# Customization Guide

*iPlanet BillerXpert 4.6 B2B Edition*

May 2002

Copyright © 2002 Sun Microsystems, Inc. All rights reserved.

Sun, Sun Microsystems, the Sun logo, iPlanet are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Federal Acquisitions: Commercial Software—Government Users Subject to Standard License Terms and Conditions

The product described in this document is distributed under licenses restricting its use, copying, distribution, and decompilation. No part of the product or this document may be reproduced in any form by any means without prior written authorization of Sun Microsystems, Inc. and its licensors, if any.

THIS DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

---

Copyright © 2002 Sun Microsystems, Inc. Tous droits réservés.

Sun, Sun Microsystems, le logo Sun, iPlanet sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et d’autre pays.

Le produit décrit dans ce document est distribué selon des conditions de licence qui en restreignent l'utilisation, la copie, la distribution et la décompilation. Aucune partie de ce produit ni de ce document ne peut être reproduite sous quelque forme ou par quelque moyen que ce soit sans l’autorisation écrite préalable de Sun Microsystems, Inc. et le cas échéant, de ses bailleurs de licence.

CETTE DOCUMENTATION EST FOURNIE “EN L'ÉTAT”, ET TOUTES CONDITIONS EXPRESSES OU IMPLICITES, TOUTES REPRÉSENTATIONS ET TOUTES GARANTIES, Y COMPRIS TOUTE GARANTIE IMPLICITE D'APTITUDE À LA VENTE, OU À UN BUT PARTICULIER OU DE NON CONTREFAÇON SONT EXCLUES, EXCEPTÉ DANS LA MESURE OÙ DE TELLES EXCLUSIONS SERAIENT CONTRAIRES À LA LOI.

# Contents

# Preface

This Handbook describes the customizations you can make to iPlanet BillerXpert B2B Edition.

This Preface discusses the intended audience, the organization of the Handbook, and provides a listing of typographic conventions used in this document. If you spend a few minutes looking through the Preface before reading the rest of the Handbook, you will be able to utilize the Handbook more effectively.

## Before You Begin

You need to use this manual if you are changing the user interface for BillerXpert B2B Edition or extending BillerXpert by changing the database schema or adding event handlers. When customizing BillerXpert B2B Edition, you also may need to refer to the *BillerXpert B2B Edition API and Schema Reference Manual* for information about BillerXpert's component business object interfaces and database schemas.

You will also need to have read or have available a book on iPlanet Application Server (IAS). You should read *Developing IAS Applications (Java)* for general background information about how to use the IAS programming constructs upon which BillerXpert Edition has been developed.

## Audience

This manual is written for web site developers and Java programmers who want to modify the appearance of BillerXpert B2B Edition or its operation. This manual assumes that you have experience with Java and have access to iPlanet Application Server (IAS) documentation.

## Organization

The book contains nine chapters, plsu an Appendix, as follows:

- Chapter 1, “Introduction to iPlanet BillerXpert B2B Edition Customization"

  This chapter discusses the BillerXpert B2B Edition architecture and how to customize BillerXpert. BillerXpert B2B Edition is implemented as a collection of classes and templates built on the iPlanet Application Server (IAS). You customize BillerXpert B2B Edition by modifying the classes and templates to meet the needs of your customers. You may also implement event handlers and component business object classes.

- Chapter 2, “Customization Techniques"

  This chapter presents a tutorial consisting of exercises that you can follow to learn how to customize BillerXpert B2B Edition. Even if you do not follow the tutorial step-by-step, you should skim these sections because they provide conceptual information about how to customize BillerXpert B2B Edition.

- Chapter 3, “Common Front-End Classes"

  This chapter discusses the common classes used by the BillerXpert front end to manipulate application business objects.

- Chapter 4, “Common Templates and Screens"

  This chapter introduces common templates and shows screens used by several different Servlets, all of which may be customized.

- Chapter 5, “Invoice Center Screens"

  This chapter shows the Invoice Center screens that may be customized.

- Chapter 6, “Search Screens"

  This chapter shows the Search screens that may be customized.

- Chapter 7, “User Center Screens"

  This chapter shows the User Center screens that may be customized.

- Chapter 8, “Process Manager"

  This chapter shows detailed information on each of the processes within process manager, and, provides sample customizations for those processes that may be customized.

- Chapter 9, “Currency Converter"

  This chapter shows the features within currency converter that may be customized.

# Conventions

This document uses the following conventions:

- The `monospace` font is used for sample code and code listings, Application Program Interface (API) and language elements (such as method names and property names), file names, commands, path names, directory names, Hypertext Markup Language (HTML) tags, and any text that must be typed on the screen.
- The *italic* font is used in code to represent placeholder parameters (variables) that should be replaced with an actual value, or items that require *emphasis*.
- Brackets ([]) are used to enclose optional parameters.
- A slash (/) is used to separate directories in a path. (Windows NT supports both the slash and the backslash.)

Chapter 1

# Introduction to iPlanet BillerXpert B2B Edition Customization

This chapter discusses the BillerXpert B2B Edition architecture and how to customize BillerXpert. BillerXpert B2B Edition is implemented as a collection of classes and templates built on the iPlanet Application Server (IAS). You customize BillerXpert B2B Edition by modifying the classes and templates to meet the needs of your customers. You may also implement event handlers and component business object classes.

This chapter presents the following topics:

- BillerXpert B2B Edition Architecture
- Customization Strategy
- Front-End Classes and Templates
- Front-End Files and Directories

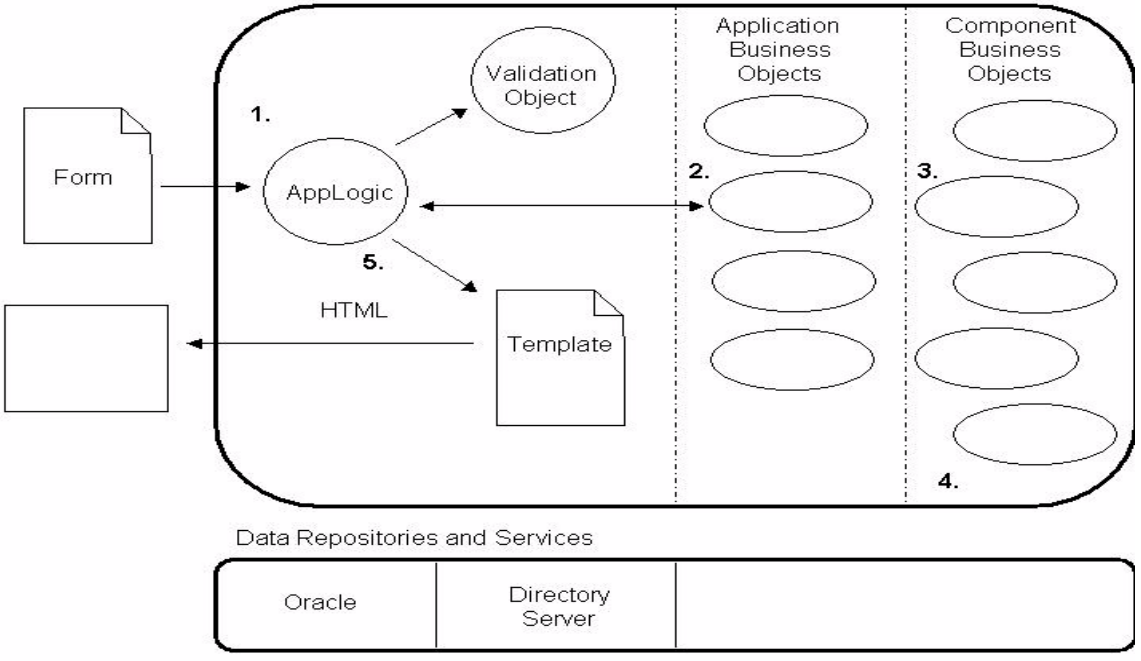
# BillerXpert B2B Edition Architecture

The entry point to BillerXpert B2B Edition is the iPlanet Application Server (iAS) Servlet. A Servlet is a set of programming instructions that accomplishes a well-defined modular task within an application. In BillerXpert, a Servlet typically takes the following actions:

- validates input from the form
- performs the business logic specified by the request
- constructs a bean object to supply data to jsp templates
- forwards a request to jsp

A Servlet creates application business objects to carry out the business logic, which in turn creates component business objects to manipulate data in a repository or communicate with a service. Figure 1-1 shows the BillerXpert B2B Edition architecture.

**Figure 1-1** BillerXpert B2B Edition Architecture



For example, consider the Sign On processing that occurs when a user logs on. By default, BillerXpert displays a form that contains fields for the user name and password. When the user submits the form, BillerXpert takes the following actions, which are numbered from 1 to 5 in Figure 1-1:

1. The Servlet retrieves the user name and password from the form and creates a validation object that verifies whether these fields are filled in, and that the constraints, such as minimum and maximum lengths, were met.
2. The Servlet creates an application business object containing this data with which it performs authentication.
3. The application business object creates a component business object; in this case, the component business object represents a customer's profile.
4. The application business object uses the component business object to retrieve the customer's profile from a data repository; in this case, repository is the Directory Server.
5. The Servlet uses the data to construct the bean object and forwards the request to the JavaServer page (JSP) to display the form. In the case of the User Center Servlet, the data is stored in BXUserCenter bean that contains the verified user name, email address, manager, etc., from the data repository.

If an error occurs, either as a result of validation or during retrieval of the customer's profile, the Servlet outputs JSP that describes the error instead of the normal processing described in the previous steps.

The previous examples noted the kinds of objects used by BillerXpert B2B Edition, which are summarized in Table 1-1:

**Table 1-1** Kinds of objects

| Object | Description |
|---|---|
| Servlet | Represents the processing of a request to BillerXpert. |
| validation object | Represents the validation logic for input data. |
| application business object | Represents the data manipulated by the servlet and methods that operate on the data. |
| component business object | Represents data in a data repository and the methods that control access to the repository. |
| Bean object | Represents data to be displayed by a JSP. |

The kinds of objects identified in Table 1-1 are implemented in Java classes. In addition to the Java classes, BillerXpert includes JSP-based templates that define the appearance of screens.

With the exception of the component business objects, all objects in Table 1-1 are part of the BillerXpert front end. BillerXpert provides source code for front-end classes, which are the classes that you can modify. BillerXpert provides class libraries for component business object classes; however, you can implement your own component business object classes, as needed.

# Customization Strategy

BillerXpert follows the iAS Servlet-implementation recommendation of using a single Servlet for a single purpose, thus keeping each Servlet small and focused on a single task. BillerXpert also separates the validation and data access into separate classes from the Servlet itself.

You are not required to follow this model; however, object-oriented programming lends itself well to this kind of application structure. For example, if you want to use additional data in application business objects, you can derive your own class to access the data. Likewise, if your form accepts different inputs, you can derive your own validation class to verify them.

Front end classes are generally useful for any kind of bill presentment application. This includes the BillerXpert common and core classes.

You can customize BillerXpert front-end in the following ways:

- modify an JSP-based template to change the appearance of a screen; this is probably the most common task that you need to perform
- modify a Servlet to change the data that is displayed on a screen; this kind of customization may require changing or replacing java beans.
- modify a validation class to handle additional input data; this may involve adding additional error codes and messages to the common portion of the front-end
- modify or create a subclass of an application business class to store and retrieve data from a data repository; a component business object must already be in support the of data you wish to access
- create a new Servlet to handle additional processing requirements, such as asynchronous events.

If BillerXpert does not provide support for the data that you want to access from a data repository, you can alter the schema to support the new data and implement your own component business object class to provide access to the data.

# Front-End Classes and Templates

Front-end classes include Servlet classes, core classes from which other front-end classes are derived or define common objects such as error codes and messages, application business object classes, output classes for placing data in JSP, and validating classes. BillerXpert provides source code for all front-end classes.

Front-end classes do not include component business objects classes; source code is not provided for these classes. For information about component business object classes and their APIs, see the *BillerXpert B2B Edition API and Schema Reference Guide*.

In addition to front-end classes, BillerXpert B2B Edition provides JSP source code that you modify to change the appearance of the screens that BillerXpert displays. The following sections identify the front-end classes and JSP templates used by BillerXpert.

## Core Classes

Core classes are BillerXpert classes that are derived from corresponding iAS-supplied classes, or they are utility classes that represent objects used system-wide. Table 1-2 shows the common and core classes.

**Table 1-2** Core classes

| Class | Description |
|---|---|
| `BXBaseServlet` | Class from which all servlet classes are derived. |
| `BXBaseView` | Abstract class from which view classes are derived. |
| `BXConfigObj` | Represents NTV list entries in configuration files. |
| `BXErrorCode` | Defines BillerXpert error codes. |
| `BXSession` | BillerXpert session class. |
| `BXValidateAttribute` | Represent attributes, such as input fields, which can be validated. |
| `BXValidateView` | Abstract class from which classes that validate views are derived. |
| `BXValidator` | Validates BillerXpert data types. |
| `BXBaseBean` | Base class for BillerXpert bean classes. |
| `BXTemplateUtil` | Additional methods to support templates. |

# Application Logic Classes

Application logic classes implement BillerXpert Servlets. There is one class for each major feature supported by BillerXpert. Table 1-3 identifies these classes.

**Table 1-3** Application logic classes

| Class | Description |
|---|---|
| `BXBillDocServlet` | Displays an invoice detail. |
| `BXBillDocSearchServlet` | Searches for an invoice or a line item. |
| `BXBillingCenterServlet` | Presents invoice summaries to the customer. |
| `BXPayHistoryServlet` | Displays current and completed payments. |
| `BXPaymentServlet` | Displays invoices with outstanding amount; user can make payment or cancel payment. |
| `BXSignOnServlet` | Logs into online billing system. |
| `BXSignOnCSRServlet` | Can be used by a CSR to simulate a user signing on. |

| | |
|---|---|
| BXSignOutServlet | Logs off of online billing system. |
| BXUserCenterServlet | Changes customer profile information. |
| BXSilentSignOnServlet | Logs onto online billing system given a valid process manager's cookie. This scenario can occur if iAS unexpectedly restarts; the current user should not be asked to sign on again. |

# Application Business Object Classes

Application business objects implement the application logic and manipulate component business objects representing records in the database. Application business objects differ from component business objects in that an application business object uses one or more component business objects and provides a higher-level representation of data; for example, an application business object representing a customer account may combine membership information and billing information from several different component business objects. Table 1-4 shows the application business object classes in BillerXpert B2B Edition.

**Table 1-4** Application business object classes

| Class | Description |
|---|---|
| BXBillerCustAcctObj | Determines information about a customer account. |
| BXBillerCustAcctUtil | Provides utility methods for customer account objects. |
| BXBillerProfileView | Retrieves biller profile information from membership server. |
| BXBillingActivity | Manipulates billing activity information. |
| BXCustomerAcctView | Retrieves customer account information from database. |
| BXFinancialAcctObj | Determines information about a financial account. |
| BXPaymentView | Manipulates payment information, including payment methods. |
| BXSearchAttributes | Determines information about search criteria. |
| BXSignUpSession | Manipulates information provided by the user during sign up. |

| | |
|---|---|
| BXStatementView | Retrieves summaries, documents, and document details for invoices. |
| BXSummaryItem | Determines information about an invoice summary. |
| BXSummaryVector | Provides support for vectors of summaries. |
| BXUserProfileObj | Manipulates user profile objects. |
| BXUserProfileView | Allows updates to user profile in LDAP. Used by a biller in Admin. |
| BXUserServiceView | Allows updates to customer service options in LDAP. Used by Biller in Admin. |
| BXViewUtil | Provides utility methods for business objects. |

# Validation Classes

Validation class objects validate input data from a JSP form on behalf of Servlet classes. Table 1-5 shows BillerXpert's validation classes.

**Table 1-5** View and validation classes

| Class | Description |
|---|---|
| BXValidateAcctView | Validates customer account information. |
| BXValidateNewBCAView | Validates new BillerXpert account information. |
| BXValidateNewPaymentView | Validates new payment information. |
| BXValidatePayView | Validates payment information. |
| BXValidatePwdView | Validates passwords. |
| BXValidateSignOnView | Validates log-on information. |
| BXValidateSignUpView | Validates registration information. |
| BXValidateStmtSearchView | Validates search information. |

# Output Classes

Output class objects create dynamic data to be displayed in a template. Beans are used to set up dynamic data. Table 1-6 shows the output classes used by BillerXpert B2B Edition.

**Table 1-6** Template classes

| Class | Description |
|---|---|
| BXBaseBean | Base bean from which all other beans are derived. |
| BXBillDocSearchBean | Sets up dynamic data for the search template. |
| BXInvoiceCenterBean | Sets up dynamic data for the invoice center template. |
| BXCustomerAcctBean | Sets up dynamic data for customer account information. |
| BXPayViewBean | Sets up dynamic data for the make payment template. |
| BXUserCenterBean | Sets up dynamic data for the user center template. |
| BXPaymentBean | Supplies data for listing Pending and Completed payments. |
| BXPaymentSearchBean | Supplies data for Invoice Search results page (following Payment Search link). |
| IBXBillDetailBean | Supplies data for Invoice Detail. |
| IBXBDSearchBean | Supplies data for Line Item Search results. |
| BXInvoiceBean | Abstract invoice info bean. |

# JSP Templates

JSP templates create web pages that are displayed in response to a request. Table 1-7 shows the templates that are invoked directly by Servlets. These templates may invoke other templates, which are not shown in Table 1-7.

**Table 1-7** Major JSP templates

| Template | Description |
|---|---|
| invoice_center.jsp | Invoice center template. |
| change_profile_confirm.jsp | Change user profile confirmation template. |
| error_msg.jsp | Error template. |

| | |
|---|---|
| logout_msg.jsp | Sign off message template. |
| Pay_invoice.jsp | Payment account invoice template. |
| pay_account-balance_paid.jsp | Payment account balance template. |
| pay_invoice_step3.jsp | Payment confirmation template. |
| statement_current-1234.jsp | Current statement template. |
| statement_search-power.jsp | Advanced search template. |
| statement_search-results.jsp | Search results template. |
| statement_search.jsp | Search template. |
| user_center.jsp | User center template. |
| banner.jsp | Top-side navigation links of BillerXpert. |
| csr.jsp | Customer Service Representative template front-end to emulate end-users(s). |
| services.jsp | Left-side navigation of BillerXpert front-end. |

# Front-End Files and Directories

After you install BillerXpert B2B Edition, you should set an environment variable, BX_HOME, to the billXpert directory from the root of the installation. For example, if you create a user to administer BillerXpert B2B Edition and install the software in the administrator's home directory, your BX_HOME environment variable should point to the administrator's $HOME/billXpert directory.

*Note*

The BX_HOME environment variable is set when you source the billxpert_env.csh or billxpert_env.sh files.

You can locate the files that you will need to modify in subdirectories of your BX_HOME directory. Table 1-8 identifies the BX_HOME subdirectories and their contents.

**Table 1-8** Source subdirectories of BX_HOME

| Subdirectory | Contents |
|---|---|
| config | BillerXpert configuration files. |

| | |
|---|---|
| fe / src / fe | makefile for front-end source code and resulting `bxfe.zip` file. |
| fe / src / fe / common | base and common classes for the BillerXpert front end. |
| fe / src / fe / user / common | front-end definition classes. |
| fe / src / fe / user / view | additional front-end classes. |
| fe / src / fetelco | makefile for biller-specific source code and resulting `bx_biller.zip` file. |
| fe / src / fetelco / applogic | biller-specific Servlets. |
| fe / src / fetelco / view | additional classes used by biller-specific Servlets. |
| fe / templates / en / telco | biller-specific templates. |
| java | destination directory for zip files and contains `billxpertb2b.conf`. |

*Note*

**NOTE** When you make changes to source code, you must move the zip files resulting from recompilation to a directory specified in your CLASSPATH environment variable; by default the zip files are located in the java subdirectory.

You should also set a `NAS_HOME` environment variable that points to where iAS is installed. It is also set when you source the `billxpert_env.csh` or `billxpert_env.sh` files. Subdirectories contain log files and the files you need to start and stop iAS Application Server and perform other operations. Table 1-9 identifies these subdirectories.

**Table 1-9** Subdirectories of NAS_HOME

| Subdirectory | Contents |
|---|---|
| ias / bin | utilities for operating iAS. |
| ias / logs | log files containing BillerXpert output. |

Chapter 2

# Customization Techniques

This chapter presents a tutorial consisting of exercises that you can follow to learn how to customize BillerXpert B2B Edition. Even if you do not follow the tutorial step-by-step, you should skim these sections because they provide conceptual information about how to customize BillerXpert B2B Edition.

This chapter contains the following sections:

- Getting Started
- Modifying makefile Files
- Changing JSP Templates
- Displaying Data
- Using Contexts
- Invoking Servlets
- Configuring Table Definitions
- Implementing Component Business Objects

## Getting Started

BillerXpert B2B Edition consists of JSP templates and Servlets and related classes that you can modify to achieve the presentation and behavior for web-based bill presentment and payment applications. In the simplest case, you can modify templates to change the appearance of bill presentment and modify Servlets to specify the data you want to use.

You are not restricted to simple kinds of customization. For example, you can change the flow of an application by changing the order in which Servlets are invoked. You can also add templates and Servlets to provide additional behavior.

BillerXpert B2B Edition also provides component business object interfaces to data repositories, such as billing data in an Oracle database or membership data in Directory server. You may use these interfaces to manipulate predetermined kinds of data within an application. You can also implement your own interfaces to manipulate additional kinds of data.

To make the task of populating data repositories easier, BillerXpert B2B Edition provides a program for loading XML-encoded data. This tool allows you to define the structure of data being loaded into BillerXpert B2B Edition. You can modify this program to load data that BillerXpert does not directly support.

The following sections discuss each of these kinds of customization and provides exercises that allow you to experiment with them. Before you start the exercises, you must set up your environment by sourcing `billxpert_env.csh` if you are using a c-shell or by executing `billxpert_env.sh` if you are using a Bourne shell. You will probably want to follow a methodology similar to the following one as you make changes to Java source files and compile them:

1. Locate the source file in one of the directories identified in Table 1-9.
2. Create a backup copy of your source file before you modify it.
3. Modify the source file and compile it.
4. Change to the directory that contains your zip file and create a backup copy of the zip file before you overwrite it with the changed file.
5. Copy the new zip file into the directory.
6. Stop and restart iAS.

These procedures assume that your makefile does not copy zip files. If it does, you should make a backup copy of your zip file before compiling.

If you are only changing a JSP template file, you only need to follow steps 1 and 2. If you modify one of the configuration files, you must stop and restart iAS (Step 6) in addition to steps 1 and 2.

These procedures assume that your makefile does not copy zip files. If it does, you should make a backup copy of your zip file before compiling.

If you are only changing an JSP template file, you only need to follow steps 1 and 2. If you modify one of the configuration files, you must stop and restart IAS (Step 6) in addition to steps 1 and 2.

# Modifying makefile Files

The BillerXpert source consists of general-purpose Java classes that are useful in building any front-end application. It has its own makefile, which you must modify to specify the location of the Java zip files and the location of the java compiler. The makefile uses environment variables that specify the locations of IAS and BillerXpert. Table 2-1 shows these environment variables.

**Table 2-1** Common environment variables used in makefile files

| Variable | Contents |
|---|---|
| BX_HOME | directory in which BillerXpert is installed. |
| NAS_HOME | directory in which IAS is installed. |
| JAVA_HOME | directory in which Java is installed. |

## Exercise 1

In this exercise, you modify the makefile so that you can compile changes you make to Java classes. Note that a sample makefile, `$BX_HOME/fe/src/fetelco/makefile` is specific to the telephone industry and is being used for the purpose of the exercise only.

### *Files*

In this exercise, you will modify the following files:

| | |
|---|---|
| `$BX_HOME/fe/src/fe/makefile` | front-end Java makefile file |
| `$BX_HOME/fe/src/fetelco/makefile` | telco-specific Java makefile file |

### *Tutorial Steps*

You will modify the makefile files to specify the correct `JAVA_HOME` and `CLASSPATH` variables.

**1.** Open the `$BX_Home/fe/src/fetelco/makefile` file and view its contents.

The Contents should look something like this:

```
JAVA_HOME    = /share/builds/components/jdk/1.2.2/SunOS
TELCO_HOME   = $(BX_HOME)/fe/src/fetelco
PKGDIR     = $(TELCO_HOME)
```

```
JAVAC = $(JAVA_HOME)/bin/javac

JAVAH = $(JAVA_HOME)/bin/javah

JAVA_ARCHIVE= $(JAVA_HOME)/bin/jar cvfM0


CLASSPATH =
$(JAVA_HOME)/lib/classes.zip:$(NAS_HOME)/classes/java/ldapjdk.jar:$
(NAS_HOME)/classes/java/kfcjdk11.jar:$(BX_HOME)/java/bx_common.zip:
$(BX_HOME)/java/bx_be.zip:$(BX_HOME)/java/bxfe.zip:$(PKGDIR):$(BX_H
OME)/java/bx_loading.zip:$(BX_HOME)/java/currconv.zip:$(NAS_HOME)/c
lasses/java/servlet.jar



ALL :J_SRCS zipall


#####################################################



J_SRCS:\
    view/BXValidateSignOnView.class   \
        view/BXValidateSignUpView.class            \
        view/BXValidateAcctView.class              \
        view/BXValidatePwdView.class               \
        view/BXValidateStmtSearchView.class        \
        view/BXValidateNewPaymentView.class        \
        view/BXValidatePayView.class               \
        view/BXBaseBean.class                      \
        view/BXInvoiceBean.class                   \
        view/BXCustomerAcctBean.class              \
        view/IBXBillDetailBean.class               \
        view/BXBillDetailBeanB1.class              \
        view/BXBDSearchVectorB1.class              \
        view/IBXBDSearchAttributes.class           \
```

```
        view/BXBDSearchAttributesB1.class          \
        view/IBXBDSearchBean.class                 \
        view/BXBDSearchBeanB1.class                \
        view/IBXValidateBDSearchView.class         \
        view/BXValidateBDSearchViewB1.class        \
        view/IBXBillDetailView.class               \
        view/BXBillDetailViewB1.class              \
        view/BXPaymentBean.class                   \
        view/DepartmentBean.class                  \
        view/BXPayViewBean.class                   \
        view/BXInvoiceCenterBean.class             \
        view/BXPaymentSearchBean.class             \
        view/BXUserCenterBean.class                \
        applogic/BXSignOnServlet.class             \
        applogic/BXSignOutServlet.class            \
        applogic/BXBillDocSearchServlet.class      \
        applogic/BXBillDocServlet.class            \
        applogic/BXBillingCenterServlet.class      \
        applogic/BXPayHistoryServlet.class         \
        applogic/BXPaymentServlet.class            \
        applogic/BXSignOnCSRServlet.class          \
        applogic/BXSilentSignOnServlet.class       \
        applogic/BXUserCenterServlet.class


#####################################################

.SUFFIXES:
.SUFFIXES: .java .class

.java.class:
```

```
    $(JAVAC) -classpath $(CLASSPATH) -d $(PKGDIR) $*.java


zipall:
    $(JAVA_ARCHIVE) bx_fetelco.zip com
```

2. Change the value of the `JAVA_HOME` variable to match the location of the Java compiler.
3. Optionally, change the `CLASSPATH` variable to include zip files that are not in the `$BX_HOME/java` subdirectory.
4. Save your changes, then open `$BX_HOME/fe/src/fe/makefile`.

