurrentAccount();
IAccountSummary summary = acct.getSummaryInfoDefaultDDN();
String summaryDates[] = summary.getStatementDates();

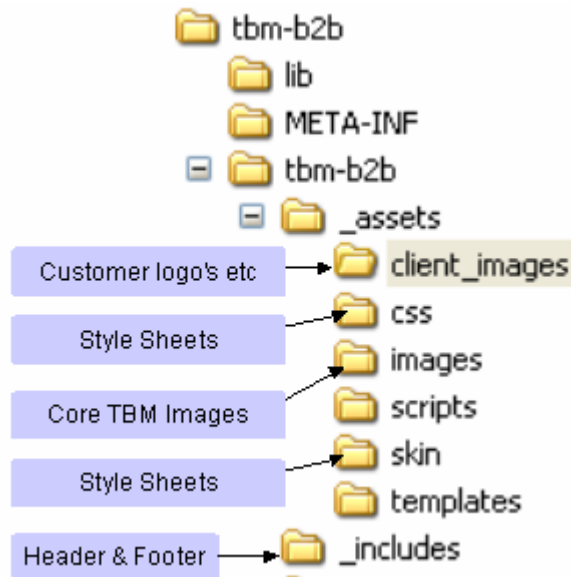
```

Look and Feel

To manage application look and feel you can:

- Manage template files such as `header/footer.jsp`
- Set a core color and text scheme using style sheets
- Manage images

Place these items in the Billing Manager directory as follows:



Define application header & footer content using:

`tbm-[b2b|b2c\_assets\_includes\header.jsp & footer.jsp`

```

<%@ include file="/_includes/taglibraries.jsp" %>
<div id="navtop">
  <div id="navtopblackbar">&nbsp;</div>
  <div id="navtoplogobar">
    
    <div id="navtoplinks">
      <a href="#">Contact Us</a> |
      <a href="<%= request.getContextPath()
        %>/logout/perform.do">Logout</a>
    </div>
  </div>
  <div id="navtopredbar">&nbsp;</div>
</div>
  
```

Message Rendering and Localization

UI text content is obtained extensively using resource properties.

Resource properties are:

- Read from `WEB-INF\classes\application[_locale].properties` based on run time locale
- Rendered by `<bean:message key="property name"> tags`

You can localize text resources by:

- Creating a locale specific application properties file
- Adding or replacing property text with localized versions

Style Sheets

Billing Manager Application style is controlled by style sheets found in:

`tbm-[b2b|b2c\_assets\css\` and `tbm-[b2b|b2c\skins\`

`\_assets\skins\skin.css`

- Primary CSS for look and feel
- Define the look and feel of pages and navigation
- Can change definitions (fonts, size etc)
- Should NEVER add or remove definitions

`assets\css\edx.css`

- Support of additional look and feel
- Exists to be customized
- Can contain new additions

`\_assets\css\hierarchy.css`

- Primary CSS for look and feel for hierarchy
- Can change definitions (fonts, size etc)
- Should NEVER add or remove definitions

Images

Images and logos exist to further define application look and feel

`\_assets\client_images\`

- Contains replaceable images

- Should contain any new images

skins

- Contains UI images for tabs etc
- Can be replaced, but must maintain size

images

- Contains core images which can be modified but no new images added
- Can be replaced, but must maintain size

4 Statement Presentation and Content Access

This chapter describes how to extend and customize views, including:

- View architecture basics
- Downloadable view support
- How to write and configure post processors
- How to write and configure live data extraction

View Architecture Basics

Billing Manager simplifies view presentation via a custom tiles controller, view JSP and view tiles.

View tiles:

- Use TBMTags via template JSP to render view content.
- Defines viewName, viewType, and DDN tiles, used by the view controller to specify content.

You can write custom template view JSP pages to manage special requirements such as download views

View Architecture

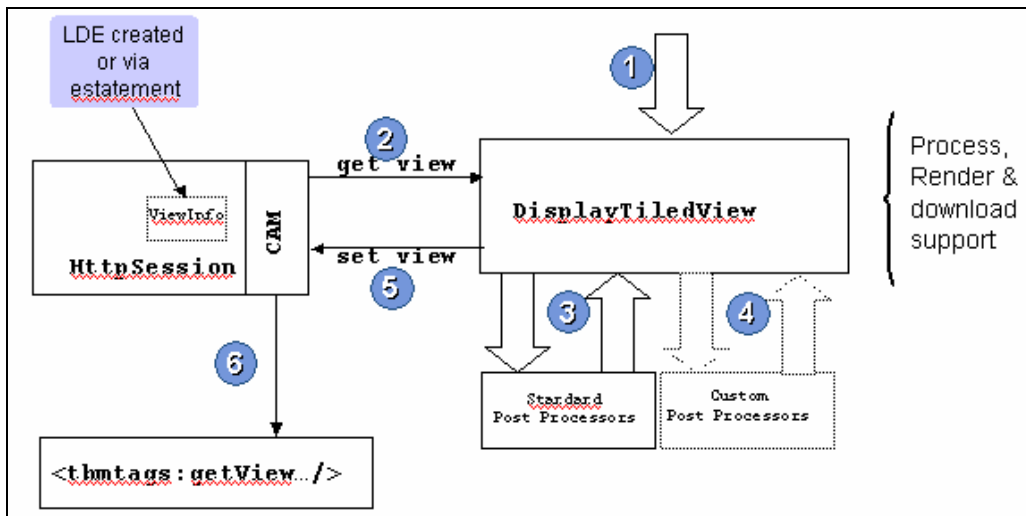
Billing Manager Supports two source data types:

- Indexed views, obtaining data from indexed DDNs and requiring no specialized actions
- Live data views, obtaining data from custom sources

Billing Manager View processing is extended based on a specialized tiles controller DisplayTiledView, which:

- Provides view rendering support
- Provides view post processing
- Is extensible, providing support for custom post processors and live data

Display Tiles Run-time Architecture



The Display Tiles Controller hooks run time requests to display views and executes as follows:

- 1 The struts/tiles framework calls the DisplayTiledView (DTV) controller.
- 2 The DTV controller then uses the CAM layer to obtain a view stream for processing.
- 3 Standard post processors and custom post processors are applied in the order defined in the view definition.
- 4 The resulting, perhaps changed, view is set back into the underlying session via CAM.
- 5 The actual JSP tied to the view via its `.view.setup` defined definition is run and the `tbmtags:getView` tag renders the view.

Downloadable View Support

Billing Manager supports creating download views:

- 1 Create and publish a download view
- 2 Use or modify download template JSP

Downloadable JSP Example:

```

<%@ include file="/_includes/taglibraries.jsp" %>
<tiles:useAttribute id="viewName" name="viewName"
  classname="java.lang.String" />
<%
String fileName=(String) request.getAttribute ("FILE_NAME");
%>

<tbmtags:getView name="<%=viewName%>" fileName="<%=fileName%>" />
}

```

Extend or modify as required

Tiles controller supports downloads via filename

_templates/display_download_view.jsp

3 Define downloadable view tile:**Download Tile Template:**

```

<definition name=".template.customdownload"
  path="/_templates/display_download_view.jsp"
  controllerClass=
    "com.edocs.application.tbm.displayview.DisplayTiledView">
  <put name="viewName" value="DefaultView" />
  <put name="viewType" value="XSLT" />
  <put name="ddn" value="TBM" />
</definition>

```

If many tiles to be defined create a custom template

Download Example:

```

<definition name=".download.someView" extends=".template.customdownload">
  <put name="viewName" value="publishedXSLTViewName" />
  [ <put name="ddn" value="TBM" /> ]
</definition>

```

Set View Name
Set DDN if required

4 Use new tile in appropriate page

Writing and Configuring Post Processors

Post processors are used to transform data in some way after it has been returned from a data source. Views can be associated with one or more Post Processors.

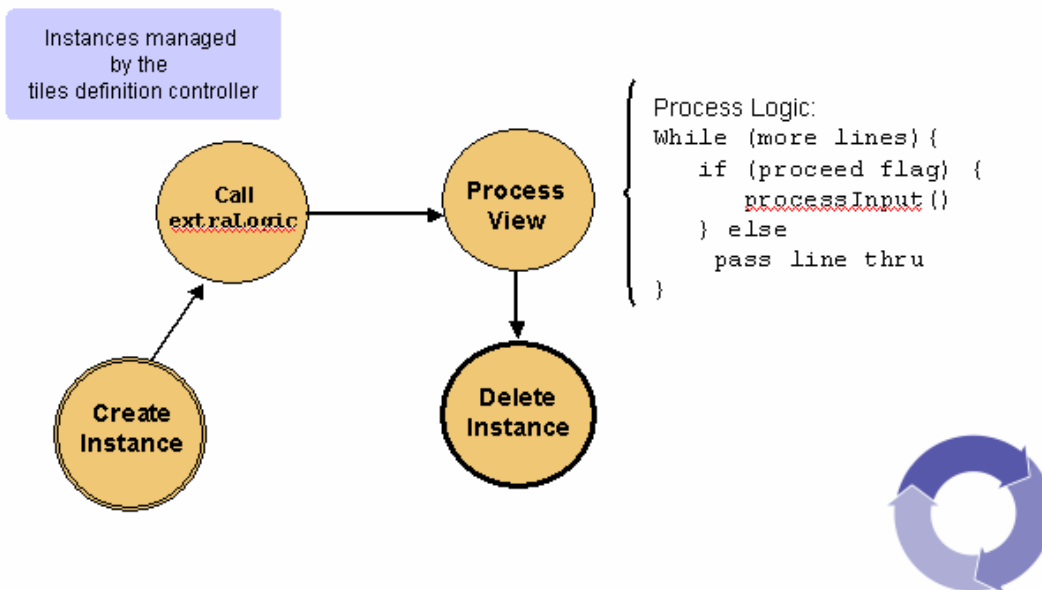
Post Processors:

- Are called after a view has been obtained from a data source
- Can transform data programmatically
- Can be parameterized for a specific view in a tiles definition
- Can be chained one after another

Examples of post processing include:

- Replacing text with a link, for example to insert personal address book data.
- Adding new text, for example replacing a phone number with a name.
- Summarizing data as it's presented (usage statistics).
- Logging information about data.

Post Processor lifestyle



How to Write a Post Processor

To write a post processor

- 1 Write a class that extends `PostProcessor` (writing two specific methods)
- 2 Register the postprocessor logical name in `app-config.properties`
- 3 Modify the tiles def which specifies the view to be post processed to reference the view logical name
- 4 Optionally modify the tiles def to supply additional arguments to the view

Each step is described here:

1. Extending PostProcessor

Post Processor Example:

```
import com.edocs.application.tbm.displayview.PostProcessor;
import com.edocs.application.tbm.core.common.ViewInfo;
import javax.servlet.http.HttpServletRequest;
public class MyPP extends PostProcessor {
    public MyPP() {}
    protected void handleExtraLogic(ViewInfo viewInfo,
                                    HttpServletRequest request,
                                    HashMap parameters)
                                    throws Exception {}
    protected String processInput(String inputLine)
                                    throws Exception {}
}
```

Required Imports

Extends

Required Methods

```
import com.edocs.application.tbm.displayview.PostProcessor;
import com.edocs.application.tbm.core.common.ViewInfo;
import javax.servlet.http.HttpServletRequest;
import org.apache.commons.logging.*;
public class MyPP extends PostProcessor{

    private static Log m_CLog = LogFactory.getLog(MyPP.class);
    //-- definitions for initial and final string to match. --
    private static final String cszBeginMatchString = "startpattern";
    private static final String cszBeginMatchTemplate = "xxxxxxxxxxx";
    private static final String cszEndMatchString = "end pattern";
    private static final String cszEndMatchTemplate = "xxxxxxxxxxx";

    // -- matching classes. --
    private OutputThroughMatch m_FindBeginMatch = null;
    private RetainThroughMatch m_RetainMatch = null;

    public MyPP() {
        m_FindBeginMatch = new OutputThroughMatch(
                                cszBeginMatchString,
                                cszBeginMatchTemplate );

        m_RetainMatch = new RetainThroughMatch(
                                cszEndMatchString,
                                cszEndMatchTemplate )
    }
    protected void handleExtraLogic(ViewInfo viewInfo,
                                    HttpServletRequest request,
                                    HashMap parameters)
                                    throws Exception {
```

```

        if (m_cLog.isDebugEnabled()) {
            String aParam = (String)parameters.get(new String("key"));
            String anotherParam = (String)parameters.get(
                new String("key2"));
        }
    }
    protected String processInput(String inputLine) throws Exception {
        if (m_cLog.isDebugEnabled())
            m_cLog.debug("MyPP -- entering with string \""+
                inputLine + "\"");

        String t_szRet = m_FindBeginMatch.process(inputLine);
        if (m_FindBeginMatch.isMatchComplete()) {
            // found the beginning, look for the end
            if (m_RetainMatch.isMatchComplete())
                setProceed(false);
        }
        return t_szRet;
    } // -- end of processInput
}

```

2. Register the Post Processor

To register the Post Processors:

- Register in app-config.properties
- Assign logical name referenced by tiles definitions

Registering a Post Processor:

In general:

```

PostProcessors=Name1[, name2...]
PostProc.Name1=fully qualified name

```

For example:

```

PostProcessors=MyPP
PostProc.MyPP=com.educs.edu.MyPostProcessor
%EDX_HOME%/thm/app-config.properties

```

3. Modify the Tiles Definition

View tiles specify post processors via the `postProcessor` tiles

Specifying a Post Processor:**In general:**

```
<definition name=".some.definition" extends=".view.setup">
  <put name="viewName" value="Name of View"/>
  ... other parameters
  <put name="postProcessor" value="MyPP"/>
</definition>
```

Must match name in
app-config.properties!

For example:

```
<definition name=".analysis.expensive.tx.chart" extends=".view.setup">
  <put name="viewName" value="Report_ExpensiveTxCH"/>
  <put name="viewType" value="HTML"/>
  <put name="ddn" value="Itemisation"/>
  <put name="subIdx" value="true"/>
  <put name="postProcessor" value="ReportChart"/>
</definition>
```

4. Provide Optional Parameters

Parameters as passed to post processors via a "postprocessname"Params tile. In the example below the "ReportChart" post processor retrieves its parameters from the "ReportCharParameters" tile. Parameters are specified as name=value pairs separated by the pipe(|) character.

Specifying a Post Processor:**In general:**

```
<definition name=".some.definition" extends=".view.setup">
  <put name="viewName" value="Name of View"/>
  ... other parameters
  <put name="postProcessor" value="{post processorName}"/>
  <put name="{post processorName}Params" value="key=value[|key2=value2...]"/>
</definition>
```

For example:

```
<definition name=".analysis.expensive.tx.chart" extends=".view.setup">
  ...
  <put name="postProcessor" value="ReportChart"/>
  <put name="ReportChartParams" value="start=MyMatch/startMask=XXXXXXX"/>
</definition>
```

{WEBAPPLICATION}/WEB-INF/tiles-*.xml

Pattern Processing

View Processors typically:

- Search for a starting and ending character sequence
- Extract the contents between the starting and ending sequence for use as a data retrieval key
- Replace the sequence

The `com.edocs.domain.telco.tagprocessing` package contains four classes for pattern processing: `OutputThroughMatch`, `ExactThroughMatch`, `RetailThroughMatch`, `DiscardUpToMatch`.

The heart of view post processing is pattern matching. Typically a post processor processes content by finding a starting and ending sequence, extracting the intermediate content and uses the content in some way. In Personal Address Book applications the intermediate content, normally a phone number, is used as a key into a table and the number itself replaced with a URL that displays address book content.

In support of pattern matching four classes are provided, all extending `ProcessThroughMatch`.

Classes of the package `com.edocs.domain.telco.tagprocessing` are:

- `OutputThroughMatch` - Returns everything up to and including the match string from the input
- `ExactThroughMatch` - Retains all input in a special internal buffer
- `RetailThroughMatch` - Retains the input up to the but not including the match
- `DiscardUpToMatch` - Discards all input to the match.

OutputThroughMatch

Return the output of a match-up through and including the match string.

OutputThoughtMatch:

```
public class OutputThroughMatch
    extends ProcessThroughMatch ... {

    OutputThougMatch(String match, String mask);
    String process(java.lang.String szInput);
    public boolean isMatchComplete();
    String getExtraData();
}
```

Create an instance

Search for a match

match found?

Return data after the match

Example

Match a give starting string including start:

```
import com.edocs.domain.telco.tagprocessing.*;
OutputThroughMatch start = new OutputThroughMatch("StartsWith+F*",
"XXXXXXXXXXXXXXXXX");

String result = start.process(inputLine);
if (start.isMatchComplete()) {
    String remaining start.getExtraData();
}
```

Match exactly the text

Match Character	Meaning
'a' or 'A'	Match the character in the provided match string or a <u>whitespace</u>
'w' or 'W'	Match <u>whitespace</u>
'x' or 'X'	Match exactly the character in the match string

Live Data Extraction

DDNs can be:

- Static - based on an a batch job for content
- Dynamic, or live - based on a data stream managed dynamically by a live data stream and the LDE Manager

Views using dynamic perform identically to traditional views, and can employ all the same mechanisms such as view post processing

Live data processing is managed via a custom data stream and the LDE manager. Developers write a custom action class, which interacts with the current session for customer information, and the view management system to access data and present it dynamically.

From the perspective of view presentment, live data streams behave identically to traditional data streams and can take advantage of post processors etc. For all practical purposes dynamic data is no different, to the view subsystem, then previously processed data.

About the LDE Manager

The LDE Manager manages dynamic view content using a least recently used algorithm. Data content returned by data sources is stored and then purged over time. Several configuration parameters control how data is managed.

The LDE Manager:

- Caches LDE Data streams for presentation, periodically expiring and clearing content
- Interacts with the view subsystem to provide dynamic content for presentation

- Is tunable based on configuration settings in `app-config.properties`

LDE Configuration settings:

`LDE_MAX_CACHE_SIZE=10485760` *default 10MB*
`LDE_EXPIRATION_PERIOD=600000` *default 10 minutes, in milliseconds*
`LDE_CLEANUP_PERIOD=300000` *default 5 minutes, in milliseconds*

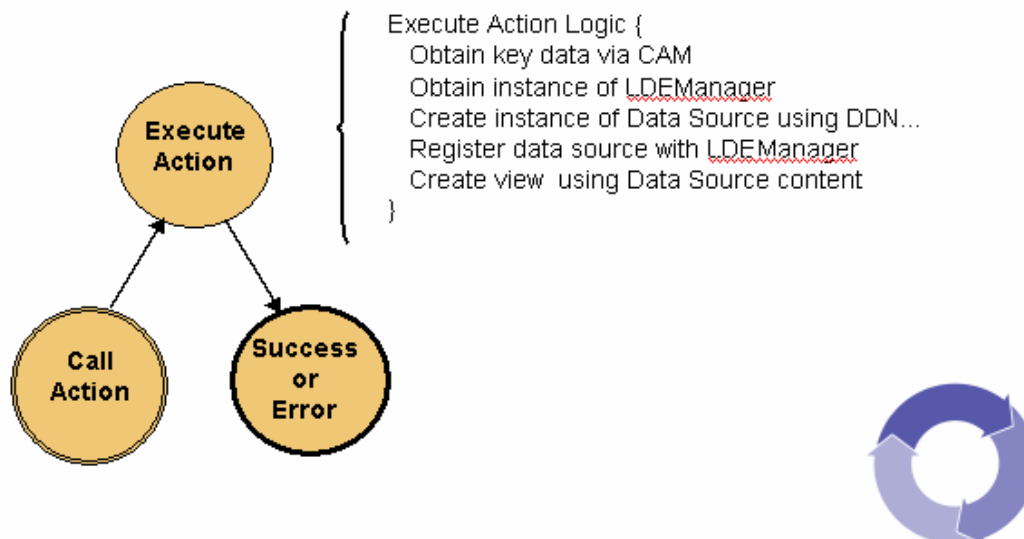
`%EDX_HOME%/tbn/app-config.properties`

The LDE Cache size represents the maximum size of view data that can be kept in memory and defaults to 10mb.

The LDE Cleanup period represents the time period used to sweep through the cache and deleted expired objects and defaults to every 5 minutes.

The LDE Expiration period represents the lifetime of a cached object and defaults to 10 minutes.

LDE Process



About LDE Data Streams

LDE Data Streams:

- Behave identically to traditional view data streams and are used by the LDE Manager to access run time data
- Must ultimately implement public interface `com.edocs.domain.telco.lde.ILDEStream`
- Often extend abstract class
`com.edocs.domain.telco.lde.ALDEStream` OR
`com. . .lde.AZipMemoryLDEStream`

ILDEStream Interface:

```

public interface ILDEStream {
    public void reset();
    public InputStream getContent();
    public long getTimestamp();
    public int getSize();
    public String getDDN();
}

```

Developing Live Data Extractors

There are five steps to writing Live Data Extractors:

- 1 Create a representative DDN for the data stream using the LDE data source
- 2 Write a class that extends `ILDEStream`, `ALDEStream` or `AZipMemoryLDEStream`
- 3 Write and configure an action class that registers the LDE Stream and forwards to view display page
- 4 Modify `edx.config.[bat|sh]` to support the LDE protocol
- 5 Define view definition, extending `.view.setup` defining the view parameters

Each step is described below.

1. Register the DDN

- 1 Login to the Siebel Command Center
For example `http://localhost:7001/edocs`
- 2 From the main window select Create New Application
- 3 Enter a suitable name
- 4 Enter the LDE data source
For example: `edx/{appname}/ejb/LdeDataSource`

2. Write the LDE Data Source Class

Writing LDE Data Classes:

- 1 Write a class that extends:
 - `ILDEStream`
 - `ALDEStream`
 - `AZipMemoryLDEStream`
- 2 Write a Constructor and store the DDN and optionally obtains configuration information.
- 3 Write a method, traditionally `getData`, which:
 - a Obtains a byte array of data
 - b Stores it using the inherited `internalSetContent` method

- Returns true or false indicating whether data is loaded.

LDE data source example

```
import java.io.ByteArrayOutputStream;
import com.edocs.domain.telco.lde.AZipMemoryLDEStream;
import com.edocs.application.tbm.config.Configuration;
import com.edocs.domain.telco.lde.LDEManager;
import java.io.*;

public class SimpleLDEDataSource extends AZipMemoryLDEStream {

    /** file to obtain content from */
    private String m_ContentFile = null;

    public SimpleLDEDataSource(String ddn) {
        super(ddn);
        loadConfig();
    }

    public boolean getData(String key) {
        byte buffer[] = null;
        //-----
        // Get a fileName, read the file and set the content
        // to the byte array
        // -----
        if (m_ContentFile != null && m_ContentFile.length() > 0) {
            File file = new File(m_ContentFile);
            int size = (int)file.length();
            buffer = new byte[size];
            try {
                InputStream fileIn = new FileInputStream(m_ContentFile);
                fileIn.read(buffer);
            } catch (IOException e) { e.printStackTrace(); }
        }

        // call setInternalContent if the byte array is not null
        if (buffer != null && buffer.length > 0) {
            internalSetContent(buffer);
        }
        else
            return false;
        return true;
    }

    private void loadConfig() {
        Configuration cConfig = Configuration.getConfiguration();
        if (cConfig != null) {
            String value = cConfig.getValue("simplelde.contentFile");
        }
    }
}
```

3. Write Action Class

Writing LDE Data Classes:

- a** Write a class that extends `EdocsAction`
- b** Implement `doAction` which typically:
 - Obtains an instance of an LDE Manager
 - Obtains the DDN and other key data required by the datasource
 - Creates an instance of the underlying data source
 - Registers the datasource with the LDE Manager
 - Creates and populate one or more views

LDE Action Class: Imports

```
import com.edocs.application.thm.core.common.*;

import com.edocs.application.thm.cam.api.*;

import com.edocs.application.thm.core.common.SessionUtils;

import com.edocs.application.thm.core.common.ViewInfo;

import com.edocs.domain.telco.lde.LDEManager;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.*;
```

Required Constants
And classes for
building edocs actions

User Information via CAM

Session

View Support

LDE Manager

Request & Response

Struts Support

LDE Action Class: `doAction` - Obtain DDN and sub key data

```
...

public class SimpleLDEAction extends EdocsAction {

    public ActionForward doAction(ActionMapping actionMapping,
                                   ActionForm actionForm,
                                   HttpServletRequest req,
                                   HttpServletResponse resp)
                                   throws RuntimeException {

        ICustomer t_cust = CAMClassFactory.getCustomer(req);
        ICustomerAccount t_acct = t_cust.getCurrentAccount();
        IAccountSummary t_sum = t_acct.getSummaryInfoDefaultDDN();
        // use t_sum for any other sub key data...
        IDDNMap ddnMap = t_cust.getDDNMap();
        String ddn = ddnMap.getDDN("Simple");

        ...
    }
}
```

LDE Action Class: `doAction` - Obtain instance of data source and LDE manager

```

. . .
// data source might take addn args

SimpleLDEDataSource ds = new SimpleLDEDataSource (ddn [,. . .]);

LDEManager ldeManager = LDEManager.getInstance();
if (ldeManager == null )
    return actionMapping.findForward(RIConstants.ERROR);

if (ds.getData() != true)
    return actionMapping.findForward(RIConstants.ERROR);

String docId = ldeManager.setContent(ds);
if (docId != null && docId.length() > 0) {
    // build view
}
. . .

```

LDE Action Class: `doAction` - build view

```

. . .
if (docId == null || docId.length() == 0)
    return actionMapping.findForward(RIConstants.ERROR);

ViewInfo view = new ViewInfo();
view.setDdn(ddn);
view.setDocId(docId);
view.setViewName("SimpleSummary"); // SimpleDetails
view.setViewType("HTML");
SessionUtils.setView(httpServletRequest, view);

return actionMapping.findForward(RIConstants.SUCCESS);
}
}

```

LDE action class example

```

import com.edocs.application.tbm.core.common.*;
import com.edocs.application.tbm.cam.api.*;
import com.edocs.application.tbm.core.common.ViewInfo;
import com.edocs.domain.telco.lde.LDEManager;
import org.apache.struts.action.*;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.commons.logging.*;

public class SimpleLDEAction extends EdocsAction {
    public ActionForward doAction(ActionMapping actionMapping,
                                ActionForm actionForm,
                                HttpServletRequest req,
                                HttpServletResponse resp)
                                throws RIException {

        try {

            ICustomer t_cust = CAMClassFactory.getCustomer(req);
            ICustomerAccount t_acct = t_cust.getCurrentAccount();
            IAccountSummary t_sum = t_acct.getSummaryInfoDefaultDDN();
            IDDNMap ddnMap = t_cust.getDDNMap();
            String ddn = ddnMap.getDDN("Simple");
            SimpleLDEDataSource ds = new SimpleLDEDataSource (ddn );
            LDEManager ldeManager = LDEManager.getInstance();
            if ( ldeManager == null )
                return actionMapping.findForward(RIConstants.ERROR);
            if (ds.getData() != true)
                return actionMapping.findForward(RIConstants.ERROR);
            String docId = ldeManager.setContent(ds);
            if (docId == null || t_szDocId.length()== 0) {
                return actionMapping.findForward(RIConstants.ERROR);
            }
            ViewInfo view = new ViewInfo();
            view.setDdn(szUnbilled_ddn);
            view.setDocId(t_szDocId);
            view.setViewName("SimpleSummary"); // SimpleDetails
            view.setViewType("HTML");
            SessionUtils.setView(req, view);

        } catch(RISessionNotFoundException re) {
            return
            actionMapping.findForward(RIConstants.ERROR);
        } catch(java.lang.Exception ex) {
            return actionMapping.findForward(RIConstants.ERROR);
        } // end try

        return actionMapping.findForward(RIConstants.SUCCESS);
    } // end doAction
}

```

4. Support the LDE Protocol

The LDE protocol is specified via the `java.protocol.handler.pkgs` property.

LDE Protocol support: `edx.config[.bat | .sh]`

By default the LDE protocol is loaded in TBM via a statement in `edx.config` similar to:

```
@set PROTOCOL_HANDERS=com.edocs.protocol
@set PROTOCOL_HANDERS="%PROTOCOL_HANDERS%|
com.edocs.domain.telco.lde.protocol"

@set JAVA_OPTIONS=%JAVA_OPTIONS% -Dedx.home="%EDX_HOME%"
-Djava.protocol.handler.pkgs=%PROTOCOL_HANDLERS%

@set CLASSPATH=%CLASSPATH%; {path}\LDEProtocol.jar
```

Pipe character

Line continued

Line continued

Protocol handler jar

5. Define View

Views which use LDE specified DDNs must specify the `skipLoadingData` tile with a value of `true`.

Example Tiles definition for a LDE based view:

```
<definition name=".my.ldebased.view" extends=".view.setup">
  <put name="viewName" value="viewName"/>
  <put name="viewType" value="HTML"/>
  <put name="ddn" value="associatedDDN"/>
  <put name="skipLoadingData" value="true"/>
</definition>
```

LDE DDN

Must be true

`{WEBAPPLICATION}/WEB-INF/tiles-*.xml`

5

Security: Enrollment, Authentication, Authorization

This chapter describes:

- Authentication concepts
- Role-based security concepts
- Implementing access controls
- Implementing custom security providers

About Role-Based Security

Security is based on:

- Authentication - The process where by a user is validated, typically using a uid/pwd combination
- Authorization - The process of determining whether an authenticated user can access a resource
- Access Rights - The definition who has the rights to access a specific resource

Billing Manager Provides:

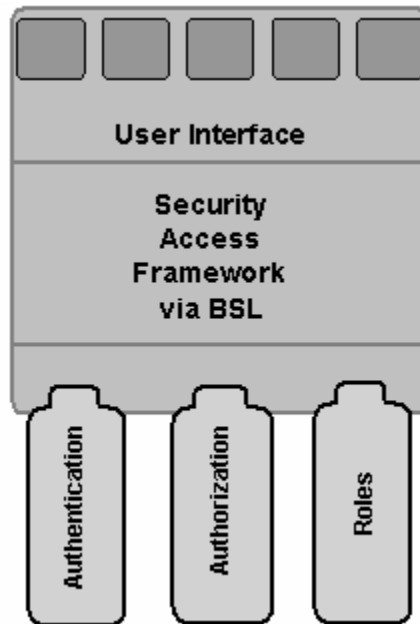
- Security Authorization framework
- Security SPIs

The Siebel Security Authorization Framework provides:

- Definitions of access rights via an XML or custom store
- Run time checking access

The Siebel BSL layer provides:

- `IAuthenticationManager` - Authenticate users
- `ISecurityProfileManager` - Create and otherwise manage user credentials & security questions
- `IUserProfileManager` - Manage userprofile data such name, address.



What is Security and Authorization Framework (SAF)?

The Security and Authorization Framework (SAF) provides the answer to the question:

- Given user, resource, and request context, should application grant or deny resource access

The SAF subsystem:

- Takes users, and resources and determines access rights based on a pluggable rules based authorization policy system
- Defaults to an XML based authorization scheme packaged with Billing Manager
- Uses the underlying security management framework to obtain user and role information

See the SAF javadoc delivered with this SDK for details about using and customizing the Security and Authorization Framework.

Authenticate

Billing Manager Authentication Subsystem:

- Includes authentication and impersonate support
- Return `ISubject` instances
- Defined by `IAuthenticationManager`

`ISubject` instances contain user* and role data for a given authenticated user

* Similar to Principals for those familiar with JAAS

Security Profile

Billing Manager Security Profile Subsystem:

- Manages `ISecurityProfile`
- Manages `ISecurityRole`
- Defined by `ISecurityProfileManager`

`ISecurityProfile` instances contain roles', secret questions and similar information for a given uid.

`ISecurityRole` instances contain role and role descriptions

User Profiles

Billing Manager User Profile Subsystem:

- Manages `IUserProfile`
- Defined by `IUserProfileManager`

`IUserProfile` instances contain name,address, email etc information for a given uid.

Security Management

Implementing Security Management involves:

- Implementing authorization
- Implementing Security Profiles including roles
- Implementing User Profiles*
- Linking common sub-systems

* User profiles are not strictly required but add value

Authorization

Implementing authorization involves:

- Creating and registering permissions on resources
- Authorizing at run time by:
 - 1 Obtaining `ISubject` instance from the BSL subsystem
 - 2 Obtaining permission instances for a given resource
 - 3 Testing access via the SAF Subsystem

Restricting URL Access

Billing Manager (Business Edition) uses a role-based URL restriction mechanism to restrict a user from accessing unauthorized pages based on assigned roles to the user.

Restricted URLs based on user assigned roles are defined in the url-access-control.cfg.xml file. This file is packaged in the ear-tbm-b2b.ear file (WEB-INF\classes folder in the ear-tbm-b2b.ear - > war-tbm-b2b.war file).

Definitions of restricted URLs for roles are in the “url-access-control.cfg.xml” file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

<beans>
<bean id="user" class="com.edocs.application.common.URLAccessManager" singleton="true">
  <property name="urlList">
    <list>
      <value>admin.*</value>
      <value>hierarchy/billing.*</value>
      <value>hierarchy/business.*</value>
      <value>addressbook/cab.*</value>
    </list>
  </property>
</bean>

<bean id="payer" class="com.edocs.application.common.URLAccessManager" singleton="true">
  <property name="urlList">
    <list>
      <value>admin.*</value>
      <value>hierarchy/billing.*</value>
      <value>hierarchy/business.*</value>
      <value>addressbook/cab.*</value>
    </list>
  </property>
</bean>
</beans>
```

Here, bean id is the **role** assigned to user and restricted URLs are defined in <property name="urlList"> under each bean.

For example, if you assigned a new role called “Biller” to a user and you want to restrict that user from viewing hierarchy and address book pages, then you would edit these lines from url-access-control.cfg.xml as shown here. You can define the URLs in a regular expression format. When you do that make sure it does not overlap with accessible URLs.

```
<bean id="Biller" class="com.edocs.application.common.URLAccessManager" singleton="true">
  <property name="urlList">
    <list>
      <value>hierarchy/business.*</value>
      <value>addressbook/cab.*</value>
    </list>
  </property>
</bean>
```

To specify roles-based page access restrictions in Billing Manager:

- 1 Extract the war-tbm-b2b.war file from the ear-tbm-b2b.ear file.
- 2 Extract the url-access-control.cfg.xml file to a folder.
- 3 Apply the changes to the file (example shown above). Verify the xml file by opening the file through IE browser. If you can see the full ml file without any error, xml file is valid.
- 4 Package the file in to WEB-INF\classes folder in war-tbm-b2b.war file.
- 5 Package the war-tbm-b2b.war file in to ear-tbm-b2b.ear file.

Restricting unauthorized pages from CSR application

You must follow the steps described above to customize the CSR application with the restricted URLs as well.

The restricted URLs based on user assigned roles are also defined in the url-access-control.cfg.xml file. This file is packaged in the ear-tbm-csr.ear file (WEB-INF\classes folder in the ear-tbm-csr.ear-> war-tbm-csr.war file).

Controlling Access to Resources

Using the SAF Sub-system

The SAF sub-system can be used with:

- Action classes to control access to resources
- In `EdocsBaseMenuItem` extended classes to control menu element visibility
- In any program element where a resource access should be protected

`RITilesMenuBean` can be used out of the box to provide security on Struts actions via SAF

SAF Components

SAF subsystem requires:

- Resource information - name etc of resource being accessed
- Roles - Defining sets of users
- `ISubject` - Instance of authorized subject attempting to access resource
- Permission object - Instance of a permission defined on the resource
- Policy files - Linking a given resource to a set of authorization rules

SAF Configuration

SAF uses `azcfg.properties`:

- to register role mappers
- to map groups to roles
- to define the name & location of the xml based policy

azcfg.properties snippet:

```
. . .
com.edocs.ps.security.azresolver=
    com.edocs.ps.security.authorization.providers.SimpleAZResolver
. . .
com.edocs.ps.security.rolemappers=
    com.edocs.ps.security.authorization.providers.SimpleRoleMapper,
    com.edocs.ps.security.authorization.providers.GroupRoleMapper
. . .
com.edocs.ps.security.rolemappers.secrole.ALL_USERS=Admin,CSR. . .
. . .
PolicyStoreData=azcfg/policy/azpolicy.xml
```

WEB-INF\classes\azcfg.properties

Role Mapping

SAF uses a pluggable role mapping engine to map groups to roles. Roles are typically defined in `azcfg.properties` and mapped from underlying roles to the roles defined in the SAF permissions files.

Roll Mapping Examples:

```
com.edocs.ps.security.rolemappers.secrole.ALL_USERS=Admin,CSR,SuperAdmin
com.edocs.ps.security.rolemappers.secrole.SUPER_ADMIN=SuperAdmin
com.edocs.ps.security.rolemappers.secrole.ADMIN_ONLY=Admin
com.edocs.ps.security.rolemappers.secrole.ALL_ADMIN=Admin,SuperAdmin
```

WEB-INF\classes\azcfg.properties

Policy Store

The SAF default policy store:

- Is defined in XML
- Contains permission elements defining a mapping between a right to access resources with a set of authorization rules

Permission elements contain:

- `<name>` - thing to protect</name>

- `<cpath>` - permission object `</cpath>`
- `<rule>` rules on who can access resources
 - `<name>` `</name>`
 - `<type>` `SecurityRole` `</type>`
 - `<value>` Comma separated list of security roles `</value>`

Permissions

- Are based on class implementations
- May imply one another to implement hierarchical rights

Policy Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE policystore PUBLIC .../>
<policystore>
  <permission>
    <name>SuperAdminSection</name>
    <cpath>com.edocs. . . ExtendsBasePermission</cpath>
    <rule>
      <name>all</name>
      <type>SecurityRole</type>
      <values>SUPER_ADMIN</values>
    </rule>
  </permission>
  ...
</policystore>
```

WEB-INF\classes\azcfg\policy\azpolicv.xml

Permission Example:

```
<permission>
  <name>special.txt</name>
  <cpath>com.MyPermission</cpath>

  <rule>
    <name>adm</name>
    <type>SecurityRole</type>
    <values>SPECIAL_ONLY
  </values>
  </rule>
</permission>
```

What are we protecting?

Who can access it?
Comma separated list

Roles come from the
underlying `ISecurityProfileManager`
implementation

Rights Engine

To check access rights via the access engine:

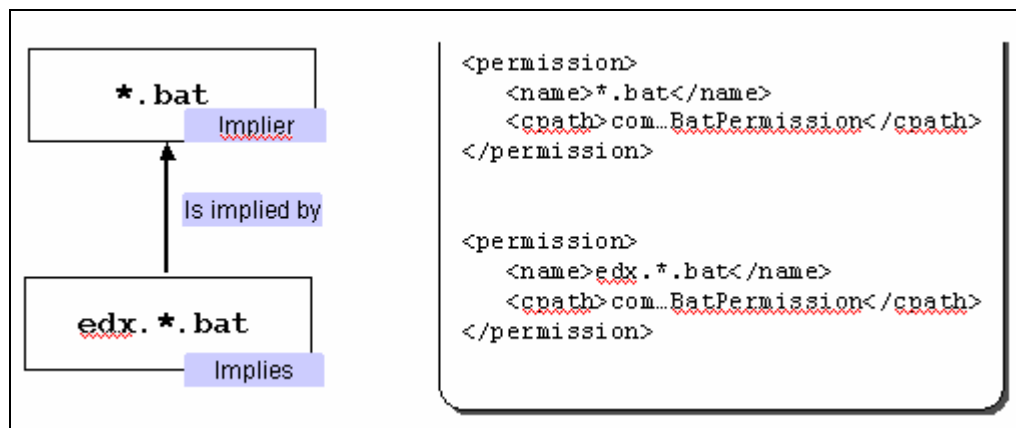
- 1 Obtains all rules defined for a permission request

- 2 Given a permission, get all mapped policy rules
If no mapped policy rules found
 - a Get implied permission
 - b Continue
- 3 For mapped policy rules check
 - a Does user have role granting access? Allow.
 - b Otherwise deny

Implied Policies

Policies can be explicit or imply one another.

Permission classes can parse resource name and return "implied" permissions.



Permission Classes

- Model resource types (for example, JSP, Text file, URL)
- Class instances instantiated with a particular name model resource instances
- Are extensions of `PermissionBase` which includes core methods such as `equals` and `hashCode`
- Can return "impliers"

Custom Impliers

Custom impliers are used when a permission may imply another.

edx.*.bat is implied by *.bat:

```
public final class BatPermission
{
    . . .
    public IPermission getImplier() throws AZException {
        IPermission perm = null;
        String name = this.getName();
        // If name does not match the pattern edx.*.bat
        // check against *.bat
        if (!name.startsWith("edx.*") && name.endsWith(".bat"))
            perm = new BatPermission("*.bat");
        }
        return perm;
    }
}
```

JSP Permission Example

This permission can be used with policies of the form:

```
<permission>
  <name>*.jsp</name>
  <cpath>JSPPermission</cpath>
  <rule>
    <name>all</name>
    <type>SecurityRole</type>
    <values>Everyone</values>
  </rule>
</permission>
<permission>
  <name>admin.jsp</name>
  <cpath>JSPPermission</cpath>
  <rule>
    <name>all</name>
    <type>SecurityRole</type>
    <values>ADMIN</values>
  </rule>
</permission>
```

To grant access to most JSPs but guard others.

```
import com.edocs.ps.security.authorization.AZException;
import com.edocs.ps.security.authorization.IPermission;
```

```

public final class JSPPermission
    extends PermissionBase
    implements IPermission {

    private static final String ANY_JSP = "*.jsp";
    private static final String JSP_EXT = ".jsp";

    public JSPPermission(String name) {
        super(name);
    }
    /**
     * Limited to matching * as 'all'.
     */
    public IPermission getImplier() throws AZException {
        IPermission perm = null;
        String name = this.getName();
        if (!name.equals(ANY_JSP) && name.endsWith(JSP_EXT)) {
            perm = new JSPPermission(ANY_JSP);
        }
        return perm;
    }
}

```

Using Programmatic Security

Steps for using programmatic security:

- 1** Write a permission that extends `BasePermission`. (See details, below.)
 - a** Implement `getImplier()` if required
- 2** Register the new permissions in the `azpolicy.xml`. (See details, below.)
- 3** Check policy where applicable:(e.g. in a action)
 - a** Obtain an `ISubject`
 - b** Create an instance of a permission representing the resource
 - c** Use SAF to check the access

1. Write permission

Permission to check access on edocs configuration files:

```
import com.edocs.ps.security.authorization.AZException;
import com.edocs.ps.security.authorization.IPermission;

public final class BatPermission extends PermissionBase implements IPermission {
    public BatPermission (String name) {
        super(name);
    }

    public IPermission getImplier() throws AZException {
        IPermission perm = null;
        String name = this.getName();
        if (!name.startsWith("edx.*") && name.endsWith(".bat")) {
            perm = new BatPermission ("*.bat");
        }
        return perm;
    }
}
```

2. Register permission

Permission Example:

```
<permission>
  <name>*.bat</name>
  <cpath>batPermission</cpath>
  <rule>
    <name>adm</name>
    <type>SecurityRole</type>
    <values>EveryoneRole</values>
  </rule>
</permission>
<permission>
  <name>edx.config.bat</name>
  <cpath>batPermission</cpath>
  <rule>
    <name>adm</name>
    <type>SecurityRole</type>
    <values>ADMIN</values>
  </rule>
</permission>
```

Obtain an ISubject

ISubject instances can be obtained from using SessionUtils.

Obtaining an ISubject within a Action Class:

```
import com.edocs.common.api.bsl.ISession;
import com.edocs.ps.security.authentication.ISubject;
import com.edocs.ps.security.authentication.SubjectUtils;

import javax.servlet.http.HttpServletRequest;

. . .