The contents look something like this:

```
JAVA_HOME = /share/builds/components/jdk/1.2.2/SunOS

NAS_HOME = /space/iplanet/ias6/ias

FE_HOME  = $(BX_HOME)/fe/src/fe

PKGDIR  = $(FE_HOME)


JAVAC = $(JAVA_HOME)/bin/javac

JAVAH = $(JAVA_HOME)/bin/javah

JAVA_ARCHIVE= $(JAVA_HOME)/bin/jar cvfM0


CLASSPATH =
$(JAVA_HOME)/lib/classes.zip:$(NAS_HOME)/classes/java/mail.jar:$(NA
S_HOME)/classes/java/servlet.jar:$(NAS_HOME)/classes/java/ldapjdk.j
ar:$(NAS_HOME)/classes/java/kfcjdk11.jar:$(BX_HOME)/java/bx_common.
zip:$(BX_HOME)/java/bx_be.zip:$(PKGDIR):$(BX_HOME)/java/bxofx.zip:$
(BX_HOME)/java/currconv.zip:$(BX_HOME)/java/ldapbeans.jar:./sax.zip
:./org_sax.zip




ALL :J_SRCS zipall


#####################################################
```

```
J_SRCS:\
       common/BXBaseView.class                          \
                common/BXSession.class                          \
                common/BXErrorCode.class                        \
                common/BXConfigObj.class                        \
                common/BXValidateView.class                \
                common/BXValidateAttribute.class        \
                common/BXValidator.class                \
                common/BXBaseServlet.class                      \
                user/view/BXUserProfileObj.class        \
                user/common/BXMembershipDBDef.class        \
                user/common/BXBillingDBDef.class        \
                user/common/BXBillingALDef.class        \
                user/common/BXPaymentALDef.class        \
                user/common/BXPaymentDBDef.class\
                user/common/BXBillerDBDef.class\
                user/view/BXBillerSignUpSession.class\
                 user/view/BXViewUtil.class                \
                 user/view/BXTemplateUtil.class                \
                 user/view/BXBillingActivity.class        \
                 user/view/BXPaymentActivity.class\
                 user/view/BXMembershipActivity.class\
                 user/view/BXBillerProfileView.class        \
                 user/view/BXSearchAttributes.class        \
                 user/view/BXSignUpSession.class                \
                 user/view/BXBillerCustAcctUtil.class        \
                 user/view/BXFinancialAcctObj.class        \
                 user/view/BXPaymentView.class                \
                 user/view/BXBillerCustAcctObj.class\
                 user/view/BXCustomerAcctView.class        \
```

```
                    user/view/BXTemplateDataBasic.class             \
                    user/view/BXTemplateMapBasic.class              \
                    user/view/BXSummaryItem.class                \
                    user/view/BXSummaryVector.class\
                    user/view/BXStatementView.class                \
                    user/view/BXDetailItem.class\
                    user/view/BXSearchVector.class\
                    user/view/BXPayHistoryDetailObj.class\
                    user/view/BXPayHistoryView.class\
                    use r/view/BXBillerServiceOptionVectors.class    \
                    user/view/BXUserProfileView.class               \
                    user/view/BXUserServiceView.class               \
                    user/view/BXCustServiceView.class               \
                    user/view/BXCustProfileView.class               \
                    user/view/BXConfigOptionView.class              \
                    applogic/LoaderDoneServlet.class

#####################################################

.SUFFIXES:
.SUFFIXES: .java .class

.java.class:
   $(JAVAC) -classpath $(CLASSPATH) -d $(PKGDIR) $*.java

zipall:
   $(JAVA_ARCHIVE) bxfe.zip com
```

**5.** Change the value of the `JAVA_HOME` variable to match the location of the Java compiler.

6. Optionally, change the `CLASSPATH` variable to include zip files that are not in the `$BX_HOME/java` subdirectory.
7. Save the file and execute make on each one.

```
cd $BX_HOME/fe/src/fetelco
make
cd $BX_HOME/fe/src/fe
make
```

### *Results*

The make commands should execute without errors and create the following files:

```
bx_fetelco.zip
bx_fe.zip
```

### *Note*

Simply creating a zip file does not register your changes. You must also move the new zip file to BillerXpert's java subdirectory or specify another directory in the CLASSPATH environment variable and restart IAS.

# Changing JSP Templates

One of the first things you will do when you customize BillerXpert B2B Edition is to modify the appearance of your web pages. You can do this by simply changing the JSP template files. After you change a template and save it, your change appears as soon as the page is loaded.

**NOTE** For testing purposes, you may wish to set your browser's disk and memory caches to 0 KBytes and force the browser to check the network for each access to avoid having the page loaded from the cache instead of downloaded each time the page is accessed.

## Exercise 2

In this exercise, you modify the JSP part of the Invoice Center template.

### *Files*

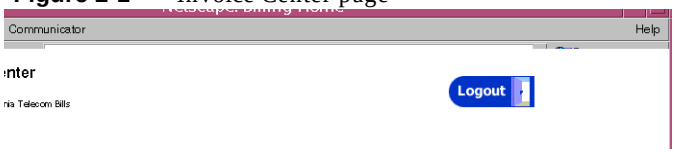
In this exercise, you will modify the following files:

| | |
|---|---|
| `invoice_center.jsp` | Invoice Center Template |

### *Tutorial Steps*

You will change the display at the top of the page from ‘Invoice Center’ to ‘Invoices for SUN’, where SUN is a company name. Figure 2-1 shows the top of the original Invoice Center page.

**Figure 2-1** Invoice Center page




1. Change your directory to the location of the template files and open the `invoice_center.jsp` template in an editor.

   ```
   cd $BX_HOME/fe/templates/en/telco
   ```

   ```
   vi invoice_center.jsp
   ```

2. Locate the line that contains ‘Invoice center’. The line is in the third table near the top of the file:

```
<table width="100%" border="0" cellspacing="0" cellpadding="0"
bgcolor="#cccccc">
          <tr>
            <td colspan="3" bgcolor="#ffffff"><img
src="/telco/images/spacer.gif" width="1
" height="8"></td>
          </tr>
          <tr>
            <td colspan="3" class="page-title"
bgcolor="#ffffff">    I
nvoice
               Center</td>
          </tr>
          <tr>
            <td colspan="3" bgcolor="#ffffff"><img
src="/telco/images/spacer.gif" width="1
" height="10"></td>
          </tr>
```

3. Add a line to be ‘`invoices for:`’ followed by the company name.

```
<tr>
            <td colspan="3" bgcolor="#ffffff"><img
src="/att/images/spacer.gif" width="1" height="8"></td>
```

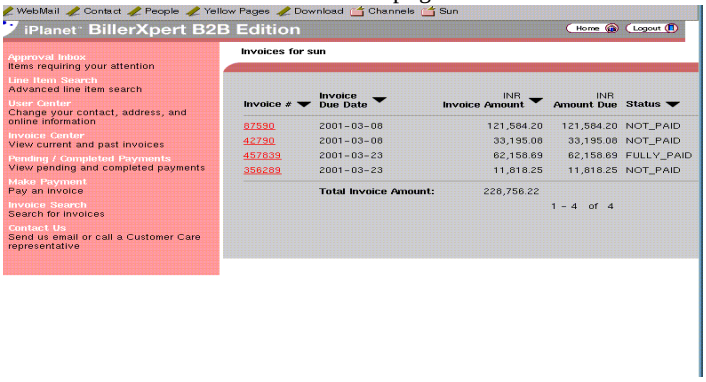
```
          </tr>
          <tr>
            <td colspan="3" class="page-title"
bgcolor="#ffffff">    
              Invoices for <jsp:getProperty name="bxsession"
property="companyID" /></td>
          </tr>
          <tr>
            <td colspan="3" bgcolor="#ffffff"><img
src="/att/images/spacer.gif" width="1" height="10"></td>
          </tr>
```

4. Save your changes and reload the page.

### *Results*

Figure 2-2 shows the top of the Invoice Center page that contains the changed text.

**Figure 2-2** Modified Invoice Center page




# Displaying Data

BillerXpert uses JSP (Java Server Page) as page templates to present application data. JSPs are browser pages defined in either HTML or XML. They can optionally contain Java code, which enables them to perform complex processing, conditionalize output, and communicate with other objects in your application. JSPs in iAS 6.0 are based on the JSP 1.1 specification.

A JSP is made up of JSP elements and template data. Template data refers to anything not in the JSP specification, including text and HTML tags. For example, the minimal JSP, which requires no processing by the JSP engine, is a static HTML page. You can call a JSP from a servlet to handle the output from a user interaction, or, since JSPs have the same access to the application environment as any other application components, you can use a JSP as a destination from an interaction.

The iAS compiles JSPs into HTTP servlets the first time they are called. This makes them available to the application environment as standard objects, and it also enables them to be called from a client using a URL.

JSPs run inside a Java process on the server. This process, called a JSP engine, is responsible for interpreting JSP-specific tags and performing the actions they specify in order to generate dynamic content. This content, along with any template data surrounding it, is assembled into a page for output and returned to the caller.

The response object contains a reference to the calling client, and this is where a JSP presents the page that it creates. If you call a JSP from a servlet using the forward() method from the RequestDispatcher interface, forward() provides the response object as a parameter to the JSP. If a JSP is invoked directly from a client, the response object is provided by the server that is managing the relationship with the caller.

In either case, the page is automatically returned to the client through the reference provided in the response object. You do not have to write any code to return a page to a client.

JSPs and some other application components can be updated at run time without restarting the server, making it easy to change the look and feel of your application without stopping service.

For more information about JSP, please refer to the *Developing IAS Applications (Java) manual.*

Code Example 2-1 in `BXUserCenterServlet` constructs a `BXUserCenterBean` and saves it to session and calls the `displayTemplate` which in turn forwards request to jsp template file `user_center.jsp`.

**Code Example 2-1** `BXUserCenterServlet`

```
public void displayUserCenter()
  {
    BXSession session  = getBXSession();
```

```
    try {
      IBXCustomerProfile custProfile = BXUserProfileObj.read(status,
session);
      BXUserCenterBean userCenter = new
BXUserCenterBean(custProfile);
      req.getSession(true).putValue("userCenter", userCenter);

      displayTemplate(USER_CENTER_PAGE);
    }
    catch (Exception e) {
          Logger.logError(getBXSession().getUserID(),
getBXSession().getBillerId(),
                                "UI_0037", "2",
BXActivities.getActivityMessage("BXFE_0002.user_
center"),
getBXSession().getEmulationID(),getBXSession().getCompanyId(),true)
;
      e.printStackTrace();
      displayErrorMsg(ERRMSG_TEMPLATE);
    }
  }
```

# Using Java Beans

JSPs support several tags for the purpose of instantiating and accessing Java beans. You use beans to perform computations in order to obtain a set of results, which are stored as bean properties. JSPs provide automatic support for creating beans and for examining their properties.

Beans themselves are separate classes created according to the Java Bean specification. It is beyond the scope of this manual to describe how to create standard Java beans. For more information about Java beans, visit the official web site `http://java.sun.com/beans`. There are many tutorials that describe how to create beans, some of which are accessible from the official site.

It is common in beans to have "getter" and "setter" methods to retrieve and set bean properties. Getter methods are named `getXxx()`, where Xxx is a property called xxx (the first letter is capitalized for the method name). If you have a corresponding setter called `setXxx()`, the setter must take a parameter of the same type as the return value from the getter.

BillerXpert provides a base bean class, `BXBaseBean`, from which all the bean classes are derived. The following code is extracted from `BXUserCenterBean.java`.

```
  public String getFirstName() {
     return firstName;
  }
public String getLastName() {
     return lastName;
  }
public void setFirstName(String val) {
     firstName = val;
  }
public void setLastName(String val) {
     lastName = val;
  }
```

`user_center.jsp` uses `BXUserCenterBean` to get property first name, last name and display them.

```
      <jsp:useBean id="userCenter"
type="com.iplanet.ecomm.billxb2b.fe.telco.BXUserCenterBean
" scope="session" />
    ...

          <tr>
            <td width="17"> </td>
            <td align="right" nowrap><b>Name</b></td>
            <td width="10"> </td>
```

```
            <td> <jsp:getProperty name="userCenter"
property="firstName" /> <jsp:ge

tProperty name="userCenter" property="lastName" /></td>

            <td width="17"> </td>

          </tr>
```

**NOTE** BillerXpert B2B's `BXSignOnServlet` creates a session object when the login has been authenticated and populates the session object with data from the customer profile record. The BXSession class defines methods to access this data. BXSession object itself is a Java Bean, some B2B's jsp files directly access this bean to retrieve session data.

## Exercise 3

In this exercise, you modify the `BXUserCenterBean` to set additional data for display and modify `user_center.jsp` to display the data.

### *Files*

In this exercise, you will modify the following files:

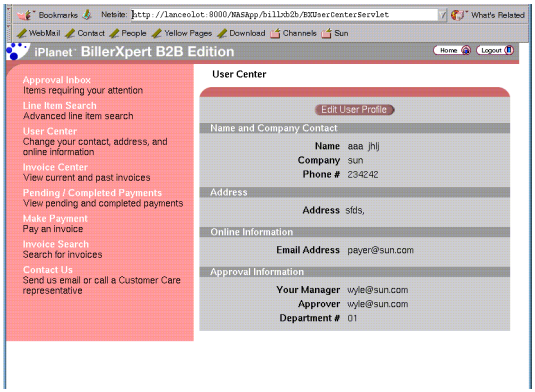
| | |
|---|---|
| `BXUserCenterBean.java` | user center javabean |
| `user_center.jsp` | user center template |

### *Tutorial Steps*

You will change the display near the top of the page from '`User Center`' to '`Good Morning/Afternoon, user_name`', where '**Good Afternoon**' is a string based on the time of day and user_name is a customer's first name, and also display user's fax number after the phone # line.

This exercise demonstrates how to use javabean and java scripting to customize look and feel. Figure 2-3 shows the original User Center page.

**Figure 2-3** User Center page




1. Change your directory to the location of the view and bean files and open the `BXUserCenterBean.java` file in an editor.

   ```
   cd $BX_HOME/fe/src/fetelco/view
   ```

   ```
   vi BXUserCenterBean.java
   ```

2. Add code to get and set fax number

```
private String fax;

   public BXUserCenterBean (IBXCustomerProfile custProfile) {
    if (custProfile != null) {
      setFirstName(custProfile.getFirstName());
      setLastName(custProfile.getLastName());
      setCompany(custProfile.getOrgName());
      setDaytimePhone(custProfile.getDayPhone());
      setEmail(custProfile.getEmail());
      ...
      setFax(custProfile.getFaxNumber());
      ...
    }
  }

  public void setFax(String val) {
     fax = val;
  }

  public String getFax() {
     return fax;
  }
```

3. save your change, change to the telco front-end directory and recompile.

```
cd $BX_HOME/fe/src/fetelco
make
```

4. Copy the `bx_fetelco.zip` file to BillerXpert's java subdirectory.

```
cp bx_fetelco.zip $BX_HOME/java/bx_fetelco.zip
```

5. Change your directory to the location of template files and open the `user_center.jsp`

```
cd $BX_HOM/fe/templates/en/your_biller_template_base
vi user_center.jsp
```

6. Locate the line that contains 'User Center', the line is in the following table near the top of the file:

```
          <tr>
            <td colspan="5" bgcolor="#ffffff"><img
src="/att/images/spacer.gif" width="1" height="8"></td>
          </tr>
          <tr>
            <td colspan="5" class="page-title"
bgcolor="#ffffff">     User Center</td>
          </tr>
```

7. Add the following scripting code after the lines shown above.

```
<%! private String greeting; %>
<% Calendar cal = Calendar.getInstance();
   if (cal.get(Calendar.HOUR_OF_DAY) < 12)
      greeting = "Good Morning, " + bxsession.getFirstName();
   else if (cal.get(Calendar.HOUR_OF_DAY) < 18)
      greeting = "Good Afternoon, " + bxsession.getFirstName();
   else
      greeting = "Good Evening, " + bxsession.getFirstName(); %>
```

8. Add the html tab to show greeting line

```
          <tr>
            <td colspan="5" bgcolor="#ffffff"><img
src="/att/images/spacer.gif" width="1" height="8"></td>
          <tr>
```

```
            <td colspan="5" class="page-title"
bgcolor="#ffffff">     <%= greeting %>
</td>
          </tr>
```

**9.** Locate the line that contains 'Phone #'

```
          <tr>
            <td width="17"> </td>
            <td align="right" nowrap><b>Phone #</b></td>
            <td> </td>
            <td> <jsp:getProperty name="userCenter"
property="daytimePhone" /></td>
            <td width="17"> </td>
          </tr>
```

**10.** Add html and jsp tag to display user's fax number

```
          <tr>
            <td colspan="5"><img src="/att/images/spacer.gif"
width="1" height="6"></td>
          </tr>
          <tr>
            <td width="17"> </td>
            <td align="right" nowrap><b>Fax #</b></td>
            <td width="10"> </td>
            <td> <jsp:getProperty name="userCenter"
property="fax" /></td>
            <td width="17"> </td>
          </tr>
```

**11.** Save the changes.

**12.** Restart IAS.

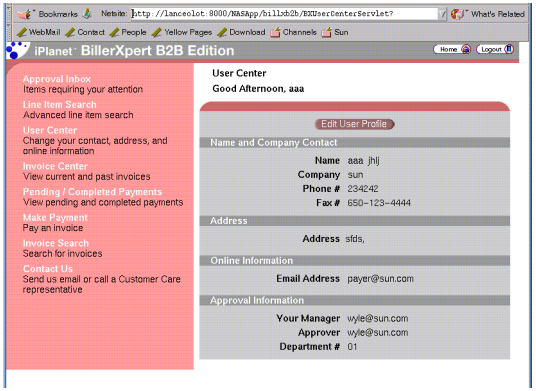
Change to the bin subdirectory of the directory in which IAS is installed and issue the following commands:

```
cd $NAS_HOME/bin
KIVAes.sh stop
KIVAes.sh start
```

### *Results*

Figure 2-4 shows the revised User Center page.

**Figure 2-4** Revised User Center Page




For more information about JSP, see the *Java Programmer's Guide.*

# Using Contexts

BillerXpert B2B Edition provides contexts that use the Servlet's IContext object. This object provides access to IAS services. In addition to managing the Servlet's IContext object, BillerXpert contexts provide database access, transaction management, status information, and the ability to write to the log file.

One of the first operations that a Servlet must perform is to initialize its context. For example, a Servlet that only accesses membership data needs only to create a membership context object. BillerXpert provides an `init()` method in its `BXBaseServlet` class, from which BillerXpert Servlets descend, that creates context objects based on the context constant that you provide as an argument to the method. Table 2-2 identifies the context constants.

**Table 2-2** BillerXpert contexts

| Context constant | Creates object | Referenced by | Description |
|---|---|---|---|
| BILLING_CONTEXT | BXBillingContext | billingContext | Billing context |
| MEMBERSHIP_CONTEXT | BXMembershipContext | membershipContext | Membership context |
| CONFIG_CONTEXT | BXConfigContex | configContext | Configuration context |
| ALL_CONTEXT | all of the above | — | All contexts |

If you do not provide an argument to the `init()` method, the method creates all contexts.

## Exercise 4

In this exercise, you change the Sign On Servlet to use both the billing and membership contexts rather than just the membership context. In the next exercise you use this change to allow both billing and membership data to be accessed instead of just membership data.

### *Files*

In this exercise, you will modify the following files:

`BXSignOnServlet.java` Sign On Servlet

### *Tutorial Steps*

You change the way the `init()` method is called to create all contexts and you will create `BXCustomerAcctView` objects using both the membership and billing contexts.

1. Change your directory to the location of the Servlet files and open the `BXSignOnServlet.java` file in an editor.

   ```
   cd $BX_HOME/fe/src/fetelco/servlet
   ```

   ```
   vi BXSignOnServlet.java
   ```

2. Locate the call to the `init()` method from the `doPost()` method.

```
public void doPost(HttpServletRequest req, HttpServletResponse resp)
```

```
                        throws ServletException, java.io.IOException
  {
/* call init(req, resp) first to set the resp before any
displayErrorMsg is called
   check to see if there's alreay a valid bxsession in req, if so,
perform logout logic
   and create new session
*/
    init(req, resp);

    if (req.getSession(true).getValue("bxsession") != null) {
       removeCookieForPM();
       req.getSession(true).invalidate();
    }

    // Retrieve the biller id were are currently working with.
    String billerid   = req.getParameter("billerid");
    // Sets the BillerID in the BXBaseServlet
    setBillerID(billerid);
    setCompanyID(null);

    init(ALL_CONTEXT, req, resp);
```

3. You can change the call to the init() method to initialize `MEMBERSHIP_CONTEXT`, `BILLING_CONTEXT` or `ALL_CONTEXT`.

4. Locate the line that creates a `BXCustomerAcctView` object in the

```
verifyLogin() method.
public boolean verifyLogin(String Username, String Password)
{
try {
  ca = new BXCustomerAcctView(membershipContext, status);
}
catch (Exception e) {
  e.printStackTrace();
  return false;
}
return ca.authenticate(Username, Password);
}
```

5. Change the constructor invocation to use both the membership and billing contexts.

```
ca =new BXCustomerAcctView
        (membershipContext,billingContext,status);
```

6. Locate the line that creates a `BXCustomerAcctView` object in the `createSession()` method.

```
public boolean createSession(String OnlineName, String Password)
{
...
  if (ca == null) {
    try (
      ca = new BXCustomerAcctView(membershipContext, status);
    }
    catch (Exception e) {
      return false;
    }
  }
  return ca.saveUserProfile(session, OnlineName);
}
```

7. Change the constructor invocation to use both the membership and billing contexts.

```
ca =new BXCustomerAcctView
        (membershipContext,billingContext,status);
```

8. Save your changes and compile the java file.

# Invoking Servlets

You invoke a servlet either by directly addressing it from an application page with a URL, or by calling it programmatically from another servlet that is already running. The basic flow of execution in BillerXpert B2B Edition is from the `index.html` to the `BXSignOnServlet`. When the `BXSignOnServlet` authenticates the login, it invokes another JSP which invokes the Process Management servlet to display Approval Inbox and all the left side navigation links which will invoke other servlets to display different information.

Some servlets require additional information when they are invoked. For example, the Payment Servlet, BXPaymentServlet, requires a parameter when it's invoked that controls the behavior of the servlet. For example, the navigation link 'Make Payment' on the left side will invoke BXPaymentServlet with value 'view' for the cmd parameter:

```
<tr>
            <td colspan="2" checked="false" valign="top"><a
href="/NASApp/billxb2b/BXPaymentServlet?cmd=view"><span
class="left-menu-link">Make Payment</span></a><br>
              <span class="small-font">Pay an invoice</span></td>
            <td width="10"> </td>
          </tr>
```

The servlet can then control it's flow based on the cmd value.

| | |
|---|---|
| `cmd=view` | list the invoices with outstanding amount due |
| `cmd=sort` | sort the invoices based on input sorting column |
| `cmd=makePayment` | pay the selected invoice |

In BillerXpert, the most common way to invoke a servlet is from a html form. The following form is defined in pay_invoice.jsp. The value of field name `cmd` is set to either 'PayInvoice' or 'sort'.

```
<script LANGUAGE="JavaScript">
function processSort(sortfield)
{
  document.form.sortfield.value = sortfield;
  document.form.cmd.value = "sort";
  document.form.submit();
}

function submitForm() {
  document.form.cmd.value = "PayInvoice";
  document.form.submit();
}
</script>
```

```
<form method="POST" name="form" action="BXPaymentServlet">
<input type="hidden" value="PayInvoice" name="cmd">
<input type="hidden" name="sortfield">
<input type="hidden" name="PAGE_INDEX" value ="1">
```

Incoming data is encapsulated in a request object. For HTTP servlets, the request object is of type HttpServletRequest. For generic servlets, the request object is of type ServletRequest. The request object contains all the parameters in a request, and you can also set your own values in the request. The latter are called attributes.

You can access all the parameters in an incoming request by using the `getParameter()` method. Code Example 2-2 shows the retrieval of the value associated with cmd.

**Code Example 2-2** The BXPaymentServlet's doPost() method

```
  public void doPost(HttpServletRequest req, HttpServletResponse
resp)
                        throws ServletException, java.io.IOException
  {
    ...

    String request = req.getParameter("cmd");
    if (request != null) {
      if (request.equals(VIEW_PAY)) {
         if (checkACL("PAYAUTH_ACL", "FEMakePayment") != 0)
        {
            status.push(BXStatus.STATUS_ERROR,
BXErrorCode.ACCESS_NOT_ALLOWED);
            displayErrorMsg(ERRMSG_TEMPLATE);
            return;
        }
        displayPayView(request);
      }
      else if (request.equals("sort")) {
         String sortfield = req.getParameter("sortfield");
```

```
         sortPayView(sortfield,session);
      }
      else if (request.equals(MAKE_PAYMENT)) {
        // process payment when user selects "Make Payment" buton
        if (checkACL("PAYAUTH_ACL", "FEMakePayment") != 0)
        {
          status.push(BXStatus.STATUS_ERROR,
BXErrorCode.ACCESS_NOT_ALLOWED);
          displayErrorMsg(ERRMSG_TEMPLATE);
          return;
        }
        makePayment();
      }
      else if (request.equals(CANCEL)) {
        displayTemplate(WAIT_PAGE);
      }
      else if (request.equals(PAY_SELECTED_INVOICE)){
          if (checkACL("PAYAUTH_ACL", "FEMakePayment") != 0){
             status.push(BXStatus.STATUS_ERROR,
BXErrorCode.ACCESS_NOT_ALLOWED);
             displayErrorMsg(ERRMSG_TEMPLATE);
             return;
          }
          displayLineItemStatusCount();
      }
    }
    else
       displayPayView(request);
  }
```

## Exercise 5

In this exercise, you set up a pull-down menu on the Sign On page that allows the user to select starting page in addition to the Approval Inbox. This exercise requires you to change code in jsp file, no modification is necessary in servlet code.

### *Files*

In this exercise, you will modify the following files:

- `index.html`
- `auth_summary.jsp`

### *Tutorial Steps*

You will add a drop-down menu on the Signon page and invoke a servlet based on the user's selection

1. Change your directory to the location of the template files and open the index.html in an editor.

   ```
   cd $BX_HOME/fe/templates/en/your_biller_template_base
   vi index.html
   ```

2. Locate the `OnlineName` and `Password` input fields.

```
                <tr>
                  <td align="right"><font size="-1"><b><font
face="Arial,Helvetica,Geneva,Swiss, SunSans-Regular">User
                    ID </font></b></font></td>
                  <td height="35">
                    <input type="text" name="OnlineName" size="23"
value="" maxlength="40">
                  </td>
                </tr>
                <tr>
                  <td align="right"><font size="-1"><b><font
face="Arial,Helvetica,Geneva,Swiss, SunSans-Regular">Password
                    </font></b></font></td>
                  <td height="35">
                   <input type="password" name="Password" size="23"
maxlength="20">
                  </td>
                </tr>
```

3. Add your drop-down menu immediately after the password row

   The menu consists of five items whose values are strings containing starting page name. The selected menu item is returned in the `StartingPage` field.

```
<tr>
                  <td align="right"><font size="-1"><b><font
face="Arial,Helvetica,Geneva,Swiss, SunSans-Regular">Starting Page
                    </font></b></font></td>
                  <td valign="middle">
                    <select name="StartingPage">
                      <option value="User Center" >User
Center</option>
                      <option value="Invoice Center" >Invoice
Center</option>
                      <option value="Line Item Search" >Line Item
Search</option>
                      <option value="Approval Inbox" >Approval
Inbox</option>
                      <option value="Payment Center" >Payment
Center</option>
                    </select>
                  </td>
                </tr>
```

**4.** in the same directory, open `auth_summary.jsp`

**5.** Locate jsp `useBean` line

```
<jsp:useBean id="bxsession"
type="com.iplanet.ecomm.billxb2b.common.BXSession" scope="session"
/>
<%! private String url; %>
<% String clusterId = bxsession.getClusterId();
   url = "/NASApp/" + clusterId +
"/Express.npm?eventId=OnDisplaySplitWorklist&__template=BX"; %>
```

**6.** retrieve the 'StartingPage' value and dispatch a servlet base on it.

```
<jsp:useBean id="bxsession"
type="com.iplanet.ecomm.billxb2b.common.BXSession" scope="session"
/>
<%! private String url; %>
<% String clusterId = bxsession.getClusterId();
   String sp = request.getParameter("StartingPage");
```

```
   if (sp.equals("User Center")) {
      url = "/NASApp/billxb2b/BXUserCenterServlet";
   }
   else if (sp.equals("Invoice Center")) {
      url =
"/NASApp/billxb2b/BXBillingCenterServlet?cmd=refresh&sortfield=2";
   }
   else if (sp.equals("Line Item Search")) {
      url = "/NASApp/billxb2b/BXBillDocSearchServlet?cmd=Advanced";
   }
   else if (sp.equals("Payment Center")) {
      url = "/NASApp/billxb2b/BXPayHistoryServlet?cmd=ViewHistory";
   }
   else if (sp.equals("Approval Inbox")) {
      url = "/NASApp/" + clusterId +
"/Express.npm?eventId=OnDisplaySplitWorklist&__template=BX";
   } %>
```

7. Save your changes, no need to restart server, just reload the `index.html` page

### *Results*

Figure 2-5 shows the SignOn page that contains the drop down menu, the menu items take you to the corresponding BillerXpert page.

**Figure 2-5** Modified Sign On Page.

# Configuring Table Definitions

LDAP is used by BillerXpert B2B Edition to map database column names to names used within BillerXpert to identify fields in component business objects, described in the section “Implementing Component Business Objects”. After you create the database schema, you should update LDAP.