ISubject subject = ISession.getUserSubject (HttpServletRequest);
```

Check permissions method**Sample Check Permissions Method:**

```
import com.edocs.ps.security.authorization.*;

public boolean checkPermission(ISubject subject,
                             IPermission permission) {

    boolean permitted = false;
    try {
        AZContext ctx = new AZContext();
        permitted = AZEngine.checkAccess(perm, subject, ctx);
    } catch (AZException e) {
        log.error("Error checking permission.", e);
    }
    return permitted;
}
```

3. Check access**Sample Check Permissions Method:**

```
ISubject subject =
SessionUtils.getUserSubject(HttpServletRequest);

Permission perm = new BatPermission("edx.config.bat");

boolean allowed = checkPermission(subject, perm);

if (allowed)
    // you may change config settings
else
    // you may NOT change config settings
```

Integrating Security Providers

This section describes how to integrate security interfaces from Billing Manager's Business Services Layer (BSL):

- Working with `IAuthenticationManager`
- Working with `ISecurityProfile`
- Working with `ISecurityProfileManager`

BSL Security SPI's

The Billing Manager Business Services Layer provides: Service Provider Interfaces (SPI's) for the purposes of providing custom:

- Authorization
- Security and user profile management

■ Company and account management

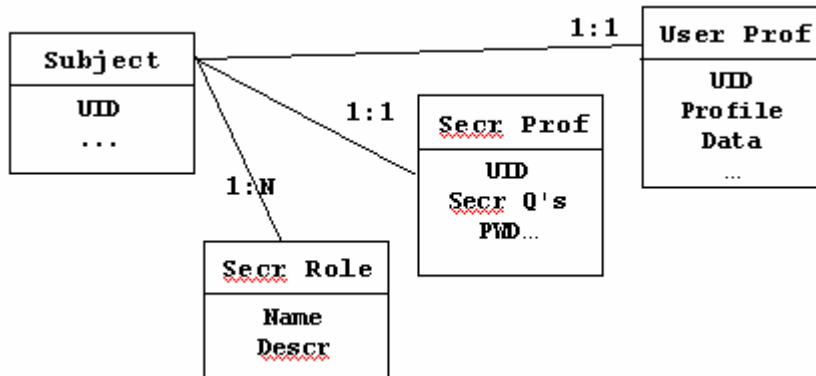
Authorization, Security and User profile management are considered Security management SPI's and are used to implement custom Single Sign-On etc.

See the BSL javadocs delivered with this SDK for implementation details about the BSL API and SPI.

BSL Security Concepts

BSL Security provides the instances of the classes required to answer the question: Given user, resource, and request context, should application grant or deny resource access

BSL is conceptually modeled as:



Subject:

- Modeled by ISubject
- Managed by IAuthenticationManager

Roles and Security Profiles:

- Modeled by ISecurityRole, ISecurityProfile
- Managed by IAuthenticationManager

User Profiles:

- Modeled by IUserProfile
- Managed by IUserProfileManager

Common Implementation Use Cases

Federate with existing authentication only:

- Only need implement IAuthenticationManager
- Cannot self-enroll, can manage security questions and user profile data
- User profile and security profile handled by default systems

Authentication and account management

- Implement `IAuthenticationManager`
- Implement `ISecurityProfileManager`
- Can self-enroll, new accounts populated back to system of record

Authentication, account management, and user profile

- Implement `IAuthenticationManager`, `ISecurityProfileManager` and `IUserProfileManager`
- Can self-enroll, new accounts populated back to system of record
- Can access pre-populated specialized User Profile data

Authentication

`IAuthenticationManager` interface implementations provide authorization services to BSL

`IAuthenticationManager` Methods:

```
public ISubject authenticate(String uid,  
                             String password);
```

Given a `UID/pwd`
Return an `ISubject` if
the combination is valid

```
public ISubject impersonate(String uid)
```

Assuming you are a valid
CSR, return a `ISubject`
reflecting the provided `UID`

`ISubject` implementations contain:

- Principal - "who" the subject is
- Principal roles - "what roles the subject belongs to"

Security Manager Profiles

`ISecurityProfileManager` interface implementations provide role and security profile services to BSL

`ISecurityProfile` `ISecurityProfileManager` Methods (partial list) :

```
public ISecurityProfile  
    createSecurityProfile(String uid, String password);
```

Create a
new "user"

```
public ISecurityProfile updateSecurityProfile  
    (ISecurityProfile profile)
```

Update a
"user"

```
public void removeSecurityProfile(ISecurityProfile profile)
```

Delete a
"user"

```
public ISecurityProfile[] getSecurityProfile(IFilter filter);
```

Get some
"users"

ISecurityRole ISecurityProfileManager Methods :

```
public ISecurityRole createRole(String name, String description);
```

Create a role

```
public ISecurityRole getRole(String name) ;
```

Find a role

```
ISecurityRole[] getAllRoles();
```

Get all roles

User Profiles

IUserProfileManager interface implementations provide user profile services to BSL:

IUserProfile IUserProfileManager **Methods (partial list):**

```
public IUserProfile createUserProfile(String uid,...);
```

Create a new user profile

```
public void removeUserProfile(IUserProfile);
```

Remove a user profile

```
public IUserProfile updateUserProfile(IUserProfile);
```

Update a user profile

```
public IUserProfile[] getUserProfile(IFilter filter)
```

Get some user profiles

Configuring BSL Providers

The BSL Subsystem: retrieves core interface instance information via a BSLConfiguration.properties. Each property represents a class replacement implementing a profile manager.

BSLConfiguration.properties:

```
bsl.AuthenticationManager=  
bsl.SecurityProfileManager=  
bsl.UserProfileManager=  
bsl.CompanyManager=  
...
```

```
{WEBAPPROOT}\WEB-INF\classes\BSLConfiguration.properties
```

6

Customer Service Representative (CSR) Access/Capabilities

Overview

The CSR application delivered with Billing Manager provides an interface to create and manage CSR administrators and organizations. Through this application a CSR can also impersonate a user. When a CSR enrolls in a CSR-enabled application, profiles are created in the database just as they are when regular users enroll. Depending on the CSR roles configured, a CSR can be limited to specific UI views and actions on behalf of another user. As with normal users, SAF authorizes access based on the permissions set for the CSR role.

CSR Access (Impersonate User)

When a CSR logs into the CSR application, he/she intends to either administer organizations, search for users or other CSR's or impersonate another user to provide support for that user. A CSR can see all the users he works with and click on an impersonate hyperlink for that user. When the impersonate hyperlink is clicked, `CSRAction.impersonate()` is executed:

```
public class CSRAction extends EdocsAction {

    private static Log log = LogFactory.getLog(CSRAction.class);

    public ActionForward impersonate(String uid,
                                    ActionMapping mapping,
                                    HttpServletRequest request,
                                    HttpServletResponse response) {

        if (log.isDebugEnabled()) {
            log.debug("Doing CSRAction: Impersonate");
        }

        ActionErrors errors = null;

        try {
            String urlString = "uid=" + uid
                               + "&pwd=" +
            SessionUtils.getUserSession(request).getUser().getSecurityProfile().
            getPassword()
                               + "&csr=" + SessionUtils.getUserId(request);
```

```

        if (log.isDebugEnabled()) {
            log.debug("Impersonate: urlString contains uid: " + uid + "
csr:" + SessionUtils.getUserId(request));
        }

        // DDN needs to be passed separately in the URL string. Else,
payment
        // initialization will fail (really!)
        String link = "/tbmb/impersonateuser.do" +
//CSRConfiguration.getConfig().getCustAppURL() +
            "?value(" + RConstants.CSR_TOKEN_ATTRIBUTE + ")="
            + RIEncryptUtil.encrypt(urlString)
            + "&ddn=" + CSRConfiguration.getConfig().getCustAppDdn();

        if (log.isDebugEnabled()) {
            log.debug("@impersonateCustomer: Set Link request attribute to
: " +
                link);
        }

        // Write a record to the Siebel log to indicate that the
        // user was impersonated
        Logger.log(new CSRActivityItem("TBM_CSR",
            "Impersonate",
            CSRConfiguration.getConfig().getCustAppDdn(),
            SessionUtils.getUserId(request), // CSR User Id.
            "",
            "",
            "STARTING",
            uid, // Log userid of user being impersonated.
            "",
            "",
            ""));

```

```

        // Use request.setAttribute() so that the parameters do not show up
        // in the URL.
        request.setAttribute("CSRImpersonationUrl", link);
        return mapping.findForward(RIConstants.SUCCESS);
    } catch (EncryptionException e) {
        log.error("EncryptionException: " + e.getMessage(), e);
        errors.add(ActionErrors.GLOBAL_ERROR, new
        ActionError(RIHierarchyMessages.ENCRYPTION_EXCEPTION));
    } catch (RIException e) {
        log.error("Exception with RI: " + e.getMessage(), e);
        errors.add(ActionErrors.GLOBAL_ERROR, new
        ActionError(RIHierarchyMessages.SYSTEM_RI_ERROR));
    }
    if (!errors.isEmpty())
        saveErrors(request, errors);

    return mapping.findForward(RIConstants.ERROR); }

```

The `CSRImpersonationUrl` request attribute above is configured in `app-config.properties`. to the `tbm-app` with the authenticated user's credentials is passed. Read the `tbm-csr-app` javadocs for more detailed information about the CSR application.

BSL and SAF provide the authentication classes needed to impersonate. Implement BSL's `IAuthenticationManager.impersonate(UID)` to authenticate both the CSR and the impersonated user (UID). Configure the CSR permissions in the SAF policy file and create permission objects that the SAF engine uses to authenticate and authorize user access.

To create tiered access for CSR's, configure the SAF policy file with permissions tied to specific roles, such as CSR SuperAdmin (all privileges), CSR Manager (most privileges), CSR (some privileges), CSRImpersonate (impersonation only). Create permission classes to filter access. Employ the `checkAccess` tile definition in UI JSPs to ask the SAF engine for access control by role (RBAC) of specific UI components.

For details about configuring SAF, see the Security chapter in this guide.

CSR Application

See the J2EE/tbm-csr-app javadoc for a description that demonstrates CSR access.

The CSR application WAR file contains the tiles (*.jsp) for the application. Under src/webroot are a variety of packages containing tiles that address key CSR view functions such as impersonating and finding a CSR's customer (*access-cust*), enrolling the CSR and searching for a customer's CSR (*manage-csr*), enrolling the customer (*manage-cust*), and searching and managing organizations (*manage-org*).

Under src/main/com/edocs/application/tbm/csr/ (compiled source) are action, form, and tag classes which comprise the model and controller of the CSR application. The common package contains a variety of CSR helper classes for login, enrollment, authentication, and configuration.

See the src/webroot/WEB-INF for the struts configuration beans and forwarding actions for this CSR application. The tiles configuration resides here also.

See the misc/app-config.properties file for how to configure access to the customer application from the CSR app, how to specify the customer application DDN that will be used during impersonation, and the list of CSR roles that are enabled.

```
#####
# Below are CSR Application properties
#####

# URL to use to access the customer application
tbm.csr.config.CUSTOMER_APPLICATION_URL=/tbm/impersonateuser.do

# Customer Application DDN that will be used during impersonation
tbm.csr.config.CUSTOMER_APPLICATION_DDN=TBM

# List of CSR Roles
tbm.csr.config.CSR_ROLE_LIST=Admin,CSR,SuperAdmin
```

7 Payment Cartridges

The default implementation delivered with Billing Manager provides a payment component which provides a rich API for accessing e-Payment services. Read the javadocs delivered for this component and customize the key classes in this implementation to suit your payment needs.

Look in `tbm-app/src/webroot/payment` to find default jsp tile pages which you can re-use and customize for your own payment UI.

Look in `tbm-app/src/webroot/WEB-INF` to find the payment tiles and struts definition files for the default implementation of Billing Manager. Examine these files to see how the UI is configured, what beans are available, and what payment actions or services are at your disposal.

Read the javadocs delivered for the Billing Manager payment component in `com.edocs.common.api.payment`. Implement the payment interfaces as needed to customize the payment service. Place new implementations in `com.edocs.payment`. Re-use or extend already provided default implementation classes in `com.edocs.payment` as needed to customize the payment service for your application. Services, cassettes, and enrollment classes always reside in `com.edocs.payment` sub-packages.

New or template payment cartridges may be implemented to support international debit cards or other payment vehicles. To start, familiarize yourself with e-Payment cartridges by reading the chapter, *Implementing a Custom Payment Cartridge*, in the *Payment Developer's Guide*. Here you learn how to create and integrate custom cartridges and how to configure these new cartridges through Command Center Payment Settings. *Implementing a Custom Payment Cartridge* references demonstration source code. See `cassette_demo.zip`, which is delivered with this SDK.

8

Downloading Views and Converting Advanced AFP to PDF

Downloading CSV, XML, and PDF Views

Billing Manager supports downloading of views in a variety of formats, including comma separated values (CSV), eXtensible Markup Language (XML), custom via eXtensible Style Sheets (XSLT) and Portable Document Format (PDF). The default implementation, tbm-b2b-app, provides several examples.

To Create a downloadable view

- 1 Create and publish view specific files defining the content of the view
- 2 Create view specific tiles definitions to render the view
- 3 Incorporate the view tiles into application as required.

Common Download View Setup

Before content can be rendered for download, various web application infrastructures are required specific to downloadable views. Billing Manager provides a tiles definition, similar to that shown below, that provides support for downloadable views. The download setup tile defines the various fields overridden by the developer when new downloads are being created. By default all downloads use the `display_download_view.jsp` specifies the view name and output file name required by the view. Note that the out-of-the-box download jsp also expect the name of a file for download, this file is normally obtained from a user form in an action class and set with code similar to

```
request.setAttribute("FILE_NAME", "myfile.csv");
```

Download set up tile from `tiles-defs-download.xml`

```
<definition name=".download.setup"
    path="/_templates/display_download_view.jsp"

    controllerClass="com.edocs.application.tbm.displayview.DisplayTiledView">
    <put name="viewName" value="Defaultview"/>
    <put name="viewType" value="XSLT"/>
    <put name="ddn" value="TBM"/>
    <put name="download" value="true"/>
</definition>
```

Download set up tile from `display_download_view.jsp`

```
<% include file="/_includes/taglibraries.jsp" %>
<tiles:useAttribute id="viewName" name="viewName"
classname="java.lang.String" />
<% String fileName=(String)request.getAttribute("FILE_NAME"); %>
<tbmtags:getView name="<%=viewName%>" fileName="<%=fileName%>"
download="true"/>
```

Views are then specified within a given Tiles definition file as shown below. Note that the view extends `.download.setup`, specifies the actual view name, view type and associated DDN using the `viewName`, `viewType` and `ddn` tiles. The specified tile would then be rendered in response to some struts action.

```
<definition name=".download.MyViewName.PDF" extends=".download.setup">
  <put name="viewName" value="MyViewNameWhichIsPDF"/>
  <put name="viewType" value="XSLT"/>
  <put name="ddn" value="TBM"/>
</definition>
```

Comma Separated Value (CSV) Downloadable views

Comma Separated Value downloadable content is produced by combining a DDF defining the universe of all downloadable fields and a TOK file defining the specific fields, their order, separator etc. that will be included in the download. The view is then rendered via a tiles definition and a struts action trigger.

Elements of a CSV Downloadable View are:

- Publishable components:
 - DDF defining input fields
 - TOK file defining comma separated outputs
- View tile definition, specified in an appropriate tiles definition, specifying a `viewType` of CSV.
- Action causing the tile rendering the view to be rendered

Token File Format:

Command separated downloads are defined by the fields listed in the `.TOK` file. There are three areas of interest in this file. The first is the delimiter value itself and is typically a `","` (comma). Second is the list of header fields, denoted by the `FIELDS` element with attribute `HEADER="TRUE"`. The `FIELDS` element is a list of the name of each header elements for the various columns being output. Content itself is generated by the set of fields listed in the `RECORDS` element. Each repeating group to be downloaded is then specified by a `RECORD` element specifying the name of the repeating group via the `NAME` attribute and the actual fields to be output using one or more `FIELD` elements, each of which specifying the individual fields via the `NAME` attribute.

Sample Token File

```
<DOCUMENT>
<DELIMITER VALUE=","/>
<TITLE VALUE=""/>
<FIELDS HEADER="TRUE">
  <FIELD NAME="AcctNum" TYPE="S" BLANKLINE="0"/>
  <FIELD NAME="CustName" TYPE="S" BLANKLINE="0"/>
  <FIELD NAME="StatementDate" TYPE="S" BLANKLINE="0"/>
</FIELDS>
<RECORDS HEADER="TRUE">
  <RECORD NAME="SummaryInfo" HEADER="TRUE" BLANKLINE="0">
    <FIELD NAME="SummaryInfoLab" TYPE="S" BLANKLINE="0"/>
    <FIELD NAME="SummaryInfoAmt" TYPE="S" BLANKLINE="0"/>
    <FIELD NAME="SummaryInfoCR" TYPE="S" BLANKLINE="0"/>
  </RECORD>
</RECORDS>
<RECORDGROUPS HEADER="TRUE"></RECORDGROUPS>
</DOCUMENT>
```

The token file, along with its associated DDF is then published with a given name and specified to be rendered for a given struts action as described in the prior section Common Download View Setup.

Note that CSV views are specified for legacy purposes, all new development should use the more flexible XSLT view type and define a style sheet transform specifying the comma-separated content.

eXtensible Markup Language (XML) views

Billing Manager supports the ability to create downloadable XML via XML Views. Generate XML data by specifying the fields that should be rendered in a DDF and then publishing the DDF as an XML view. The content is rendered using a view tile as shown below.

Download set up tile for XML

```
<definition name=".download.MyViewName.xml" extends=".download.setup">
  <put name="viewName" value="MyViewNameWhichIsXML"/>
  <put name="viewType" value="XML"/>
  <put name="ddn" value="TBM"/>
</definition>
```

Elements of a XML Downloadable View are:

- DDF specifying content, published as a XML view
- View tile definition, specified in an appropriate tiles definition, specifying a viewType of XML.
- Action causing the tile rendering the view to be rendered

eXtensible Stylesheet Language (XSLT) views

Billing Manager supports the transform of data via XSLT views. XSLT views generate XML data by specifying the fields that should be rendered in a DDF and then publishing the DDF as an XML view. The content is rendered using a view tile as shown below.

Billing Manager uses XML to read, write, and transform data using the universal standard of XSLT. Billing 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 Billing Manager

Download set up tile for xslt

```
<definition name=".download.MyViewName.xslt" extends=".download.setup">
  <put name="viewName" value="MyViewNameWhichIsXSLTBased"/>
  <put name="viewType" value="XSLT"/>
  <put name="ddn" value="TBM"/>
</definition>
```

Elements of a XML Downloadable View are:

- Publishable components:
 - DDF defining input fields
 - Stylesheet specifying transform as XSL file
- View tile definition, specified in an appropriate tiles definition, specifying a `viewType` of XSLT.
- Action causing the tile rendering the view to be rendered

Example Token File

This template processes the elements in the a table element specified within the DDF as **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.

```

<?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"/>
  <xsl:variable name="newline">
    <xsl:text>&#13;&#10;</xsl:text>
  </xsl:variable>
  <xsl:variable name="separator">
    <xsl:text>","</xsl:text>
  </xsl:variable>
  <!-- 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:value-of select="$separator"/>
    <!-- Insert the column data -->
    <!-- Trim any extra whitespace from the data value -->
    <xsl:value-of select="normalize-space(SummaryInfoLab)"/>
    <xsl:value-of select="$separator"/>
    <xsl:value-of select="normalize-space(SummaryInfoAmt)"/>
    <xsl:value-of select="$newline"/>
  </xsl:template>
</xsl:stylesheet>

```

Example Generated Content

```

ivn-1/po-0/bc-17152/pc-7/dd-20011214,1,PREVIOUS BALANCE,285.12
. . .
ivn-1/po-0/bc-17152/pc-7/dd-20011214,14,LATE FEE,1.61
ivn-1/po-0/bc-17152/pc-7/dd-20011214,15,TOTAL CURRENT AMOUNT,117.17

```

Portable Document Format (PDF_FO) views

Billing Manager supports the production of PDF format files on the fly via data via PDF_FO views. PDF_FO views produce files by using XSLT:FO objects to specify the placement of data on a page and DDF files to specify page content.

Please note that PDF_FO views are designed for the production of 2-3 page output and typically generate 1-2 pages per second when rendered. For generating longer output please speak to a Siebel Professional Services representative.

Download set up tile for PDF_FO

```
<definition name=".download.MyViewName.pfg" extends=".download.setup">
  <put name="viewName" value="MyViewNameWhichGeneratedPDF"/>
  <put name="viewType" value="PDF_FO"/>
  <put name="ddn" value="TBM"/>
</definition>
```

Elements of a PDF_FO Downloadable View are:

- Publishable components:
 - DDF specifying content,
 - FO based Stylesheet specifying resulting format, as .TXSL file
 - images.jar – jar file containing any referenced images
 - fonts.jar – jar file containing any referenced fonts
 - PMCONFIG.xml – file defining the characteristics of the fonts, sizes kerning etc.
- View tile definition, specified in an appropriate tiles definition, specifying a viewType of PDF_FO.
- Action causing the tile rendering the view to be rendered

Example XSLT Stylesheet using FO for PDF generation

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format"
  xmlns:fox="http://xml.apache.org/fop/extensions"
  xmlns:java="http://xml.apache.org/xalan/java"
  version="1.0" xml:space="preserve">
  <xsl:template match="/">
    <fo:root linefeed-treatment="preserve">
      <fo:layout-master-set>
        . . .
```

Mapping a DDF to XML

Mapping a DDF to XML is the first step in the process content as XML or for creating XSLT or PDF views.

About XML DTDs for Billing Manager

The Billing 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 Billing Manager—each DDF defines its own DTD. However, all Billing 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 that 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.
- Billing Manager does not prevent collisions among FIELD, TABLE, and GROUP names as Billing Manager constrains DDF item names to be unique within the DDF.

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

Further 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

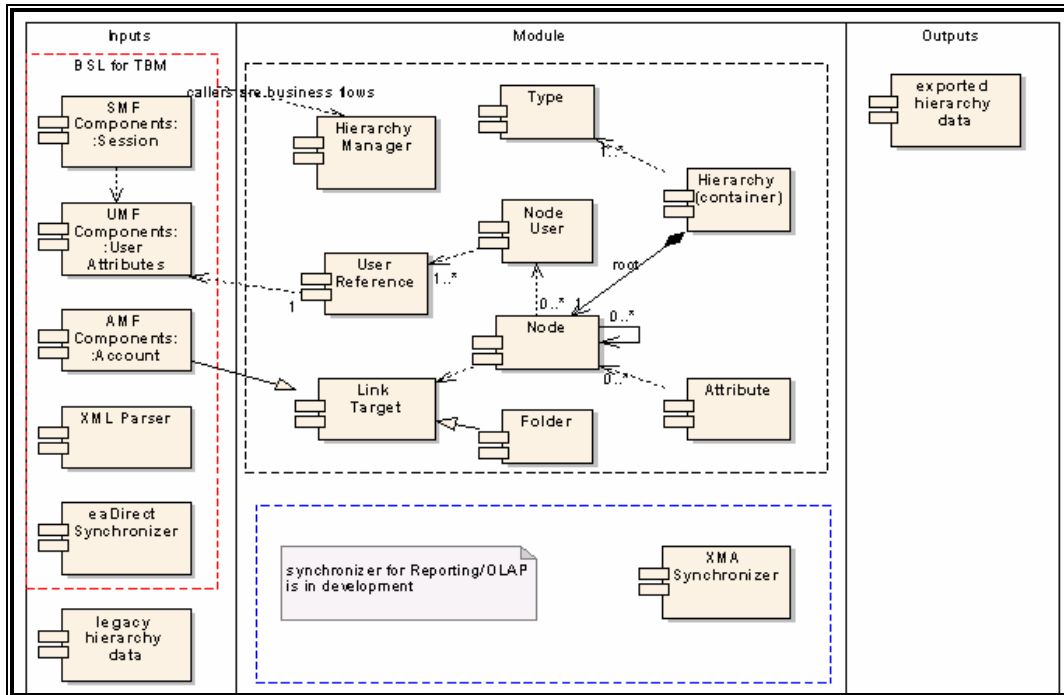
Holzner, Steven, *Inside XSLT*, New Riders, 201 West 103rd Street, Indiana 46290, July 2001

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/>

9 BSL Hierarchy Connector

Overview



Hierarchy - A system organized in the shape of a pyramid, with each row of objects, sometimes called nodes, linked to objects directly beneath it. A hierarchy contains a root directory at the top of the pyramid and subdirectories below it.

Node - A position in the hierarchy. A node can associate with a link target to indicate what the node represents. Users can be assigned to a node in the hierarchy to determine their access to that node and its subtree. Nodes are organized into parent-child relationships in the hierarchy. Nodes may have any number of name:value attributes.

Link target - A business object that can be associated to a hierarchy node.

URI - universal resource identifier to reference objects that exist in other modules. When a relationship is many-to-many, Hibernate requires that both sides of the relationship exist within the module domain. In order to fully use Hibernate for the hierarchy module, create a reference class within the module to do the many-to-many mapping between source object and reference object (proxy for original target object). In the Hierarchy module, UserReference class is a reference object for the IUser object from the UMF module.

Folder - Predefined link target object inside hierarchy module. Folder's life cycle is managed by its containing hierarchy.

Hierarchy User - A user with permission to view/modify (but not create) a hierarchy. For example CSR or any user in the system as long as they are given the permission.

Permission - A permission allows a user to view or take action on information they have access to, based on the node they are assigned to in the hierarchy. Permissions include view summary, view detail, pay, report, manage hierarchy, assign users, assign permissions, etc.

The Business Services Layer contains a connector package that provides import, synchronization, and hierarchy management for Billing Manager applications. BSL connector employs several sub-systems to create, modify, and synchronize hierarchical information from a variety of sources, including XML files and eStatement DDNs.

Use the BSL connector in concert with other BSL components, the Billing Manager CAM component, the Hierarchy module, Struts action classes, Tiles, and an optional third-party jsp tree tag library to display hierarchies of data in your Billing Manager application. See the J2EE/tbm-app directory for a reference implementation of hierarchies.

See the BSL javadoc delivered with this SDK for detailed implementations of hierarchy import and synchronization interfaces.

The BSL connector relies implicitly on a hierarchy module that is delivered in another Billing Manager jar. All basic hierarchy functionality is provided for by this hierarchy module, which should not need to be customized, as it provides basic hierarchy support: adding, deleting, editing, and searching at the node level. Also programmatic linking of users to nodes provides a means of hierarchy-based access control at this level. A key core class often used is `com.edocs.common.hierarchy.core.HierarchyManager`, which implements `IHierarchyManager` and employs several hierarchy module interfaces and classes to perform the low-level management of hierarchy elements.

See the hierarchy module javadoc delivered with this SDK for detailed descriptions of key hierarchy interfaces and default implementations.

How to Create and Manage Hierarchies via XML

To manage hierarchies, you can use the following DTD file to validate an XML file that specifies the create, update, and remove operations for hierarchy management:

```
<!ELEMENT HierarchyList (HierarchyToBeCreated?, HierarchyToBeUpdated?,
HierarchyToBeRemoved?)>
<!ELEMENT HierarchyToBeCreated (Hierarchy*)>
<!ELEMENT HierarchyToBeUpdated (UpdateHierarchy*)>
<!ELEMENT HierarchyToBeRemoved (RemoveHierarchy*)>
<!-- =====
*   Section for creating hierarchy
=====
-->
```

```

<!--ELEMENT Hierarchy (Name, Description, NewHierarchyNode)>
<!--ATTLIST Hierarchy
      id CDATA #REQUIRED
      companyId CDATA #REQUIRED
      type (billing | business | report) "business"
>
<!--ELEMENT NewHierarchyNode ((LinkTargetReference | SimpleNodeFolder), Alias?,
CanBeAccessedBy?, ChildNodeList?)>
<!--ELEMENT ChildNodeList (NewHierarchyNode*)>
<!--ELEMENT SimpleNodeFolder (Name?, Description, Attribute*)>
<!--ATTLIST SimpleNodeFolder
      id CDATA #REQUIRED
>
<!--ELEMENT Attribute EMPTY>
<!--ATTLIST Attribute
      name CDATA #REQUIRED
      value CDATA #REQUIRED
>
<!--ELEMENT LinkTargetReference (Name?, Description?)>
<!--ATTLIST LinkTargetReference
      id CDATA #REQUIRED
      type (BillingAccount | BusinessUnit | CostCenter) "BillingAccount"
      action (Link | Create | Override) "Link"
>
<!-- Use this element to describe the list of the users who have granted access to
the node and all nodes below in the hierarchy. -->
<!--ELEMENT CanBeAccessedBy (UserExternalReference*)>
<!--ELEMENT UserExternalReference EMPTY>
<!--ATTLIST UserExternalReference
      id CDATA #REQUIRED
>
<!--ELEMENT Alias (#PCDATA)>
<!--ELEMENT Description (#PCDATA)>
<!--ELEMENT Name (#PCDATA)>
<!--ATTLIST HierarchyID
      value CDATA #REQUIRED
>

```

```

<!-- =====
*   Section for updating hierarchy
=====
-->
<!ELEMENT UpdateHierarchy (NodesToBeCreated?, NodesToBeUpdated?,
NodesToBeRemoved?)>
<!ATTLIST UpdateHierarchy
    id CDATA #REQUIRED
    companyId CDATA #REQUIRED
>
<!ELEMENT NodesToBeCreated (NewNode*)>
<!ELEMENT NewNode (ParentNode, NewHierarchyNode)>
<!ELEMENT NodesToBeUpdated (UpdateNode*)>
<!ELEMENT UpdateNode (ParentNode, ChildNode)>
<!ELEMENT ParentNode EMPTY>
<!ATTLIST ParentNode
    id CDATA #REQUIRED
    type (BillingAccount | BusinessUnit | CostCenter | Folder)
"BillingAccount"
>
<!ELEMENT ChildNode EMPTY>
<!ATTLIST ChildNode
    id CDATA #REQUIRED
    type (BillingAccount | BusinessUnit | CostCenter | Folder)
"BillingAccount"
>
<!ELEMENT NodesToBeRemoved (NodeToDelete*)>
<!ELEMENT NodeToDelete EMPTY>
<!ATTLIST NodeToDelete
    id CDATA #REQUIRED
    type (BillingAccount | BusinessUnit | CostCenter | Folder)
"BillingAccount"
>
<!-- =====
*   Section for removing hierarchy
=====
-->
<!ELEMENT RemoveHierarchy EMPTY>
<!ATTLIST RemoveHierarchy
    id CDATA #REQUIRED
    companyId CDATA #REQUIRED
>

```

Example Hierarchy XML File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HierarchyList SYSTEM "eDocs_Hierarchy_Interchange1.0.dtd">
<HierarchyList>
  <HierarchyToBeCreated>
    <Hierarchy id="100" companyId="COMP-0001" type="billing">
      <Name>eDocs Billing</Name>
      <Description>This is a sample hierarchy</Description>
      <NewHierarchyNode>
        <SimpleNodeFolder id="20001">
          <Name>HR Accounts</Name>
          <Description>This folder list all accountfor HR Department
          </Description>
          <Attribute name="Address " value="1 Appl Hill, Natick, MA"/>
          <Attribute name="Phone " value="508-652-8700"/>
        </SimpleNodeFolder>
        <CanBeAccessedBy>
          <UserExternalReference id="user1"/>
          <UserExternalReference id="user2"/>
        </CanBeAccessedBy>
        <ChildNodeList>
          <NewHierarchyNode>
            <LinkTargetReference id="acct20001" type="BillingAccount">
              <Name>Account #2001 </Name>
              <Description>This is the account 20001</Description>
            </LinkTargetReference>
          </NewHierarchyNode>
          <NewHierarchyNode>
            <LinkTargetReference id="acct20002" type="BillingAccount">
              <Description>This is the account 20002</Description>
            </LinkTargetReference>
          </NewHierarchyNode>
        </ChildNodeList>
      </NewHierarchyNode>
    </Hierarchy>
    <Hierarchy id="200" companyId="COMP-0002" type="billing">
      <Name>ePays Billing</Name>
      <Description>This is a sample billing hierarchy</Description>
      <NewHierarchyNode>
        <LinkTargetReference id="c30001" type="CostCenter"/>
        <CanBeAccessedBy>
          <UserExternalReference id="user10"/>
          <UserExternalReference id="user20"/>
        </CanBeAccessedBy>
      </NewHierarchyNode>
    </Hierarchy>
  </HierarchyToBeCreated>
</HierarchyList>
```

```

<ChildNodeList>
  <NewHierarchyNode>
    <SimpleNodeFolder id="40001">
      <Name>PS Team</Name>
      <Description>This folder list all accounts for PS Department
      </Description>
      <Attribute name="Company Address "
        value="2 Apple Hill, Natick, MA"/>
      <Attribute name="Company Phone " value="508-652-8700"/>
    </SimpleNodeFolder>
    <Alias>Text</Alias>
    <CanBeAccessedBy>
      <UserExternalReference id="user11"/>
      <UserExternalReference id="user22"/>
    </CanBeAccessedBy>
    <ChildNodeList>
      <NewHierarchyNode>
        <LinkTargetReference id="acct30003"
          type="BillingAccount"/>
      </NewHierarchyNode>
      <NewHierarchyNode>
        <LinkTargetReference id="acct30004"
          type="BillingAccount"/>
      </NewHierarchyNode>
    </ChildNodeList>
  </NewHierarchyNode>
  <NewHierarchyNode>
    <LinkTargetReference id="acct30001" type="BillingAccount"/>
  </NewHierarchyNode>
  <NewHierarchyNode>
    <LinkTargetReference id="acct30002" type="BillingAccount"/>
  </NewHierarchyNode>
</ChildNodeList>
</NewHierarchyNode>
</Hierarchy>
</HierarchyToBeCreated>
<HierarchyToBeUpdated>
  <UpdateHierarchy id="200" companyId="COMP0002">
    <NodesToBeCreated>
      <NewNode>
        <ParentNode id="40001" type="Folder"/>
        <NewHierarchyNode>
          <LinkTargetReference id="acct30005" type="BillingAccount"/>
          <Alias>Text</Alias>
        </NewHierarchyNode>
      </NewNode>
    </NodesToBeCreated>
    <NodesToBeUpdated>
      <UpdateNode>
        <ParentNode id="c30003" type="CostCenter"/>
        <ChildNode id="acct30001" type="BillingAccount"/>
      </UpdateNode>
    </NodesToBeUpdated>
  </UpdateHierarchy>
</HierarchyToBeUpdated>

```

```

        </UpdateNode>
    </NodesToBeUpdated>
    <NodesToBeRemoved>
        <NodeToDelete id="acct30001" type="BillingAccount"/>
    </NodesToBeRemoved>
</UpdateHierarchy>
</HierarchyToBeUpdated>
<HierarchyToBeRemoved>
    <RemoveHierarchy id="100" companyId="COMP-0001"/>
    <RemoveHierarchy id="200" companyId="COMP-0002"/>
</HierarchyToBeRemoved>
</HierarchyList>

```

To manage the hierarchies specified in the above XML, use or extend `connector/tasks/HierarchyImporter.java` and register it in a job in the Command Center. See the chapter, *Custom Jobs*, for details on how to do this.

HierarchyImporter loads hierarchies from XML files. HierarchyImporter calls `com.edocs.common.bsl.connector.HierarchyXMLParser` to read the file. See the BSL connector/tasks javadoc for implementation details.

Synchronizing Hierarchies with eStatement Indexer Data

Another way to create hierarchies for the Billing Manager application is to obtain the information from the eStatement Indexer database table using a Command Center job. Use or extend `connector/tasks/BillingDataSynchronizer.java` and register it in a job in the Command Center.

BillingDataSynchronizer synchronizes pre-existing hierarchies with eStatement indexer data. BillingDataSynchronizer calls `BillingHierarchyLoader(aBillingHierSyncHandler)` to perform the synchronization.

The classpath resource, `hierarchy.cfg.xml`, is used to configure the hierarchy synchronization handler bean. See the default example delivered with Billing Manager in `bsl/config`:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

```

```
<beans>
  <bean id="billingHierarchyHandler"
    class="com.edocs.common.bsl.connector.BillingHiersynchHandler"
    singleton="false">
    <property name="serviceBuilder">
      <ref local="bslServiceBuilder"/>
    </property>
    <property name="billingHierarchyName">
      <value>BILLING</value>
    </property>
    <property name="defaultCompanyName">
      <value>__default__</value>
    </property>
  </bean>
</beans>
```

```

    <property name="companyColumnName">
      <value>CustName</value>
    </property>
  </bean>
  <bean id="bslServiceBuilder"
    class="com.edocs.common.bsl.connector.BusinessServiceBuilder"
    singleton="true">
    <property name="userId">
      <value>admin</value>
    </property>
    <property name="userPassword">
      <value>edocs</value>
    </property>
  </bean>
  <bean id="billingHierarchyBuilderProxy"
    class="org.springframework.aop.framework.ProxyFactoryBean">
    <property name="target">
      <ref local="billingHierarchyHandler"/>
    </property>
    <!-- <property name="interceptorNames">
      <value>afterReturningInterceptor</value></property> -->
    <property name="interceptorNames">
      <value>billingSyncAdvisor</value>
    </property>
  </bean>
  <bean id="billingSyncAdvisor"
    class="org.springframework.aop.support.DefaultPointcutAdvisor">
    <property name="pointcut">
      <bean
class="com.edocs.common.hierarchy.connector.BillingHierSyncPointcut">
      </property>
      <property name="advice">
        <ref local="afterReturningInterceptor"/>
      </property>
    </bean>
    <!-- Replace class name with your own interceptor class name -->
    <bean id="afterReturningInterceptor"
class="com.edocs.common.bsl.connector.tasks.AccountStatementInterceptor"/>
  </beans>

```

The synchronizer may not pick up all the indexed data that is available from eStatement by default. To extract more data or do additional processing after the synchronization handler processes each row in the indexer table, create an interceptor and register it in `hierarchy.cfg.xml`.

In BSL's connector/tasks package, *AccountStatementInterceptor* implements `org.springframework.aop.AfterReturningAdvice`. But you can also create an interceptor by extending `com.edocs.common.hierarchy.api.BillingHierAfterReturningInterceptor` and overriding `afterReturningHook()`:

```
/**
```

```

* Override this method to do any additional processing.
* Called every time a row in the indexer table is processed.
* Each row represents either a parent account or a nested
* sub-account or Mobile Telephone Number (MTN). Uses Z-Context
* field to determine how to sort rows into unresolved and accounts
* hashtables.
*
* @param handler handler object that synchronizes billing hierarchy with
* indexed billing information.
* @param zPrimary Z_PRIMARY field value in the current row
* @param zContext Z_CONTEXT field value in the current row
* @param rowData all other field vlaues in the current row.
* Each field is indexed by the column name.
*/

protected void afterReturningHook(IBillingHierSyncHandler handler, String
zPrimary, String zContext, HashMap rowData) throws Exception {
    String statementDate = (String)rowData.get("StatementDate");

    String[] amount = {(String)rowData.get("TotalAmtDueAmt"),
    (String)rowData.get("MTNCurrChgs"), (String)rowData.get("MTNCurrChgsCR")};

    log.debug("AccountStatementInterceptor.afterReturningHook(): Z_PRIMARY
= " + zPrimary + "; Z_CONTEXT=" + zContext + "; StatementDate=" + statementDate
+
        "; TotalAmtDueAmt=" + amount[0] + "; TotalMtnAmt="+amount[1] +
        "; TotalMtnAmt="+amount[2]);

    if (zContext==null || zContext.trim().length()<1) {
        // Top-level Account
        _store(zPrimary, statementDate, _resolveAmount(amount));

    } else {
        // Nested Account or MTN
        if (statementDate!=null) {
            _store(zPrimary, statementDate, _resolveAmount(amount));

```

```

    } else {
        BillingStatement statement =
        (BillingStatement)accounts.get(zContext);
        if (statement==null) {
            // Add MTN to the unresolved pile
            Vector unMtns = (Vector)unresolved.get(zContext);
            if (unMtns==null) {
                unMtns = new Vector();
                unresolved.put(zContext, unMtns);
            }

            // add to unresolved list
            String[] data = {zPrimary, _resolveAmount(amount)};
            unMtns.add(data);

        } else {
            // Parent Account found
            _store(zPrimary, statement.getStatementDate(),
            _resolveAmount(amount));
        }
    }
}
}
}

```

`AfterReturningHook()` is called after each row in the indexer table is processed by the synchronizer. The `rowData` `HashMap` contains all the values in the record, keyed by the column name. You can also get table name, `ivn`, and `ddn` through the handler class which is passed in as the first argument.

Be sure to add the interceptor class to the bean definition in *hierarchy.cfg.xml* as follows:

```

<bean id="afterReturningInterceptor"
class="com.edocs.common.bsl.connector.tasks.AccountStatementInterceptor"/>

```

Place *hierarchy.cfg.xml* anywhere in the deployment classpath. In order to compile the interceptor class, the following jars should be in our CLASSPATH:

- Aopalliance-1.0.jar
- Spring-1.0.2.jar

Hierarchy APIs

See the BSL connector javadocs for implementation details about high-level hierarchy management. Refer to the hierarchy module javadocs for implementation details about low-level hierarchy structures.

Customization of low-level hierarchy management may never be required, as most of the work needed to manage hierarchies at this level is fully implemented as follows. The package to look for in the hierarchy module jar is `com.edocs.common.hierarchy.api`.

Creating Hierarchies Programmatically

To create hierarchy a new hierarchy

```

IHierarchyManager hMgr = new HierarchyManager()
String companyId = new String("COMP-1");
String hierName = new String ("BILLING");
IHierarchyType hierType = Hierarchy.BUSINESS;
IHierarchy hier = hMgr.createHierarchy(companyId, hierName, hierType);
....

```

To create hierarchy from an existing hierarchy

```

IHierarchyNode rootNode = hier.getRoot();
IHierarchy anotherHier = createHierarchyFromNode(root);

```

Adding Entities to a Hierarchy

Any Business Objects that implement `IHierarchyLinkTarget` can be added into Hierarchy:

```

public interface IBusinessObject {
    public String getExternalID();
    public String getIdentityURI();
    public String getDisplayName();
    public String getTypeCode();
}

public interface IHierarchyLinkTarget extends IBusinessObject
    boolean isContainer();

```

To add a link target to Hierarchy

```

IHierarchyLinkTarget linkAcct = bs.findAccount("acc100");
IHierarchyNode acctNode = hMgr.addLinkTarget(rootNode ,linkAcct);

```

To add a folder to Hierarchy:

```

IHierarchyFolder hFolder = new HierarchyFolder("HR", "Human Resource",
                                                "This is HR folder");
hFolder.addAttribute(new Attribute("Phone: ", "508-123-8700"));
IHierarchyNode fNode = hMgr.addFolder hMgr.addFolder(rootNode, hFolder);
}

```

To add a node to Hierarchy:

```

hMgr.addNode(fNode, acctNode);

```

Finding Hierarchies, Nodes, and Folders

Developers decide how much data to load: Lazy (metadata only) or Bulk Loads.

To load Hierarchy MetaData (container) only

```

getAllHierarchyMetaData()
getAllHierarchyMetaData(hierarchyType);
getAllHierarchyMetaDataForUser(userName);
getAllHierarchyMetaDataForUser(hierarchyType, userName);

```

To load Hierarchy including its tree structure

```

getHierarchy("COMP-1", "BILLING", true);
Returns the whole tree
getHierarchyForUser("COMP-1", "BILLING", "jwang", true);
Returns a list of root nodes that the given user has access to

```

To load Hierarchy meta data and non-leaf node only

```

getHierarchyStructure("COMP-2", "BUSINESS")

```

To locate a node in Hierarchy:

```

IHierarchyNode findNodeByIdentityURI(IHierarchy hier, String uri)

```

```

IHierarchyNode[] findObjNodeByIdentityId(String identityId)
IHierarchyNode[] findObjNodeByIdentityId(IHierarchy hier, String identityId)
IHierarchyNode[] findObjNodeByIdentityId(IHierarchyNode hierNode, String identityId)

IHierarchyNode[] findObjectsNodeByURI(String uri)
IHierarchyNode findObjNodeByURI(IHierarchy hier, String uri)
IHierarchyNode findObjNodeByURI(IHierarchyNode hierNode, String uri)

```

To locate a folder in Hierarchy:

```

■ IHierarchyNode [] findFolderNode(IHierarchy hier, String name)
■ IHierarchyNode[] findFolderNode(IHierarchyNode node, String name)

```

Updating Nodes

To update a node:

```

updateFolder(IHierarchy hierarchy, IHierarchyFolder folder);
updateNode(IHierarchyNode hNode);

```

To move node to a different parent:

```

moveNodeTo(IHierarchyNode node, IHierarchyNode destinationNode)

```

To delete a node:

```

deleteNode(IHierarchyNode nodeToBeDeleted)

```

Giving User Access to Nodes

To give user access to a node:

```

addUserAccess(IHierarchyNode hNode, String userId)
addUserAccess(IHierarchy hier, IHierarchyLinkTarget target,
               String userId)

```

To remove user access from a node:

```
removeUserAccess(IHierarchyNode hNode, String userId)

removeUserAccess(IHierarchy hier, IHierarchyLinkTarget target,
                  String userId)
```

Permission types come from user object.

How to Implement Hierarchy-Based Access Control (HBAC)

HBAC provides another level of security for your Billing Manager application. When a user enrolls, a security profile registers one or more roles for the user. The SAF sub-system tests the logged-in user's permission to access key parts of the application. When a user is assigned to specific nodes in a business structure hierarchy, the access of the user is further limited via HBAC.

To implement HBAC, you can use the *CanBeAccessedBy* element in an XML file to add user access to specific nodes in the hierarchy as follows:

```
...
<CanBeAccessedBy>
|           <UserExternalReference id="user1"/>
|           <UserExternalReference id="user2"/>
</CanBeAccessedBy>
...
```

You can do this programmatically by extending the BSL core/BusinessServices class or re-implementing bsl/api/IBusinessServices methods to assign users to specific nodes according to your needs. You might constrain hierarchy access by role as well. Or the logged-in user's account list might be filtered through CAM according to account-level pattern matching. The flexibility provided by Billing Manager makes the final solution entirely up to you.

See the CAM and BSL javadocs for default implementations available.

How to Search for and Find Objects Within Hierarchies

Implement BSL's api/IBusinessServices.java or use BSL's core/BusinessServices class, which implements IBusinessServices, to search existing hierarchies for pertinent information.

Using IBusinessServices methods you can locate and return whole hierarchies or just the metadata for hierarchies. You can also search for objects within hierarchies. There is a method for every search that is based on a specific filter parameter.

See the BSL/api javadoc for implementation details.

Tuning Performance for Large Hierarchies

If a service provider's customer has tens of thousands of accounts or service agreements, there are a number of screens that will either time out or have an unacceptable latency. More importantly, even if the screens did load they are not designed to allow a user to navigate large quantities of hierarchy data (accounts, service agreements). Service Manager offers several parameters to tune performance in such an environment. Generally, there are several thresholds which limit result sets in specific situations. If the threshold is exceeded, the system displays a message informing the user. Search functionality is provided to find a specific result set, and the user can set their hierarchy position to get results for a specific hierarchy branch.

Retrieval Threshold

The retrieval threshold sets the maximum number of records which can be returned in a backend query. The retrieval threshold applies to screens throughout Billing Manager. Adjust the retrieval threshold by changing the value for the `hibernate.querythreshold` parameter in the `hibernate.properties` and `hierarchy.hibernate.properties` files in `$TSM_HOME/config`.