LDAP is a collection of NTV lists, which are lists of *name-type-value* tuples. NTV lists for component business objects and the loader program define the data structures available to BillerXpert and the loader program. The NTV list consists of the following information:

1. Name of the NTV list.
2. Database table or LDAP schema that corresponds to the list.
3. A list of attributes that define the names that can be used programmatically.

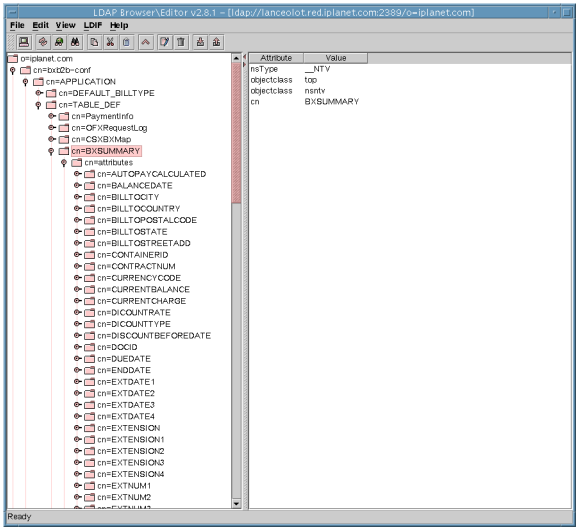
   Each attribute is itself an NTV list that contains the following information:

An attribute contains information about the database table:

1. Table or schema and field associated with the attribute.
2. Ordinal number of the a column represented by the attribute name.
3. Length of the database field.
4. Type of field, such as INT for integer or STRING for string.

Figure 2-6 shows the NTV list:

**Figure 2-6** NTV List




## Exercise 6

In this exercise, you examine the structure of the LDAP file. In this example, you examine the database in the `$NAS_HOME` directory.

### *Files*

In this exercise, you will examine the LDAP database contents.

### *Tutorial Steps*

Examine the LDAP database contents.

1. Change to the iPlanet Application Server (IAS) configuration directory and open the Directory Server.

   ```
   cd $NAS_HOME/..
   ```

   ```
   startconsole
   ```

2. Log in as the IAS Administrator.

   lSearch for the `BXSummary` entry as in the previous figure.

# Implementing Component Business Objects

Component business object classes represent records in a database. For information about these classes, see the *BillerXpert B2B Edition API & Schema Reference Manual.* You can modify these classes to define your own component business object classes.

To implement a component business object class, you must define accessor methods for each field in the record and also implement the following methods, which are defined in the `IBXEntity` interface:

| | |
|---|---|
| `ejbCreate()` | Creates a record in the database. |
| `ejbStore()` | Updates a record in the database. |
| `ejbRemove()` | Removes a record from the database. |
| `ejbLoad()` | Reads a record, identified by object ID, from the database. |
| `ejbFindByPrimaryKey()` | Reads a record, identified by key, from the database. |
| `ejbFindAll()` | Reads selected records from the database. |
| `getName()` | Determines the class name. |
| `dump()` | Displays the fields |

You may define other methods as needed.

Chapter 3

# Common Front-End Classes

This chapter discusses the common classes used by the BillerXpert B2B Edition front end to manipulate application business objects.

This chapter presents the following topics:

- Overview
- BXBaseServlet Class Reference
- BXBaseView Class Reference
- BXConfigObj Class Reference
- BXErrorCode Class Reference
- BXSession Class Reference
- BXBaseBean Class Reference
- BXValidateView Class Reference
- BXValidateAttribute Class Reference
- BXValidator Class Reference
- BXViewUtil Class Reference

## Overview

Table 3-1 shows the classes that are available to any kind of BillerXpert application.

**Table 3-1** Core classes

| Class | Description |
|---|---|
| `BXBaseServlet` | Class from which servlet classes are derived. |
| `BXBaseView` | Abstract class from which view classes are derived. |
| `BXConfigObj` | Represents NTV list entries in configuration files. |
| `BXErrorCode` | Defines BillerXpert error codes. |
| `BXSession` | BillerXpert session class. |
| `BXValidateAttribute` | Represent attributes, such as input fields, which can be validated. |
| `BXValidateView` | Abstract class from which classes that validate views are derived. |
| `BXValidator` | Validates BillerXpert data types. |
| `BXTemplateUtil` | Additional methods to support templates. |
| `BXViewUtil` | Additional methods to support view objects. |

# BXBaseServlet Class Reference

Instances of the `BXBaseServlet` class represent BillerXpert Servlets. You typically define a subclass of the `BXBaseServlet` class to implement your application logic.

## Methods

Summary list:

| | |
|---|---|
| **Initialization** | |
| `init()` | Initializes database contexts. |
| **Biller and Company Support** | |
| `setBillerID()` | Sets the biller id. |
| `set CompanyID()` | Sets the company id. |
| **Context management** | |
| `createBXBillingContext()` | Creates a billing context. |

| | |
|---|---|
| `createBXMembershipContext()` | Creates a membership context. |
| `createBXConfigContext()` | Creates a configuration context. |
| `releaseBXDBContext()` | Releases database contexts. |
| **Session management** | |
| `getBXSession()` | Returns the current session object. |
| `createBXSession()` | Creates a BillerXpert session object. |
| `verifyUserSession()` | Verifies that a valid session exists. |
| **Displaying templates and error information** | |
| `displayTemplate()` | Displays a template. |
| `displayErrorMsg()` | Displays an error message template. |
| **Activity Logging** | |
| `logActivity()` | Logs activity. |
| **Access Control** | |
| `CheckACL()` | Checks if the Access Control List (ACL) role is authorized. |

# Class Definition

### *Package*

`com.iplanet.ecomm.billxb2b.common`

### *Syntax*

```
public class BXBaseServlet extends Servlet{ ... }
```

## Context Flags

You can use a context flag to specify the contexts to create when you call a Servlet's `init()` method:

### *Syntax*

```
protected final int ALL_CONTEXT = 0;
protected final int BILLING_CONTEXT = 1;
protected final int MEMBERSHIP_CONTEXT = 2;
protected final int CONFIG_CONTEXT = 4;
```

| | |
|---|---|
| ALL_CONTEXT | Create one of each kind of context object. |
| BILLING_CONTEXT | Create a billing context object. |
| MEMBERSHIP_CONTEXT | Create a membership context object. |
| CONFIG_CONTEXT | Create a configuration context object. |

## Context Objects

You can use the following context objects within your Servlets; these objects are created for you when you call the Servlet's `init()` method.

### *Syntax*

```
protected BXBillingContext billingContext= null;
protected BXMembershipContext membershipContext = null;
protected BXConfigContext configContext = null;
```

| | |
|---|---|
| billingContext | Reference to the current billing context. |
| membershipContext | Reference to the current membership context. |
| configContext | Reference to the current configuration context. |

## Session, Status, and Application Information

The following session, status, and application information can be used within your Servlet:

```
Syntax BXSession m_sessionProxy = null;
protected BXStatus status = null;
```

| | |
|---|---|
| m_sessionProxy | Reference to the current session object. |
| status | Reference to the stackable status object. |

# Methods

This section describes the methods of the `BXBaseServlet` class.

## createBXBillingContext( )

Creates a billing context.

### *Syntax*

```
void createBXBillingContext();
```

### *Discussion*

The `createBXBillingContext()` method creates a `BXBillingContext` object and associates it with the `billingContext` variable. The `createBXBillingContext()` method is protected.

## createBXConfigContext( )

Creates a configuration context.

### *Syntax*

```
void createBXConfigContext();
```

### *Discussion*

The `createBXConfigContext()` method creates a `BXConfigContext` object and associates it with the `configContext` variable. The `createBXConfigContext()` method is protected.

## createBXMembershipContext( )

Creates a membership context.

### *Syntax*

```
void createBXMembershipContext();
```

### *Discussion*

The `createBXMembershipContext()` method creates a `BXMembershipContext` object.

## Create BXSession ()

Creates a BillerXpert session object.

### *Syntax*

```
BXSession createBXSession();
BXSession createBXSession(String OnlineName)
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `OnlineName` | A string that contains the user's online name. |

### *Returns*

A `BXSession` object that represents the current session.

### *Discussion*

The first form of the `createBXSession()` method creates a session object only if an object representing the current session does not exist. The second form of the `createBXSession()` method creates a new session object if the online name is not the same as the current online name or if an object representing the current session does not exist. The `createBXSession()` method assigns a reference to a newly created session object to the `m_sessionProxy` variable.

## displayErrorMsg( )

Displays an error message template.

### *Syntax*

```
void displayErrorMsg(String templateFile);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `templateFile` | A string that contains the pathname of the JSP template. |

### *Discussion*

The method invokes the specified template and attempts to display the error message and code, which is stored in the current status object. If the status object is empty, the `displayErrorMsg()` method displays the message "Unknown Error," for error code BXFE_10000. The `displayErrorMsg()` method calls the `displayTemplate()` method to perform this action.

### *Note*

The `displayErrorMsg()` method releases all database contexts.

### *Example*

Code Example 3-1 displays the Sign On error message template if the Servlet cannot be initialized.

**Code Example 3-1** Sign On Error message template

```
private static final String ERRMSG_TEMPLATE =
    "/signon_msg.jsp";
...
if (!init(MEMBERSHIP_CONTEXT))
displayErrorMsg(ERRMSG_TEMPLATE);
```

## displayTemplate( )

Displays a template.

### *Syntax*

```
void displayTemplate( String template);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `template` | A string that contains the pathname of the JSP template. |

### *Discussion*

The method forwards the request to the specified jsp. The `displayTemplate()` method calls the `getRequestDispatcher()` method to perform this action.

### *Note*

The `displayTemplate()` method releases all database contexts.

## getBXSession( )

Returns the current session object.

### *Syntax*

```
BXSession getBXSession();
```

### *Returns*

A `BXSession` object that represents the session object.

### *Discussion*

The `getBXSession()` method returns the session object referenced by the `m_sessionProxy` variable. If the variable is null, the `getBXSession()` method creates a session object and assigns it to the `m_sessionProxy` variable.

## init( )

Initializes database context.

### *Syntax*

```
void init(HttpRequest request. HttpResponse response);
boolean init(int whichContext);
```

### *Parameters*

The method has the following parameters:.

| | |
|---|---|
| `whichContext` | An integer that specifies the database context. |

### *Returns*

A boolean value that is true if the operation was successful; otherwise, the value is false.

### *Discussion*

The first form of the method initialize request and response objects. The second form of the method creates a context object for the specified context using a context flag to specify the context. The `init()` method is a protected method.

## logActivity( )

Logs activity.

### *Syntax*

```
boolean logActivity(IValList activity);
boolean logActivity(String act, String activityDetail, String
customerID, String activityData, string emulationID);
```

### *Parameters*

The method has the following parameters:.

| | |
|---|---|
| `activity` | The activity object. |

| | |
|---|---|
| `act` | The activity name. |
| `activityDetail` | The activity detail. |
| `customerID` | The customer id. |
| `activityData` | The activity data. |
| `emulationID` | The emulation ID. |

### *Discussion*

Log activity logs activity information to the database.

## setBillerId( )

Sets the biller ID.

### *Syntax*

```
Void setBillerID (String billerId);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `billerId` | A string which specifies the biller id. |

### *Discussion*

The method `setBillerID()` is a protected method. This method will only be called by the sign on Servlet `BXSignOnServlet.java` to set the biller id.

## setCompanyID( )

Sets the compnay ID.

### *Syntax*

```
Void setCompanyID (String companyId);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `companyId` | A string which specifies the company id. |

### *Discussion*

The method `setCompanyID()` is a protected method.

## releaseBXDBContext( )

Releases database contexts.

### *Syntax*

```
void releaseBXDBContext();
```

### *Discussion*

All database contexts are released. The `releaseBXDBContext()` method is a protected method.

## verifyUserSession( )

Verifies that a valid session exists.

### *Syntax*

```
boolean verifyUserSession();
```

### *Returns*

A boolean value that is true if a valid session exists; otherwise, it is false.

### *Discussion*

If this method returns a false value, the user is not logged into BillerXpert or the session is timed out. You should call this method before taking any other action in your Servlet's `execute()` method.

## checkACL( )

Checks if Access Control List (ACL) role is authorized.

### *Syntax*

```
int checkACL (String perm);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `perm` | The permission to check for |

### *Returns*

**0** if the the specific permission is granted, or, **-1** if the specific permission is not granted.

# BXBaseView Class Reference

The `BXBaseView` class is an abstract class from which other view classes descend. A summary list can be seen in Table 3-2.

### *Methods*

**Table 3-2** Summary list for BXBaseView class reference.
:

| | |
|---|---|
| **Constructor** | |
| `BXBaseView()` | Initializes some variables. |
| **Context management** | |
| `setBillingContext()` | Associates a billing context with this object. |
| `setMembershipContext()` | Associates a membership context with this object. |
| **Associating a status object** | |
| `set()` | Associates a status object with this object. |
| **Debugging support** | |
| `debug()` | Writes the specified message to the system output stream. |

## Class Definition

### *Package*

`com.iplanet.ecomm.billxb2b.common`

### *Syntax*

```
public abstract class BXBaseView { ... }
```

# Context Objects

You can use the following context object references within your view; these references are associated with context objects when you instantiate a subclass of `BXBaseView` or call one of the methods defined in this class that set the context.

### *Syntax*

```
protected BXBillingContext billingContext            = null;
protected BXMembershipContext membershipContext      = null;
protected BXConfigContext configContext              = null;
```

| | |
|---|---|
| `billingContext` | Reference to a billing context. |
| `membershipContext` | Reference to a membership context. |
| `configContext` | Reference to a configuration context. |

# Status Object

The following status information can be used within a view object, if the reference is set:

### *Syntax*

```
protected BXStatus status           = null;
```

| | |
|---|---|
| `status` | Reference to the current status object. |

# Constructors

## BXBaseView( )

Initializes variables.

### Syntax

```
BXBaseView(BXStatus stat);
BXBaseView(BXBillingContext context, BXStatus stat);
BXBaseView(BXMembershipContext context, BXStatus stat);
BXBaseView(BXConfigContext context, BXStatus stat);
```

### *Parameters*

The constructor has the following parameters:

| | |
|---|---|
| `context` | The context object to associate with this view object. |
| `stat` | The `BXStatus` object to associate with this view object. |

### *Discussion*

The first constructor does not associate a context with this object. Each of the other constructors sets the specified kind of context.

# Methods

This section describes the methods of the `BXBaseView` class.

## debug( )

Writes the specified message to the system output stream.

### *Syntax*

```
void debug(String msg);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `msg` | A string that contains the message. |

## set( )

Associates a status object with this object.

### *Syntax*

```
void set(BXStatus stat);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `stat` | The `BXStatus` object to associate with this object. |

### setBillingContext( )

Associates a billing context with this object.

### *Syntax*

```
void setBillingContext(BXBillingContext context);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `context` | The `BXBillingContext` object to associate with this object. |

# BXConfigObj Class Reference

The `BXConfigObj` class provides access several configuration parameters. You should use the static strings defined in this class instead of the methods.

## Class Definition

### *Package*

```
com.iplanet.ecomm.billxb2b.common
```

### *Syntax*

```
public class BXConfigObj { ... };
```

### *Variables*

You can use the following static variables within your application:

### *Syntax*

```
public final static String BILLER_NAME_DEF
    = "biller_section.biller-name";
```

```
public final static String MERCHANT_ID_DEF
   = "biller_section.merchant-id";

public static final String SESSION_TIMEOUT = "BX_FE_SESSION_TIME";
public static final String NUM_STMTS = "BX_FE_NUM_STMTS";

public final static String BILLER_ID = BXConfigObj.getBillerID();
public final static String BILLER_NAME =
                BXConfigObj.getBillerID();
public final static String MERCHANT_ID =
                BXConfigObj.getMerchantID();
```

| | |
|---|---|
| BILLER_NAME_DEF | Biller name definition section |
| MERCHANT_ID_DEF | Merchant ID definition section |
| SESSION_TIMEOUT | Session timeout value |
| NUM_STMTS | Number of statements |
| BILLER_ID | Biller ID |
| BILLER_NAME | Biller name |
| MERCHANT_ID | Merchant ID |

# BXErrorCode Class Reference

The BXError class defines constants for error codes; it lists all errors used by the BillerXpert front end. To define your own error codes and constants, you can create your own subclass of this class or simply add error codes to this class.

## Class Definition

### *Package*

```
com.iplanet.ecomm.billxb2b.mail_handler
```

### *Syntax*

```
public class BXErrorCode
{
  public final static String VALIDATE_UNKNOWN          =
"BXFE_000001";
```

```
  public final static String UNKNOWN_ERROR            =
"BXFE_100000";
  public final static String DB_BILLINGMGR_ERROR       =
"BXFE_100001";
  public final static String DB_MEMBERMGR_ERROR        =
"BXFE_100002";
  public final static String AL_INVALID_REQUEST        =
"BXFE_100003";
  public final static String DB_PAYMENTMGR_ERROR       =
"BXFE_100004";
  public final static String ACCESS_NOT_ALLOWED       =
"BXFE_100005";



  // Sign on view


  public final static String ONLINENAME_MISSING        =
"BXFE_001000";
  public final static String PASSWORD_MISSING          =
"BXFE_001001";
  public final static String ONLINENAME_INVALID        =
"BXFE_001002";
  public final static String PASSWORD_INVALID          =
"BXFE_001003";
  public final static String SESSION_CREATE_FAILED     =
"BXFE_001004";
  public final static String USER_NOT_FOUND            =
"BXFE_001005";
  public final static String USER_STATUS_NOTACTIVE     =
"BXFE_001006";



  public final static String BCA_MISSING               =
"BXFE_002000";
  public final static String NUM_MONTH_MISSING         =
"BXFE_002001";
  public final static String BCA_INVALID               =
"BXFE_002002";
  public final static String NUM_MONTH_INVALID         =
"BXFE_002003";
```

```
  public final static String BCA_NOT_FOUND            =
"BXFE_002004";


  // simple and advance search
  public final static String SEARCH_BCA_MISSING       =
"BXFE_002500";
  public final static String SEARCH_BALANCE_MISSING   =
"BXFE_002501";
  public final static String SEARCH_AMOUNT_MISSING    =
"BXFE_002502";
  public final static String SEARCH_DURATION_MISSING  =
"BXFE_002503";
  public final static String SEARCH_BCA_INVALID       =
"BXFE_002504";
  public final static String SEARCH_BALANCE_INVALID   =
"BXFE_002505";
  public final static String SEARCH_AMOUNT_INVALID    =
"BXFE_002506";
  public final static String SEARCH_DURATION_INVALID  =
"BXFE_002507";
  public final static String SEARCH_TIME_INVALID      =
"BXFE_002508";
  public final static String SEARCH_TIME_AFTER_ERROR  =
"BXFE_002509";
  public final static String SEARCH_TIME_BEFORE_ERROR =
"BXFE_002510";


  // statement, details
  public final static String BILLDOC_NOT_FOUND        =
"BXFE_003000";
  public final static String DOC_CURRENT_NOT_FOUND    =
"BXFE_003001";
  public final static String BCA_NOT_SELECTED         =
"BXFE_003002";
  public final static String TEMPLATE_BASE_NOT_FOUND  =
"BXFE_003003";


  // change user profile
  public final static String PWD_MATCH_FAILED         =
"BXFE_004000";
  public final static String NEW_PASSWORD_MISSING     =
"BXFE_004001";
```

```
  public final static String REPEAT_PASSWORD_MISSING   =
"BXFE_004002";
  public final static String NEW_PASSWORD_INVALID      =
"BXFE_004003";
  public final static String REPEAT_PASSWORD_INVALID   =
"BXFE_004004";


  public final static String UP_CONTACT_EMPTY          =
"BXFE_004500";


  public final static String UP_FIN_ACCT_NOT_FOUND     =
"BXFE_004600";



  // user registration errors
  public final static String REG_BCA_MISSING           =
"BXFE_005000";
  public final static String REG_BALANCE_MISSING       =
"BXFE_005001";
  public final static String REG_BCA_INVALID           =
"BXFE_005002";
  public final static String REG_BALANCE_INVALID       =
"BXFE_005003";
  public final static String REG_BANK_ROUTING_INVALID  =
"BXFE_005004";
  public final static String REG_BANK_ACCT_INVALID     =
"BXFE_005005";
  public final static String REG_BANK_NICKNAME_INVALID =
"BXFE_005006";
  public final static String REG_CC_TYPE_INVALID       =
"BXFE_005007";
  public final static String REG_CC_NUMBER_INVALID     =
"BXFE_005008";
  public final static String REG_CC_EXP_MONTH_INVALID  =
"BXFE_005009";
  public final static String REG_CC_EXP_YEAR_INVALID   =
"BXFE_005010";
  public final static String REG_CC_NICKNAME_INVALID   =
"BXFE_005011";
  public final static String REG_FIRSTNAME_MISSING     =
"BXFE_005012";
```

```
  public final static String REG_LASTNAME_MISSING      =
"BXFE_005013";

  public final static String REG_FIRSTNAME_INVALID     =
"BXFE_005014";

  public final static String REG_LASTNAME_INVALID      =
"BXFE_005015";

  public final static String REG_MIDDLENAME_INVALID    =
"BXFE_005016";

  public final static String REG_PREFIX_INVALID        =
"BXFE_005017";

  public final static String REG_SUFFIX_INVALID        =
"BXFE_005018";

  public final static String REG_CA_INVALID            =
"BXFE_005019";

  public final static String REG_EMAIL_INVALID         =
"BXFE_005020";

  public final static String REG_CC_BRAND_MISSING      =
"BXFE_005021";

  public final static String REG_CC_NUMBER_MISSING     =
"BXFE_005022";

  public final static String REG_CC_EXP_MONTH_MISSING  =
"BXFE_005023";

  public final static String REG_CC_EXP_YEAR_MISSING   =
"BXFE_005024";

  public final static String REG_CC_HOLDERNAME_MISSING =
"BXFE_005025";

  public final static String REG_CC_ADDRESS1_MISSING   =
"BXFE_005026";

  public final static String REG_CC_CITY_MISSING       =
"BXFE_005027";

  public final static String REG_CC_STATE_MISSING      =
"BXFE_005028";

  public final static String REG_CC_POSTALCODE_MISSING =
"BXFE_005029";

  public final static String REG_CC_BRAND_INVALID      =
"BXFE_005030";

  public final static String REG_CC_HOLDERNAME_INVALID =
"BXFE_005031";

  public final static String REG_CC_ADDRESS1_INVALID   =
"BXFE_005032";

  public final static String REG_CC_CITY_INVALID       =
"BXFE_005033";

  public final static String REG_CC_STATE_INVALID      =
"BXFE_005034";
```

```
  public final static String REG_CC_POSTALCODE_INVALID =
"BXFE_005035";


  public final static String REG_CC_VISA_BAD           =
"BXFE_005036";
  public final static String REG_CC_MASTERCARD_BAD     =
"BXFE_005037";
  public final static String REG_CC_AMEX_BAD           =
"BXFE_005038";
  public final static String REG_CC_DISCOVER_BAD       =
"BXFE_005039";
  public final static String REG_CC_DINERSCLUB_BAD     =
"BXFE_005040";


  public final static String REG_CC_ONLINENAME_MISSING =
"BXFE_005041";
  public final static String REG_CC_ONLINENAME_INVALID =
"BXFE_005042";
  public final static String REG_CC_PASSWORD_MISSING   =
"BXFE_005043";
  public final static String REG_CC_PASSWORD_INVALID   =
"BXFE_005044";


  public final static String REG_CC_DATE_INVALID       =
"BXFE_005045";
  public final static String REG_CC_MOD10_FAILED       =
"BXFE_005046";


  public final static String REG_CK_ROUTING_MISSING    =
"BXFE_005047";
  public final static String REG_CK_ACCT_MISSING       =
"BXFE_005048";
  public final static String REG_CK_ROUTING_INVALID    =
"BXFE_005049";
  public final static String REG_CK_ACCT_INVALID       =
"BXFE_005050";
  public final static String REG_CK_NICKNAME_INVALID   =
"BXFE_005051";
  public final static String REG_CC_BAD_CARD_FORMAT      =
"BXFE_005052";
  public final static String REG_CC_NUMBER_BLANK_INVALID =
"BXFE_005053";
```

```
  public final static String REG_EMAIL_MISSING         =
"BXFE_005054";
  public final static String REG_PHONE_MISSING         =
"BXFE_005055";


  public final static String REG_PAY_METHOD_MISSING    =
"BXFE_005056";
  public final static String REG_CK_MOD10_FAILED       =
"BXFE_005057";
  public final static String REG_CK_BAD_ROUTING_FORMAT =
"BXFE_005058";



  // bca related view
  public final static String BCA_ADDRESS1_INVALID      =
"BXFE_005500";
  public final static String BCA_ADDRESS2_INVALID      =
"BXFE_005501";
  public final static String BCA_ADDRESS3_INVALID      =
"BXFE_005502";
  public final static String BCA_CITY_INVALID          =
"BXFE_005503";
  public final static String BCA_STATE_INVALID         =
"BXFE_005504";
  public final static String BCA_POSTALCODE_INVALID    =
"BXFE_005505";
  public final static String BCA_ADDRESS1_MISSING      =
"BXFE_005506";
  public final static String BCA_CITY_MISSING          =
"BXFE_005507";
  public final static String BCA_STATE_MISSING         =
"BXFE_005508";
  public final static String BCA_POSTALCODE_MISSING    =
"BXFE_005509";


  // pay view
  public final static String PAY_BCA_NOT_FOUND         =
"BXFE_006000";
  public final static String PAY_NOT_SELECT            =
"BXFE_006001";
```

```
  public final static String PAY_METHOD_NOT_SELECT     =
"BXFE_006002";

  public final static String PAY_INVALID_PAYDATE       =
"BXFE_006003";

  public final static String PAY_AFTER_DUEDATE         =
"BXFE_006004";

  public final static String PAY_BEFORE_CURRENTDATE    =
"BXFE_006005";

  public final static String PAYMENT_CANCEL_NOT_SELECT =
"BXFE_006006";

  public final static String PAY_PAST_DUE              =
"BXFE_006007";


   // Error Code related to AutoPay

   public final static String NO_ACCT_FOR_AUTOPAY         =
"BXFE_006200";

   public final static String NO_USER_FOR_ACCT           =
"BXFE_006201";

   public final static String FAILED_TO_DSP_ADD_AUTOPAY       =
"BXFE_006202";

   public final static String NO_AUTOPAY_SCHEDULE       =
"BXFE_006203";

   public final static String FAILED_TO_DSP_EDIT_AUTOPAY       =
"BXFE_006204";

   public final static String FAILED_TO_ADD_AUTOPAY       =
"BXFE_006205";

   public final static String FAILED_TO_UPDATE_AUTOPAY       =
"BXFE_006206";

   public final static String FAILED_TO_DELETE_AUTOPAY       =
"BXFE_006207";

   public final static String AUTOPAY_AMT_ERR = "BXFE_006208";