The retrieval threshold has the largest impact of any of the thresholds in tuning performance. The smaller the number, the quicker the system will perform in a large hierarchy environment. However, there is some inconvenience to the user when he/she experiences an over-threshold condition, because of the necessity to set position or search for the needed result. So the developer needs to balance the performance gain with the productivity loss to find the appropriate value for this threshold.

Display Tree Threshold

The display tree threshold sets the maximum number of records per hierarchy level which will display in a hierarchy tree presentation. This threshold applies to the Service Manager Overview page as well as the Service Manager Hierarchy pages. Adjust the display tree threshold by changing the value for the `displayTreeThreshold` parameter in the `app-config.properties` file in `$TSM_HOME/config`.

The display tree threshold applies only to tree displays, which are relatively expensive operations. Note that the threshold applies per hierarchy level, so that, for example, in a business structure which has many folders, each with many subfolders, where the display tree threshold is set to 100, the system would display 100 folders at hierarchy level 1, 100 folders at hierarchy level 2, and so on. Also note that this threshold would only be applied if the value is set to a value lower than the value for the retrieval threshold: if the backend query returns a value limited by the retrieval threshold, and that value is already under the value for the display tree threshold, then the latter threshold would never be applied.

Display List Threshold

The display list threshold limits the number of records displayed per page in list displays. It acts as a paging threshold and applies to most paged lists throughout Service Manager. Adjust the display list threshold by changing the value for the `ps.riconfig.MAX_ENTRIES` parameter in the `app-config.properties` file in `$TSM_HOME/config`.

In most cases, the default value will be appropriate. Please note that this value applies to paged screens in Service Manager and also the CSR application. If the value is not applied, the system defaults to the value for the same parameter in the `RIconfiguration.properties` file (inside the ear file). The value in the configuration file can be overridden for a particular table in a JSP page by using the optional `pageSize` parameter for the `HTMLTableTag` custom tag.

Dropdown Threshold

The dropdown threshold limits the number of records displayed in a dropdown. This applies to the Context Change Module, which allows the user to navigate between accounts and/or service agreements to which they have access. If the dropdown threshold is exceeded, the dropdown is not displayed. Instead, the default information is displayed along with a message that the threshold is exceeded. The user can narrow their result set by setting their position to the appropriate account, service agreement, business structure, or folder. Adjust the dropdown threshold by changing the value for the `tbm.ccm.dropDown.threshold` parameter in the `app-config.properties` file in `$TSM_HOME/config`.

Setting this value to a relatively low value will help the user to avoid having dropdowns which are so long as to be unwieldy. However, there is some inconvenience to the user when he/she experiences an over-threshold condition, because of the necessity to set position to obtain the needed result. So the developer needs to balance the performance gain with the productivity loss to find the appropriate value for this threshold.

Large Enterprise User Threshold

The large enterprise user threshold determines whether or not to display the “Top Five Service Agreements” table in the side navigation bar of Service Manager. If the user has access to more service agreements than the threshold, this table is not displayed. Adjust the large enterprise user threshold by changing the value for the `largeEnterpriseUserThreshold` parameter in the `app-config.properties` file in `$TSM_HOME/config`.

The “Top Five Service Agreements” table requires querying the amount due for all the service agreements to which the user has access. This query has been optimized, but it is still rather expensive. If this unacceptably degrades performance, the large enterprise user threshold can be reduced so that the table is not displayed. In most cases, the default value will be acceptable.

10 Profile Management

Overview

The Business Services Layer contains interfaces and default implementations for user profile management, security profile management, and company profile management. The packages for these components are as follows:

- User profile management – bsl/umf
- Security profile management – bsl/authentication
- Company profile management – bsl/cmf

Managing the User Profile

When a user enrolls, a user profile is created and stored in the database. The user profile may be programmatically managed through the BSL via implementations of `IUserProfileManger` and other interfaces in the umf package. The BSL controller, `IBusinessServices`, calls these umf classes to access user profile information. See the BSL/umf javadoc for default implementations that may be extended to customize user profile management.

Managing the Security Profile

When a user enrolls, a security profile is created and stored in the database. The security profile is an important aspect of SAF and role-based access control. The security profile may be programmatically managed through the BSL via implementations of `ISecurityProfileManger` and other interfaces in the authentication package. The BSL controller, `IBusinessServices`, calls these authentication classes to access security profile information. See the BSL/authentication javadoc for default implementations that may be extended to customize security profile management.

Managing the Company and Company User Profiles

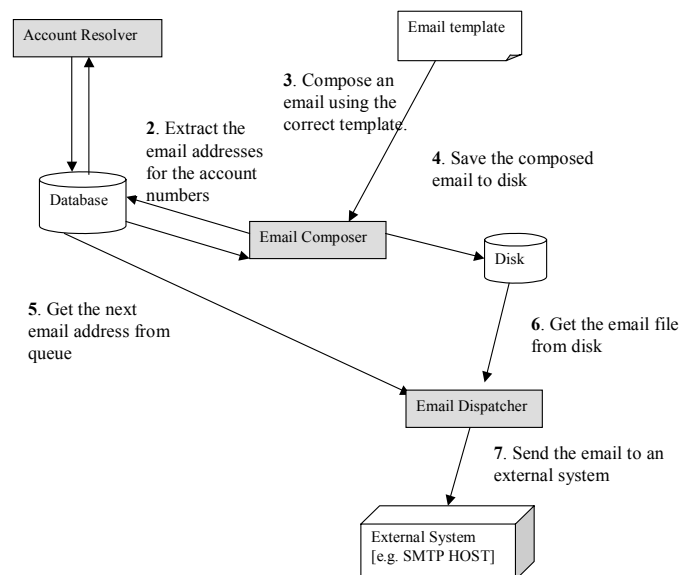
When a user enrolls, a company and company user profile may be created and stored in the database. The company profile may be programmatically managed through the BSL via implementations of `ICompanyProfileManger` and other interfaces in the cmf package. The BSL controller, `IBusinessServices`, calls these cmf classes to access company and company user profile information. See the BSL/cmf javadoc for default implementations that may be extended to customize company profile management.

11 Notifications

Email Overview

The Billing Manager Email system is designed to send emails based on various notification and payment lifecycle events. Such events include bill notifications, payment due, payment sent etc. The email subsystem supports a rich XSLT template based facility for support of both HTML and text based emails as well as pluggable support for other messaging providers, such as SMS. Both payment and enrollment-based activities can result in message generation. The diagrams below detail the major components of the email subsystem.

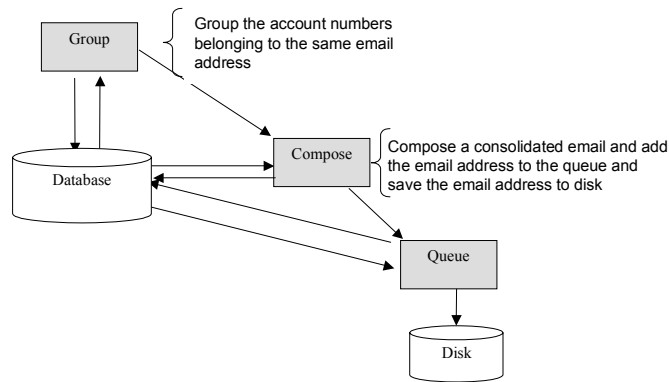
General Email Subsystem



Conceptually the Billing Manager email subsystem is composed from:

- The email composer is triggered by various email jobs and interacts with the Billing Manager database to determine what email should be sent. For each message generated a message is created, and stored.
- Account Resolver is a pluggable component which returns email addresses for a given account number.
- The email dispatcher takes stored emails and sends them based on selected external transport type, for example, SMTP.

Email Grouping via Composer



The Email Composer is further broken down into three components which function together to group multiple account numbers by email address in support of message roll up. Composer components are:

- Group account numbers by send to address
- Compose a group message based on a template
- Create a grouped message for a given queue.

Configuring Email Messaging

Email delivery is configured via properties in `app-config.properties`. At a minimum you must specify the name of a mail host via the property `mail.host`. You may also specify an SMTP host and a variety of other mail specific properties as shown below:

`app-config.properties` snippet associated with mail configuration

```

# SMTP server for email notification for enrollment

mail.smtp.host= exchangeus.edocs.com
mail.host=exchangeus.edocs.com

##
mail.queue.storageDirectory=mailqueue
mail.xml.templateFilename=templates.xml
#
mail.transport.protocol=SMTP
mail.queue.threadMax=10
mail.queue.elementsPerThread=30
mail.queue.dispatcherSleepPeriod=5
mail.queue.hangingTimeout=10
  
```

Email Configuration properties

Property	Description
mail.smtp.host mail.host	Fully qualified IP address or name of a host running the SMTP which can be used to send email.
mail.xsl.templateFilename	Template XML Style sheet, located in %EDX_HOME%\config\, used for composing email messages. Default:templates.xml.
mail.queue.storageDirectory	Directory in %EDX_HOME%\config\ used to temporarily store undeliverable emails. Default:mailqueue.
mail.queue.threadMax	Maximum number of mail threads to create when sending emails. Default 10 threads.
mail.queue.elementsPerThread	Email messages are sent in batches, by thread. The maximum number of messages each thread should send per batch. Default 30 messages.
mail.queue.dispatcherSleepPeriod	Period, in seconds, the dispatcher should sleep between sending emails, to allow other threads to complete sends before removing queued messages.
mail.queue.hangingTimeout	Period, in seconds, the dispatcher should wait before deciding the mail host is not responding and queue messages. Default 15 seconds

Email Notifications

Billing Manager supports a number of email notifications directly out of the box. These notifications fall into two categories, bill ready notification and payment notifications. The text of each notification is managed via an XML style sheet containing text and variables that can be modified for each notification. Variable text itself is defined via XSLT Style sheets. The style sheet contains a number of elements that should be customized to reflect your install.

Email templates reflect the content of each of the various notifications supported by Billing Manager. Each notification template email started with an XSLT template match statement similar to `<xsl:template match="notificationType">`. Where *notificationType* is one of the known notification types.

Each notification contains one or more variables that may be substituted within the body text of the message. Additionally each variable may contain sub-elements. In support of apply style sheet transforms the variables themselves, and then sub variables, are provided to the notification as snippets of XML. As such the content may be processed with XSLT statements. The Bill Notification email template style sheet is shown below:

```
<xsl:template match="BillNotification">
From: yourcustomerservice@yourco.com
Subject: tbm account bill ready notification
Content-Type: text/html

    <html>
    <b>Following accounts registered under your email address
        have their bills ready. </b><br/>
        <xsl:for-each select="acct">
        Account payment is overdue for account #:
            <xsl:value-of select="number"/>
        </xsl:for-each>
    <br/>
    This is an automatically generated email.
    Please do not reply to this email address.
    </html>
</xsl:template>
```

Notification Email Variables

Notification	Description & Example
Bill Notification	Account Numbers within accounts. <code><acct></code> <code><number>00000001</number></code> <code><number>00000002</number></code> <code></acct></code>
Enrollment Notification	Username and password <code><user>someUser</user></code> <code><pass>password</pass></code>
RecurringPayment Notification	User, Account number for recurring payment, amount of payment <code><uid>someUser</uid></code> <code><actnum>00000001</actnum></code> <code><amount>15.00</amount></code>
RecurringPaymentUpdate Notification	User, Account number for recurring payment, amount of payment <code><uid>someUser</uid></code> <code><actnum>00000001</actnum></code> <code><amount>15.00</amount></code>
QuickPayment Notification	Quick payment amount, transaction status message <code><amount>15.00<amount></code> <code><msg>explanation</msg></code>
QuickPaymentFailure Notification	Quick payment amount, transaction status message <code><amount>15.00<amount></code> <code><msg>explanation</msg></code>

The email subsystem merges the message template with runtime information to create the email message. The messenger then calls the gateway, configured in `app-config.properties`, to send the email message.

Modifying Email Addresses Programmatically

Subscribers may have zero or more email addresses associated with a login id. The Billing Manager B2C and B2B applications provide support for managing email addresses via the Profiles tab pages. However, developers may create custom interactions which support add, delete and update operations on a given set of email addresses. The Javadoc for the BSL subsystem provides a complete description of the fields and methods associated with email addresses. The example below shows how you might interact with the Business Services Layer to obtain the current set of email addresses, add a new email address, and associate an updated set of email addresses back with a user.

Add Email Address Example Action

```

public ActionForward doAction(ActionMapping mapping,
                              ActionForm form,
                              HttpServletRequest request,
                              HttpServletResponse response)
    throws Exception {
    // Obtain user associated with the current login ID
    String uid = SessionUtils.getUserId(request);
    IBusinessServices bs1 = new BusinessServices();
    IUser u = bs1.findUserByUID(uid);
    try {
        // Using the current user obtain the current
        // list of email addresses
        Set emails = u.getNotificationEmails();
        if (shouldNewAddyBePrimaryEmailAddy? == true) {
            Iterator it = emails.iterator();
            while (it.hasNext()) {
                IEmail e = (IEmail)it.next();
                if (e.isPrimary()) e.setPrimary(false);
                break;
            }
        }
        // create a new email address instance, set it primary
        // add to the current set, and then update the user in backing store
        IEmail newEmailAddress = new DefaultEmail("some@somplace.com");
        newEmailAddress.setPrimary(true);
        emails.add(newEmailAddress);
        bs1.updateUser(u);
    } catch (Exception e) {
        . . .
        return mapping.findForward(RIConstants.SUCCESS);
    }
}

```

12 Address Books

Address Book Overview

Billing Manager supports personal and corporate address books. These address books are displayed in Billing Manager LDE views, so each has its own DDN. Configure address book DDN's in `app-config.properties` as follows:

```
# PAB DDN
```

```
DDN.PersonalAddressBook=PAB
```

```
# CAB DDN
```

```
DDN.CorporateAddressBook=CAB
```

See the javadocs delivered for the `LDEToolkit` component and Chapter 3, Statementing and Content Access for details about Billing Manager Views and Live Data Extraction for Views.

Personal Address Books

Billing Manager supports personal address books and follows a standard application flow architecture involving Struts and Tiles. Inspect the WAR file delivered in the Billing Manager application EAR to see the configuration and `jsp` files described below.

The top-level Billing Manager UI involves templates defined in `webroot/WEB-INF/tiles-defs.xml` and `struts-config.xml`. Billing Manager delivers a tabbed interface placing a personal address book sub-tab under a profile tab. This is configured in `tiles-defs.xml` as follows:

```
<tiles-definitions>

<definition name=".template.main" path="/_templates/main_template.jsp">
    <put name="title" value="welcome to Telco wireless - we work for you" type="string"/>
    <put name="header" value="/_includes/header.jsp" type="page"/>
    <put name="sectionHeader" value="/_includes/blank_page.jsp" type="page"/>
    <put name="sectionHeadline" value="" type="string"/>
    <put name="subSectionHeadLine" value="" type="string"/>
    <put name="selectDisplayMenu" value="/_includes/blank_page.jsp" type="page"/>
    <put name="tabs" value=".template.tabs" type="definition"/>
</definition>
```

```

        <put name="tab" value="Account Sum"/>
        <put name="subtab" value="" type="string"/>
        <put name="action" value="/_includes/blank_page.jsp" type="page"/>
        <put name="leftSideNav" value="/_includes/blank_page.jsp"
type="page"/>
        <put name="summary" value="/_includes/blank_page.jsp"
type="page"/>
        <put name="module1" value="/_includes/blank_page.jsp"
type="page"/>
        <put name="module2" value="/_includes/blank_page.jsp"
type="page"/>
        <put name="module3" value="/_includes/blank_page.jsp"
type="page"/>
        <put name="module4" value="/_includes/blank_page.jsp"
type="page"/>
        <put name="module5" value="/_includes/blank_page.jsp"
type="page"/>
        <put name="module6" value="/_includes/blank_page.jsp"
type="page"/>
        <put name="module7" value="/_includes/blank_page.jsp"
type="page"/>
        <put name="view" value="/_includes/blank_page.jsp" type="page"/>
        <put name="sidebar" value="sidebar.simple" type="definition"/>
        <put name="footer" value="/_includes/footer.jsp" type="page"/>
    </definition>
<definition name=".template.sectionHeader"
path="/_includes/section_header.jsp"/>
    <!-- Top Navigation Definitions -->
    <definition name=".template.tabs" path="/_templates/tab_template.jsp">
        <putList name="tabList">
            <item value="Overview"
link="/hierarchy/main/dashboard/show.do"
classtype="com.edocs.application.tbm.core.menus.EdocsBaseMenuItem"/>

            ...
        </putList>
        <putList name="subtabList"/>
    </definition>

```

```

</definition>

...

<!-- Profile Sub Tab List -->

<definition name=".tabs.profile" extends=".template.tabs">
    <putList name="subtabList">
        <item value="Personal Profile"
link="/profile/personal_profile.do"
classtype="com.edocs.application.tbm.core.menus.EdocsBaseMenuItem"/>
        <item value="Notifications"
link="/profile/notifications.do"
classtype="com.edocs.application.tbm.core.menus.EdocsBaseMenuItem"/>
        <item value="Personal Address Book"
link="/profile/personal_address_book.do"
classtype="com.edocs.application.tbm.core.menus.EdocsBaseMenuItem"/>
    </putList>
</definition>

```

The profile sub tab list above indicates that when the Personal Address Book subtab is selected, navigation proceeds to the Struts action, */profile/personal_address_book*.

Struts-config.xml defines this action as follows:

```

<action path="/profile/personal_address_book"
forward=".main.profile.personalAddressBook">
    </action>

```

The action forwards to another tile that is defined in tiles-defs.xml as follows:

```

<!-- Personal Address Book -->

<definition name=".main.profile.personalAddressBook"
extends=".template.main">
    <put name="sectionHeader" value=".template.sectionHeader"
type="definition"/>
    <put name="sectionHeadline" value="Profile"/>
    <put name="subSectionHeadLine" value="Personal Address Book"/>
    <put name="tabs" value=".tabs.profile" type="definition"/>
    <put name="tab" value="Profile"/>
    <put name="subtab" value="Personal Address Book"/>
    <put name="leftSideNav" value="/_includes/side_navbar.jsp"
type="page"/>

```

```

        <put name="module1" value="/personaladdressbook.do"
type="definition"/>
        <put name="module3" value="/addressbook/pab-addlink.jsp"
type="page"/>
    </definition>

```

Personal address book modules above (module1 and module3) are configured in webroot/WEB-INF/tiles-defs-addressbook.xml and struts-config-addressbook.

module3 above places an add link on the page. The link points to the Struts action class, com.edocs.application.tbm.b2b.actions.PersonalShowAddressAction, in struts-config-addressbook:

```

<!-- Show the PAB AddNew Page -->
        <action path="/addressbook/pab-addnew/show"
type="com.edocs.application.tbm.b2b.actions.PersonalShowAddressAction"
name="addressBookForm" scope="request"
parameter=".main.addressbook.pab.addnew" validate="false">
            <!-- <set-property property="checkLogin" value="false" /> -
        ->
        <forward name="success"
path=".main.addressbook.pab.addnew"/>
    </action>

```

The action class forwards to .main.addressbook.pab.addnew which pops up another tile, .module.addressbook.pab.addnew, both defined in tiles-defs-addressbook:

```

<definition name=".main.addressbook.pab.addnew" extends="template.popup">
    <put name="module1" value=".module.addressbook.pab.addnew"/>
</definition>
<definition name=".module.addressbook.pab.addnew"
extends="template.module">
    <put name="form" value="/addressbook/pab-addnew.jsp" type="page"/>
    <put name="activeModule" value="addressbook"/>
    <putList name="moduleNames">
        <add value="addressbook"/>
    </putList>
    <putList name="moduleLinks">
        <add value="#" />
    </putList>
</definition>

```

Module1 above is configured to read the struts definition, /personaladdressbook.do, defined in struts-config-addressbook.xml:

```
<!-- ===== Action Mapping Definitions ===== -->

<action-mappings
type="com.edocs.application.tbm.core.common.EdocsActionMapping">

    <!--
    ***** -->

    <!-- ***** LDE ACTIONS
    ***** -->

    <!--
    ***** -->

    <action path="/personaladdressbook"
type="com.edocs.application.tbm.b2b.actions.PersonalAddressBookRequestAction"
name="addressBookForm" scope="request" validate="false">

        <!-- <set-property property="checkLogin" value="false" /> -
    ->

        <forward name="success"
path=".module.addressbook.personaladdressbook"/>
    </action>
```

The Struts action class, *com.edocs.application.tbm.b2b.actions.PersonalAddressBookRequestAction*, loads the address book view into another tile, *.module.addressbook.personaladdressbook*, defined in *tiles-defs-addressbook.xml*:

```
<definition name=".module.addressbook.personaladdressbook"
extends=".view.setup">

    <put name="viewName" value="Pab_Display"/>
    <put name="viewType" value="HTML"/>
    <put name="ddn" value="PersonalAddressBook"/>
    <put name="useExistingViewInfo" value="true"/>

</definition>
```

This tile extends *.view.setup*, which is defined in *tiles-defs.xml*:

```
<!-- Views Setup (Base definition for rendering the views created by Billing
Manager) -->

<definition name=".view.setup" path="/_templates/display_view.jsp"
controllerClass="com.edocs.application.tbm.displayview.DisplayTiledView">

    <put name="viewName" value="Defaultview"/>
    <put name="viewType" value="HTML"/>
```

```

        <put name="ddn" value="TBM"/>
    </definition>

```

Here are other popup tiles available in tiles-defs-addressbook.xml. Note that you can also define a list of module names and links for the session here.

```

<!-- ***** ->
<!-- ***** PERSONAL ADDRESS BOOK popup tiles ***** -
->
<!-- ***** -
->

<definition name=".main.addressbook.pab.add" extends=".template.popup">
    <put name="module1" value=".module.addressbook.pab.add"/>
</definition>

<definition name=".module.addressbook.pab.add"
extends=".template.module">
    <put name="form" value="/addressbook/pab-add.jsp" type="page"/>
    <put name="activeModule" value="addressbook"/>
    <putList name="moduleNames">
        <add value="addressbook"/>
    </putList>
    <putList name="moduleLinks">
        <add value="#"/>
    </putList>
</definition>

<definition name=".main.addressbook.pab.edit" extends=".template.popup">
    <put name="module1" value=".module.addressbook.pab.edit"/>
</definition>

<definition name=".module.addressbook.pab.edit"
extends=".template.module">
    <put name="form" value="/addressbook/pab-edit.jsp" type="page"/>
    <put name="activeModule" value="addressbook"/>
    <putList name="moduleNames">
        <add value="addressbook"/>
    </putList>
    <putList name="moduleLinks">

```

```

        <add value="#" />
    </putList>
</definition>
<definition name=".main.addressbook.pab.addnew"
extends=".template.popup">
    <put name="module1" value=".module.addressbook.pab.addnew" />
</definition>
<definition name=".module.addressbook.pab.addnew"
extends=".template.module">
    <put name="form" value="/addressbook/pab-addnew.jsp" type="page" />
    <put name="activeModule" value="addressbook" />
    <putList name="moduleNames">
        <add value="addressbook" />
    </putList>
    <putList name="moduleLinks">
        <add value="#" />
    </putList>
</definition>