}
```

# BXSession Class Reference

The BXSession class represents a BillerXpert session. It contains the user profile information of the user that is currently logged in and some application configuration variables. You use the BXSession object to maintain state information within a Servlet and to transfer this information between Servlets and JSPs. Each BillerXpert Servlet has access to the session object by calling the Servlet's `getBXSession() method`.

## Methods

Summary list:

| | |
|---|---|
| **Constructor** | |
| `BXSession()` | Creates a BXSession object. |
| **User and service profile manipulation** | |
| `isUserProfile()` | Determines whether a user profile exists. |
| `setUserProfile()` | Specifies that a user profile exists. |
| `getProfileExtension()` | Determines the profile extension data. |
| `setProfileExtension()` | Specifies the profile extension data. |
| `getServiceExtension()` | Determines the service extension data. |
| `setServiceExtension()` | Specifies the service extension data. |
| `parseExtension()` | Determines the elements of an extension field. |
| **Getting and setting object fields** | |
| **Advertising support** | |
| `getAdTargetString()` | Creates a target string for an advertisement based on user profile options. |
| **Getting and setting object fields** | |
| `saveValString()` | Specifies a value for a session data field and saves all session data. |
| `setValString()` | Specifies a value for a session data field. |
| `getUserID()` | Determines the user ID. |
| `setUserID()` | Specifies the user ID. |
| `getAccount()` | Determines the account number. |
| `setAccount()` | Specifies the account number. |

| | |
|---|---|
| `getPrefix()` | Determines the name prefix. |
| `setPrefix()` | Specifies the name prefix. |
| `getSuffix()` | Determines the name suffix. |
| `setSuffix()` | Specifies the name suffix. |
| `getLastName()` | Determines the last name. |
| `setLastName()` | Specifies the last name. |
| `getFirstName()` | Determines the first name. |
| `setFirstName()` | Specifies the first name. |
| `getMiddleName()` | Determines the middle name. |
| `setMiddleName()` | Specifies the middle name. |
| `getNickName()` | Determines the nick name. |
| `setNickName()` | Specifies the nick name. |
| `getDescription()` | Determines the description. |
| `setDescription()` | Specifies the description. |
| `getAddress()` | Determines the address. |
| `setAddress()` | Specifies the address. |
| `getDayPhone()` | Determines the day-time phone number. |
| `setDayPhone()` | Specifies the day-time phone number. |
| `getNightPhone()` | Determines the night-time phone number. |
| `setNightPhone()` | Specifies the night-time phone number. |
| `getFaxNumber()` | Determines the fax number. |
| `setFaxNumber()` | Specifies the fax number. |
| `getEmail()` | Determines the e-mail address. |
| `setEmail()` | Specifies the e-mail address. |
| `getOnlineName()` | Determines the online name. |
| `setOnlineName()` | Specifies the online name. |
| `getPassword()` | Determines the password. |
| `setPassword()` | Specifies the password. |
| `getLanguage()` | Determines the language. |
| `setLanguage()` | Specifies the language. |

| | |
|---|---|
| `getPrimaryContact()` | Determines the primary contact. |
| `setPrimaryContact()` | Specifies the primary contact. |
| `getSecondaryContact()` | Determines the secondary contact. |
| `setSecondaryContact()` | Specifies the secondary contact. |
| `getSalutation()` | Determines the salutation. |
| `setSalutation()` | Specifies the salutation. |
| `getDefaultPaymentMethod()` | Determines the default payment method. |
| `setDefaultPaymentMethod()` | Specifies the default payment method. |
| `getNumberMonth()` | Determines the number of months data to display. |
| `setNumberMonth()` | Specifies the number of months data to display. |
| `getSelectedAccount()` | Determines the current account. |
| `setSelectedAccount()` | Specifies the current account. |
| `getSortField()` | Determines the sort field. |
| `setSortField()` | Specifies the sort field. |
| `getBCAs()` | Determines the list of billing customer accounts. |
| `setBCAs()` | Specifies the list of billing customer accounts. |
| `getSearchAttributes()` | Determines the search attributes. |
| `setSearchAttributes()` | Specifies the search attributes. |
| getManager() | get user's manager name |
| setManager() | set user's manager name |
| getApprover() | get user's approver name |
| getReferenceCode() | get user's dept number |
| setReferenceCode() | set user's dept number |
| getBillerId() | get biller name |
| setBillerId() | set biller name |
| getCompanyId() | get company name |

| | |
|---|---|
| setCompanyId() | set company name |
| getServicePage() | get variable to determine which nav links to display |
| setServicePage() | set this variable based on user's privilege |
| getTemplateBase() | get biller's jsp template base |
| setTemplateBase() | set biller's jsp template base |
| getSignOnPage() | get this biller's sign on page name |
| setSignOnPage() | set this biller's sign on page name |
| getPageLength() | get the number of records to display on one page |
| setPageLength() | set this number |
| getSessionTimeout() | get the timeout number from configuration |
| setSessionTimeout() | set the timeout |
| getClusterId() | get biller's cluster id |
| getBaseCurrency() | get biller's base currency |
| setBaseCurrency() | set biller's base currency |
| getSecondaryCurrency() | get company's display currency |
| setSecondaryCurrency() | set company's display currency |
| **Debugging** | |
| `debug()` | Displays session data. |

# Class Definition

## *Package*

`com.iplanet.ecomm.billxb2b.commom`

## *Syntax*

```
class BXSession implements Serializable.
```

# Constructors

## BXSession( )

Creates a BXSession object.

### *Syntax*

```
BXSession;
```

### *Discussion*

You do not typically create a BXSession object directly; rather, you call your Servlet's `createBXSession()` method.

# Methods

This section describes the methods of the `BXSession` class.

## debug( )

Displays session data.

### *Syntax*

```
void debug();
```

### *Discussion*

All session object key and value pairs are written as informational messages to the log file.

## getAccount( )

Determines the account number.

### *Syntax*

```
String getAccount();
```

### *Returns*

A string that contains the customer's account number.

### *Discussion*

This method returns the value associated with the "Account" key in the session object.

## getAddress( )

Determines the address.

### *Syntax*

```
String getAddress();
```

### *Returns*

A string that contains the address.

### *Discussion*

This method returns the value associated with the “Address” key in the session object.

## getAdTargetString( )

Creates a target string for an advertisement based on user profile options.

### *Syntax*

```
String getAdTargetString()
```

### *Returns*

A string that contains the target information.

### *Discussion*

The target string consists of the e-mail address, language, and profile extensions, separated by delimiters.

## getBCAs( )

Determines the list of billing customer accounts.

### *Syntax*

```
Vector getBCAs();
```

### *Returns*

A vector object that contains the accounts.

### *Discussion*

This method returns the value associated with the “bcas” key in the session object.

## getDayPhone( )

Determines the day-time phone number.

### *Syntax*

```
String getDayPhone();
```

### *Returns*

A string that contains the phone number.

### *Discussion*

This method returns the value associated with the “DayPhone” key in the session object.

## getDefaultPaymentMethod( )

Determines the default payment method.

### *Syntax*

```
String getDefaultPaymentMethod();
```

### *Returns*

A string that contains the phone number.

### *Discussion*

This method returns the value associated with the “DefaultPaymentMethod” key in the session object.

## getDescription( )

Determines the description.

### *Syntax*

```
String getDescription();
```

### *Returns*

A string that contains the description.

### *Discussion*

This method returns the value associated with the “Description” key in the session object.

## getEmail( )

Determines the e-mail address.

### *Syntax*

```
String getEmail();
```

### *Returns*

A string that contains the e-mail address.

### *Discussion*

This method returns the value associated with the “email” key in the session object.

## getFaxNumber( )

Determines the fax number.

### *Syntax*

```
String getFaxNumber();
```

### *Returns*

A string that contains the fax number.

### *Discussion*

This method returns the value associated with the “FaxNumber” key in the session object.

## getFirstName( )

Determines the first name.

### *Syntax*

```
String getFirstName();
```

### *Returns*

A string that contains the name.

### *Discussion*

This method returns the value associated with the “Firstname” key in the session object.

## getLanguage( )

Determines the language.

### *Syntax*

```
String getLanguage();
```

### *Returns*

A string that contains the language.

### *Discussion*

This method returns the value associated with the “Language” key in the session object.

## getLastName( )

Determines the last name.

### *Syntax*

```
String getLastName();
```

### *Returns*

A string that contains the name.

### *Discussion*

This method returns the value associated with the “Lastname” key in the session object.

## getMiddleName( )

Determines the middle name.

### *Syntax*

```
String getMiddleName();
```

### *Returns*

A string that contains the name.

### *Discussion*

This method returns the value associated with the “Middlename” key in the session object.

## getNickName( )

Determines the nick name.

### *Syntax*

```
String getNickName();
```

### *Returns*

A string that contains the name.

### *Discussion*

This method returns the value associated with the “NickName” key in the session object.

## getNightPhone( )

Determines the night-time phone number.

### *Syntax*

```
String getNightPhone();
```

### *Returns*

A string that contains the phone number.

### *Discussion*

This method returns the value associated with the “NightPhone” key in the session object.

## getNumberMonth( )

Determines the number of months data to display.

### *Syntax*

```
int getNumberMonth();
```

### *Returns*

A string that contains the number.

### *Discussion*

This method returns the value associated with the “NumberMonth” key in the session object.

## getOnlineName( )

Determines the online name.

### *Syntax*

```
String getOnlineName();
```

### *Returns*

A string that contains the name.

### *Discussion*

This method returns the value associated with the “OnlineName” key in the session object.

## getPassword( )

Determines the password.

### *Syntax*

```
String getPassword();
```

### *Returns*

A string that contains the encrypted password.

### *Discussion*

This method returns the value associated with the “Password” key in the session object.

## getPrefix( )

Determines the name prefix.

### *Syntax*

```
String getPrefix();
```

### *Returns*

A string that contains the prefix.

### *Discussion*

This method returns the value associated with the “Prefix” key in the session object.

## getPrimaryContact( )

Determines the primary contact.

### *Syntax*

```
String getPrimaryContact();
```

### *Returns*

A string that contains the primary contact.

### *Discussion*

This method returns the value associated with the “PrimaryContact” key in the session object.

## getProfileExtension( )

Determines the profile extension data.

### *Syntax*

```
String getProfileExtension();
```

### *Returns*

A string that contains the profile extension.

### *Discussion*

This method returns the value associated with the “ProfileExtension” key in the session object.

## getSalutation( )

Determines the salutation.

### *Syntax*

```
String getSalutation();
```

### *Returns*

A string that contains the salutation.

### *Discussion*

This method returns the value associated with the “Salutation” key in the session object.

## getSearchAttributes( )

Determines the search attributes.

### *Syntax*

```
String getSearchAttributes();
```

### *Returns*

A string that contains the search attributes.

### *Discussion*

This method returns the value associated with the “searchAttributes” key in the session object.

## getSecondaryContact( )

Determines the secondary contact.

### *Syntax*

```
Vector getSecondaryContact();
```

### *Returns*

A vector that contains the secondary contacts.

### *Discussion*

This method returns the value associated with the “SecondaryContact” key in the session object.

## getSelectedAccount( )

Determines the current account.

### *Syntax*

```
String getSelectedAccount();
```

### *Returns*

A string that contains the account.

### *Discussion*

This method returns the value associated with the “SelectedAccount” key in the session object.

## getServiceExtension( )

Determines the service extension data.

### *Syntax*

```
String getServiceExtension();
```

### *Returns*

A string that contains the service extension.

### *Discussion*

This method returns the value associated with the “ServiceExtension” key in the session object.

## getSuffix( )

Determines the name suffix.

### *Syntax*

```
String getSuffix();
```

### *Returns*

A string that contains the suffix.

### *Discussion*

This method returns the value associated with the “Suffix” key in the session object.

## getUserID( )

Determines the user ID.

### *Syntax*

```
String getUserID();
```

### *Returns*

A string that contains the ID.

### *Discussion*

This method returns the value associated with the “UserID” key in the session object.

## setAccount( )

Specifies the account number.

### *Syntax*

```
void setAccount( String value );
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | A string that specifies the account. |

## setAddress( )

Specifies the address.

### *Syntax*

```
void setAddress(String value);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | A string that specifies the address. |

### *Discussion*

This method changes the value associated with the “Address” key in the session object.

## setBCAs( )

Specifies the list of billing customer accounts.

### *Syntax*

```
void setBCAs(Vector bcaList);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `bcaList` | A Vector object that specifies the accounts. |

### *Discussion*

This method changes the value associated with the “bcas” key in the session object.

## setDayPhone( )

Specifies the day-time phone number.

### *Syntax*

```
void setDayPhone( String value);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | A string that specifies the phone number. |

### *Discussion*

This method changes the value associated with the “DayPhone” key in the session object.

## setAccount( )

Specifies the account number.

### *Syntax*

```
void setAccount( String value );
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | A string that specifies the account. |

## setAddress( )

Specifies the address.

### *Syntax*

```
void setAddress(String value);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | A string that specifies the address. |

## setBCAs( )

Specifies the list of billing customer accounts.

### *Syntax*

```
void setBCAs(Vector bcaList);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `bcaList` | A Vector object that specifies the accounts. |

### *Discussion*

This method changes the value associated with the “bcas” key in the session object.

## setDayPhone( )

Specifies the day-time phone number.

### *Syntax*

```
void setDayPhone(String value);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | A string that specifies the phone number. |

### *Discussion*

This method changes the value associated with the “DayPhone” key in the session object.

## setDefaultPaymentMethod( )

Specifies the default payment method.

### *Syntax*

```
void setDefaultPaymentMethod(String value);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | A string that specifies the payment method. |

### *Discussion*

This method changes the value associated with the “DefaultPaymentMethod” key in the session object.

## setDescription( )

Specifies the description.

### *Syntax*

```
void setDescription( String value);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | A string that specifies the description. |

### *Discussion*

This method changes the value associated with the “Description” key in the session object.

## setFaxNumber( )

Specifies the fax number.

### *Syntax*

```
void setFaxNumber( String value);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | A string that specifies the fax number. |

### *Discussion*

This method changes the value associated with the “FaxNumber” key in the session object.

## setFirstName( )

Specifies the first name.

### *Syntax*

```
void setFirstName( String value );
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | A string that specifies the name. |

### *Discussion*

This method changes the value associated with the “FirstName” key in the session object.

## setLanguage( )

Specifies the language.

### *Syntax*

```
void setLanguage( String value );
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | A string that specifies the language. |

### *Discussion*

This method changes the value associated with the “Language” key in the session object.

## setLastName( )

Specifies the last name.

### *Syntax*

```
void setLastName( String value );
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | A string that specifies the name. |

### *Discussion*

This method changes the value associated with the “LastName” key in the session object.

## setMiddleName( )

Specifies the middle name.

### *Syntax*

```
void setMiddleName( String value );
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | A string that specifies the name. |

### *Discussion*

This method changes the value associated with the “MiddleName” key in the session object.

## setNickName( )

Specifies the nick name.

### *Syntax*

```
void setNickName( String value );
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | A string that specifies the name. |

### *Discussion*

This method changes the value associated with the “NickName” key in the session object.

## setNightPhone( )

Specifies the night-time phone number.

### *Syntax*

```
void setNightPhone(String value);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | A string that specifies the phone number. |

### *Discussion*

This method changes the value associated with the “NightPhone” key in the session object.

## setNumberMonth( )

Specifies the number of months data to display.

### *Syntax*

```
void setNumberMonth(int value);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | A string that specifies the number of months. |

### *Discussion*

This method changes the value associated with the “NumberMonth” key in the session object.

## setOnlineName( )

Specifies the online name.

### *Syntax*

```
void setOnlineName( String value);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | A string that specifies the name. |

### *Discussion*

This method changes the value associated with the “OnlineName” key in the session object.

## setPassword( )

Specifies the password.

### *Syntax*

```
void setPassword( String value);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | A string that specifies the password. |

### *Discussion*

This method changes the value associated with the “Password” key in the session object.

## setPrefix( )

Specifies the name prefix.

### *Syntax*

```
void setPrefix( String value );
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | A string that specifies the prefix. |

### *Discussion*

This method changes the value associated with the “Prefix” key in the session object.

## setPrimaryContact( )

Specifies the primary contact.

### *Syntax*

```
void setPrimaryContact( String value);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | A string that specifies the primary contact. |

### *Discussion*

This method changes the value associated with the “PrimaryContact” key in the session object.

## setProfileExtension( )

Specifies the profile extension data.

### *Syntax*

```
void setProfileExtension( String value);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | A string that specifies the profile extension. |

### *Discussion*

This method changes the value associated with the “ProfileExtension” key in the session object.

## setSalutation( )

Specifies the salutation.

### *Syntax*

```
void setSalutation(String value);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | A string that specifies the salutation. |

### *Discussion*

This method changes the value associated with the “Salutation” key in the session object.

## setSearchAttributes( )

Specifies the search attributes.

### *Syntax*

```
void setSearchAttributes(String value);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | A string that specifies the salutation. |

### *Discussion*

This method changes the value associated with the “searchAttributes” key in the session object.

## setSecondaryContact( )

Specifies the secondary contact.

### *Syntax*

```
void setSecondaryContact( Vector value);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | A vector that specifies the secondary contacts. |

### *Discussion*

This method changes the value associated with the “SecondaryContact” key in the session object.

## setSelectedAccount( )

Specifies the current account.

### *Syntax*

```
void setSelectedAccount( Vector value);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | A vector that specifies the account. |

### *Discussion*

This method changes the value associated with the “SelectedAccount” key in the session object.

## setServiceExtension( )

Specifies the service extension data.

### *Syntax*

```
void setServiceExtension( String value);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | A string that specifies the service extension. |

### *Discussion*

This method changes the value associated with the “ServiceExtension” key in the session object.

## setSortField( )

Specifies the sort field.

### *Syntax*

```
void setSortField( String value );
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | A string that specifies the sort field. |

### *Discussion*

This method changes the value associated with the “SortField” key in the session object.

## setSuffix( )

Specifies the name suffix.

### *Syntax*

```
void setSuffix( String value );
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | A string that specifies the suffix. |

### *Discussion*

This method changes the value associated with the “Suffix” key in the session object.

## setUserID( )

Specifies the user ID.

### *Syntax*

```
void setUserID(String value);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | A string that specifies the ID. |

### *Discussion*

This method changes the value associated with the “UserID” key in the session object.

# BXBaseBean Class Reference

Instances of the `BXBaseBean` class represent BillerXpert bean data. You typically create a `BXBaseBean` object to define the data you want to display.

## Methods

Summary list:

**Constructor**

| | |
|---|---|
| `BXBaseBean()` | Creates a `BXBaseBean` object. |

**Pagination Support**

| | |
|---|---|
| `getHREF()` | Creates a link to a given page number. |
| `setPagingData()` | Sets the page count and URL. |
| `setpageIndex()` | |
| `getPageIndex()` | Gets the pagination output. |

**Template manipulation**

| | |
|---|---|
| `format()` | Formats a double-precision floating point number. |

## Class Definition

### *Package*

```
com.iplanet.ecomm.billxb2b.fe.telco
```

### *Syntax*

```
public class BXBaseBean { ... }
```

## Constructor

### BXBaseBean( )

Creates a `BXBaseBean` object.

### *Syntax*

```
BXBaseBean();
```

# Methods

This section describes the methods of the `BXBaseBean` class reference.

## format( )

Formats a double-precision floating point number.

### *Syntax*

```
String format(double value, int maxDigits);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | The double-precision floating point number to format. |
| `maxDigits` | The number of digits to include to the right of the decimal point. |

### *Returns*

A string that contains the formatted number.

## getHREF( )

Creates a link to a given page number.

### *Syntax*

```
String getHREF(int pageNumber);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `pageNumber` | Thepage's HTML anchor of a given page number. |

### setPagingData( )

Sets the page count and URL.

#### *Syntax*

```
void setPagingData (int count, int page, String url);
```

#### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `count` | The total page count. |
| `page` | The current page index. |
| `url` | The url the hyperlinks will invoke. |

### setPageIndex( )

Sets the page count and URL.

#### *Syntax*

```
void setPageIndex (String url);
```

### getPageIndex( )

#### *Syntax*

```
String getPageIndex()
```

#### *Returns*

A string that contains' the page's index.

## BXTemplateUtil Class Reference

The `BXTemplateUtil` class provides several static methods that you can use in your derived subclasses of `BXBaseBean`.

#### *Methods*

Summary list:

**Convenience methods**

| | |
|---|---|
| `addressToString()` | Creates a string from an `IBXAddress` object. |
| `format()` | Formats a double-precision floating point number. |
| `dateToString()` | Creates a string from a `Date` object. |
| `formatCurrency()` | Formats currency based on a given Locale. |

# Class Definition

### *Package*

```
com.iplanet.ecomm.billxb2b.user
```

### *Syntax*

```
public class BXTemplateUtil { ... }
```

# Methods

This section describes the methods of the `BXTemplateUtil` class.

## addressToString( )

Creates a string from an `IBXAddress` object.

### *Syntax*

```
static String addressToString(IBXAddress address);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `address` | An `IBXAddress` object that contains the address. |

### *Returns*

A string that contains the date.

### *Discussion*

The string contains the three address lines separated by the `<br>` tag, followed by the city, state, and postal code in the format of `'<br>city, state, postal'`.

## format( )

Formats a double-precision floating point number.

### *Syntax*

```
String format(double value, int maxDigits);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | The double-precision floating point number to format. |
| `maxDigits` | The number of digits to include to the right of the decimal point. |

### *Returns*

A string that contains the formatted number.

## dateToString( )

Creates a string from a `Date` object.

### *Syntax*

```
static String dateToString(Date date)
```

```
static String dateTo String (Date date, Locale locale)
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `date` | A Date object that contains the date. |
| `locale` | The locale to format the date with. |

### *Returns*

A string that contains the date.

### *Discussion*

The format of the date is ‘day, month date, year’, such as ‘Saturday, July 12, 1997’.

## setCurrency( )

Maps the currency value to a specified key.

### *Syntax*

```
public Void setCurrency (String key, double value);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `key` | The key to map the value to. |
| `value` | The currency value. |

## formatCurrency( )

Formats the currency based on a given locale.

### *Syntax*

```
public static String formatCurrency(double value, int maxDigits,
Locale locale);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `value` | The double-precision floating point number to format. |
| `maxDigits` | The number of digits to include to the right of the decimal point. |
| `locale` | The locale in which to format. |

# BXValidateView Class Reference

The `BXValidateView` class is an abstract class from which validation view classes are derived. Your subclass must implement the `validate()` and `setError()` methods.

# Methods

Summary list:

**Constructor**

| | |
|---|---|
| `BXValidateView()` | Creates a BXValidateView object. |

**Abstract methods**

| | |
|---|---|
| `validate()` | Performs validation of input parameters. |
| `setError()` | Specifies the input parameters that contain errors. |

**Utility methods**

| | |
|---|---|
| `parmCount()` | Determines the number of parameters to validate. |
| `errorCount()` | Determines the error count. |
| `getParmList()` | Obtains the list of input parameters. |
| `insertError()` | Adds a parameter to the list of errors. |
| `getErrorCodeByName()` | Determines the error code by name. |
| `getNumErrors()` | Determines the number of errors. |
| `getMinLength()` | Determines the minimum length allowed for a parameter. |
| `printInParms()` | Writes the list of input parameters to the log file. |

# Class Definition

## *Package*

```
com.iplanet.ecomm.billxb2b.common
```

## *Syntax*

```
public abstract class BXValidateView { ... }
```

# Variables

The following `IValList` objects and status information can be used within your AppLogic:

### *Syntax*

```
protected IValList parmList;
protected IValList errorList;
protected BXStatus status;
```

| | |
|---|---|
| `parmList` | An `IValList` object that contains the parameters to be validated. |
| `errorList` | An `IValList` object that contains the parameters that failed validation. |
| `status` | Reference to the stackable status object. |

# Constructors

## BXValidateView

Initializes variables.

### *Syntax*

```
BXValidateView();
```

### *Discussion*

Your subclass constructor should call this constructor.

# Methods

This section describes the methods of the `BXValidateView` class.

## errorCount( )

Determines the error count.

### *Syntax*

```
public int errorCount()
```

### *Returns*

The error count.

## getErrorCodeByName( )

Determines the error code by name.

### *Syntax*

```
public int getErrorCodeByName (String fieldName)
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `fieldName` | The error code name. |

### *Returns*

The error code as an integer value.

## getMinLength( )

Determines the minimum length allowed for a parameter.

### *Syntax*

```
static int getMinLength(String className, String attrName);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `className` | The name of the class in which the parameter is defined. |
| `attrName` | The name of the input parameter. |

### *Returns*

An integer that specifies the minimum length.

### *Discussion*

The minimum value is stored in the `tabledef.conf` file.

### *See Also*

For information about the `tabledef.conf` file, see "Configuring Table Definitions," on page 45.

## getNumErrors( )

Determines the number of errors.

### *Syntax*

```
int getNumErrors();
```

### *Returns*

An integer that specifies the number of errors.

## getParmList( )

Obtains the list of input parameters.

### *Syntax*

```
IValList getParmList();
```

### *Returns*

An `IValList` object that specifies the number of errors.

## insertError( )

Adds a parameter to the list of errors.

### *Syntax*

```
void insertError(String fieldName, int errorCode);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `fieldName` | The a string that contains the name of the input parameter. |
| `errorCode` | An integer that specifies the error. |

## parmCount( )

Determines the number of parameters to validate.

### *Syntax*

```
int parmCount();
```

### *Returns*

An integer that specifies the number of parameters.

## printInParms( )

Writes the list of input parameters to the log file.

### *Syntax*

```
void printInParms();
```

### *Discussion*

Use this method for debugging only; it will decrease performance. The `printInParms()` method is a protected method.

## setError( )

Specifies the input parameters that contain errors.

### *Syntax*

```
abstract void setError();
```

### *Discussion*

The `setError()` method determines the parameters that are invalid and sets the appropriate error code in the status object. You must implement a `setError()` method in your derived subclass. The `setError()` method is a protected method.

## validate( )

Performs validation of input parameters.

### *Syntax*

```
abstract boolean validate();
```

### *Returns*

A boolean value that is true if the input parameter is valid; otherwise the value is false.

### *Discussion*

You must implement a `validate()` method in your derived subclass.

# BXValidateAttribute Class Reference

The `BXValidateAttribute` class represents the validation criteria. A `BXValidateAttribute` object specifies the criteria for a single attribute or parameter. A `BXValidateView` object typically uses a list of `BXValidateAttribute` objects to perform validation.

## Methods

Summary list:

**Constructor**

| | |
|---|---|
| `BXValidateAttribute()` | Creates a `BXValidateAttribute` object. |

**Getting attribute characteristics**

| | |
|---|---|
| `getAttribute()` | Determines the attribute name. |
| `getAttributeType()` | Determines the attribute type. |
| `getFormatString()` | Determines the format string. |
| `isRequired()` | Determines whether the attribute is required. |
| `getMin()` | Determines the minimum length allowed for an attribute. |
| `getMax()` | Determines the maximum length allowed for an attribute. |
| `getDateMin()` | Determines the minimum date allowed for an attribute. |
| `setDateMin()` | Specifies the minimum date allowed for an attribute. |
| `getDateMax()` | Determines the maximum date allowed for an attribute. |
| `setDateMax()` | Specifies the maximum date allowed for an attribute. |

# Class Definition

### *Package*

```
com.iplanet.ecomm.billxb2b.common
```

### *Syntax*

```
public class BXValidateAttribute { ... }
```

# Constructors

## BXValidateAttribute

Creates a `BXValidateAttribute` object.

### *Syntax*

```
BXValidateAttribute(String field, int type);
BXValidateAttribute( String field,
   int type,
   String format,
   boolean req,
   int min,
   int max);
```

### *Parameters*

The constructor has the following parameters:

| | |
|---|---|
| `field` | A string that contains the attribute name. |
| `type` | An integer that specifies the attribute type. |
| `format` | A string that contains the format of a string attribute. |
| `req` | A boolean value that specifies whether the attribute is required (true) or not (false). |
| `min` | An integer that specifies the minimum value. |
| `max` | An integer that specifies the maximum value. |

### *Example*

The following code adds two `BXValidateAttribute` objects to a list of validation criteria:

```
int oMinLength = 2;
int oMaxLength = BXValidateView.getMaxLength("BXCustomerProfile",
                                             "LOGINNAME");
int pMinLength = 2;
int pMaxLength = BXValidateView.getMaxLength("BXCustomerProfile",
                                             "PASSWORD");
Vector list = new Vector(2);
list.addElement(new BXValidateAttribute(ONLINE_NAME,
                BXValidator.BX_TYPE_STRING,
                "", true, oMinLength, oMaxLength));
list.addElement(new BXValidateAttribute(PASSWORD,
                BXValidator.BX_TYPE_STRING,
                "", true, pMinLength, pMaxLength));
```

### *See Also*

For information about attribute types, see “Attribute Types,” on page 112.

## Methods

This section describes the methods of the `BXValidateAttribute` class.

### getAttribute( )

Determines the attribute name.

#### *Syntax*

```
String getAttribute();
```

#### *Returns*

A string that specifies the name.

### getAttributeType( )

Determines the attribute type.

#### *Syntax*

```
int getAttributeType();
```

### *Returns*

An integer that specifies the type.

### *See Also*

For information about attribute types, see “Attribute Types,” on page 112.

## getDateMax( )

Determines the maximum date allowed for an attribute.

### *Syntax*

```
Date getDateMin();
```

### *Returns*

A `Date` object that specifies the date.

## getDateMin( )

Determines the minimum date allowed for an attribute.

### *Syntax*

```
Date getDateMin();
```

### *Returns*

A `Date` object that specifies the date.

## getFormatString( )

Determines the format string.

### *Syntax*

```
String getFormatString();
```

### *Returns*

A string that specifies the format.

## getFormatString( )

Determines the format string.

### *Syntax*

```
String getFormatString();
```

### *Returns*

A string that specifies the format.

## getMax( )

Determines the maximum length allowed for an attribute.

### *Syntax*

```
int getMax();
```

### *Returns*

An integer that specifies the maximum value.

## getMin( )

Determines the minimum length allowed for an attribute.

### *Syntax*

```
int getMin();
```

### *Returns*

An integer that specifies the minimum value.

## isRequired( )

Determines whether the attribute is required.

### *Syntax*

```
boolean isRequired();
```

### *Returns*

A boolean value that is true if the attribute is required; otherwise, it is false.

## setDateMax( )

Specifies the maximum date allowed for an attribute.

### *Syntax*

```
void setDateMax(Date max);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `max` | A `Date` object that specifies the date. |

### setDateMin( )

Specifies the minimum date allowed for an attribute.

### *Syntax*

```
void setDateMin(Date min);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `min` | A `Date` object that specifies the date. |

# BXValidator Class Reference

The `BXValidator` class provides a static `validate()` method that you can use in your BXValidateView subclass to perform validation. The `BXValidator` class also provides attribute type definitions, which you can use when you create a `BXValidateAttribute` object.

### *Methods*

Summary list:

**Validation**

| | |
|---|---|
| `validate()` | Performs validation of a vector of elements associated with a validation view. |

## Class Definition

### *Package*

`com.iplanet.ecomm.billxb2b.common`

### *Syntax*

```
public class BXValidator { ... }
```

## Attribute Types

You can use the following attribute types to specify the kind of attribute:

### *Syntax*

```
public static final int BX_TYPE_CURRENCY = 0;
public static final int BX_TYPE_DATETIME = 1;
public static final int BX_TYPE_DOUBLE = 2;
public static final int BX_TYPE_EMAIL = 3;
public static final int BX_TYPE_INT = 4;
public static final int BX_TYPE_PERCENT = 5;
public static final int BX_TYPE_PHONE = 6;
public static final int BX_TYPE_STRING = 7;
public static final int BX_TYPE_URL = 8;
public static final int BX_TYPE_REGEXP = 9;

BX_TYPE_CURRENCY Currency

BX_TYPE_DATETIME Date and time

BX_TYPE_DOUBLE Double

BX_TYPE_EMAIL E-mail

BX_TYPE_INT Integer

BX_TYPE_PHONE Phone number

BX_TYPE_STRING String

BX_TYPE_URL URL

BX_TYPE_REGEXP Regular expression
```

## Methods

This section describes the methods of the `BXValidator` class.

### validate( )

Performs validation of a vector of elements associated with a validation view.

#### *Syntax*

```
Static boolean validate (Vector elements, BXValidateView view)
```

#### *Parameters*

The method has the following parameters

| | |
|---|---|
| `:elements` | A `Vector` object that contains references to `BXValiidateAttribute` objects that identify the attributes to validate and the criteria to use. |
| `view` | A `BXValidateView` object that contains the data values to validate. |

#### *Returns*

A boolean value that is true if all elements of the vector are valid; otherwise, the value is false.

#### *Discussion*

The `validate()` method performs validation of the data in the specified validation view object using the criteria in the vector of elements. The `validate()` method matches each attribute name in the vector with the key in the view object's parameter list and performs validation using the associated criteria. An entry is inserted into the view's error list if the data is not valid. The `validate()` method is static.

# BXViewUtil Class Reference

The `BXViewUtil` class provides validation and conversion methods that you can use within your `BXValidateView` subclass.

## Methods

Summary list:

**Validation**

| | |
|---|---|
| `ccValidate()` | Performs credit card validation. |
| `routingNumberValidate` | Performs routing number validation. |
| **Conversion** | |
| `stringToVector()` | Converts a string to a vector. |
| `vectorToString()` | Converts a vector to a string. |
| `stringToAddress()` | Converts a string to an address. |
| `addressToString()` | Converts an address to a string. |
| `stringToDate()` | Converts a string to a date using the default format. |
| `stringToDateLongFormat()` | Converts a string to a date using the long format. |
| `getYear()` | Converts a year to a string that contains two digits. |
| `getYearLong()` | Converts a year to a string that contains four digits. |
| `getMonthNumber()` | Converts a month into its numeric representation. |
| `getMonthString()` | Converts a string that contains the numeric month into its name. |
| `parseExtension()` | Creates a hash table with the contents of an extension field. |

# Class Definition

### *Package*

```
com.iplanet.ecomm.billxb2b.user
```

### *Syntax*

```
public class BXViewUtil { ... }
```

# Methods

This section describes the methods of the `BXViewUtil` class.

## addressToString( )

Converts an address to a string.

### *Syntax*

```
static String addressToString(IBXAddress address);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `address` | An `IBXAddress` object that contains the address. |

### *Returns*

A string that contains the date.

### *Discussion*

The string contains the three address lines separated by the `<br>` tag, followed by the city, state, and postal code in the format of '`<br>city, state, postal`'.

## ccValidate( )

Performs credit card validation.

### *Syntax*

```
static boolean ccValidate( BXStatus status,
   String cardBrand,
   String cardNumber)
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `status` | A `BXStatus` object to receive the resulting status. |
| `cardBrand` | A string that contains the card brand. |
| `cardNumber` | A string that contains the card number. |

### *Returns*

A boolean value that is true if the brand and number are valid; otherwise it is false.

## getMonthNumber( )

Converts a month into its numeric representation.

### *Syntax*

```
static String getMonthNumber(String month);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `month` | A string that contains the month. |

### *Returns*

A string that contains the two-digit representation of the month.

### *Discussion*

You can specify the month in any case. The month number for January through September has a leading zero, e.g. ‘01’.

## getYear( )

Converts a year to a string that contains two digits.

### *Syntax*

```
static String getYear(String year);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `year` | A string that contains a two or four-digit representation of the year. |

### *Returns*

A string that contains the two-digit representation of the year.

### *Discussion*

The last two digits of the year are returned. Error checking is not performed.

## getYearLong( )

Converts a year to a string that contains four digits.

### *Syntax*

```
static String getYearLong(String year);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `year` | A string that contains a two or four-digit representation of the year. |

### *Returns*

A string that contains the four-digit representation of the year.

### *Discussion*

The year must be two digits. Dates before 1990, are returned in the 21st century, e.g. ‘89’ is returned as ‘2089’.

## parseExtension( )

Creates a hashtable with the contents of an extension field.

### *Syntax*

```
Hashtable parseExtension(String ext);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `ext` | A string that contains the extension data. |

### *Returns*

A `Hashtable` object that hashes the keys of the extension and associates them with their respective values in the hashtable.

### *Discussion*

The format of an extension data is key=value and multiple elements are separated by commas.

## stringToAddress( )

Converts a string to an address.

### *Syntax*

```
static IBXAddress stringToAddress(String str);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `string` | An `IBXAddress` object that contains the address. |

### *Returns*

An `IBXAddress` object that contains the address.

### *Discussion*

The string contains the three address lines, the city, state, and postal code separated by semicolons ( ; ).

## stringToDate( )

Converts a string to a date using the default format.

### *Syntax*

```
static Date stringToDate(String date);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `date` | A string that contains the date. |

### *Returns*

A `Date` object that contains the date.

### *Discussion*

The string contains the short form (default) of the date.

## stringToDateLong( )

Converts a string to a date using the long format.

### *Syntax*

```
static Date stringToDateLongFormat(String date);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `date` | A string that contains the date. |

### *Returns*

A `Date` object that contains the date.

### *Discussion*

The string contains the long form of the date.

## stringToVector( )

Converts a string to a vector.

### *Syntax*

```
static Vector stringToVector(String strval);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `strval` | A string that you want to convert. |

### *Returns*

A `Vector` object that contains the string data.

### *Discussion*

Each substring, separated by a semicolon ( ; ) becomes an element in the vector.

## vectorToString( )

Converts a vector to a string.

### *Syntax*

```
static String vectorToString(Vector list);
```

### *Parameters*

The method has the following parameters:

| | |
|---|---|
| `list` | A `Vector` object that contains the items to convert. |

### *Discussion*

Each element of the vector is cast to a string. The result of the cast is appended to the returned string. The elements of the vector are separated by semicolons ( ; ) in the string.

Chapter 4

# Common Templates and Screens

This chapter introduces common templates and shows screens used by several different Servlets, all of which may be customized in BillerXpert B2B Edition.

This chapter contains the following sections:

- Common Templates
  - Services Template
  - Sign On Screen
  - Sign Off Screen
  - Error Screen
  - Sign On Error Screen

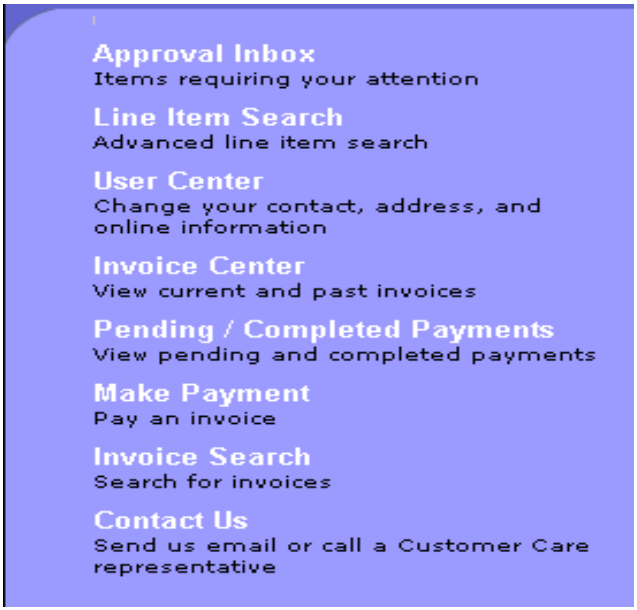
## Common Templates

The following sections describe templates that are invoked from several different Servlets or are included in many other templates. The following templates are commonly used throughout BillerXpert:

| Template | Description |
|---|---|
| `services.jsp` | Main service menu |
| `banner.jsp` | Top menu for Home and Logout links. |

Figure 4-1 shows the common elements of most biller company-related screens in BillerXpert. The services menu appears on the left-hand side of these screens. The Banner menu appears on the top side of these screens.

**Figure 4-1** Common templates




# Services Template

The `services.jsp` template is included by most templates that display an entire screen. This template displays hierarchal references and associated descriptive text and is responsible for invoking a Servlet for each menu item. The template is dynamically generated based on the Access Control List (ACL), which allows certain privileges to a group of users.

# Sign On Screen

### *Template*

`index.html`

### *Servlet*

`BXSignOnServlet, BXSignUpServlet`

The Sign On screen appears when the user enters the URL for the BillerXpert application. Figure 4-2 shows the Sign On screen.

**Figure 4-2** Sign On screen


# Sign Off Screen

### *Template*

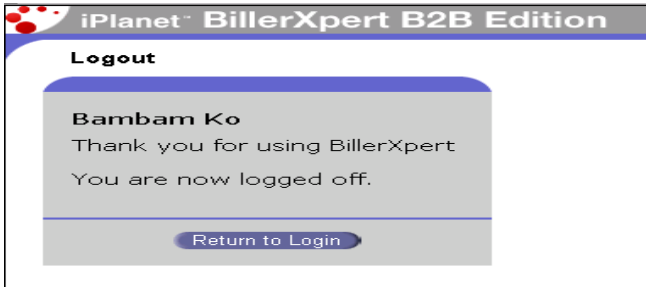
`logout_msg.jsp`

### *Servlet*

`BXSignOutServlet`

The Sign Off screen appears when the user clicks on the log off button on any of the user screens. Figure 4-3 shows the Sign Off screen.

**Figure 4-3** Sign Off screen




# Error Screen

## *Template*

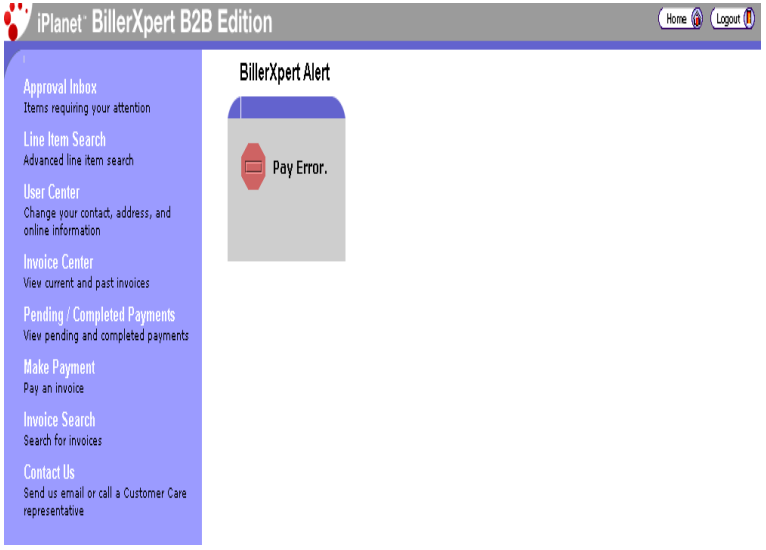
`error_msg.jsp`

## *Servlet*

`BXSignOnServlet`

The `error_msg.jsp` template displays an error message and allows the user to return to the Billing Center screen. Figure 4-4 shows the screen.

**Figure 4-4** Error screen



# Sign On Error Screen

## *Template*

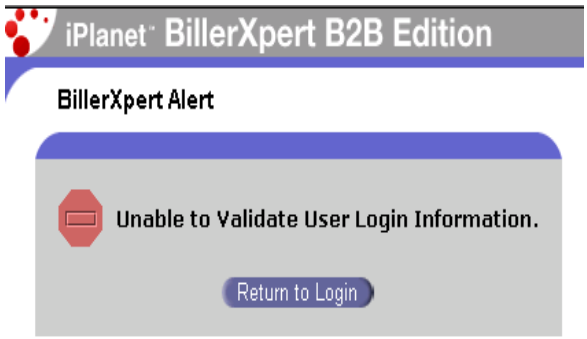
`signon_err_msg.jsp`

## *Servlet*

`BXSignOnServlet`

The `Signon_err_msg.jsp` template displays an error message and allows the user to return to the Home screen. Figure 4-5 shows the screen.

**Figure 4-5** Error screen



Chapter 5

# Invoice Center Screens

This chapter shows the Invoice Center screens that may be customized in BillerXpert B2B Edition.

This chapter contains the following sections:

- Invoice Center Screen
- Pay Balance Screen
- Payments Screen

## Invoice Center Screen

### *Template*

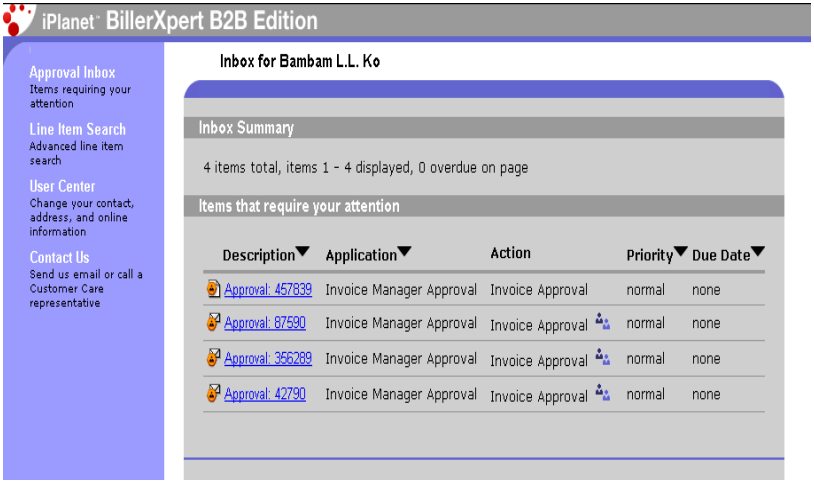
`invoice_center.jsp`

### *Servlet*

`BXBillingCenterServlet`

The Invoice Center screen displays summary information about the user's invoices. Figure 5-1 shows the Invoice Center screen.

**Figure 5-1** Invoice Center screen




# Statement Screen

### *Template*

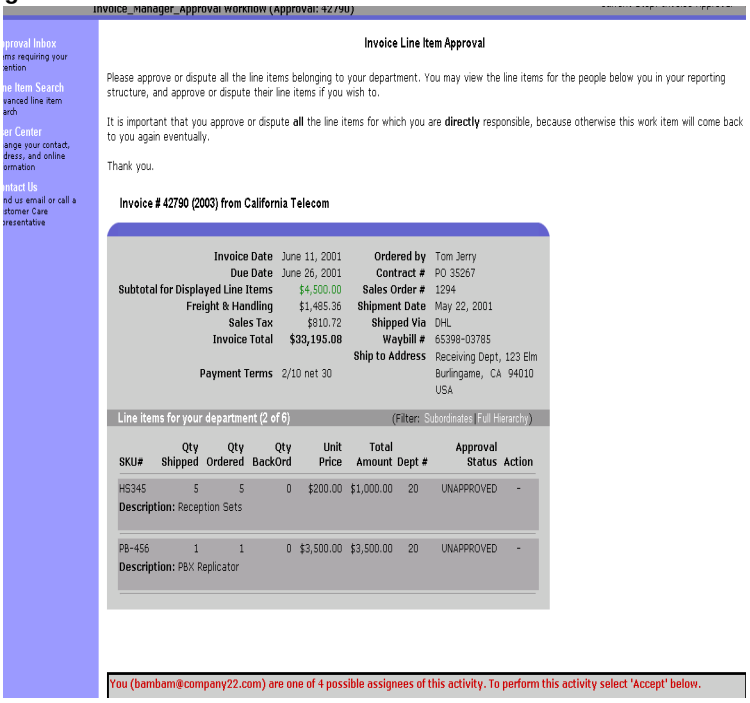
`invoice_view.jsp`

### *Servlet*

`BXBillDocServlet`

A Statement screen appears when the user selects an account summary from the Invoice Center screen or when the user chooses to display current statements. Figure 5-2 shows a statement.

**Figure 5-2** Statement screen




# Pay Balance Screen

### *Template*

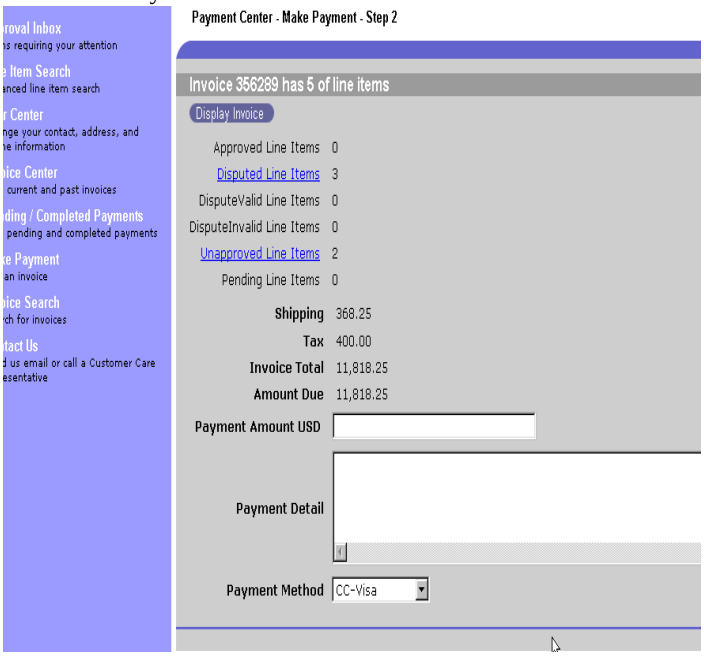
`pay_invoice.jsp`

### *Servlet*

`BXPaymentServlet`

A Pay Balance screen appears when the user selects Pay Balance. This screen lists the accounts that are available for payment. This screen displays the due date, default payment method, and the total amount due for each account number. From this screen, the user can select which accounts to pay. Figure 5-3 shows the Pay Balance screen.

**Figure 5-3** Pay Balance screen




# Payments Screen

### *Template*

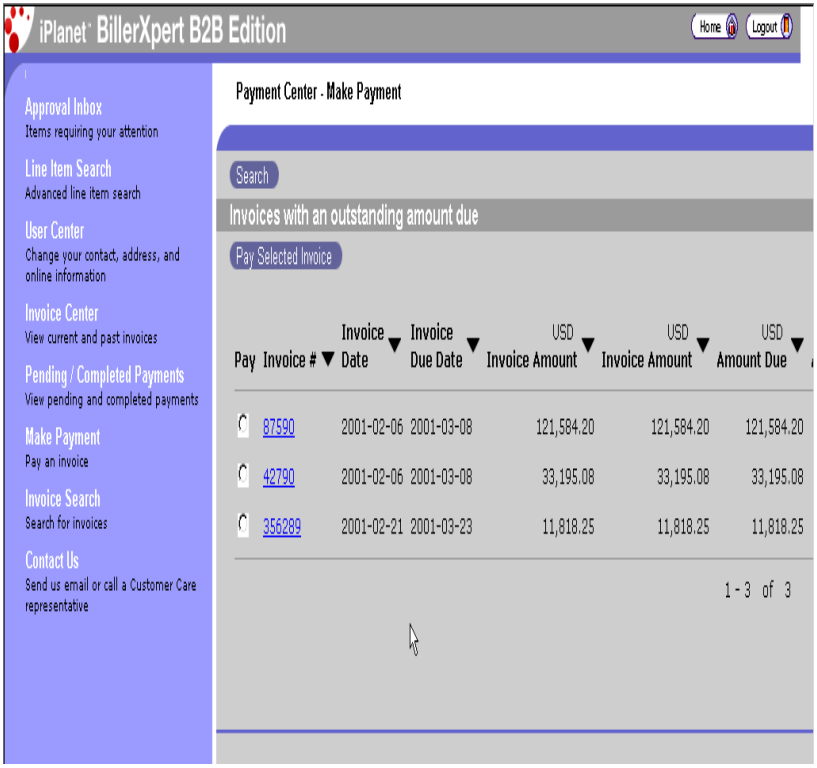
`payments.jsp`

### *Servlet*

`BXPayHistoryServlet`

The Payments screen appears when the user selects Payments. This screen lists the status of all of the user's payments, account information, and payment status. Figure 5-4 shows the Payments screen.

**Figure 5-4** Payments screen



Chapter 6

# Search Screens

This chapter shows the Search screens that may be customized in BillerXpert B2B Edition.

Statement Search screens allow the user to choose which statements to display. The user can make the selection from a screen with the most common parameters or from a screen that allows selection of any parameter. These searches are respectively called simple search and power search.

This chapter contains the following sections:

- Statement Search Screen
- Statement Search Result Screen

## Statement Search Screen

### *Template*

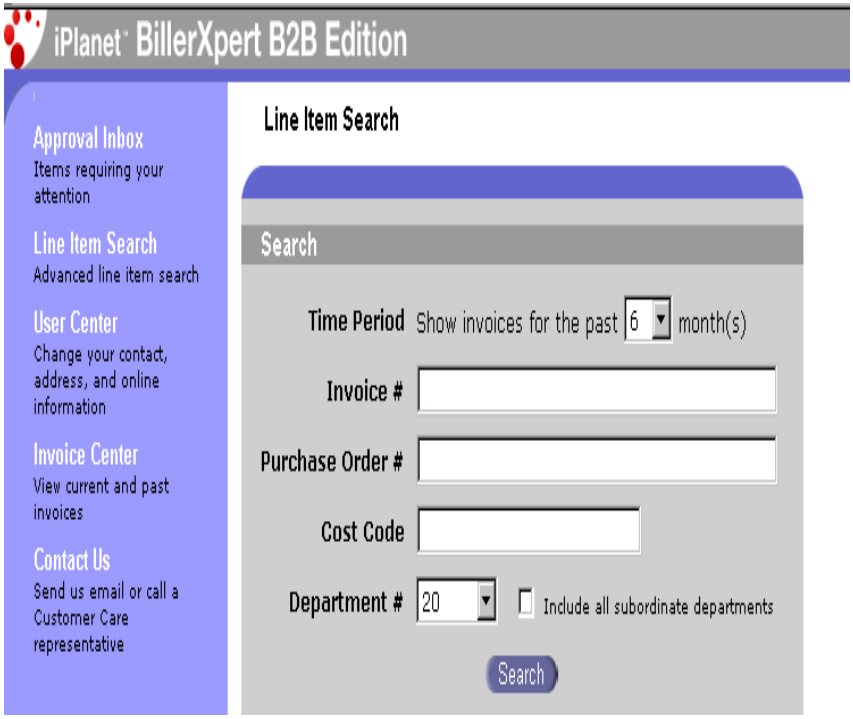
`statement_search.jsp`

### *Servlet*

`BXBillDocSearchServlet`

Figure 6-1 shows the Statement Search screen.

**Figure 6-1** Statement Search screen




# Statement Search Result Screen

### *Template*

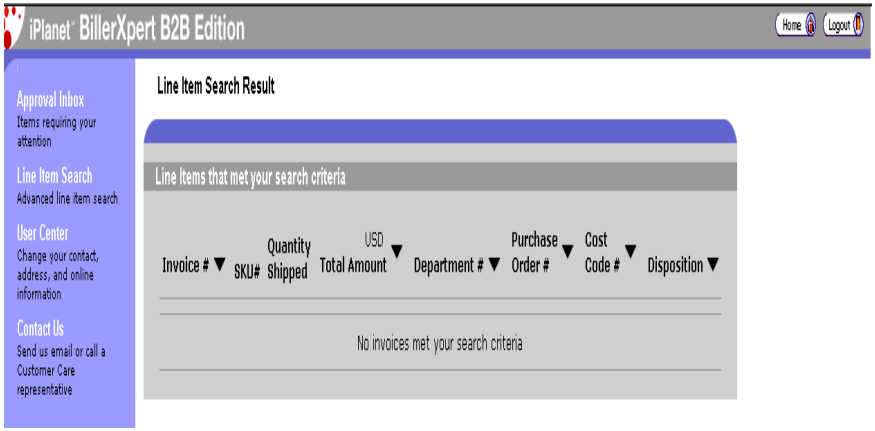
`statement_search-results.jsp`

### *Servlet*

`BXBillDocSearchServlet`

When a search completes, BillerXpert displays a results screen that shows the calls that match the search criteria. You can also sort the results. Figure 6-2 shows the screen.

**Figure 6-2** Statement Search Results screen



Chapter 7

# User Center Screens

This chapter shows the User Center screens that may be customized in BillerXpert B2B Edition.

This chapter contains the following sections:

- User Center Screen
- Name and Address Screen
- Online Name and Password Screen

## User Center Screen

### *Template*

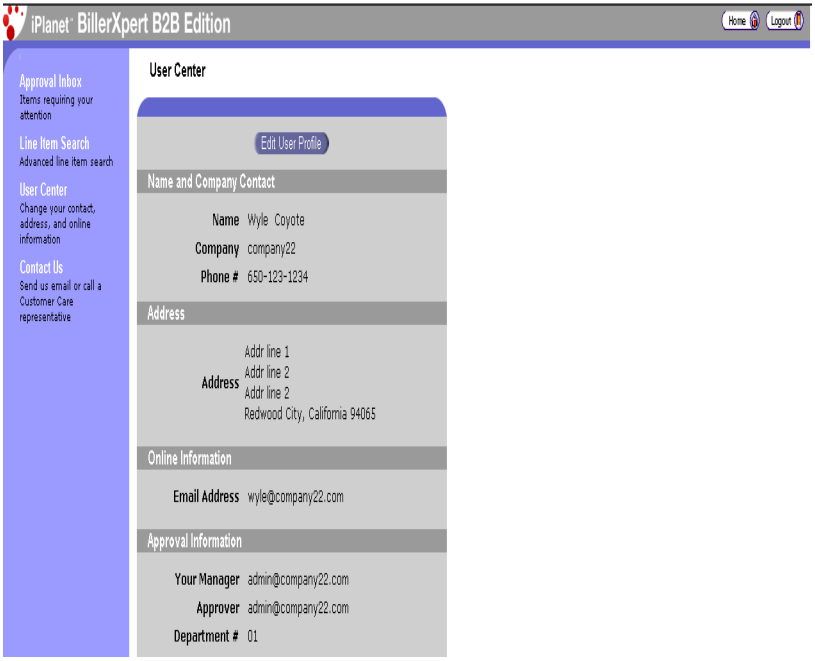
`user_center.jsp`

### *Servlet*

`BXUserCenterServlet`

The user can display the User Center screen to view or change their current information. To change current information, the user can click on `Edit User Profile`. The following sections show the change screens for each of the information types shown in the User Center screen. Figure 7-1 shows the User Center screen.