```

The jsp pages in the tile definitions above often point to two submit actions, "show.do" and "save.do", as defined in struts-config-addressbook:

```

<!-- ***** -->

    <!-- ***** PERSONAL ADDRESS BOOK
***** -->

    <!--
***** -->

    <!-- Show the PAB Add Page -->
    <action path="/addressbook/pab-add/show"
type="com.edocs.application.tbm.b2b.actions.PersonalShowAddressAction"
name="addressBookForm" scope="request" parameter=".main.addressbook.pab.add"
validate="false">
        <!-- <set-property property="checkLogin" value="false" /> -
->

        <forward name="success" path=".main.addressbook.pab.add" />
    </action>
    <!-- Save the new PAB -->

```

```

        <action path="/addressbook/pab-add/save"
type="com.edocs.application.tbm.b2b.actions.PersonalSaveAddressAction"
name="addressBookForm" scope="request" parameter="/addressbook/pab-
complete.jsp" validate="true" input="pab_add">

        <!-- <set-property property="checkLogin" value="false" /> -->
-->

        <forward name="success" path="/addressbook/pab-
complete.jsp"/>
    </action>

    <!-- Show the PAB Edit Page -->

    <action path="/addressbook/pab-edit/show"
type="com.edocs.application.tbm.b2b.actions.PersonalShowAddressAction"
name="addressBookForm" scope="request"
parameter=".main.addressbook.pab.edit" validate="false">

        <!-- <set-property property="checkLogin" value="false" /> --
->

        <forward name="success" path=".main.addressbook.pab.edit"/>
    </action>

    <!-- Save the PAB Edit -->

    <action path="/addressbook/pab-edit/save"
type="com.edocs.application.tbm.b2b.actions.PersonalSaveAddressAction"
name="addressBookForm" scope="request" parameter="/addressbook/pab-
complete.jsp" validate="true" input="pab_edit">

        <!-- <set-property property="checkLogin" value="false"/> --
>

        <forward name="success" path="/addressbook/pab-
complete.jsp"/>
    </action>

    <!-- Show the PAB AddNew Page -->

    <action path="/addressbook/pab-addnew/show"
type="com.edocs.application.tbm.b2b.actions.PersonalShowAddressAction"
name="addressBookForm" scope="request"
parameter=".main.addressbook.pab.addnew" validate="false">

        <!-- <set-property property="checkLogin" value="false" /> --
->

        <forward name="success"
path=".main.addressbook.pab.addnew"/>
    </action>

    <!-- Save the PAB Delete -->

```

```

        <action path="/addressbook/pab-delete/save"
type="com.edocs.application.tbm.b2b.actions.PersonalDeleteAddressAction"
name="addressBookForm" scope="request" validate="false" input="pab">

        <!-- <set-property property="checkLogin" value="false" /> -
->

        <forward name="success"
path="/profile/personal_address_book.do"/>

    </action>

```

Note that the form used in these jsp pages is configured in a form-bean in struts-config-addressbook.xml. Read the javadocs for com.edocs.application.tbm.b2b.forms.AddressBookForm:

```

<struts-config>

    <!-- ===== Form Bean Definitions =====
-->

    <form-beans>

        <form-bean name="addressBookForm"
type="com.edocs.application.tbm.b2b.forms.AddressBookForm"/>

    </form-beans>

```

To customize a personal address book for the Billing Manager application, read the javadocs for the tbm-b2b-app application, paying particular attention to the Struts action and form classes delivered for personal address books. Open the WAR file delivered with the tbm-b2b-app ear to see the tiles delivered in webroot/_templates and webroot/addressbook. Also see the Struts and Tiles configuration files delivered in webroot/WEB-INF.

Extend or use com.edocs.application.tbm.b2b.forms.AddressBookForm and com.edocs.application.tbm.b2b.actions.Personal*AddressActions classes as needed. These classes employ the AddressBook component delivered with Billing Manager. To customize the AddressBook component, read the javadocs delivered for the com.edocs.domain.telco.addrbook classes. See Section 11.3, AddressBook Component for a description of lower level services provided by Billing Manager.

Corporate Address Books

Corporate address books follow the same application architecture used for personal address books as discussed in Section 11.1 above. See corresponding corporate address book sections defined in the configuration files described above.

AddressBook Component

The AddressBook component services both personal and corporate address book applications. The AddressBook component is delivered in the Billing Manager EAR. Read the javadocs delivered for `com.edocs.domain.telco.addrbook` for implementation details.

The AddressBook component consists of the following classes:

`com.edocs.domain.telco.addrbook.ABException` (exception class for addressbook use)

`com.edocs.domain.telco.addrbook.AddrBookViewProcessor` (extends `RUIViewProcessor` and provides address tag substitution capability for views)

`com.edocs.domain.telco.addrbook.AddressBook` (abstract superclass of `PersonalAddressBook` and `CorporateAddressBook`)

`com.edocs.domain.telco.addrbook.AddressBookFactory` (gets/creates personal or corporate address books from/for the session)

`com.edocs.domain.telco.addrbook.AddressBookRequestZLDESTream` (creates an `LDEToolkit` stream given a `ddn`, account number, and addressbook)

`com.edocs.domain.telco.addrbook.AddressBookViewHelper` (provides stream filtering)

`com.edocs.domain.telco.addrbook.AddressEntry` (an address)

`com.edocs.domain.telco.addrbook.CABDataSource` (corporate address book data source)

`com.edocs.domain.telco.addrbook.CorporateAddressBook` (extends `AddressBook`)

`com.edocs.domain.telco.addrbook.PABDataSource` (personal address book data source)

`com.edocs.domain.telco.addrbook.PersonalAddressBook` (extends `AddressBook`)

The Struts action classes delivered in the Billing Manager application WAR were described in Section 11.1 in the context of their interface with personal address book tiles. Here we describe their interface with the AddressBook component. The Struts action classes involved:

`com.edocs.application.tbm.b2b.actions.AddressActionBase`

`com.edocs.application.tbm.b2b.actions.PersonalAddressBookRequestAction`

`com.edocs.application.tbm.b2b.actions.PersonalShowAddressAction`

`com.edocs.application.tbm.b2b.actions.PersonalSaveAddressAction`

`com.edocs.application.tbm.b2b.actions.PersonalDeleteAddressAction`

`com.edocs.application.tbm.b2b.actions.CorporateAddressBookRequestAction`

`com.edocs.application.tbm.b2b.actions.CorporateShowAddressAction`

`com.edocs.application.tbm.b2b.actions.CorporateSaveAddressAction`

```
com.edocs.application.tbm.b2b.actions.CorporateDeleteAddressAction
```

These action classes extend the Billing Manager core component, `EdocsAction`. These action classes call the `AddressBook` component to configure and load an addressbook into an e-Statement view (`PersonalAddressBookRequestAction`) or to show (`PersonalShowAddressAction`), save (`PersonalSaveAddressAction`), or delete (`PersonalDeleteAddressAction`) addresses from the address book.

Displaying the Address Book

The `PersonalAddressBookRequestAction` is configured in the action mapping `/personaladdressbook` in `struts-config-addressbook.xml`. This action class loads the personal address book into a view as follows:

```
try {
    log.debug("Entering
PersonalAddressBookRequestAction.doAction()");
    LDEManager ldeManager = LDEManager.getInstance();
    AddressBook addressBook =
AddressBookFactory.getSessionPersonalAddressBook(httpServletRequest);
    ICustomer customer =
CAMClassFactory.getCustomer(httpServletRequest);
    String accountNumber =
customer.getCurrentAccount().getAccountNumber();
    String ddN = customer.getDDNMap().getDDN("PersonalAddressBook");
    log.debug("addressbook ddN " + ddN);

    AddressBookRequestZLDEStream ldeStream = new
AddressBookRequestZLDEStream(ddN, accountNumber, addressBook);

    String docId = ldeManager.setContent(ldeStream);

    viewInfo viewInfo = new ViewInfo();
    viewInfo.setDdn(ddN);
    viewInfo.setDocId(docId);
    viewInfo.setViewName("Pab_Display");
    viewInfo.setViewType("HTML");

    SessionUtils.setView(httpServletRequest, viewInfo);
```

This action instantiates an LDEToolkit component, `com.edocs.domain.telco.lde.LDEManager`, and the AddressBook component, `AddressBook`. It uses the CAM layer component, `CAMClassFactory`, to get the current customer, account number, and personal address book DDN. It instantiates a live data extraction (LDE) stream (`AddressBookRequestZLDEStream`) which it uses to set the view content. `ViewInfo` parameters are populated and the view is set in the session for display. (For more information about LDE, see Chapter 3.4 Live Data Extraction.)

Once the personal address book is loaded and ready for display, the action class forwards to the "success" jsp tile defined in `struts-config-addressbook`:

```
<!-- ***** -->

<!-- ***** LDE ACTIONS
***** -->

<!--
***** -->

<action path="/personaladdressbook"
type="com.edocs.application.tbm.b2b.actions.PersonalAddressBookRequestAction
" name="addressBookForm" scope="request" validate="false">

    <!-- <set-property property="checkLogin" value="false" /> -
->

    <forward name="success"
path=".module.addressbook.personaladdressbook"/>
</action>
```

`module.addressbook.personaladdressbook` in `tiles-defs-addressbook` extends `.view.setup` from `tiles-defs.xml`. Finally, `webroot/_templates/display_view.jsp` is called to display the personal address book view.

(From `tiles-defs.xml`)

```
<!-- Views Setup (Base definition for rendering the views created by
eStatement) -->

<definition name=".view.setup" path="_templates/display_view.jsp"
controllerClass="com.edocs.application.tbm.displayview.DisplayTiledView">

    <put name="viewName" value="Defaultview"/>
    <put name="viewType" value="HTML"/>
    <put name="ddn" value="TBM"/>

</definition>
```

(From `tiles-defs-addressbook.xml`)

```
<definition name=".module.addressbook.personaladdressbook"
extends=".view.setup">
```

```

<put name="viewName" value="Pab_Display"/>
<put name="viewType" value="HTML"/>
<put name="ddn" value="PersonalAddressBook"/>
<put name="useExistingViewInfo" value="true"/>
</definition>

```

Showing Addresses

When an address is selected from the personal address book view, the Struts action class, `PersonalShowAddressAction`, instantiates `AddressBookForm` and calls the `AddressBook` Component, `AddressBookFactory`, to get the current personal address book from the session or create one, if it does not exist. `PersonalAddressBook` gets an instance of `PABDataSource` to persist the address book. Then it passes an `AddressBookForm` and an `AddressBook` as parameters in a call to `AddressActionBase.populateForm()`, which calls `PersonalAddressBook.getEntry()` to instantiate another `AddressBook` component, `AddressEntry`. Once the form is populated, the Struts action forwards to `addressbook/pab-add.jsp`.

See the javadocs delivered for the `AddressBook` component for more implementation details.

Saving and Deleting Addresses

After an address has been shown, it may be updated, saved, or deleted. A submit button on the `addressbook/pab-add.jsp` page forwards to a `save.do` Struts action.

The Struts actions, `PersonalDeleteAddressAction` and `PersonalSaveAddressAction`, instantiate `AddressBookForm` and call the `AddressBookFactory` to get the current personal address book from the session. Then the action classes pass these parameters in calls to `AddressActionBase.updateAddressBook()`, which deletes, updates, or saves new `AddressEntry` information to the `PABDataSource` via `PersonalAddressBook.deleteEntry()`, `PersonalAddressBook.updateEntry()`, and `PersonalAddressBook.addEntry()`.

See the javadocs delivered for the `AddressBook` component for more implementation details.

Address Book View Processing

The `AddressBook` component provides an address book processor for filtering address book views. LDE view streams are strings that may be manipulated to change presentation in certain ways. The `AddressBook` component, `AddrBookViewProcessor`, takes a `ViewInfo` object, processes the input stream from the `ViewInfo` object inserting address book entries where address book tags appear (via `AddrBookViewHelper`), and then returns a new `ViewInfo` object with the address book entries inserted.

`AddressBookViewHelper` creates a "(Don't) Show Nicknames" toggle link on the address book view and substitutes phone numbers with nickname hyperlinks, if the nickname function is toggled on. See `webroot/_assets/scripts/addrbook.js` for the javascript functions.

Address Book Post-Processing of Billing Manager Views

As for address book views, support for substitution of phone numbers with nickname hyperlinks is provided for all Billing Manager views. See the javadocs delivered for the `DisplayView` component and Chapter 3, *Statementing and Content Access* for details.

To configure a view to use an address book post processor, set name="postprocessor" and name="usePersonalAddressBook" in tiles-defs.xml as follows:

```
<!--View Display Detailed Usage (Home) Collapsed-->
<definition name=".statement.detailedusage.view" extends=".view.setup">
    <put name="viewName" value="B2B_DetHomeCol"/>
    <put name="usePersonalAddressBook" value="true"/>
    <put name="postProcessor" value="AddressBook"/>
    <put name="useViewInfoForFilters" value="true"/>
    <put name="subIdx" value="true"/>
</definition>
```

The AddressBookPostProcessor resides in the DisplayView component package, com.edocs.application.tbm.displayview. Override this class as needed and configure it in app-config.properties:

```
#post processor classes
```

```
PostProcessors=PostProcessEngine
```

```
PostProc.PostProcessEngine=com.~.displayview.PostProcessorComposition
```

```
PostProcessor.classes=addressbook.class
```

```
addressbook.class=com.~.displayview.AddressBookPostProcessor
```

13 Custom Jobs

This chapter 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 chapter 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

Billing 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 *Siebel Billing Manager Administration Guide*.

However, there may be times when you will want to expand these predefined jobs to fit your needs. For cases like this Billing 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 Billing Manager and/or your own custom task by defining what is referred to in Billing 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.

The following illustration shows a new custom Indexer job type in the Command Center for the latter example.

Task 2: Indexer	
DDF Path:	/opt/EDCSbd/AppProfiles/Alex2/DOC_CONFIG/Indexer/20010319155326/
Doc Date:	----Today's Date----
Index Field List:	Name* EndDate* PaymentDueDate* >> << TotalCharges* StartDate* TaxesAndSurcharges* AmountReceived* StatementDate*
Task 3: ShellCmdTask	
Shell Command:	enter your shell command here
Environment Vars:	enter semi-colon separated vars here
Task 4: IXLoader	
Skip Rows:	0
Split Size:	0
Optional Field Count:	0
Load Method:	Direct
Task 5: AutoIndexVolAccept	
Action on Index Volume:	Auto Accept

Defining a New Job Type

This section 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 Billing Manager database using the Oracle utility **sqlplus**. Within this SQL script you define:

- 1 The job name
- 2 The tasks and the order in which they will run

3 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': 'AutoIndexVolAccept'

    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 will be:

```
jtid := pwc_job_types.create_job_type ('myIndexer');
```

Specify Job Tasks

The next step is to specify what tasks will be part of the new job, and in what order will 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):

- 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 for the *SDK API Specification*.

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 need to run it against the Oracle database used by Billing Manager (as described in the Installation and Configuration Guides). For example, if the script is named **myindexer.sql** and placed in **/opt/EDCSbd/db** (the default database location for Billing Manager), you could run the following in SQL*Plus:

```
$ sqlplus -s edx_dba/edx@edx.db @ /opt/EDCSbd/db/myindexer.sql
```

The above command presumes you are using the default names for the Billing 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 in 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. The error message will be logged into the log file by Billing Manager. However, any errors within the shell command are not logged and 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:

- Scanner
- Indexer
- ShellCmdTask
- IXLoader
- AutoIndexVolAccept

As mentioned in the previous section, 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;
```

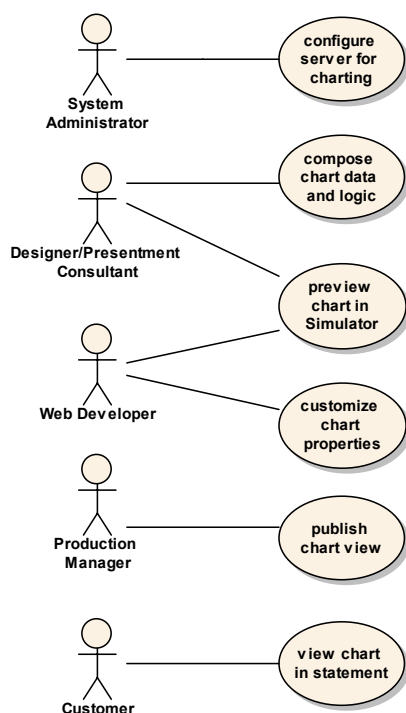
14 Charting

Introduction and Components

Billing 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 Billing 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 Billing 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 Siebel 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 will refresh dynamically with each new version set of statement data.

The sections in this chapter describe each of these tasks in detail.

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

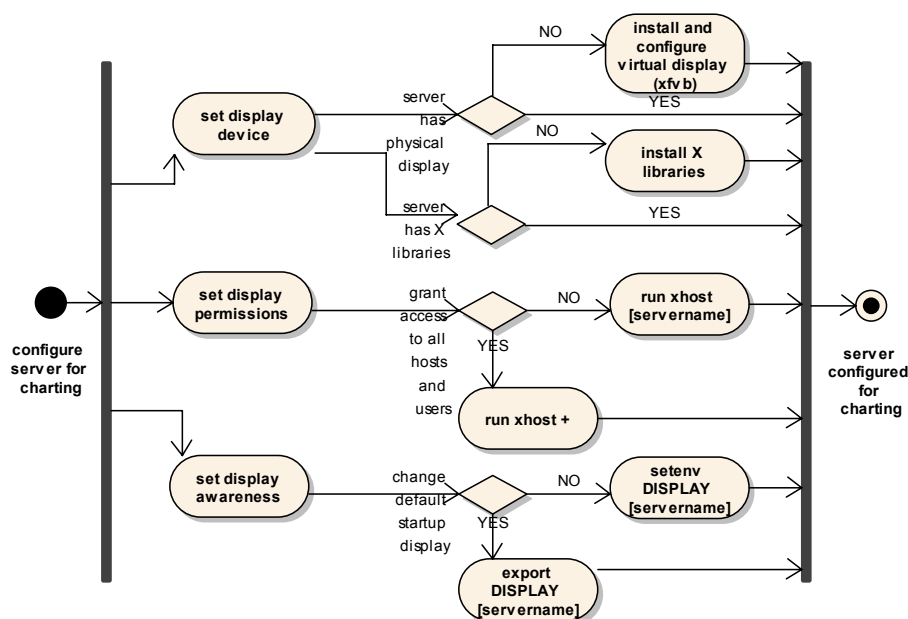
CAUTION: The configuration steps in this chapter apply primarily to **deployment** servers. Servers in a **production** environment often have physical display devices with graphics support, so that configuration may not be an issue. **Always test charts** (with the rest of your web application) **on 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, Siebel 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 will need 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 will 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. Siebel recommends that you add **xhost** to your environment PATH variable.

To grant X server display access to all available hosts and users, type:

```
xhost +
```

For other options, see [Syntax and Parameters](#).

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 will be denied.
+	Turns off access control; grants access to all host names and user names, even if not on the X server list.
-	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 will use 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 *Siebel Billing Manager Installation Guide*.

Configuring a Headless Server for Charting

If your deployment environment does not have a physical display **and** X libraries, your "headless" server will need 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 **smitt** 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):

- 1 Download **xvfb** from ftp://www.ferret.noaa.gov/special_request/xvfb/solaris/
- 2 Install to `/usr/X11R6`. **xvfb** will be installed in the `/bin` directory.
- 3 Enable X display permission on your web server with the command **xhost +**.
- 4 To set the current display to use the frame buffer for graphics display, set your DISPLAY variable, for example:

```
DISPLAY=:1; export DISPLAY
```

This will send any graphics output going to display 1 to shared memory.

- 5 Run **xvfb** as a background process.