**Figure 7-1** User Center screen




# Name and Address Screen

### *Template*

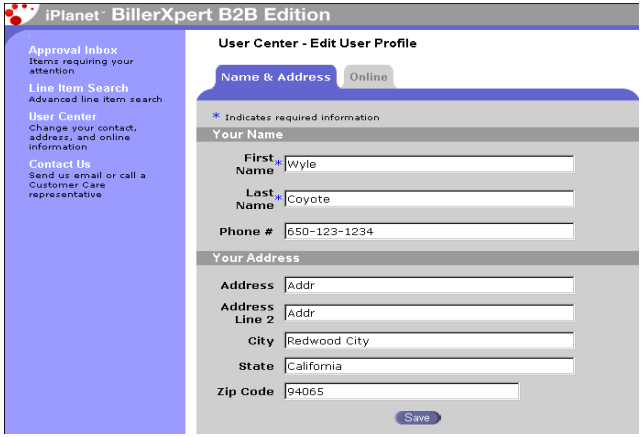
`user_center-editor-address.jsp`

### *Servlet*

`BXUserProfileServlet`

The user can update their name and address from the Edit User profile screen. Figure 7-2 shows the Edit User Profile screen.

**Figure 7-2** Name and Address screen




# Online Name and Password Screen

## *Template*

`user_center-editor-password.jsp`

## *Servlet*

`BXUserProfileServlet`

The user can change their user identification and password from the Online Name and Password screen. Figure 7-3 shows the Online Name and Password screen.

**Figure 7-3** Online Name and Password screen

Chapter 8

# Process Manager

This chapter discusses the Process Manager component of BillerXpert B2B Edition.

This chapter presents the following topics:

- Introduction
- iPlanet BillerXpert B2B Edition Application Model
- Processes
- Customizing Processes
- BillerXpert Data Fields In Process Manager
- Modifying the Display of Invoice Fields
- Creating New Invoice Data Fields
- Context Data Fields

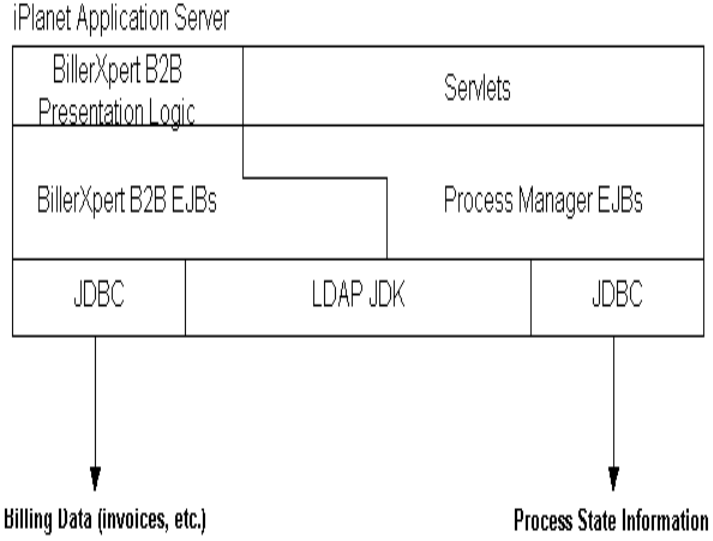
## Introduction

iPlanet Process Manager is a web based workflow system used to manage the routing of workflow through a predefined process. BillerXpert leverages Process Manager for invoice approval routing, dispute handling, enrollment processing, and invoice distribution. Process Manager also gives the ability to alter any of these processes to better model the requirements of an individual business.

# iPlanet BillerXpert B2B Edition Application Model

The BillerXpert B2B system consists of multiple systems working together. The image depicted in Figure 8-1 shows the conceptual view of how these systems fit together.

**Figure 8-1** BillerXpert Conceptual Overview




# Processes

BillerXpert B2B Edition is shipped with several predefined sample processes that demonstrate some common tasks that can be preformed in the system. Each of these processes can be modified, or new ones created, to model the businesses' specific requirements. The sample processes are described in the following sections.

## Enrollment Process

The Enrollment process handles the process of buying company's end user sign up and registration. It is also used for existing users to change their password. To start the enrollment process:

- Go to the login screen for the company that you want to register with, for example: `http://yourserver:yourport/telco`
- Press the “Register” button
- The associated form is displayed, as seen in Figure 8-2 below.
- Complete all necessary information as described in the balance of this section and press “Submit”.

**Figure 8-2** The Registration Form is displayed

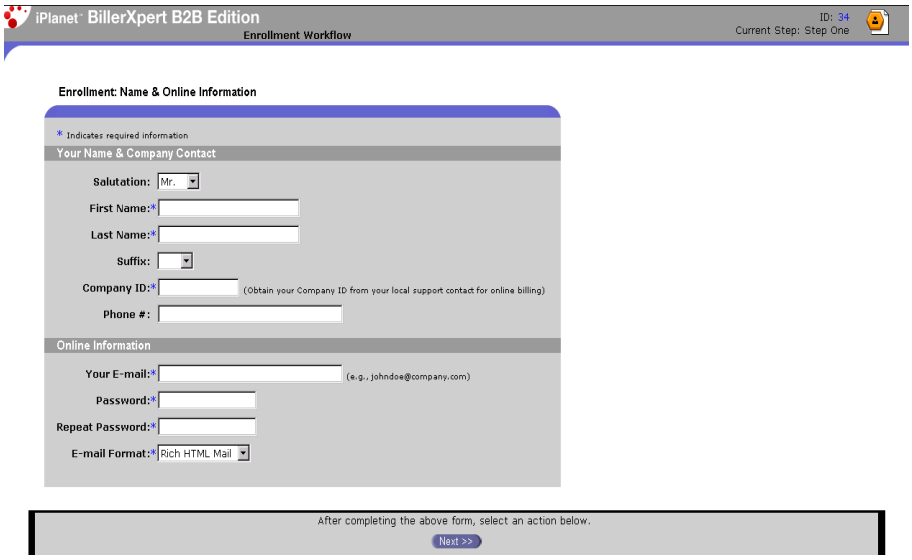
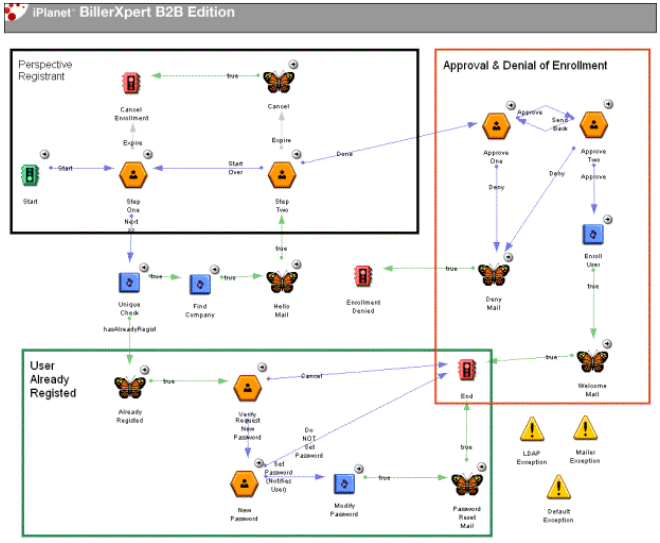
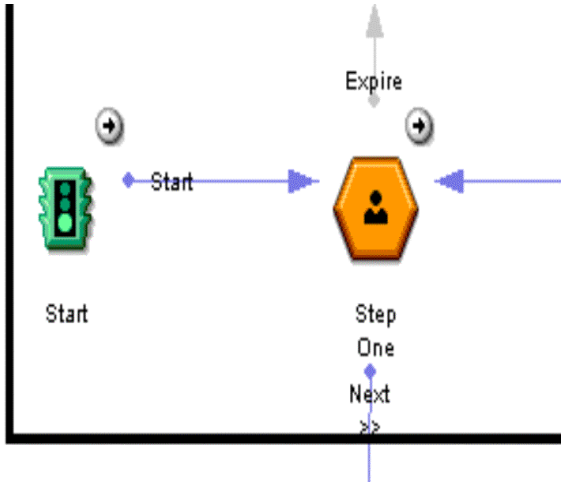


**Figure 8-3** Enrollment Process




When a user new to the system clicks the “Enroll Now” button in the BillerXpert B2B Edition login screen, the user is entered into this process at step one, as seen in Figure 8-4. The steps involved in in the enrollment process are as follows:

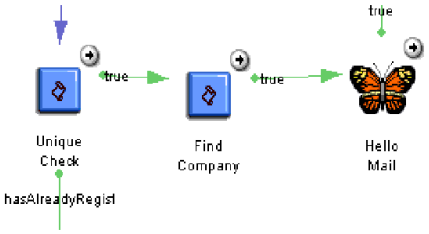
**Figure 8-4** Gather Initial User Data




### *Step One*

The user enters information about themself in the form. This information includes name, email address, desired password, and a company ID. A person in their company gives the company ID to the user before they sign up.

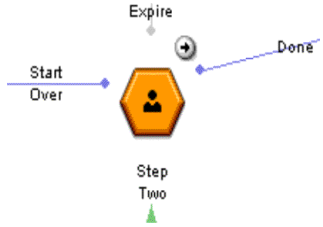
**Figure 8-5** Check for Duplicate User and Send Email




Once the user clicks the “Next>>” button, the system checks the user’s email address to see if they have signed up before. If they already have an account on the system, the system follows a path that allows the user to change their password if they want to.

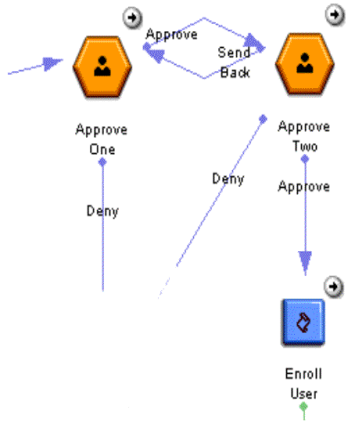
If the user is not a registered user, the system finds the company that the user is registering with (by using the company ID), and then sends a confirmation email to the user (to ensure the correct email was entered.) This is shown in Figure 8-6.

**Figure 8-6** User Confirmation




The email that the user receives includes an active link to a page that confirms all the personal information that is required to create an account. This step is shown in Figure 8-7.

**Figure 8-7** Confirmation and Approval




Once the user confirms the information, they click the “Done” button and the process takes the account creation request to an Administrator in the user’s company. The Administrator uses company resources to assign the user’s manager.

The process then moves onto the user's manager, where the task is to specify the user's department number and approver.

Once approved by the manager, the user is enrolled into the system and receives a confirmation email.

# Approval Process Assumptions

The BillerXpert B2B approval processes operate under the following assumptions.

1. Every Invoice can have Multiple Line Items.

   Each line item may have a unique department number assigned to it. Process Manager startss another process to enter the department number if the department number is either not entered or not recognized.

2. Each Department Must Have an Owner.

   The owner must be known in the People tree in the Corporate directory. If not, an exception occurs. The owner must have a Manager, and, that manager must be known in the People tree.

3. When an Exception occurs, the next step in a given process is assigned to an Adminsitrator.

The invoice approval cycle for Process Manager is driven by four invoice status codes, which are explained in the following section.

# Process Manager Status Codes

| Invoice Status Code | Definition |
|---|---|
| UNAPPROVED | The line item has never been reviewed. |
| PENDING | The line item has been reviewed and has received initial approval, but not final approval. |
| APPROVED | The line item has been reviewed and received final approved. |
| DISPUTED | The line item has been disputed. |

Each of these codes apply to a single line item, though the status will effect the flow of all line items on a multi-line item invoice. More simply stated, a single line item can hold up an entire invoice.

## Approval and Dispute Processes

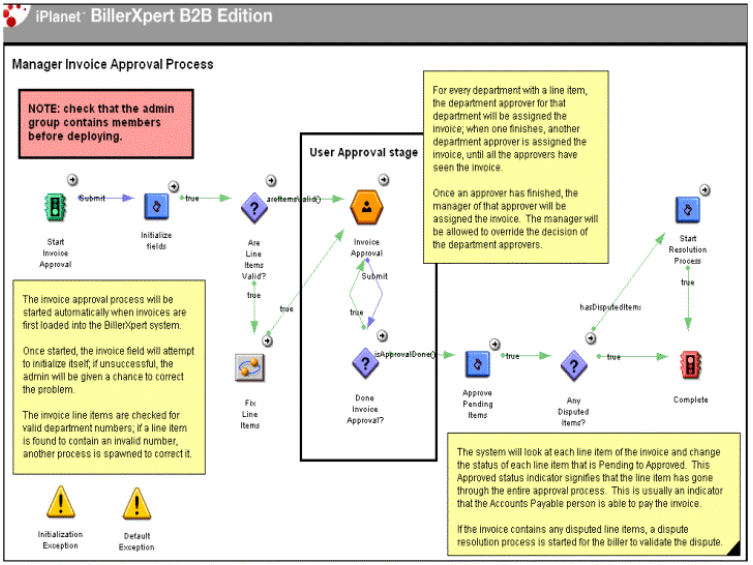
iPlanet BillerXpert B2B Edition ships with four predefined approval processes. These processes are designed to show companies the flexibility and capabilities of the BillerXpert B2B in modeling their specific business requirements. Biller or Buyer companies can create their own processes, or, modify existing processes to meet their own needs. By default, BillerXpert uses the Manager Approval Process but you can change this to suit your needs. The different approval processes are covered in the following section.

## Manager Approval Process

The manager approval process assigns the invoice initially to owners of the departments listed on the invoice.

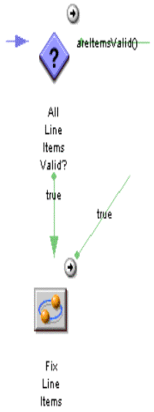
After the department owners approve the line items on the invoice, the invoice is assigned for approval by the department owners' managers (as defined in LDAP). This continues until the process determines that there are no additional approvals required. This process is shown in Figure 8-8.

**Figure 8-8** Manager Approval Process




Looking through the process in detail, the initial step in the process initializes some common variables that are required throughout the process. After the variables are initialized, the process then examines the invoice to determine if there are department numbers assigned to each line item, as seen in Figure 8-9.

**Figure 8-9** Department Check




If there are missing department numbers on the line items of the invoice, the invoice enters another process where the invoice is presented to the company administrator to assign the correct department numbers.

Once the department numbers are assigned to each line item, the invoice is assigned to the department owner, as shown in figure 8-10. The assignment script (which specifies who is required to perform the activity) for the invoice approval activity is recalculated each time someone approves the invoice, based on who owns the Unapproved line items. The department approver cycle continues until there are no unapproved line items remaining.

As an approver displays an invoice, they can customize the view according to their needs. By pressing the “Filter” button, they may choose “Subordinates” to view their invoices, as well as those assigned to their direct subordinates. To view all line items belonging to anyone below them in the department hierarchy, select “Full Hierarchy”.

### *Note on SuperApprovers*

The Invoice Manager Approval process and the Invoice Approver Approval process both contain an application group (that is, a group defined within the process) called *SuperApprovers*. All members of this group always get to see all line items in the invoice. SuperApprovers can step in and perform invoice review tasks on behalf of anybody else.

For example, if a manager leaves the company while they have approval tasks in their inbox, a SuperApprover can take over these tasks. For details on how to re-assign work items, please see the Process Manager documentation online at `http://docs.iplanet.com/docs/manuals`.

**NOTE** When the user views their Inbox, all invoices containing line items assigned to them are displayed. If there are multiple line items with multiple department numbers associated with them, a user may press “submit” on a given invoice line item assigned to them without Approving or Disapproving the item. This action serves as a ‘refresh” and the invoice is put back into the queue for others to view and disposition. On a multiple line item invoice with multiple department numbers, only one user can accept their given line items at a time.

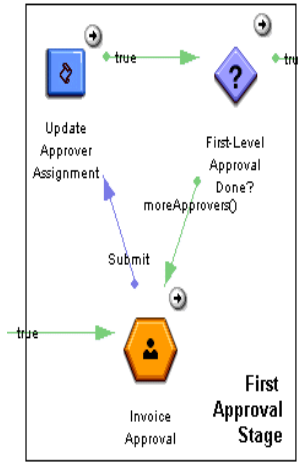
The typical scenario is that a user will:

1. View the Invoice
2. Approve or Dispute each line item

   If you Approve a line item, the status changes to PENDING

   If you Dispute a line item, the status changes to DISPUTED.

**Figure 8-10** User Approval




Once the initial approver (or approvers) approves the invoice, it requires the approval of their manager. The process has an automated step called “Prepare for Manager Approval” that recomputes the new approvers to be the managers of the previous approvers. A manager is allowed to override the decision of their subordinates regarding invoice line item approval or dispute. As in the previous activity, the assignment script for this activity is *recalculated* each time someone approves the invoice.

*An explanation of How Manager Assignment Works is given in the next subsection*. The manager approval cycle continues until the higher level managers approve or dispute all relevant line items in the invoice. This part of the process can be seen in Figure 8-11.

## How Manager Assignment Works

1. When an invoice is preparing for management approval, the process first creates a list of managers. At the same time, the system creates a list of line items that need to be looked at by a manager.

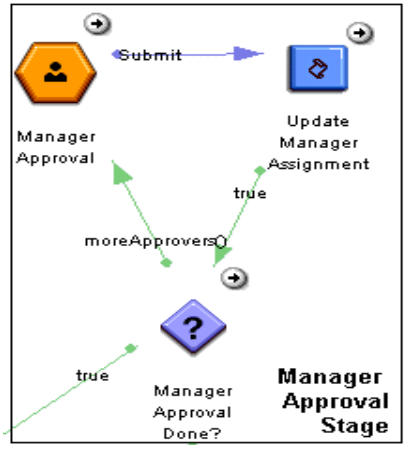
2. When the manager submits an activity, the system removes the manager from the list of managers, and also removes the line items that were changed from the list of line items. This occurs even if only some of line items are approved.

3. The situation might occur where a manager does not approve or dispute all of their line items. In this case, the manager is still removed frm the manager list as usual, with the consequence that the manager list becomes empty while there are still remaining line items. In this case, the process recalculates the manager list based on the remaining line items.

4. When there are no line items remaining, the Manager Approval stage is complete, regardless of whether any managers remain in the list of managers. This handles the case where someone else, such as a SuperApprover, performs one of the manager's approval tasks.

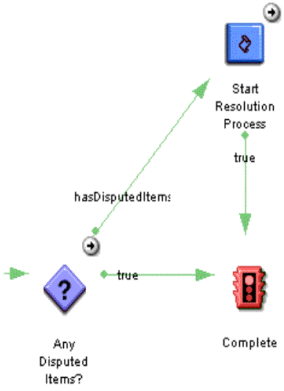
**NOTE** Even if an item has been delegated to another manager, the original manager still appears in the List of Managers.

**Figure 8-11** Manager Approval




Once the approvals are complete, the process determines if any of the line items are marked for dispute. This step can be seen in Figure 8-12. If there are items that are in dispute, the process invokes a subprocess that sends the disputed items to the billing company for resolution. The approved items are marked approved for payment by accounts payable.

**Figure 8-12** Dispute Check

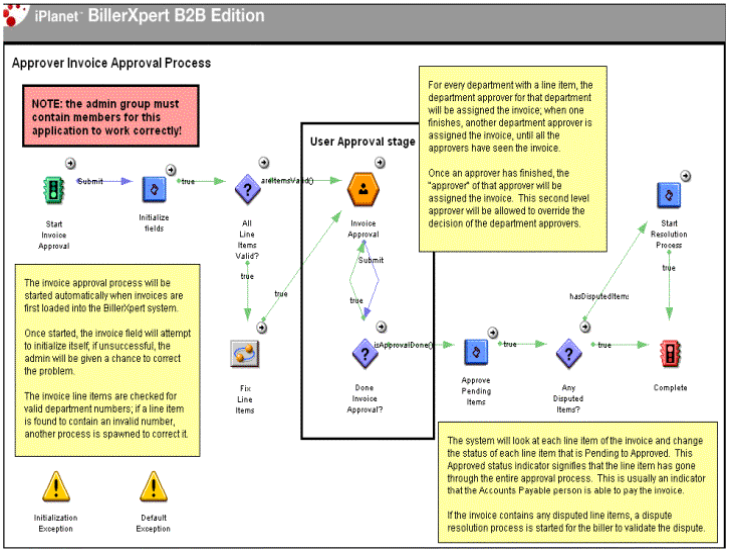



# Approver Process

The Approver Process has three levels of approval. All line items are approved by the department owner and their manager. Any line item that exceeds a pre-specified limit requires approval by a third person, defined as the department owner: approver (nsapprover in LDAP). The process invokes `Invoice_Manager_Approval` to handle the first two rounds of approval. Then it tests invoice line items; if any are greater than the defined limit, then it goes through the third round of approval.

This process is depicted in Figure 8-13: Approver Process.

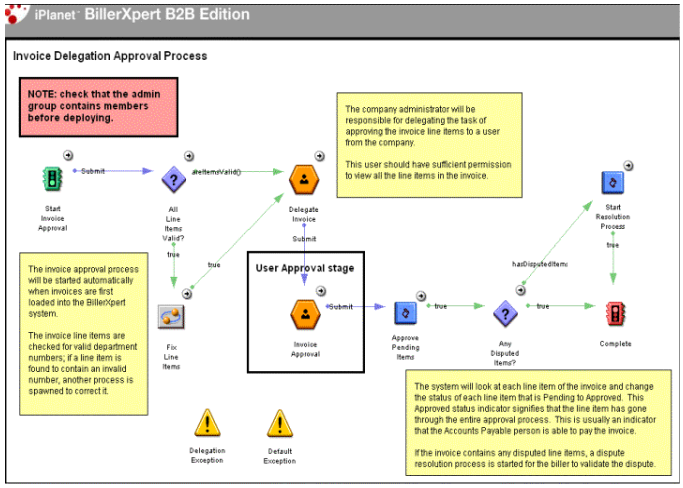
**Figure 8-13** Approver Process




# Delegation Approval Process

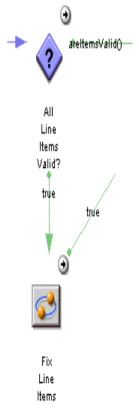
In the Delegation process, initially the invoice is assigned to a group of people within the buying company. This group of people is responsible for reviewing the invoice and then assigning it to the correct person within the company for the final approval. This process most closely follows the process that is in use at most companies today: the invoices come in to a centralized location and accounts payable personnel forward copies of the invoices to the correct people for their approval signature before payment is executed as seen in figure Figure 8-14.

**Figure 8-14** Delegation Approval Process



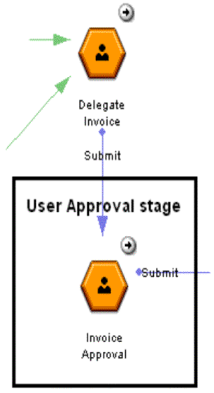

When the process starts, it checks to see if all the line items on the invoice have an assigned department number, and if not, assigns it to someone to set department numbers. This can be seen in Figure 8-15.

**Figure 8-15** Department Check




The invoice is then assigned to the company administrator. The administrator examines the invoice to determine who within the company should approve the invoice. This person is then assigned the invoice for approval. This step is seen in Figure 8-16.

**Figure 8-16** Approval Delegation

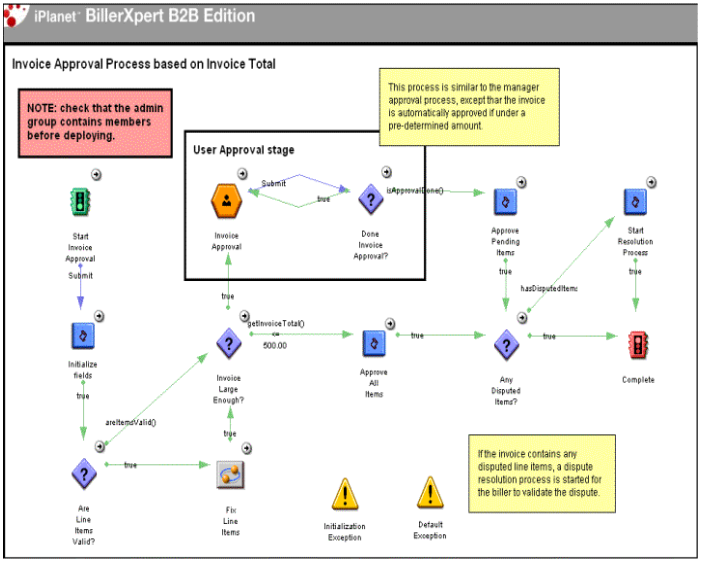



The process then marks the line items that have been approved valid for payment and forwards the disputed line items to the billing company for resolution.

# Approval Amount Process

The review and approval of invoices is an expensive task within companies, some invoices may not be worth the time required for people within the company to examine and approve. The amount approval process, as seen in Figure 8-17, examines the invoice and automatically approves invoices with an amount under 500 dollars. Although this amount can be customized, the amount of 500 dollars will be used as an example in this section.

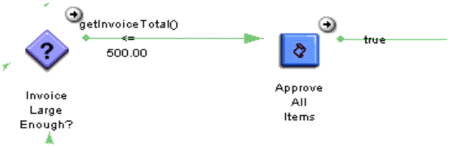
**Figure 8-17** Amount Approval Process




The process checks to see if the total amount of the invoice is less than 500 dollars. If the invoice total is less than 500 dollars, the process marks all of the line items approved and valid for accounts payable to pay.

To customize this amount, open the process in Process Manager and change the value in the following transition:

**Figure 8-18** Automatic Approval




If the invoice amount is over 500 dollars, the Manager Approval process is invoked as a sub process to continue the approval process for the invoice, as shown in Figure 8-18. Once the approvals are complete, the invoice line items that are not in dispute are marked as valid for accounts payable to pay, and the items marked in dispute are forwarded to the billing company for dispute resolution.

# Invoice Loading Process

If you create your own process, it must have an entry point called `Start_Approval` that has `docid` and `bcaid` data fields. It must have a “Start” transition (action) `nsapprovalprocessname` in LDAP for a company. This indicates the name of the approval process to start. If no value, then the process name in the data field is `defaultprocess`.

During the loading of invoices, the invoices need to be disseminated to the appropriate buying company for approval. Once the Loader program has completed taking the raw XML invoice data and bringing it into the system, the Loader program initiates the Invoice Loader Process. This process examines the new invoice data and initiates the correct process for the invoice at the appropriate company.

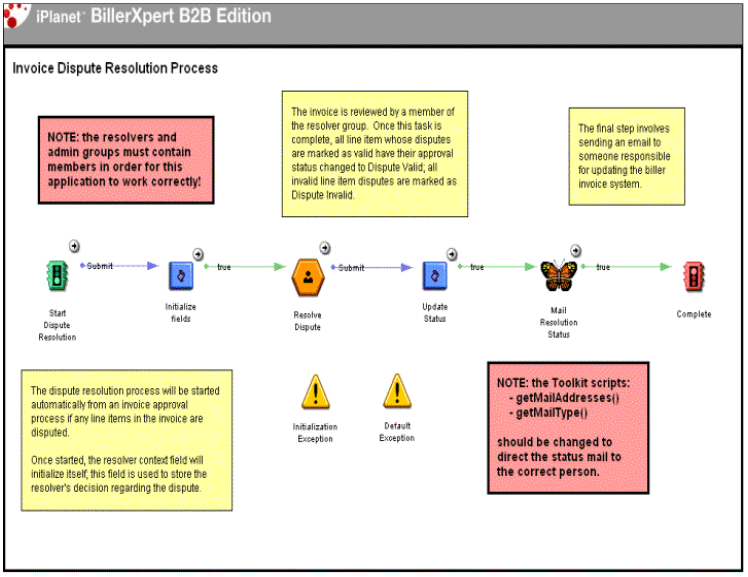
**NOTE** Before loading invoices, be sure to install the BXLoaderDone process. When the Biller Admin loads invoices, the BillerXpert B2B automatically starts a BXLoaderDone process, which in turn starts the applicable approval processes.

he BillerXpert B2B requires the Process Manager loading process to be named BXLoaderDone. If you modify the BXLoaderDone process, for example, to change the default approval process, be sure to redeploy it with the exact same name.

# TDispute Handling

As buying companies dispute specific items on an invoice, this process, shown in Figure 8-19, is used by the Billing Company to handle the dispute.

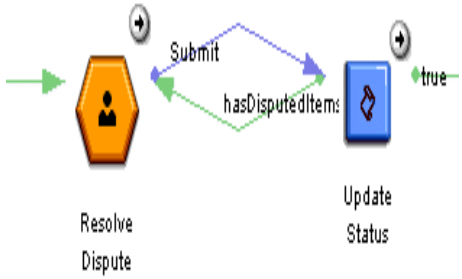
**Figure 8-19** Dispute Resolution Process




Invoices that contain disputed items are processed through this process. Invoices are assigned to the group called Resolvers in the Billing company. Any member of this group can examine the invoice and determine if the disputes are valid. The resolver can select specific disputed items and mark them as either valid or invalid. This step can be seen in Figure 8-20.

It is likely that the biller company will want to nominate particular people to handle the resolution task. To do this, add members to the `resolvers` group in the Process Manager Builder. In the Application Tree View, open the Groups and Roles node, double click on `resolvers`, and use the Inspector window to add the appropriate people to the resolvers group. Redeploy the application to have the changes take effect.

**Figure 8-20** Resolver Examines Disputes




Once the resolver finishes, the system sends an email to the resolvers, detailing the status of the disputes to the identified people. This is shown in Figure 8-21.

**Figure 8-21** Mail Resolution

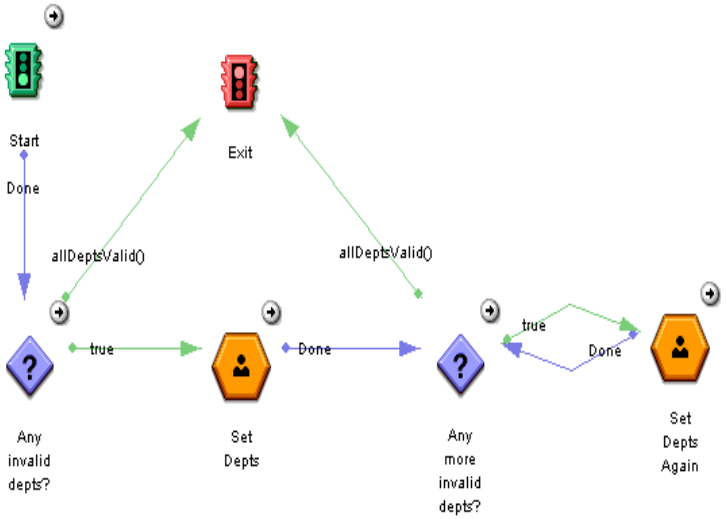
# Set Valid Departments Process

The set valid departments process has steps for entering valid department numbers for line items in an invoice.

Each of the approval processes checks if the invoice they are managing has any line items with invalid departments. If any department numbers are invalid, the approval process starts a new process for correcting the department numbers.

The set valid departments process is shown in Figure 8-22.

**Figure 8-22** Set Valid Departments Process




As with all the approval processes, the Start entry point is invoked automatically when necessary. The `Any invalid Depts?` step is an automatic step that checks if the invoice has any line items that do not have valid departments. If all departments are valid, the process terminates immediately at the Exit exit point.

If any departments are invalid, the next step is a user activity that displays a form. The form shows the line items that do not have valid department numbers and asks the user to fix the department numbers.

When the user submits the form, the process checks again if all department numbers are valid. If they are, the process exits. If they are not, the process goes to another user activity, which again displays a form asking the user to fix the department numbers. The process cannot finish until all department numbers are valid.

# Customizing Processes

One of the benefits of building the processes for handling approvals, disputes, enrollment, and loading through iPlanet Process Manager is that the business logic can be easily customized to match a company's specific business requirements. Modifying the existing processes to more closely reflect the situation within a particular company does this.

## Customizing the Sample Processes

This section discusses the following modifications that you can make to the pre-defined processes that ship with the BillerXpert B2B edition of Process Manager:

- Specifying the Approval Process to Use
- Specifying Resolvers
- Modifying Email Messages
- Changing the Amount for Automatic Approval
- A Note About Creating New Approval Processes

Use the Process Builder to modify these applications. For information about creating and modifying processes in the Process Builder, see the *Process Manager Builder's Guide* at `http://docs.iplanet.com/docs/manuals`.

### Specifying the Approval Process to Use

The BXLoaderDone process determines which approval process to initiate for each BillerXpert invoice that is loaded.

To specify the default approval process for a company, set the value of the nsapprovalprocess name attribute for the company in LDAP. To specify the default approval process for all companies in case any of them don't have a value for nsapprovalprocessname, set the value of the defaultProcess data field in the process.

If you want to modify or completely change the way that the loader process picks the approval process to use, then modify the `resolveApprovalProcessName()` script in the BXLoaderDone application.

By default, the `resolveApprovalProcessName()` script does the following:

- checks if the default approval process for the current company has already been defined, if so, uses returns it and exits
- if not, calls `getDefaultApprovalProcessForCompany` ( companyID ), which in turn
    - finds and returns the value of the `nsapprovalprocessname` attribute of the company in LDAP
    - if it can't find the value of this attribute, it returns the value of the defaultProcess data field defined in the process
- `resolveApprovalProcessName()` stores the default process name in a variable so it is available when needed again then returns the process name

## Specifying Resolvers

If any approvers dispute one or more line items of an invoice during an approval process, the system automatically starts a Dispute Resolution process. By default, this process assigns the resolution task by executing the `assignToResolvers()` script. This script assigns the resolution task to members of the resolvers group. If there are no such members, the task is assigned to the billing company administrators, and if there are none of those, the task is assigned to the Biller Company's Administrator.

It is likely that the biller company will want to nominate particular people to handle the resolution task. To do this, add members to the `resolvers` group in the Process Manager Builder. In the Application Tree View, open the Groups and Roles node, double click on `resolvers`, and use the Inspector window to add the appropriate people to the resolvers group. Redeploy the application to have the changes take effect.

## Modifying Email Messages

Both the Dispute Resolution process and Enrollment process send email messages at various steps in the process. It is unlikely that you would need to change the calculation of the email address for the Enrollment process since it sends emails messages to the user who is attempting to enroll, using the email address supplied by the user in the first step of the process. However you may want to change the calculation of the email addresses in the Dispute Resolution proces

The final step of the Dispute Resolution process sends an email to various people to tell them to update the BillerXpert invoice system. By default, the process sends emails to the resolvers, to the email address registered with the cluster and to `root@localhost`. However, it is likely that the email needs to go to a particular person or set of people who are responsible for maintaining the invoices in the BillerXpert database. To specify who receives the email, modify the value of the `getMailAddresses()` script to return a JavaScript string or an array of strings containing the email addresses of the people who need to receive the update. If the value is a string, multiple addresses must be separated by commas, for example:

```
function getMailAddresses()
{
    var email = "admin1@yourcompany.com, admin2@yourcompany.com";
    return email;
}
```

To change the body of the email messages that get sent either by the Dispute Resolution or Enrollment processes, modify the modify the templates in the templates directory in the application's directory. For information about using templates, see the section "What Templates Can Do". Although that section was written for the discussion of templates for displaying invoice fields, the general rules about what templates can do applies to all HTML templates used in Process Manager applications.

## Enrollment

The Enrollment process times out and cancels the process if the enrollee does not act quickly enough after initiating registration. You can customize the following timeout periods:

- amount of time between when a user presses the "Register" button to start the registration and when they complete the initial registration form

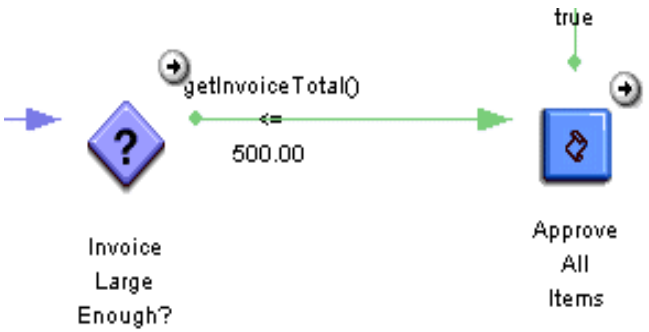
- amount of time between when the email containing the link for continuing the enrollment process is sent to the enrollee and when they submit the form that is displayed by clicking the link

To change the time out periods, open the process in the Builder, double click on the appropriate activity (Step One or Step Two) to open the properties editor, and change the value of the expiration setter script.

### Changing the Amount for Automatic Approval

The AmountApproval process automatically approves invoices whose total amount is below a certain dollar amount, as shown in Figure 8-23. To change this minimum amount, open the process in the Builder and change the value of the following transition:

**Figure 8-23** Automatic Approval




### A Note About Creating New Approval Processes

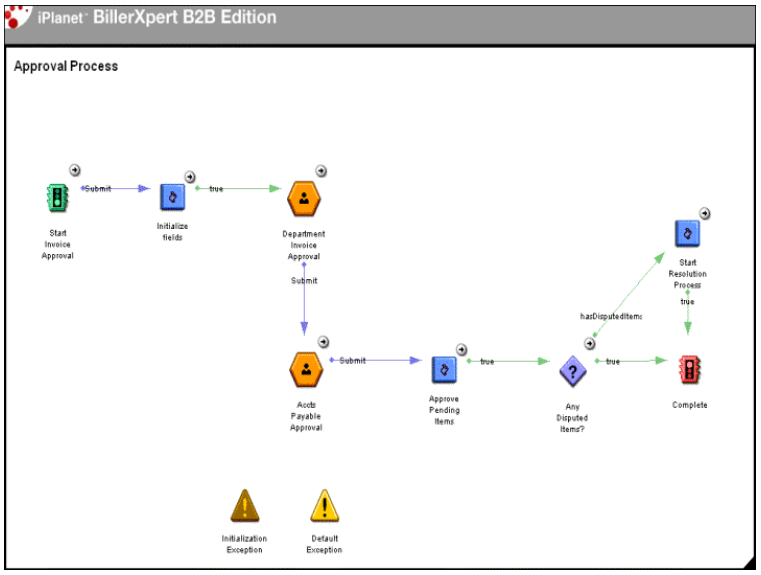
All approval processes must have an entry point called "Start Approval Process" that must have a transition (action) called "Start". If you create your own approval process, be sure to create an entry point that satisfies these conditions otherwise BXLoaderDone will not be able to start it. The approval processes should also have docid and bcaid data fields for the document id number and the customer account number, which are dynamically figured out for each invoice by BXLoaderDone.

## Further Customization

You may need to substantially modify the existing processes to better model your company’s approval requirements. Choose the existing process that most closely resembles your company’s process.

The first step in modifying the processes is to carefully determine the process that the company wishes to follow. Say, for example, a company would like to have a process for invoice approvals where the invoice comes into a department and is assigned to the department's approver (as determined in the organization structure). After the department approver approves it, the invoice needs to go to a centralized invoice approval desk with accounts payable before it is finally approved for payment. The process map for this invoice flow would look something like the one shown in Figure 8-24.

**Figure 8-24** Custom Approval Process




This process is built using the Process Builder and uses the Delegator Invoice Approval as a template to build this customized process. The following steps would need to be part of the process:

1. Start Invoice Approval

   This entry point step is to initiate the process. Typically an entry point includes data that is used throughout the process and typically has some interaction with the end user, but since BillerXpert B2B initiates the process itself, no user interaction is required.

2. Initialize Fields

   This automated activity within the process initializes some of the fields that the process needs throughout the rest of the process. These variables include the invoice number, the billing company identification, etc.

3. Department Invoice Approval

   The approvers within this process are defined by a group of users within LDAP. This LDAP group of approvers is the assignees (the people that are assigned the work to do within the process) for this setup. Any person within this group can accept an invoice to review and approve. Only one person is needed to approve an invoice and once he has approved it, the invoice moves to the next step in the process.

4. Accounts Payable Approval

   At this step, a user in the centralized accounts payable approval desk reviews the invoice and either gives it a final approval or disputes it.

5. Approve Pending Items

   Invoice line items are not immediately marked as approved once a single person in the process approves them. Line items that are approved by people in the process are put into a temporary state called 'pending' in which they are pending final approval. Since the process doesn't know when the final approval on the line is, a simple script is run to turn all 'pending' line items to 'approved.' It is only at the 'approved' state, that accounts payable knows that the line item has made it through all of the approvals within the approval process. This step in the process comes after all of the approval steps.

6. Any Disputed Items?

   Disputed line items need to be sent to the Billing Company to be handled through their dispute resolution process. This step checks all the line items on the invoice for any disputed items. If there are disputed items, the process progresses to the Start Resolution Step. If there are no disputed items, the process ends.

7. Start Resolution Process

   The start resolution process simply triggers the resolution process within the Billing Company for this particular invoice for resolution.

8. Complete

   The process is completed for this invoice.

### Removal of Checking Departments

The reason why this process removed the checking for valid departments step is that this check is only required if the process uses department codes to route individual line items of an invoice to the designated approvers for the particular departments. Since this process has a well-defined approval path (i.e., all invoices going to a set group of people), department codes are not required at the line item level of the invoice, and therefore, we do not need to check to see if they exist at the beginning of this process.

## How Process Manager Handles Multiple Billers

Process Manager uses clusters (which are different than the clusters used by the Application Server). A cluster contains a configuration directory, a corporate user directory, a relational database, one or more iPlanet Web Servers and one or more iPlanet Application Servers.

BillerXpert B2B edition creates a new Process Manager cluster for each biller. Thus each biller has a cluster and each cluster has a biller. Process Manager applications are bootstrapped on a per-biller basis. Thus if you have four billers, you will have four clusters that each contain all the bootstrapped applications.

Each biller has an attribute in LDAP called `nsclusterid` that specifies the cluster name for the biller.

BillerXpert B2B loads invoices on a per-biller basis. When invoice loading is complete, the loader sends an event to start the `BXLoaderDone` process on the cluster belonging to the appropriate biller. The `BXLoaderDone` process in turn starts approval processes on the same cluster as itself for all new invoices that were loaded for the appropriate biller. Thus each approval process operates in the context of the biller who sent the invoice being approved.

Any JavaScript script in any Process Manager process can get the LDAP entry for the biller for the current cluster by calling the global function `__bx_findBiller()`. To get the biller name, call `get("o")` on the results of `__bx_findBiller()`:

- var biller = `__ bx_findBiller();`
- var billerName = `biller.get("o")`

Other useful global functions available to Process Manager JavaScript scripts and HTML templates include:

- `__ bx_findBiller()`

  gets the LDAP entry for the current biller

- `__bx_getBillerLoginURL ( )`

  returns the URL for the current  biller's login page

- `__bx_getLDAPConfig4BillerCluster ()`

  gets the LDAP configuration for the current cluster

- `__bx_getLDAPConfig4BSP ()`

  gets the LDAP configuration for the BSP for the BillerXpert B2B installation.

# BillerXpert Data Fields In Process Manager

The remainder of this chapter discusses the data fields that are used by the approval processes to display invoices.

The Process Manager Builder that ships with BillerXpert B2B edition includes several data fields for displaying invoices in various ways, as follows:

- The BillerXpert Invoice Field displays the invoice line items that the user is responsible for approving or disputing.

  The approval processes (manager approval, approver approval, amount approval, and delegator approval) use this data field.

- The BillerXpert Departments Invoice Field displays the department numbers for line items as a selectable menu.

  This data field is used by the process for setting department numbers.

- The BillerXpert Invoice Resolution Field displays line items under dispute.

  This data field is used by the invoice dispute resolution process.

Process Manager Builder also provides two additional BillerXpert-specific data fields:

- BillerXpert Delegation Field -- allows the user to pick delegatees.
- Context Data Field -- stores multiple data values. For more information, see the section Creating New Invoice Data Fields.

You insert invoice fields, delegation fields and context data fields into a Process Manager process just as you would insert any other fields. These data fields are available through the `Insert>Data Field` menu option. For details of using data fields in a process, see the *Process Manager Builder's Guide* at `http://docs.iplanet.com/docs/manuals`

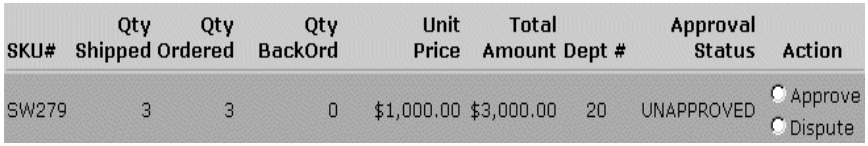
If none of the existing invoice data fields is exactly what you want, you can create your own custom data field. For details, see the section “Creating New Invoice Data Fields.”

The following sub-sections discuss each of the data fields in more detail.

# BillerXpert Invoice Field

The `InvoiceField` data field is the base invoice field class which shows the line items in the invoice that the end user is responsible for approving. To approve or dispute a line item, the user selects Approve or Dispute in the Action radio button as illustrated in Figure 8-25.

**Figure 8-25** Selecting an Action radio button




The approval status can have several values. The approval processes uses the following values:

- UNAPPROVED -- this line item has not been reviewed yet
- PENDING -- this line item has received initial approval
- DISPUTED -- this line item has been disputed
- APPROVED -- this line item has received final approval

Most of the approval processes have multi-stage approval, first one person reviews each line item, then another person reviews it, and possibly even another. When a user approves a line item and submits the form, the approval status is changed from UNAPPROVED to PENDING. If the approval status is already PENDING it remains that way. If the value was previously DISPUTED, it is changed to PENDING since each approver can override the results of previous approvers.

The PENDING status indicates that the line item has received initial approval but is not fully approved yet, therefore if anyone from accounts payable views the invoice they will know that the line item is not ready to be paid.

If the user disputes a line item, the approval status for that line item is set to DISPUTED. Obviously, the accounts payable personnel will know that disputed line items should not be paid, thus there is no need to distinguish between whether the line item has been initially or finally disputed.

Each of the approval processes have an automated work item that changes all PENDING line items to APPROVED when all approvals have been completed.

This invoice field has the following properties that can be set in the Builder:

- Name of `bcaid` field -- The name of the field that holds the account number. Any process that uses this field must also contain a textfield with this name.
- Name of `docid` field -- the name of the field that holds the *document id* number. Any process that uses this field must also contain a textfield with this name.

  The `bcaid` and `docid` identify which invoice to display. If either of these fields do not have valid values, the invoice cannot be displayed.
- Name of Assignee Context Field -- the name of a context field that the process uses for storing information about users and departments. If you create a process that does not contain a context field of this name, then it can only use the standard pre-defined display format, as discussed below.
- Name of Current Line Items Field -- the name of a context field that the process uses for recording the ids of line items that were modified during the current work item. If your process does not contain a context field of this name, then it cannot test for the line items that were modified during the current work item.

## Which Line Items Are Displayed

In some cases the invoice displays all line items to a user and in other cases it displays only the line items that the user is responsible for approving. In some situations too, it allows the user to choose whether to view only their own line items, the line items of their direct subordinates, or the line items of everyone below them in the company hierarchy.

Several things affect which line items are displayed to which users:

- whether the user is a member of the SuperApprovers groups
- line item resolution
- `displayFormat` property

### *SuperApprovers Group*

The Invoice Manager Approval process and the Invoice Approver Approval process both contain an application group (that is, a group defined within the process) called SuperApprovers. All members of this group always see all line items in the invoice.

### *DisplayFormat Property*

The invoice fields use HTML templates to display themselves in a form. See the section *How Invoice Data Fields are Displayed* for more information about HTML templates. These templates check the value of the `displayFormat` property for particular values when deciding which line items to display. This property is set on each instance of the field that is displayed on each form. To set the property, open the form in the Builder, select the field in the form, right click on the field and select Properties from the menu.

The information that the invoice fields check for in the `displayFormat` property includes:

- title -- determines the title of the line items section of the invoice
- acl -- determines which line items to display to the current user
- lock -- determines whether or not the invoice can be dynamically redisplayed to view a different selection of line items, such as line items for direct subordinates or all subordinates
- limit -- determines the name of the data field, if any, that specifies the minimum limit for the total price of line items to be displayed

### *More About ACLs (Access Control Lists)*

The value of `acl` in the `displayFormat` property affects how the invoice field is displayed in an HTML page. When an invoice field is displayed in an HTML page, a set of buttons allows the user to redisplay the invoice in various modes, unless the display format was locked in the Builder.

To specify the default display format for an invoice field in a particular form, specify one of the following values for the `displayFormat` property of the invoice fields in the relevant form:

- `acl=d0`

  display only the user's direct line items (or the line items for the person of whom the user is a manager or approver.)

- `acl=d1`

display the user's direct line items and the line items for the people one level below them in the LDAP hierarchy.

- `acl=dX`

  display the user's direct line items and all the line items for everyone below them in the LDAP hierarchy.

- `acl=all`

  display all line items in the invoice.

- `acl=none`

  display no line items in the invoice.

In some cases the invoice field displays links allowing the user to view only their own line items, their direct subordinates line items, or the line items that are owned by everyone in their hierarchy. At other times the display is fixed; the user cannot dynamically change the line item selection. To lock the display format so that it cannot be dynamically changed at runtime, add `lock` to the front of the value as follows:

- `lock acl =d0`
- `lock acl =d1`
- `lock acl =dX`
- `lock acl = all`
- `lock acl = none`

You can also set the display format to use other kinds of access control mechanisms, such as `acl=aclByManager` and `acl=aclByApprover`. If you do this, the process must contain a context data field whose name matches the “Name of Context Field” property on the invoice field. These alternate access control mechanisms can be locked if desired, for example, `lock acl=aclByApprover`.

To use an alternate access control mechanism, you need to write a script that adds entries to the context field. Each entry should be keyed by a the kind of acl (for example, `aclByManager`), and the value should be a Vector of entries keyed by userid. Each of these entries would be a list of department numbers for which the user has access to view the line items that belong to that department. The Manager approval and Approver approval process use this kind of access control mechanism in the steps that display the invoice to the manager or the approver.

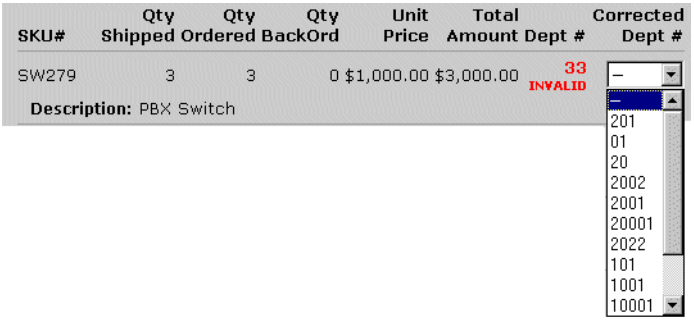
The invoice field cannot resolve alternate access control mechanisms, such as `aclByManager`, if the process does not contain a context field of the appropriate name.

By default, if the `displayFormat` property of an invoice field does not have a value for `acl`, the field shows the user's own line items and the line items for everybody below them in the LDAP hierarchy. However, if the `displayFormat` has a value that is not one of the recognized values, the field does not show any line items when it is displayed.

# BillerXpert Departments Invoice Field

The `DeptInvField` data field, which is a subclass of `InvoiceField`, shows line items that have invalid departments. The user is presented with a selectable list of department numbers so they can specify a correct department number, as shown in Figure 8-26.

**Figure 8-26** Selecting a department number




When the user selects a department number, the selected value is installed as the line item's department number.

This custom data field has the following properties that can be set in the Builder:

- Name of BCA Id field -- The name of the field that holds the account number. Any process that uses this field must also contain a textfield with this name.
- Name of DocId field -- the name of the field that holds the invoice number. Any process that uses this field must also contain a textfield with this name.

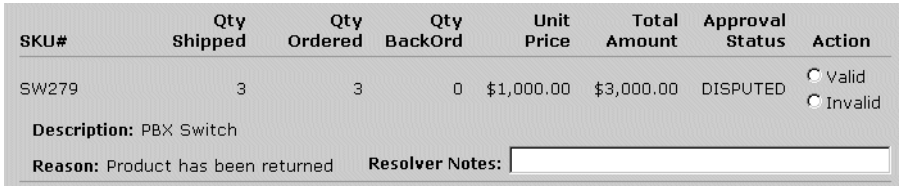
The BCAId and DocId identify which invoice to display. If either of these fields do not have valid values, the invoice cannot be displayed.

The dynamic display filters for this data field include “All”, “Valid Depts” and “Invalid Depts” so that the user can choose whether to view all line items or just those with either valid or invalid department numbers.

# BillerXpert Invoice Resolution Field

The `DisputeInvoiceField` data field, which extends `InvoiceField`, shows the line items in the invoice that are under dispute. Each line item presents a radio button that lets the user specify whether the dispute is valid or invalid as shown in Figure 8-27.

**Figure 8-27** Selecting an Action radio button for a disputed item




When the user (such an administrator at the Biller company) specifies that the dispute is valid, the process changes the approval status of the line item to DISPUTE_VALID. When the user specifies that the dispute is not valid, the approval status changes to DISPUTE_INVALID.

This field has three properties that can be set in the Builder:

- Name of BCA Id field -- The name of the field that holds the account number. Any process that uses this field must also contain a textfield with this name.
- Name of DocId field -- the name of the field that holds the invoice number. Any process that uses this field must also contain a textfield with this name.

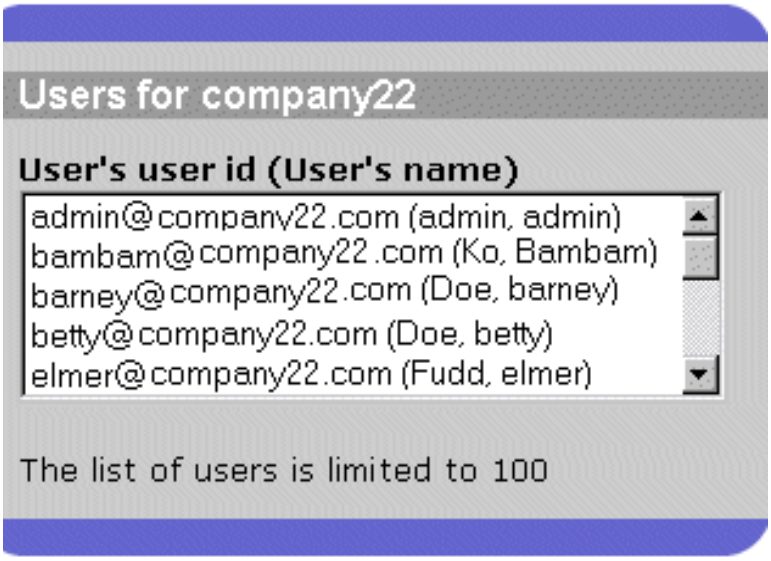
  The BCAId and DocId identify which invoice to display. If either of these fields do not have valid values, the invoice cannot be displayed.
- Name of Resolution Context field -- the name of the context field that holds the the resolver's decision regarding the dispute. Any process that uses this field must contain a context field with this name.

## BillerXpert Delegation Field

The delegation field is used to manage lists of delegatees. This field uses a context field to store a String array of the selected list of delegatees. This field displays a list of all the users contained in the department tree for the company along with the list of users currently selected.

- Company Id -- the company ID.
- Name of delegatee context field -- the name of the context field that holds the the lists of delegatees. Any process that uses this field must contain a context field with this name.

**Figure 8-28** BillerXpert Delegation Field




# Modifying the Display of Invoice Fields

The BillerXpert Invoice data fields (the invoice fields, context field and delegation field) are implemented as Java classes, as subclasses of `RhinoBasicCustomField`, which extends `BasicCustomField`. Rhino is an open-source implementation of JavaScript written in Java which is embedded into Java applications (in this case, the application is Process Manager) to provide scripting to end users.

In the BillerXpert B2B version of Process Manager, the display mechanism for the invoice fields is scripted in Rhino JavaScript, but the `update()` method is in Java. To modify the display of an invoice, you can modify the HTML templates that are used to display it. To modify the way an invoice is updated when a form displaying the invoice is submitted, you need to modify the Java classes used by the invoice field.

# How Invoice Data Fields are Displayed

When displaying any of the special data fields that are shipped with BillerXpert B2B Edition, Process Manager uses an HTML template file that has the same name as the data field. For example, the field `InvoiceField` is displayed by the HTML template file `InvoiceField.html`, while the field `DeptInvField` is displayed by the HTML template file `DeptInvField.html`.

Process Manager provides default HTML template files for the invoice fields, the context field and the delegation field. You can customize these templates if desired. You can also customize the templates on an application by application basis, or on a cluster by cluster basis (in effect, a biller by biller basis).

How do the processes decide where to find the template files? The process looks for the HTML template file in a directory structure that corresponds to the package. Let's consider the case of the `InvoiceField` data field, whose Java package is `com.netscape.pm.bx.fields.invoice`. In this case, the process looks for `InvoiceField.html` in the directory *`someroot`*`/com/netscape/pm/bx/fields/invoice`.

The next question is, what is *`someroot`*?