```
/usr/X11R6/bin/xvfb :1 -screen 0 800x600x24 &
```

The "&" will kill the command window and leave the task running in the background.

This procedure will create 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 will draw 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.

TIP: **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 Billing Manager **Composer** tool to define data objects and custom tags in HTML templates for Billing Manager **applications**. Defining a **chart tag** for a **table** or **group** will display 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 will 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 *Siebel Billing Manager Administration Guide*.

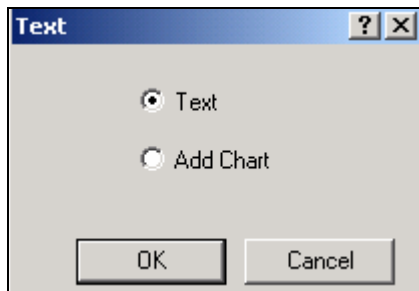
Inserting a Chart Tag in the Composer

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.

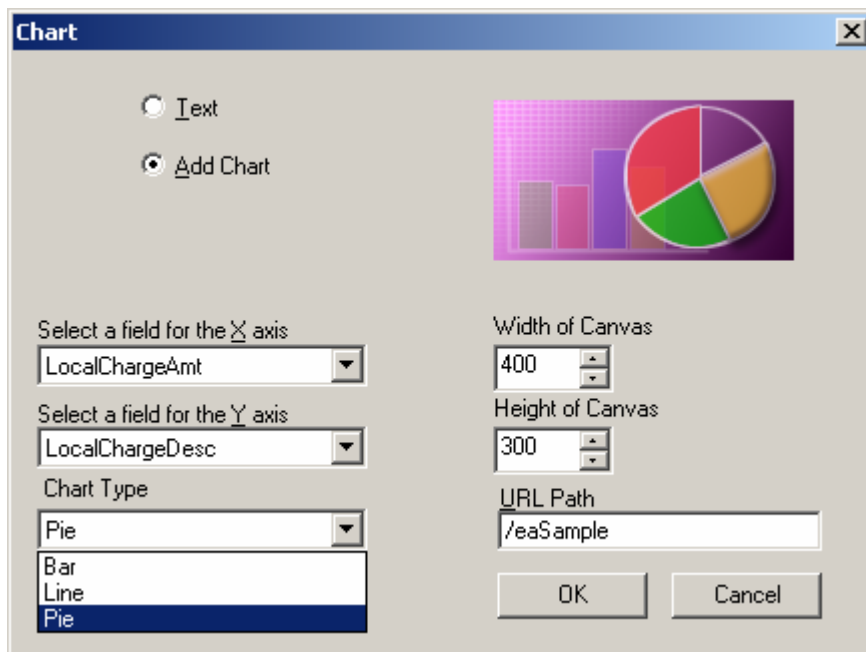
TIP: Tables to be charted must have **at least two fields**, one of which must contain **numeric values**. The Composer converts any values in non-numeric field types to numbers.

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 **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 will still generate 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](#).

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.

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 Siebel 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**.

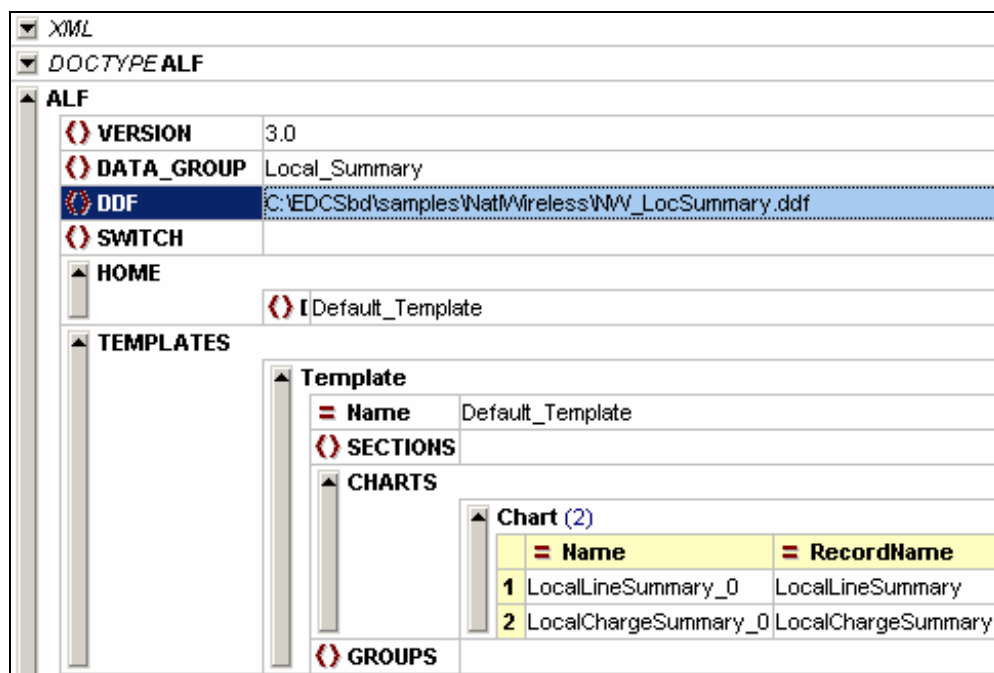
If you created a second chart for the same table, the Composer would generate the chart tag **[E]LocalChargeSummary_1,U[/E]** and the properties file **LocalChargeSummary_1.properties**.

When you publish an HTML view, you will 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 Billing 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. In this illustration (created in Altova XML Spy), the ALF file for **NW_LocSummary.ddf** has had two charts added, **LocalLineSummary_0** and **LocalChargeSummary_0**.



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](#).

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](#).

TIP: You can chart the same data table in two different charts. This will increment 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 will need to use the Chart Simulator API, which is a command-line Java tool.

Before you simulate your chart, you will 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](#), which includes procedures for [Using com.edocs.app.chart.Simulator](#).

Customizing Chart Properties

The Composer allows you to set only a few chart properties directly. To customize the format and design of your charts, you will 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 sub element 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.	
UnixChart	By default, the Composer sets UNIXChart=Pie To create other types of charts, set the chart type in the properties file.	Pie

You will notice other attributes listed in the ALF. These attributes are placed in the ALF for backward compatibility with previous versions of Billing Manager and have no effect on the current version of Billing 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 will not be 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, you will need to set the URL T/F flag in the ALF for HidePieLegend attribute to 1. This **only** applies to pie charts.

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 will appear 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.

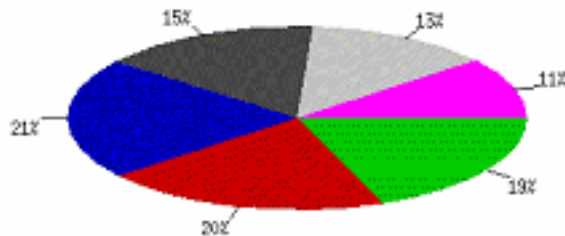
This section illustrates each available chart type for this example dataset.

```
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

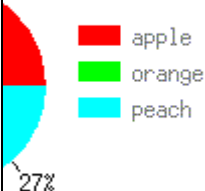


Pie chart with one slice per data point.

To define pie properties:

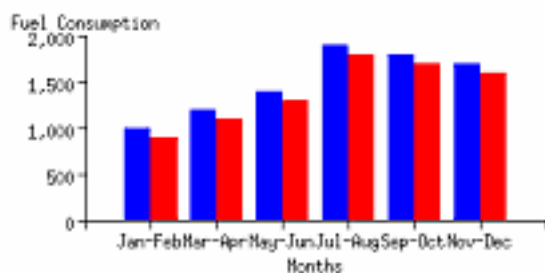
Pie.* properties should be defined only when Type=Pie. These properties control the aspect ratio (height and width) of the pie; the angle, size, and colors of the slices; and the labels and legends. For a round pie, set **Pie.Height** and **Pie.Radius** based on the height and width of the chart in pixels.

Property	Default	Description
PieLabelPosition	=2	Defines the position of the pie slice labels.
PieStartDegrees	=0	Defines the angle of the first pie slice.
PieTextLabelOn	=false	true displays pie slice name, for example <i>College Fund</i>
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.
Pie.Width	=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
<pre>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</pre>	

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

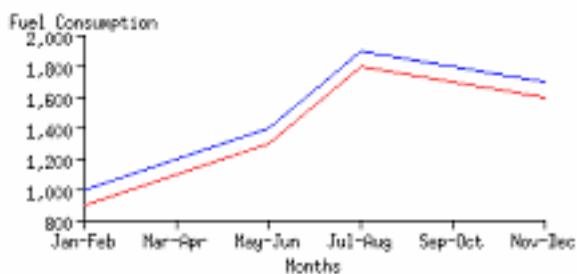


Displays each data series **vertically** in a single color (sometimes called a column chart). To display horizontally, use **HorizBar**. To display different colors for each bar in a series, use **IndBar** (horizontal) or **IndColumn** (vertical).

To define bar properties:

Property	Default	Description
Bar.Baseline	=0.0	Value from which bars will 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 will 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



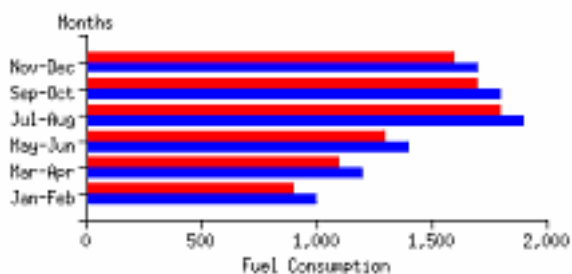
Displays data values as lines on a graph, without value labels for each data point.

To define LineClip property:

To clip line values at the boundary of the plot area, set **LineClip=true**. Default is **Clip=false**.

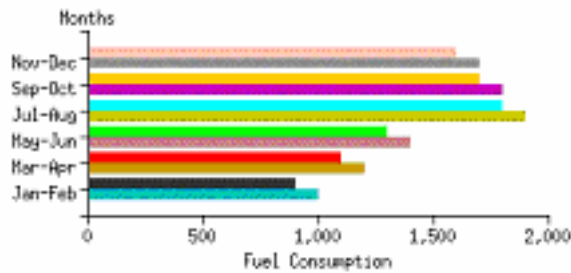
Vertical bar chart with High and Low values indicated.

Horizontal Bars (HorizBar)



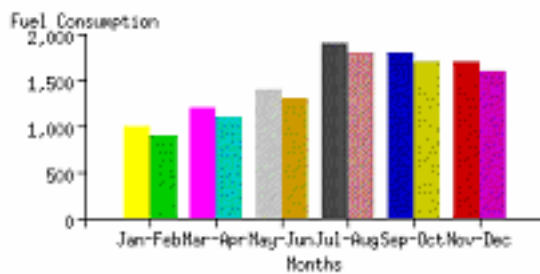
Displays each data series **horizontally** in a single color. To display vertically, (sometimes called a column chart). To display different colors for each bar in a series, use **IndBar** (horizontal) or **IndColumn** (vertical).

Individually Colored Bars (IndBar)



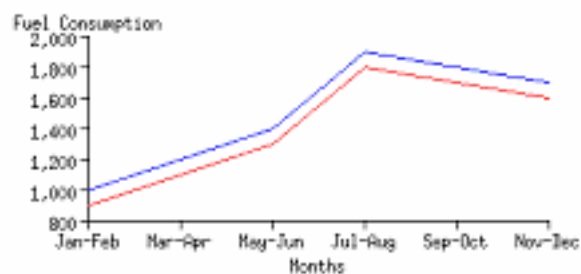
Horizontal bar chart that uses a separate color for each bar.

Individually Colored Columns (IndColumn)



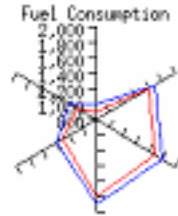
Vertical bar chart that uses a separate color for each bar.

(LabelLine)



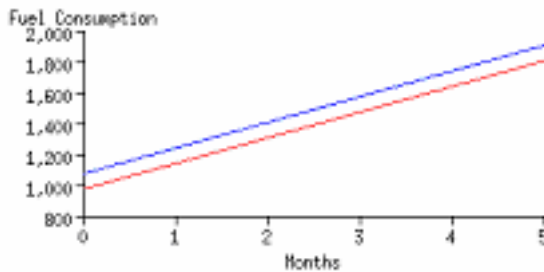
Displays data values as lines on a graph, with user-defined labels on the X-axis.

Polar Chart (Polar)



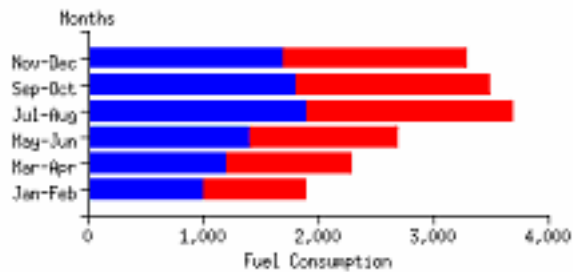
A chart that looks like a radar screen. Plots only one data value, but the scale is determined by all the data.

Regression (Regress)



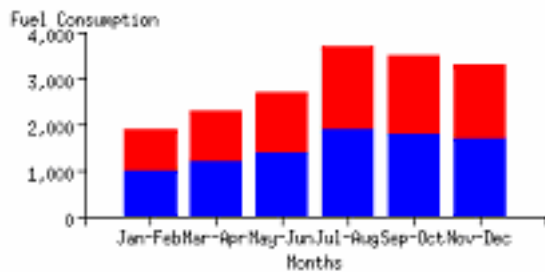
A chart that plots OLS regression for data values.

Stacked Bars (StackBar)



A chart that stacks data values horizontally.

Stack Column Chart (StackColumn)



A chart that stacks data values vertically.

Stick Chart (Stick)

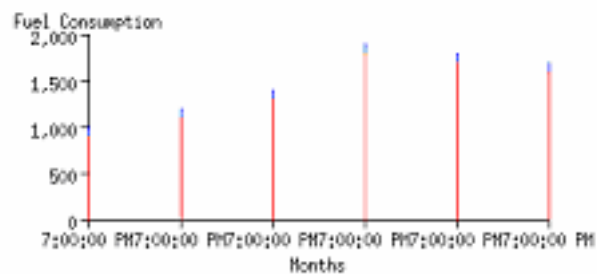


Chart that draws a vertical line to the Y-axis height of each data value.

StickBar Chart (StickBar)

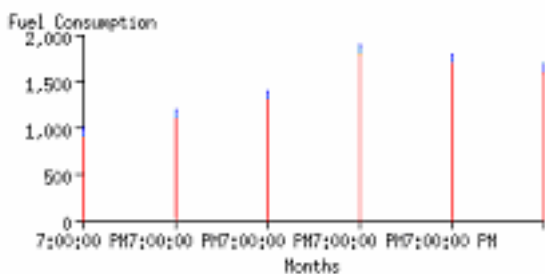


Chart that draws a vertical line to the Y-axis height of each data value.

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).
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).

Property	Default	Description
Background.SubTitleFont.name	Times New Roman	Uses available values from <code>java.awt.font</code> .
Background.SubTitleFont.size	12	Uses available values from <code>java.awt.font</code> .
Background.SubTitleFont.style	plain	Uses available values from <code>java.awt.font</code> .
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 <code>java.awt.font</code> .
Background.TitleFont.size	12	Uses available values from <code>java.awt.font</code> .
Background.TitleFont.style	plain	Uses available values from <code>java.awt.font</code> .
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).
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 will be 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).
X.Axis.LabelColor.Red	0	RGB value of red (0-255).
X.Axis.LabelFont.Name	Times Roman	Uses available values from java.awt.font.

Property	Default	Description
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 .
X.Axis.Plotarea.LIX	0.2	Shifts the horizontal position of the axis start on the canvas. Negative values shift left, positive shift right.

Property	Default	Description
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 Y 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.LIX	0	Lower x left corner $0 < y < 1.0$, 1.0 = full width
Legend.LIY	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 the for individual colors). For example, if a Pie has six slices but only five favorite colors specified, the favorite colors will not be 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 [Appendix A: Default Chart Properties and ALF Attributes](#).

```
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=RemembertosetXaxisitle!
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 of similar charts, you may find it convenient to modify the default properties as a base template.

The web application contains an instance of **edx_servlet.jar** in the **lib** directory of the EAR file **ear-eastatement-ear**. If you edit **ChartDefaults.properties**, then add the edited version of **edx_servlet.jar** to your classpath.

Unjar the EAR file, and then unjar **edx_servlet.jar** to find **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, 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 Billing 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*.

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 Siebel 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 the Simulator .

Publishing Charts

The Billing 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 Billing 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, in this example **NatlWireless**.
- 2 Create, configure, and run an Indexer job to index your dataset.
- 3 Publish an HTML web view, in this example

The screenshot shows the 'edocs' web application interface. On the left is a blue sidebar with buttons: Browse, Create, Fetch, Delete, and Help. The main content area is titled 'Create a version set for HTML'. It contains the following fields and buttons:

- Application:** A dropdown menu with 'NatlWireless' selected.
- View Type:** A text field containing 'HTML'.
- View Name:** A text field containing 'LocSummaryChart'.
- DDF File:** A text field containing 'atlWireless\NW_LocSummary.ddf' with a 'Browse...' button to its right.
- ALF File:** A text field containing 'atlWireless\NW_LocSummary.alf' with a 'Browse...' button to its right.
- HTML Template:** A text field containing 'atlWireless\NW_LocSummary.htm' with a 'Browse...' button to its right.
- At the bottom are three buttons: 'Submit', 'Clear', and 'Help'.

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 will display 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, for example
 C:/EDCSbd/samples/NatlWireless/LocChargeSummary_0.properties
 This file must have an extension of ***.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 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

The previous sections describe how to use the Siebel 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 be creating 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:portNum/eStatements/Chart?app=Charter\
&ddn=MyApp&viewName=MyView&W=200&H=300\ &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 Billing Manager, with comments on where and how to begin customization.

To customize your servlet, you will need to 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 Billing Manager 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();

```

You will 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 will determine 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,viewName));
```

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 [Example: Custom Error Message](#).

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 c1 = new ChartClient(chartProps, width, height);
c1.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,viewName));
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 c1 = new ChartClient(chartProps, width, height);
c1.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 Billing Manager Publisher? A Chart view requires a chart properties file; make sure you have published the one associated with your Billing 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 will 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's chart dialog box offers three, , and Pie. However, the Composer will create only a Pie chart by default. 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 you will need to set the **HidePieLegend** property to 1. This will only work 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 [Display Devices and xvfb](#) and Configuring a Headless Server for Charting.

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.

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

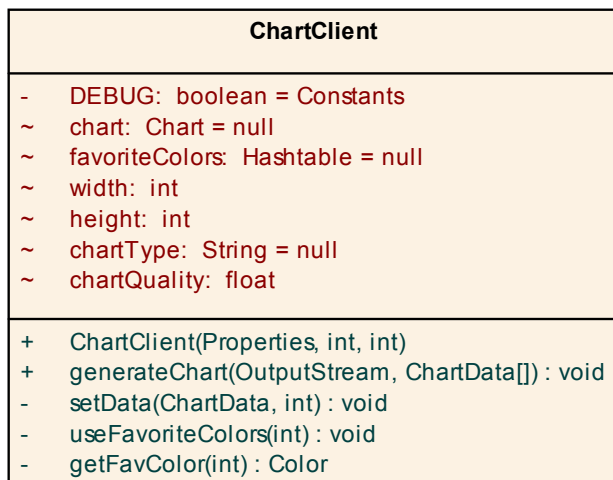
Contains a constructor and methods to draw a chart as a JPEG graphic.

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

Type	Description
	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
<code>properties</code>	Default property list. See <code>java.util.Properties</code> .
<code>canvasWidth</code>	Integer specifying the chart width in pixels.
<code>canvasHeight</code>	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
<code>out</code>	Defines an output stream, for example to generate the output for a servlet response object.
<code>data</code>	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 *Billing Manager API Specification*.

Class 3Diagram

ChartData
- DEBUG: boolean = Constants ~ XAxisTitle: String = null ~ YAxisTitle: String = null ~ labels: String[] = null ~ xValues: double[] = null ~ yValues: double[] = null ~ highValues: double[] = null ~ lowValues: double[] = null ~ closeValues: double[] = null ~ seriesName: String = null
+ ChartData(String[], double[]) + ChartData(String[], double[], String, String) + ChartData(String[], double[], String, String, String) + ChartData(String) - addPercentToLabel(double[], ArrayList) : String[] + ChartData(String, double[], double[], double[], double, String, String) + getLabels() : String[] + getXValues() : double[] + getCloseValues() : double[] + getHighValues() : double[] + getLowValues() : double[] + getYValues() : double[] + setXAxisTitle(String) : void + getXAxisTitle() : String + setYAxisTitle(String) : void + getYAxisTitle() : String + getSeriesName() : String + toString() : String - doubleArray2String(String, double[]) : String - stringArray2String(String, String[]) : String

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.

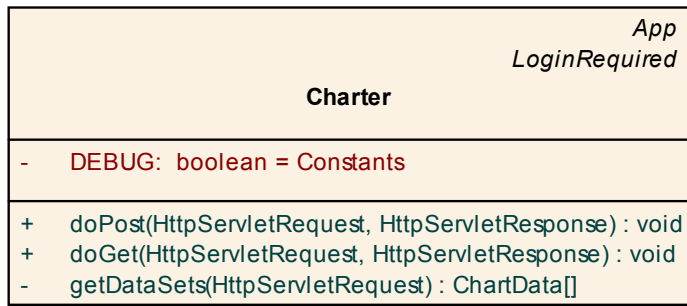
Parameters

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 the Simulator .
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 *Billing Manager API Specification*.

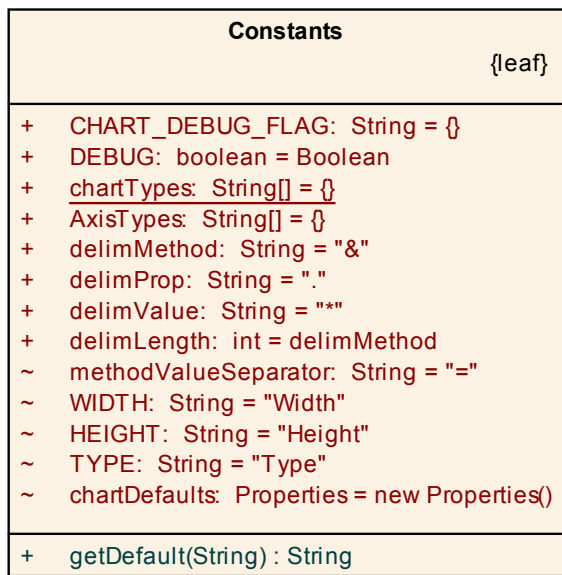
Class Diagram



Class Constants

Contains a constructor and default fields for defining custom chart parameters. For details, see the *Billing Manager API Specification*.

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 *Billing Manager API Specification*.

Class Diagram

PublisherCommon	
+	WebRoot: String = getRootDir
	APPRoot: String = getRootDir
	WEB_VIEW_TYPES: String[] = {}
-	getRootDir(String) : String
+	getResource(String, String) : byte[]
+	getVersionSet(String, String, String) : IVersionSetReader
+	<u>timeDisplayPublisher(long) : String</u>

Class PublisherWrapper

Contains a constructor, fields, and methods to retrieve chart properties as an input stream. For details, see the *Billing Manager API Specification*.

Class Diagram

PublisherWrapper	
+	CHART_TYPE: String = "CHART"
	CHART_PATH: String = "CHART_PATH"
	CHART_PROPS_EXT: String = ".properties"
+	getChartPropsStream(String, String) : ByteArrayInputStream

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](#). For details, see the *Billing Manager API Specification*.

Class Util

Utility class to support the Charting API. For details, see the *Billing Manager API Specification*.

Class Diagram

Util
- <u>DEBUG: boolean = Constants</u>
+ getColor(String, Properties) : Color + getFont(String, Properties) : Font - inRGBRange(int) : boolean + arrayListToStringArray(ArrayList) : String[] + arrayListToDoubleArray(ArrayList) : double[] + getBoolean(String) : boolean + inArrayList(ArrayList, String) : boolean + debugProps(Properties) : void + getPropVal(String, Properties) : String + <u>getFloat(String) : float</u>

Default Properties and Attributes

See ChartDefaults.properties in edx_servlet.jar, and NW_LocSummary.alf in estatment/samples/NatlWireless.

15 Internationalization (i18n)

About Internationalization in Billing Manager

Multi-byte characters are supported in Billing Manager by using struts, which provide internationalization support.

Struts use resource bundles to store application resources based on the user's locale. Each resource file contains keys with their values, so the application can load desired values from the selected locale.

The user interface allows each user to change the language used for their session. All the display texts, messages, warnings and image locations must be placed in the required resource file and their keys must be used in the required files.

Architecture

Message Resources

Message keys are divided in to three sections in the application properties file.

They are;

- Presentation files (UI)

These files are JSPs, tile definitions xml, validations xml, Struts-config xml

- Views (published for DDNs) (Not applicable for CBM B2C, B2B CSR applications)

Custom framework to present data. These files often contain short messages and headers

- Backend – JAVA files

Model and Business Logic of Struts.

Tile Definitions

Tile definition files contain text messages to display in the JSP templates. These text messages are usually the page title, page name, related tab/sub tab names etc.

File Naming Convention

Tile definitions in the Billing Manager application are grouped based on related functionality, and stored in separate property files respectively.

For example, the followings tile definitions found in *tiles-defs-addressbook.xml* file:

```
main.addressbook.cab.add
main.addressbook.cab.addnew
main.addressbook.cab.delete
```

Properties for the constants defined in the above tile definitions are stored in the following property files based on the belonging module.

Property File Name	Supporting Language
tiles.addressbook_en.properties	English
tiles.addressbook_fr.properties	French
tiles.addressbook_es.properties	Spanish

Property names are defined with appending prefix "tiles" and unique postfix. This makes the property keys unique.

```
tiles. <tile definition name>.<unique name> = [value]
```

For example, the following tile definition uses a constant called title. Title is loaded from the property file using the key **tiles.main.addressbook.cab.delete.title**.

```
<definition name=".main.addressbook.cab.delete"
    extends=".template.popup">
    <put name="title"
    value="tiles.main.addressbook.cab.delete.title"
    type="string"/>
    <put name="module1"
    value=".module.addressbook.cab.delete"/>
</definition>
```

The JSP template files where those constants are displayed are required to be modified as follows:

```
<tiles:useAttribute name="title"/>
<bean:message key="<%=title.toString()%>"/>
```

Tab names defined in the tile definition are categorized based on the module. For example, top-level tab names stored with the following key names:

```
tiles.tabs.accountTab
tiles.tabs.statementsTab
tiles.tabs.paymentsTab
```

JSPs

JSP files contain display texts, images, alt texts etc. Display texts in JSP files are set through struts bean message tag, as shown in the following example:

```
<bean:message key="jsp.administration.company_usersJsp.searchForUser"/>
```

Images are set through struts html image tags (input type), as shown in the following example:

```
<html:image  
  pagekey="image.common.button.submit"  
  altkey="jsp.login.loginJsp.button.submit.alt"/>
```

Images are set through struts html img tags, as shown in the following example:

```
<html:img page key="image.common.button.cancel"  
          altkey="jsp.common.alt.cancel"/>
```

File Naming Convention

Display texts used in JSP files are stored in property files defined based on related module:

```
jsp.<module name>_<locale>.properties
```

Property Naming Convention

Property key names used in JSPs are defined in following convention:

```
jsp.<module name>.<jsp name>.<unique key> = [value]
```

For example, the login JSP page contains following key for user id:

```
jsp.login.loginJsp.label.userId
```

Validation XML

Validation XML file contains error messages to display as a result of validating the input fields of the user pages.

File Naming Convention

A single property file stores error messages defined in the validator xml file.

```
validator_<locale>.properties
```

Property Naming Convention

Property keys for error messages in the validation xml are defined in the following naming convention.

Module specific keys

```
validator.<module name>.<unique key> = [value]
```

Common Keys

`validator.common.<unique key> = [value]`

For example, an account number constant in the payment module is defined in validation xml as:

```
validator.payment.accountNumber
```

View

Display texts in the views are translated by a post processor. HTML views are translated using message post processor, `MessageResourcesPostProcessor`, while XML type of views are translated through XSLT post processor, `XSLTDownloadPostProcessor`.

Support for New Languages

Locate the `RIConfiguration.properties` file which is packed with `ear-tbm-b2b.ear` each application EAR file. When you extract the EAR file there is a jar file called `tbm-cfg.jar` in the lib directory, which contains the `RIConfiguration.properties` property file. `RIConfiguration.properties` contains the "`ps.riconfig.LANGUAGES`" key which defines the internationalized languages.

```
# Languages
# This property contains the language id and the key of the languages that
# needs to display in the UI
ps.riconfig.LANGUAGES=[en,
jsp.profile.change_languageJSP.language.english];[fr,
jsp.profile.change_languageJSP.language.french];[es,
jsp.profile.change_languageJSP.language.Spanish]
```

These are the locale specific keys (en for English, fr for French, es for Spanish) and each associated with a key for display name in the change language UI. To add a new language, use the following steps:

- 1 Modify the entry in '`RIConfiguration.properties`' file to reflect new language.

```
ps.riconfig.LANGUAGES=[en,
jsp.profile.change_languageJSP.language.english];[fr,
jsp.profile.change_languageJSP.language.french];[es,
jsp.profile.change_languageJSP.language.Spanish];[de,
jsp.profile.change_languageJSP.language.german]
```

This example shows the entry to support German language

```
[de, jsp.profile.change_languageJSP.language.german]
```

- 2 Navigate to the location '`WEB-INF\classes`' in the WAR file (which is in the application ear file) , which includes the property files for the display names. Those property files are named '`ApplicationResources_XX.properties`' where XX define the locale specific key.

For example, `ApplicationResources_en.properties` is the English language key, value pairs.

Create a copy of `ApplicationResources_en.properties` into the appropriate name for the new language.

For example:- `ApplicationResources_de.properties`

- 3 Replace all the values defined in the new file with the required language. (in this example it should be replaced with the German language).
- 4 Add the newly defined language key to all the '`ApplicationResources_XX.properties`' property files in the '`WEB-INF\classes`' in the WAR file with appropriate value.

For example, in `ApplicationResources_en.properties` property file it should be `jsp.profile.change_languageJSP.language.german= German`

16 Multiple DDN

Overview

Customer Activity Manager (CAM) is a session object that reflects the current state and position of a user within the CBM system. CAM provides the web layer (that is, tiles, JSP, and action classes) with easy access to:

- currently accessible billing accounts, phones, statements, and their attributes
- currently selected billing account
- currently selected sub account (phone)
- all the information needed to render a statement, report or statement/report attributes on the screen
- in B2B applications, the current hierarchy position information

CAM describes nearly all its functions through interfaces, and realizes those interfaces by locating the name of the class implementing the interface in a configuration file, and then instantiating an instance of that class. Configurable interfaces include:

Property	Interface/Meaning	Default*
cam.DDNMap	IDDNMap	<cppre>.DDNMap
cam.Customer	ICustomer	< cppre >.Customer
cam.BillingAccountManager	IBillingAccountManager	< cppre >.BillingAccountManager
cam.AccountSummaryManager	IAccountSummaryManager	< cppre >.AccountSummaryManager
cam.AccountSummary	IAccountSummary	< cppre >.AccountSummary
cam.CustomerAccount	ICustomerAccount	< cppre >.CustomerAccount
cam.AuthorizationManager	IAuthorizationManager	< cppre >.AuthorizationManager
cam.UserProfileManager	IUserProfileManager	< cppre >.UserProfileManager
cam.DateComparator	IDateComparitor	< cppre >.DateComparator
cam.InstanceInitialization	If true, loads all summaries for accounts at CAM initialization.	false
cam.StatementDateFormat	Format for statement date in summary information.	"MMMM dd, yyyy"

* < cppre > = com.edocs.application.tbm.cam

These interfaces allow the CBM web layer to operate with complex enrollment/login systems, and operate without using BSL.

About Multiple DDN Support

CAM supports multiple DDNs. For version 5.1, CAM was extended to:

- Correctly support the IDDNMap interface

- Correctly support multiple summary info instances
- Cleanup the code to partition DDNs
- Changes in both SessionUtils and DisplayTiledView

There are three kinds of multiple DDN models:

- Split Stream, Single Account
- Account Level Stream Mapping
- Customer Level Stream Mapping

These three models may ALL be implemented in the same custom application. This section describes each of these techniques.

Split Stream, Single Account

Many Telecommunication companies print either limited Call Data Records or no Call Data Records (CDRs) on invoices sent to customers. The value of on-line presentment and reports are limited if we don't show all CDRs, so customers frequently opt for creating a second billing stream that contains the complete CDR data. The result is:

- **A Summary Stream** – provides the information contains the information that shows on the printed invoice,
- **An Itemization Stream** – provides the CDR data used to produce the summary stream.

The two streams have the same account number, statement date, and sometimes invoice number. This data is used to "link" the two streams for presentment view.

Account Level Stream Mapping

Customers may have several data streams, each formatted differently, each from a different billing system. In this case, a customer specific algorithm determines the data stream that contains a particular account. The algorithm is usually either:

- A database lookup from account number to stream indicator, or
- An algorithm that parses an account number to find the appropriate stream

CBM needs to provide a mechanism for mapping an account to a DDN through either lookup or algorithm.

Customer Level Stream Mapping

In some cases, all the accounts associated with a particular customer are contained in a single bill stream. In this case, some customer specific algorithm determines the data stream associated with all the accounts for that particular customer. The algorithm is usually either based upon:

- a database lookup, or

- Geographic data (like zip code).

High Level Architecture

High-Level Description

The changes to support multiple DDNs fall into several categories:

- Configuration Extensions (configuration files and tiles).
- Supporting Summary information for each Customer Account.
- Mapping Abstract to Realized DDNs.
- Miscellaneous Changes/Updates.

This section provides background information on CAM and the DDN architecture, and then describes each of the major areas outlined above.

CAM DDN Background

Abstract vs. Realized DDNs

One of the most confused aspects of CAM and view presentment is the notion of an abstract versus realized (sometimes called real) DDN.

DDN names are configurable. That means that the Administrator chooses the DDN names and puts them in the app-config.properties file. Then Administrator should create DDNs with the same names through the command center. This feature provides flexibility to define and create customer specific DDN.

Previously, there was a single DDN that was referenced in a property file, which is referred to as DDN lookup.

Version 5.1 of CAM introduces the concept of an Abstract DDN. Previously, every DDN lookup used a DDN mapper object, which provides the opportunity to change the lookup string to just about anything, based on the account number and an object passed in. This changed string was used to look up the realized (real) DDN in the application configuration (app-config.properties) property file. Before version 5.1, realized DDN names were stored under the key DDN.<abstract ddn name>. The DDN Mapper is considered just a pass through, adding the DDN in front of the name passed in.

The Important Definitions

Abstract DDN – The name given to a type of data stream, index values, and group of associated views that is used throughout an application to obtain information and render information from a data stream with those characteristics. The abstract DDN name refers to the type of usage and view naming convention for a data stream. It does NOT refer to the structure of the data or the structure of the views themselves.

Realized DDN – The name given to a specific implementation of a data stream, index, and views. The realized DDN name is the name you used when creating an “application” in the command center.

Mapping – The process of converting an abstract DDN to a realized DDN at runtime, allowing an application to access information and render information without knowing the detailed structure of the data stream, index, or views. Mapping is implemented in the IDDNMap interface of CAM.

Configuration Extensions

Multiple DDNs introduce three categories of DDN:

- **Application default abstract DDN Name** – Stored in app-config under APP_DDND. The value here is used to define which of the abstract DDNs may be used for getting summary information.
- **Billing account default DDND** – The DDND on which all getSummaryInfo calls for a specific billing account will be performed.
- **Billing account alternate DDNs** – One or more DDNs that are not the billing account default DDND, but for which docids are searched using the account number and invoice date.

App-Config DDND Naming Conventions

The released app-config.properties file uses the same DDND for both billing account level and service agreement level (phone itemization) views. With Multi-DDND support, the following abstract DDND names are used for the account level summary and phone level itemization data.

- **DDND_APP = Summary** – Indicates that all getSummaryInfo calls will be against the Summary DDND for all streams.
- **DDND.Summary<.optional distinguishing characteristic>** -- Used for all Billing Account level views and retrieving summary information.
- **DDND.Itemization<.optional distinguishing characteristic>** -- Used for all Service Agreement level views.
- **DDND.Unbilled<.optional distinguishing characteristic>** -- Used for unbilled live data views.
- **DDND.Reports<.optional distinguishing characteristic>** -- Used for batch report views.
- **DDND.AddressBook** – Used for address book views.

Note that both the Summary and Itemization views can be pointing to the same realized DDND for cases where there is a single data feed containing both summary and itemization information.

Also note that the distinguishing characteristic and its preceding are optional. This allows for cases where all accounts or all items with the implementation selected distinguishing characteristic use the same DDN for a particular function. In versions before 5.1, AddressBook is a good example of that.

The generic format for a DDN entry in the app-config.properties file is:

```
DDN.<data_type_name><.optional_distinguishing_characteristic>
```

Where:

- **DDN.** – indicates that this is a DDN definition in the app-config file.
- **data_type_name** – is the abstract reference name used in tiles definitions to indicate the “type of data” this abstract DDN supports.
- **optional_distinguishing_characteristic** – is the unique account level characteristic that determines which realized DDN among all the DDNs supporting the “type of data” that will render a view for a particularly selected account.

Tiles Definitions

The tiles definitions for statement views reflect the new convention of Summary and Itemization for abstract DDN names.

Abstract DDN-names simply specify the data_type_name in the tiles definition. This is because the IDDNMap implementation calculates then DDN based upon account data. The generic CBM release simply specifies “Summary” and “Itemization” as the DDN names.

This data_type_name only description enables a very important feature: IT IS POSSIBLE TO USE ONE SET OF PAGES TO HOST VIEWS FOR ANY NUMBER OF DDNS THAT IMPLEMENT THE SAME VIEW TYPES, EVEN IF THOSE VIEWS LOOK COMPLETELY DIFFERENT and work off completely different data streams.

For example, if you build an application that supports both fixed and mobile invoices, you can create a single page set for both, supporting what appears to be a consolidated view of the bill. Presentment consultants simply configure the views so that views that display the same “kind” of data have the same name and appear under the same abstract DDN name. The realized DDNs and the underlying bill stream format can be completely different!

The user CAN optionally specify the abstract DDN with the distinguishing characteristic. This allows UI developers to be clear about their intent, so that if one of the billing streams in a multiple DDN application has a special page, the tiles definition truly indicates that that page is special.

A Tiles definition in a multiple DDN situation might look like this:

```
*** insert tiles definition from tele2 here ***
<..... extends “view.Display.Definition”>
  <put name=“DDN” value=“summary”/>
  <put name=“viewName” value=“Summary_View”/>
< .../>
```

This single DDN could work across data streams defined in the configuration file as:

```
DDN.summary.mobile = "mobile_summary"  
DDN.summary.fixed = "fixed_summary"
```

Where both DDNs had a view named `Summary_View`. Create a consistent naming scheme across all DDNs and one set of tiles can present both.

Examples

App-Config Example

The following example of an `app-config.properties` file shows a system with three DDNs. Mobile data has a summary and itemization DDN, fixed data has a single stream that contains both summary and itemization data.

```
APP_DDN = Summary  
DDN.Summary.mobile = mobile_summary  
DDN.Itemization.mobile = mobile_itemization  
DDN.Unbilled.mobile = unbilled  
DDN.Reports.mobile = mobile_reports  
  
DDN.Summary.fixed = fixed_singlestream  
DDN.Itemization.fixed = fixed_singlestream  
DDN.Unbilled.fixed = unbilled  
DDN.Reports.fixed = fixed_reports
```

In this example, `APP_DDN` describes the abstract name of the DDN used for searches on indexed (`getSummaryInfo`) data.

The mobile and fixed suffixes are determined by looking at billing account information to decide if the account is a mobile account or fixed account.

Mobile data is provided as two data streams, one with account level summary information, the other with call detail (CDR) data for all the phones in an account. Both streams contain account level separators that provide account number, invoice number, and date. The itemization stream is subindexed to locate mobile phone data quickly.

Fixed data is provided as a single stream that includes both account summary information and call detail (CDR) data for all the phones in an account. The phone level data is subindexed to locate fixed phone data quickly.

The implementation of `IDDNMap` works as follows:

- 1 If the default DDN is being requested, it looks up the abstract name of the default DDN in `app-config` through `APP_DDN`.
- 2 It then determines if the account is fixed or mobile by querying the supplied billing account.
- 3 It creates a property DDN.`<abstractname>.<fixed|mobile>` and returns the realized DDN name.

Extending the Concept to Other Customer Situations

This concept is easily extended by changing step 2 in the preceding example with a lookup in another table.

For example, one way to decide which billing system to query can be based upon a portion of the account number. If we implement this algorithm in a custom IDDNMap implementation, then we can generate a DDN using:

DDN.<abstractname>.<bs1|bs2|bs3|bs4|bs5> for each of five billing systems!

Using the Naming Convention for Single DDN Implementations

The same naming convention may be used in Single DDN implementations. In this type of implementation, the Summary and Itemization views map to the same DDN. An implementation of IDDNMap hardcodes the distinguishing characteristic to MAIN. This results in the following app-config.properties entries:

```
APP_DDN = Summary
DDN.Summary = TBM
DDN.Itemization = TBM
DDN.Unbilled = UNBILLED
DDN.Reports = REPORTS
DDN.AddressBook = ADDRESSBOOK
```

Specifying Live Data DDNs

Live Data DDNs are treated just like other DDNs, with the same lookup mechanism, following the same pattern for properties.

Supporting Summary Information for Each Customer Account

Since it is possible for a single invoice to have data spread across more than one realized DDN, the multi-DDN functionality uses a convention for performing data lookups.

Summary information, invoice date, and the document id associated with billing account level data is always obtained on the Summary DDN.

Detail Information and the document id associated with views built against other data streams are obtained from summary information on those DDNs. DETAIL DDNs MUST ALWAYS NAME THE BILLING ACCOUNT AND DATE FIELD AND FORMAT THEM IN EXACTLY THE AS THE CORRESPONDING SUMMARY DDN.

Date Formats Must Be The Same Across all DDNs

In CBM it is critical that all date formats be configured the same across all DDNs in an application. Both CAM.Configuration and RI.Configuration require the date format that the application uses be the same.

Mapping Abstract to Realized DDN

The mapping described previously all comes together in the implementation of the IDDNMap interface.

The IDDNMap interface returns a realized DDN, based upon the abstract DDN and account number, and a flexible selection criteria object passed in. This interface is defined as (notes are shown in bold):

```
public interface IDDNMap {

    *** returns a billing account alternate DDN for a specific billing account (or billing account default ddn if you know that) ***

    /**
     * Gets an actual DDN based upon an abstract DDN,
     * an account number, and/or a set of selection criteria.
     *
     * @param abstractDDN the abstract DDN to translate
     * @param accountNumber the number of the account to translate
     * @param selectionCriteria a set of selection criteria to do the
     *                           translation on
     * @return String the real DDN based upon the selection criteria
     *
     * @throws CustomerInfoException EX_DDN_NOT_FOUND, EX_NO_DDN_MAP,
     *                               EX_DDN_INVALIDARG
     */
    public String getDDN(String abstractDDN,
                        String accountNumber,
                        Object selectionCriteria) throws
CustomerInfoException;

    *** returns a billing account alternate DDN for the CURRENT billing account as selected in CAM ... This should be deprecated. ***
}
```

```

/**
 * Gets the DDN for the current account using whatever selection
 * criteria are the default for that account.
 *
 * @param abstractDDN the abstract DDN name to translate
 * @return String the DDN based upon the selection criteria
 *
 * @throws CustomerInfoException EX_DDN_NOT_FOUND, EX_NO_DDN_MAP,
 *                               EX_DDN_INVALIDARG
 */
public String getDDN(String abstractDDN) throws CustomerInfoException;

```

***** deprecated method that can be removed.*****

```

/**
 * Gets the default DDN.
 *
 * @deprecated
 * @see getDefaultDDN
 */
public String getDDN() throws CustomerInfoException;

```

***** returns the application default abstract DDN name. This is used ONLY internally to billing accounts to get the “property name” of the default DDN it actually returns “part” of a DDN so the call name is somewhat of a misnomer *****

```

/**
 * Gets the default DDN from the app-config.properties file
 * where the property key is APP_DDN.
 *
 * @return
 * @throws CustomerInfoException EX_DDN_NOT_FOUND, EX_NO_DDN_MAP,
 *                               EX_DDN_INVALIDARG
 */
public String getDefaultDDN() throws CustomerInfoException;

```

***** returns the billing account default DDN *****

```

/**
 * Gets an actual default DDN based
 * on an account number and/or a set of selection criteria.
 *
 * @param accountNumber the number of the account to translate
 * @param selectionCriteria a set of selection criteria to do the
 *                           translation on
 * @return String a DDN based upon the selection criteria
 *
 * @throws CustomerInfoException EX_DDN_NOT_FOUND, EX_NO_DDN_MAP,
 *                               EX_DDN_INVALIDARG
 */

```

```
public String getDefaultDDN(String accountNumber,  
                             Object selectionCriteria) throws  
CustomerInfoException;  
}
```

The new default implementation of `IDDNMap`, called `MultiDDNMap.java` accesses the `App-Config.properties` file according to the specifications outlined above. The object passed in is the `IBillingAccount` interface. This interface is extended to support a method `getSystemType` which returns the optional distinguishing characteristic portion of the name against used to look up the realized DDN in the `App-Config.properties` file.

If `App-Config.properties` doesn't contain a mapping with that full name, then it looks under the DDN.<abstract name> to find the realized DDN. This allows backward compatibility with existing PS configurations.

Index

A

- ALF, 160, 165, 167, 169, 170
- Architecture
 - Account Resolver, 127
 - Action Classes, 18
 - BSL, 19
 - Business Services Layer, 19
 - CAM, 18
 - Compose, 128
 - Customer Account Management, 18
 - Email Composer, 127
 - Email Dispathter, 127
 - ePayment, 19
 - eStatement, 19
 - Group, 128
 - Hierarchy, 19
 - LDE, 18
 - Live Data Extraction, 18
 - Payment Gateway, 19
 - SAF, 18
 - Security Access Framework, 18
 - View Post Processing, 18
 - View Processing, 18
- Auditing Data Streams, 158
- AutoIndexVolAccept task, 156
- Axis Properties, 180

B

- Background Properties, 178
- Bar, 173

C

- Chart
 - creating, 165
- Chart view, 167, 191

- ChartClient, 192
- Charter, 192
- com.edocs.app.chart, 188
- Communications eBilling Manager
 - Action EdocsAction, 18
 - Action EdocsActionForm, 18
 - BSL, 22
 - Bulk Load & Synchronization, 24
 - Business Services Layer, 22
 - Business Services Layer-Service Provider interface, 23
 - CAM, 22
 - Content Retrieval, 20
 - Customer Account Manager, 22
 - Display Tiles Controller, 23
 - Indexing, 21
 - LDE, 23
 - Live Data Extraction, 23
 - Payment Gateway, 21
 - Payment Utils, 21
 - RBAC, 24
 - Role Based Access Control, 24
 - SAF, 24
 - Security Access Framework, 24
 - View Post Processing, 24
- Composer, 160, 165, 166, 167

D

- DDF, 165
- DDN, 157

E

edx_servlet.jar, 188

F

Favorite Colors Properties,
184
FIELD, 104, 105
Fonts, 161

G

General Properties, 178
GROUP, 105

H

HiLoBar, 171
HorizBar, 173, 174
HorizHiLoBar, 171
HTML template, 165, 166

I

IndBar, 173, 175
IndColumn, 173, 175
Indexer, 160
Indexer job, 147
Indexer task, 147
Input and output arguments,
149, 154
IXLoader task, 147

J

JOB_NAME, 157

L

LabelLine, 175
Legend Properties, 183
Line, 174

M

Mapping
tables as charts, 165

N

NatlWireless.alf, 170
NW_LocSummary.alf, 165

O

Oracle *sqlplus* utility, 148

P

Parameters, 190
Pie, 165, 169, 172
Plot Area Properties, 179
Polar, 176
PREVIOUS_STATUS, 157
PublisherWrapper, 203

Q

Queue, 128

R

Regress, 176

S

Scanner, 147
Scanner task, 147
Servlet, 160
Shell command task, 149
SHELL_INPUT, 157
Siebel Communications
Billing Manager, 159
Simulator, 160, 168, 189
Speedo, 171
SQL scripts task, 148
sqlplus, 148
StackBar, 176
StackColumn, 177
STATUS, 157
Stick, 177
StickBar, 177

T

TABLE, 105

V

Verify, 158

W

war-easample.war, 188
web.xml, 192

X

xvfb, 161, 162, 164