The process looks in the following places in the given order until it finds the appropriately-named HTML template file:

1. The `templates` directory in the particular application's folder.

   The template files in here affect all instances of the relevant field in the particular application. To customize the display of a data field for a single application, put the template files in this directory. If the appropriate directories do not exist, go ahead and create them.

   For example, the template for `InvoiceField` whose package is `com.netscape.pm.bx.fields.invoice` would be: (breaks added for readability only!)

   *`ias-install-dir`*`/bpm/clusters/`*`mycluster`*`/applications/`*`myapp`*`/templates/com/netscape/pm/bx/fields/invoice/InvoiceField.html`

2. In the particular application's folder.

   The template files in here affect all instances of the relevant field in the particular application. The template files here have the same effect as those in Step 1. It is better to group your template fields in the `templates` folder, but this part of the search path is included for internal compatibility reasons.

   For example, the template for `InvoiceField` whose package is `com.netscape.pm.bx.fields.invoice` would be: (breaks added for readability only!)

   *ias-install-dir*/bpm/clusters/*mycluster*/
   applications/*myapp*/
   com/netscape/pm/bx/fields/invoice/InvoiceField.html

3. In the `templates` directory in the directory of the particular cluster.

   The template files in here act as default templates for all instances of the relevant field in all applications in the particular cluster. To customize the display of a data field for all applications in a cluster, put the template files in this directory. If the appropriate directories do not exist, go ahead and create them.

   For example, the template for `InvoiceField` whose package is `com.netscape.pm.bx.fields.invoice` would be: (breaks added for readability only!)

   *ias-install-dir*/bpm/clusters/*mycluster*/templates/
   com/netscape/pm/bx/fields/invoice/InvoiceField.html

4. In the `fields` directory in *ias-install-dir*/bpm/resources/server/.

   The template files in here act as default templates for all instances of all the relevant fields in all applications in all clusters. To customize the default display of a data field for all situations, customize the template in this directory. If you create a new data field, you'll need to create the appropriate directories.

   For example, the template for `InvoiceField` whose package is `com.netscape.pm.bx.fields.invoice` would be: (breaks added for readability only!)

   *ias-install-dir*/bpm/resources/server/fields/
   com/netscape/pm/bx/fields/invoice/InvoiceField.html

# What Templates Can Do

The HTML template for use in Process Manager applications can contain the following:

- HTML code
- Client-side JavaScript
- JavaScript Rhino code
- Other template files

## JavaScript Rhino Code

Inside JavaScript rhino scripts you can call Process Manager functions and variables, for example `getProcessInstance()`. For a list of the public JavaScript API for Process Manager, see the *JavaScript API Reference* in the `Builder's Guide` at `http://docs.iplanet.com/docs/manuals`.

The HTML templates for displaying data fields can use the following implicit variables:

- `field` -- is the field data. For example, in the case of an invoice field, `field` evaluates to the invoice.

  For example,

  ```
  var nlitems  = field.lineItems();
  ```

  invokes the `lineItems()` method on the current `Invoice` object to get the number of line items.

- `fieldManager --` is the data field Java instance itself, for example the instance of `InvoiceField`. For example:

  ```
  var bSuperAppr = fieldManager.isSuperApprover (participant, pi);
  ```

  invokes the `isSuperApprover()` method on the current instance of `InvoiceField`.

- `fm` -- same as `fieldManager`

To embed rhino JavaScript enclose the code inside script tags `<% %>`, for example:

```
<% if (bxsum.getCurrentBalance() > 1000){ %>
```

Be sure to leave a space after `<%` and before `%>`. To write anything literal to the page, such as `<FONT COLOR="RED">` you must be outside the <% %>tag. In the following example, `<FONT COLOR="RED">` only gets written if the condition is true.

```
<% if (condition) { %>
<FONT COLOR=RED>
<% } %>
```

This particular code has the same effect as:

```
<%
    if (condition)
    {
        document.writeln("<FONT COLOR=RED>");
    }
%>
```

To write the content of an embedded script directly to the page, use `<%=`. For example, the following code dynamically figures out the color. Note that the closing "`>`" for the FONT tag is not embedded in `<% %>` tags since it is literal HTML text.

```
<% if (condition) { %>
<FONT COLOR= <%= calculateColor() %>
>
<% } %>
```

## Including Other Template Files

HTML template files can include other HTML template files by using the `evaluateTemplate()` JavaScript function.

JavaScript variables and functions defined in one template file can be accessed by other template files that it includes, or any template files that are evaluated after it.

For example, `InvoiceField.html` is shown in Code Example 8-1. It makes use of `evaluateTemplate()` to evaluate other HTML template files. The file `__code1.html` defines some functions that are used inside `__summary.html` and `__approvalLineItems.html`.

**Code Example 8-1** InvoiceField.html

```
<!-- InvoiceField Display: <%= fm.getName()  %>  -->

<%= evaluateTemplate (
"com/netscape/pm/bx/fields/invoice/__code1.html")  %>

<%= evaluateTemplate (
"com/netscape/pm/bx/fields/invoice/__approvalCode.html")  %>

<%= evaluateTemplate (
"com/netscape/pm/bx/fields/invoice/__summary.html" ) %>

<%= evaluateTemplate (
"com/netscape/pm/bx/fields/invoice/__approvalLineItems.html" ) %>

<%= evaluateTemplate (
"com/netscape/pm/bx/fields/invoice/__end.html") %>

<!-- /InvoiceField Done: <%= fm.getName() %> -->
```

# An In-Depth Look at the InvoiceField Templates

This section briefly examines the template files that are used to display the `InvoiceField` class to give you an idea of how the template files are used. For details on the template files used by the other classes, such as `DisputeInvoiceField`, see the source code. You can find the HTML template files used by the `InvoiceField`, `DisputeInvoiceField` and `DeptInvField` classes in *`ias-dir`*`/bpm/resources/server/fields/com/netscape/pm/bx/fields/invoice`.

The main template file for `InvoiceField` is `InvoiceField.html`. It calls `evaluateTemplate()` to include the following files:

- `__code1.html`

  This file defines some useful functions such as formatting functions.

  The following two functions can be used to make sure that if a value is null it is converted to something else so it can be included in the invoice display.

  `__checkNull(arg)` returns the arg if not null, otherwise returns NA.

  `__checkNull2(arg)` returns the arg if not null, otherwise returns 0.

  Other useful functions defined here include:

`__formatDate ( d )` formats a date.

`__formatCurrency ( c )` formats a number as currency.

`__formatNumber ( n )` formats a number.

This file defines the following variables:

`bxsum` -- the invoice summary, which is an `IBXSummary` object. `IBXSummary` provides methods for accessing a BillerXpert B2B invoice summary, such as `getCurrentBalance()`, `getDueDate()` and so on. For a full list of the methods on `IBXSummary`, see the *API & Schema Reference Manual*.

`bxdoc` -- the invoice document, which is an `IBXDocument` object. `IBXDocument` provides methods for accessing a BillerXpert B2B invoice document , such as `getCreateDate()`, `getInvoiceNum()` and so on. For a full list of the methods on `IBXDocument`, see the *API & Schema Reference Manual*.

You can use `bxsum` and `bxdoc` to refer to the invoice summary or invoice document respectively in any template file that is included after this one.

- `__approvalCode.html`

  This template file does some preliminary work, such as setting the title for the list of line items. It defines some functions used when displaying standard invoice field line items, such as `getDisputeCodes()`, which returns an array of dispute codes.

- `__summary.html`

  Writes the invoice summary.

- `__approvalLineItems.html`

  Writes the line items. It performs some preliminary calculations, such as how to resolve the line items (that is, how to select which line items to display), then it writes a table that contains a row for each line item. For information about modifying the display of line items, see the section “Modifying the Way Line Items are Displayed.”

- `__end.html`

  Ends the display of the field.

# Example Invoice Display Customization

So long as you have some familiarity with JavaScript, you can easily modify the display of an invoice, for example you can customize the display of the summary or the line items and you can modify the list of dispute codes that appears in the popup menu.

A more advanced customization is to change the way the field picks which line items to display. The `InvoiceField` uses `Resolver` Java classes that have a `resolve()` method, which tests a line item and returns true or false depending on whether it should be displayed or not. To change the way line items are selected for display, you need to create an appropriate `Resolver` class, as discussed in *Deciding Which Line Items to Display* on page 217. This is an advanced customization that requires knowledge of Java.

If you change the form elements that appear in the invoice display, you need to modify the `update()` method in the Java class, as discussed in *Form Elements in the Invoice Display* on page 220. This is an advanced customization that requires knowledge of Java.

## Customizing the Invoice Summary

To customize the invoice summary, edit `__summary.html`.

This file starts off by defining some functions that are used by buttons in the page. It's best not to modify these unless you know what you're doing. Then it writes out a table that displays the invoice summary. You'll notice that this table is mainly written in HTML, but drops into JavaScript when it needs to calculate something such as the invoice total. It calls methods on `bxsum` and `bxdoc`, which refer to the `IBXSummary` and `IBXDocument` objects used by the invoice. For a full list of the `IBXSummary` and `IBXDocument` methods see the *API & Schema Reference Manual.*

For an example customization, you could display the invoice total in red if it is greater than 10000. In this case you would edit the section that writes the table cell displaying the invoice total as shown in Code Example 8-2:

**Code Example 8-2** Example of customization to display the invoice total in red

```
<td align=right><b>Invoice Total</b></td>
<td> </td>

<td align=right><b>
<% if (bxsum.getCurrentBalance() > 1000){ %>
<FONT COLOR="red">
<% } else { %>
<FONT>
<% } %>
<%= __formatCurrency ( bxsum.getCurrentBalance() ) %>
</FONT></b>
</TD>
```

## Modifying the Way Line Items are Displayed

To modify the way that line items are displayed for the `InvoiceField`, look in `__approvalLineItems` for the section that starts with

```
while (lineItems.hasMoreElements() )
```

This iterates over the line items and writes information about them in table cells, such as the following cell, which displays the quantity shipped:

```
<td nowrap align='right'><%= __formatNumber ( li.getQtyShipped() )
%></td>
```

You can make modifications here. The variable `li` refers to the current line item and is an `IBXLineItem` object. You can call any `IBXLineItem` method on it, for example, `li.getQtyOrdered()`, `li.getUnitPrice()`, `li.getTotalAmount()` and so on. For a full list of the `IBXLineItem` methods see the *API & Schema Reference Manual*.

If you add or remove columns, be sure to make the corresponding change in the column headings too.

## Changing Dispute Codes

To change the list of dispute reasons, edit the function `getDisputeCodes()` in `__approvalCode.html` in the appropriate directory. This function must return an array of dispute reasons, for example:

```
function getDisputeCodes ()
{
```

```
return new Array("Did not buy it", "Not mine", "Never received it",
"Did not receive full quantity", "Product has been returned",
"Arrived too late", "Discount has not been applied", "Price differs
from quoted price", "Other" );
}
```

## Deciding Which Line Items to Display

The invoice fields use auxiliary objects called Resolvers to figure out which line items to display. The `Resolver` Java class implements the `IResolver` interface, which declares the following method:

```
public boolean resolve (IBXLineItem li)
```

The `resolve()` method returns true if the line item is to be displayed or false if not.

The `InvoiceField` class has a method:

```
public Vector resolveItems ( IResolver res )
```

This method calls `resolve()` on the appropriate resolver for each line item in the invoice. It returns a vector containing all the line items for which `resolve()` returns true, which is effectively a vector containing the line items to be displayed.

To resolve line items in a different way, write a new `Resolver` class that implements `IResolver`. Define the `resolve(IBXLineItem)` method to return true or false. The `resolve()` method always takes just a line item as its argument. If the criteria for resolution involves other factors, define the constructor to take account of those factors. The constructor can take any arguments you like.

For example, in the following `Code Example 8-3`, `LimitResolver` has a `resolve()` method that returns true only if the total amount of the line item exceeds a given limit. This resolver is used in the Approver Approval process to determine which line items to display in the third approval stage.

**Code Example 8-3** public class LimitResolver

```
public class LimitResolver implements IResolver
{
 // The price limit
 private int moLimit = 0;

 public LimitResolver ( int limit )
 {
    moLimit = limit;
 }

 // Returns true or false, depending on whether the
 // line item's total amount exceeds the limit or not.
 public boolean resolve ( IBXLineItem li )
 {
    return (li.getTotalAmount() > moLimit);
 }
}
```

For another example of resolvers, look at the source code for `Resolver.java`, which is more complicated. Also see `DepartmentResolver.java`, which resolves line items based on whether their department number is valid or not.

You can string resolvers together to resolve a line item using multiple criteria by using the following resolvers:

- `AndResolver`

  This resolver takes an array or vector of other resolvers and resolves true for a line item only if all the other resolvers resolve true for it. For example, in the following code, the `finalResolver` resolves true for a line item only if the `QuantityBackOrderedResolver` and the `LimitResolver` both resolve it as true.

```
resolver1 = new QuantityBackOrderedResolver();
resolver2 = new LimitResolver(limit);

finalResolver = new AndResolver([resolver1,resolver2]);
```

- `OrResolver`

This resolver takes an array or vector of other resolvers and resolves true for a line item if any of the other resolvers return true for it. For example, in the following code, the `finalResolver` resolves true for a line item if either the `QuantityBackOrderedResolver` or the `LimitResolver` resolve it as true.

```
resolver1 = new QtyBackOrdResolver();
resolver2 = new LimitResolver(limit);

finalResolver = new AndResolver([resolver1,resolver2]);
```

To use resolvers in the HTML template that displays an invoice:

1. Create the appropriate `Resolver` class. Unless you have explicitly imported the appropriate Java package into the HTML template, include the full Package name, for example:

   ```
   var resolver =
   Packages.com.netscape.pm.bx.fields.invoice.QtyBackOrdResolver;
   ```

2. Call `resolveItems()` on the invoice, passing the resolver. Remember here you can use the implicit variable `field` to represent the value of the data field, which in this case is the `Invoice` object.

   ```
   var lineItemsToDisplay = field.resolveItems(finalResolver);
   ```

3. Your code can now iterate over `lineItemsToDisplay`, which is a vector of the line items that have been resolved as true (that is, the line items to display) and display each line item as desired.

## Form Elements in the Invoice Display

The display mechanics for the invoice fields are implemented in JavaScript, but the update mechanics are implemented in Java. If you change the form elements that get displayed per line item, for example, you make quantity ordered be an editable text field, you'll need to make a corresponding change in the `update()` method in the appropriate Java file. The `update()` method processes the form elements when the form is submitted, and updates the invoice accordingly. Thus if you add a new form element to each line item, it won't be processed unless you modify the Java `update()` method to handle it.

If you know what you're doing and you go ahead with adding new form elements or changing the existing form elements, there are a couple of functions that are helpful. The `update()` Java method must process the form elements of the same names as the JavaScript display mechanism writes out. For example, if the display mechanism writes a text field named `123_totalamount`, the `update()` method should process a parameter named `123_totalamount`.

The display mechanism generates a unique name for each form element. Since each element is typically associated with a single line item, the name is constructed using the line item id and a key, for example, `action`. The display code can call `paramName(lineitemid, "key")` to generate a unique name for each form element, for example, `paramName(lineitemid, "action")` for the approve / dispute radio buttons.

The following code is an extract from `__invalidLineItems.html`, which is the template that writes the line items in the invoice in the Set Valid Departments process. In this case, each line item is displayed with the department number shown as a select field so that the user can select a department number for the line item. The salient thing to note in this code is the use of

```
name="<%= paramName( lineitemid, "SET_DEPT" ) %>"
```

to generate the name of the form element that contains the selectable department number.

```
<%
    var lineitemid = "";
    while ( lineItems.hasMoreElements() )
    {
        var li = lineItems.nextElement();
        var lineitemid = li.getPrimaryKey();
%>
<! -- Write out the row for each line item -->
<tr>
    more content in the row goes here

<!-- Write the table cell to show the current dept num in red -->

<td nowrap align='right'><font color=red>
    <b><%= li.getDeptNum() %><BR>
    INVALID</b></font>
</td>

<!-- Write the table cell to show the select list of dept nums -->
<td align='right' nowrap>
    <select style="font-size: 10pt"
        name="<%= paramName( lineitemid, "SET_DEPT" ) %>">
        <option selected>--</option>
        <% for (var i=0; i<validDepts.length; i++ )
            { %>
                <option><%= validDepts[i] %></option>
            <% } %>
    </select>
</td>
</tr>

<!-- end the while loop that is iterating over the line items -->
<% } %>
```

The following code shows a stripped-down version of the Java `update()` method for the `InvDeptField` class. For the full code, see the source code in `InvDeptField.java`. Here we show the salient points of the method that illustrate how to get form element names. What's happening here is that for each line item

displayed, the method takes the new department number and puts in the `IBXLineItem` object. The new department number is indicated by the value of a field in the form. Note the use of the `(String) parmMap.get("SET_DEPT")` to get the value of the “set department” form element for the current line item.

```
public void update( IProcessInstance pi, IPMRequest rq )
    throws Exception
{
    // parseLineId2Parms is defined in InvoiceField
    HashMap lineId2Parms = parseLineId2Parms ( rq );

    // Load the invoice and update all
    // the line items with department numbers
    Invoice inv = (Invoice) pi.getData( getName() );

    // Perform the update.
    for (Iterator ids = lineId2Parms.keySet().iterator();
        ids.hasNext(); )
    {
        String id = (String) ids.next();
        IBXLineItem li = inv.getLineItem ( id, true );

        HashMap parmMap = (HashMap) lineId2Parms.get( id );

        // Set the deptnum in the BXLineItem
        String newdeptnum = (String) parmMap.get( "SET_DEPT" );
        li.setDeptNum ( newdeptnum);
        }
}
```

# Creating New Invoice Data Fields

If none of the predefined invoice fields suits your needs, you can create a custom data field to display and manage the invoice as you want. For example, you could define an invoice data field that displays the summary information and the total cost of the invoice without displaying individual line items.

# Data Field Classes

The BillerXpert Invoice data fields (the invoice fields, context field and delegation field) are implemented as Java classes, as subclasses of `RhinoBasicCustomField`, which extends `BasicCustomField`. Rhino is an open-source implementation of JavaScript written in Java which is embedded into Java applications (in this case, the application is Process Manager) to provide scripting to end users.

The main difference between `RhinoBasicCustomField` and `BasicCustomField` is that `RhinoBasicCustomField` allows you to script the display mechanism in rhino JavaScript instead of having to code it all in Java. See the section *How Invoice Data Fields are Displayed* on page 210 for details.

The source code for the invoice data fields is available in `bpm/support/src` in the Process Manager server installation so you can see how they are implemented. When creating a new invoice data field, you can create a data field from scratch or use an existing invoice field as a superclass, depending on your needs. In general it's best to create a subclass of `InvoiceField`, which provides base functionality for managing invoices.

For information about creating custom data fields, see *Chapter 2, Writing Custom Data Fields* in the *Process Manager Programmers Guide* at `http://docs.iplanet.com/docs/manuals`. This chapter provides details of implementing data fields as subclasses of `BasicCustomField`. Everything that applies to `BasicCustomField` also applies to `RhinoBasicCustomField`. However, to take advantage of having the display mechanism be scriptable through rhino JavaScript, do not override the `display()` methods at all, instead create HTML template files as discussed in *How Invoice Data Fields are Displayed* on page 210.

# IInvoice

The `InvoiceField`, `DeptInvField` and `DisputeInvoiceField` data field classes manage `IInvoice` objects as their data. The `IInvoice` interface provides methods for interacting with the appropriate `IBXDocument` object, `IBXSummary` object and `IBXLineItem` objects.

- `public IBXDocument getDocument ( )`

  `IBXDocument` provides methods for accessing a BillerXpert B2B invoice document , such as `getCreateDate()`, `getInvoiceNum()` and so on. For a full list of the methods on `IBXDocument`, see the *API & Schema Reference Manual*.

- `public IBXSummary getSummary( )`

`IBXSummary` provides methods for accessing a BillerXpert B2B invoice summary, such as `getCurrentBalance()`, `getDueDate()` and so on. For more a full list of the methods on `IBXSummary`, see the *API & Schema Reference Manual*.

- `public Enumeration getLineItems( )`

  returns an `Enumeration` of `IBXLineItem` objects.

  `IBXLineItem` provides methods for accessing line items in a BillerXpert B2B invoice, such as `getSkuNumber()`, `getUnitPrice()`, `getQtyShipped()` and so on. For more a full list of the methods on `IBXLineItem`, see the *API & Schema Reference Manual*.

The `IInvoice` interface also provides utility methods that are used by the data field classes, such as `getInvoiceDepts()`, which returns a hashmap of the departments that own line items in the invoice. For more information, see the javadoc in the Process Manager server installation at *`bpm/support/src`*.

# Utility Classes and Resources

The invoice fields use the additional auxiliary classes:

- `com.netscape.pm.bx.util.DepartmentTree.java` -- interacts with LDAP to access users and departments.

  `DepartmentTree` provides methods for getting connection to the LDAP server, for finding the department numbers in the Department hierarchy, for finding which department a user belongs to, and so on. The full package for `DepartmentTree` is `com.netscape.pm.bx.util.DepartmentTree`. For a full list of the methods provided by `DepartmentTree`, check the javadocs or the source code.

- `com.netscape.pm.bx.BXResources.java` -- contains BillerXpert resources.
- `com.netscape.pm.bx.BXDefs.java` -- contains BillerXpert definitions.
- `com.netscape.pm.bx.BXResources_en_US.properties` -- contains some necessary properties.

# Context Data Fields

The context field lets you store any number of objects that implement `java.io.Serializable`. Context feilds are useful for storing data that is more complex than just a single value, but does not need to be dynamically generated or retrieved, thus does not require the creation of its own custom data field.

## Using Context Fields

The value of a context field is a Java Hashmap that contains serializable objects. After modifying the value of a context field, be sure to call `setData()` on that field to ensure that it gets saved. The invocation of `setData()` triggers Process Manager to save the value of a data field. Thus if you modify an object that is the value of a context field but do not call `setData()` on the field, your changes are not saved.

The following code creates an object and stores it in a context field. This code might be included in a JavaScript script such as an automation script or a completion script. This example assumes that the class `myObject` implements `java.io.Serializable`.

```
var pi = getProcessInstance();
var value1 = pi.getData("field1");
var value2 = pi.getData("field2");
// Create the object to be stored
if (value1 > value2)
{
 var myObject = new Packages.com.some.package.myObject(value1);
}
else
{
 var myObject = new Packages.com.some.package.myObject(value2);
}
// Initially, the value of a context field is an empty Hashmap.
var myHashmap = pi.getData("myContextField");
// Add the object to the hashmap
myHashmap.put("myKey", myObject);
// Put the hashmap back into the data field
pi.setData("myContextField", myHashmap);
```

The following code shows how to retrieve the object from the context field in a JavaScript script:

```
var pi = getProcessInstance();
var myHashmap = pi.getData("myContextField");
var myObject = myHashmap.get("myKey");
```

The Invoice_Manager_Approval process uses two context data fields, one called `AssigneeList` and one called `tempLineItems`. The `AssigneeList` field stores information about the assignees and the departments whose invoices they are allowed to view. The `tempLineItems` field keeps track of which line items are being worked on in the current work item.

## Cleaning Up Context Fields

A context field saves its value by serializing all the objects in its hashmap to a file. The path to this file is saved as the entity key for the context field. It is good practice to clean up the file at the end of the process, as illustrated in the following code, which is extracted from the `cleanup()` automation script in the Invoice_Manager_Approval process.

```
// Extracted from cleanup() automation script
var pi = getProcessInstance();

// Get the context field entity key
// This is the absolute path for the serialization file.

try
{
    // Delete the file for the tempLineItems context field
    path = pi.getEntityKey( "tempLineItems" );
    if( path != null )
    {
    f = new
        Packages.com.netscape.pm.sample.fields.ContextFile(path);
    f._delete();
    }
}
catch( e )
{
    e.printStackTrace();
    // ignore
}
```

# API Documentation and Compilation Information

For information about the methods used by the invoice field and delegation field classes, see the javadoc in the Process Manager server installation at *bpm/support/docs* For the Java source code for the data fields and their auxiliary classes, see *bpm/support/src*. For the JavaScript source code for the viewer classes, look in *bpm/resoruces/scripts/bx*.

When compiling the invoice field or delegation field classes and their associated classes, you need to include the BillerXpert jar files in the compilation class path for access to BillerXpert classes. You also need to include the Process Manager classpath in the compilation for access to certain Process Manager classes that the fields access, such as `BasicCustomField`. The Process Manager classpath is typically `ias6/bpm/classes` in the installation directory.

If your compilation complains that it cannot find `javax.ejbObject`, then include *server-root*`/ias/lib/java/javax.jar` as an input jar.

The invoice data fields import packages in the `com.kivasoft` packages. You can find these packages in the `kfcjdk11.jar` file in the iPlanet Application Server installation if you need them.

## *See Also*

For more information on Process Manager, please refer to the Process Manager documentation online at `http://docs.iplanet.com/docs/manuals`.

Chapter 9

# Currency Converter

iPlanet BillerXpert B2B Edition provides a Currency Converter module to support currency conversion capabilities for both banks and billers. The Currency Converter module is a plugable component that provides host application API's to help convert currencies. It is EURO compliant.

Within the Currency Converter module, there is also a loader program, which is used to load the currency conversion entries. The loader program takes in an Extensible Markup Language (XML) file of conversion data and loads it into the conversion table. Updates to the currency conversion table are performed by the administrator either manually, or via an external scheduling software program, such as *ECXpert*.

*ECXpert is an Internet commerce exchange application that enables an enterprise to automate and manage the processes that occur between organizations over the Internet and existing private networks. ECXpert enables encrypted transmission of documents and messages among heterogeneous trading partner systems, and can transform information from one format into another.*

## Currency Conversion Process

The currency conversion process is as follows:

1. Currency codes and related information are loaded at the time of installation. These codes are International Organization for Standardization (ISO) standard and should not change. The Euro currencies are identified through the use of a special flag.

2. The latest currency rates are loaded into the database using a loader program. This program reads the data in XML format from a file and loads the rate table. Each load is tracked on the basis of conversion date. Each load is tracked in the database by a sequentially generated id, fileid. *Please refer to the database schema for more details.*
3. When the conversion method is called, it retrieves the appropriate fileid by using the date specified to find the latest track for that date. This search is on the basis of conversion date (not the load date).
4. Once the appropriate load is determined, the required rate is searched from the `currencyconversion` table, using the fileid retrieved in step 3.
5. If the currency is a Euro currency, the rate between the other currency and Euro is retrieved. This rate is then used to convert the amount from one currency to another. In case of Euro currencies, the conversion always passes through Euro, i.e. Euro currency -> Euro -> other currency. For conversion rules regarding the EURO, please refer to the official site `http://europa.eu.int/eur-lex/en/lif/dat/1997/en_397R1103.html`

# Database Schema

Table 9-1 lists and describes the three tables contained in the Currency Converter database schema.

**Table 9-1** The three tables contained in the Currency Conveter database schema.

| Table Name | Purpose |
|---|---|
| `currencyloadtrack` | Stores information about when and where each load took place. |
| `Currency` | Master data is loaded to this table only. |
| `currencyconversion` | Displays exchange information for each load. |

More information about each of these three tables can be found in the following sections.

## currencyloadtrack

Table 9-2 decribes the data fields used in the `currencyloadtrack` table.

**Table 9-2** Data fields for the `currencyloadtrack` table.

| Contents | Convention Used | Null | Description |
|---|---|---|---|
| FILEID | NUMBER(15) | not null, | Sequence number, which is internally generated. |
| FILEPATH | VARCHAR2(240) | not null, | Complete path of XML file loaded. |
| RECORDCOUNT | NUMBER(10) | not null, | Number of records. |
| LOADDATE | DATE | not null, | The date the file was loaded. |
| STATUS | VARCHAR2(16) | not null, | Status code of the job. |
| SOURCE | VARCHAR2(240) | null, | Source of conversion data, e.g. Reuters, BOA, etc. |
| CONVDATE | DATE | not null, | Currency conversion date time. |

## Currency

Table 9-3 decribes the data fields used in the `Currency` table.

**Table 9-3** Data fields for the `Currency` table.

| Contents | Convention Used | Null | Description |
|---|---|---|---|
| CURCODE | CHAR(3) | not null, | 3-character code, e.g., USD, EUR. |
| CURID | NUMBER(10) | not null, | ISO currency id., numeric. |
| MINORUNIT | NUMBER(5) | not null, | Number of units of minor unit. e.g., for USD: 2. |
| CURDESC | VARCHAR2(240) | null, | Literal discription for the currency. |
| CREATEDATE | DATE | null, | The date the record was created. |
| EUFLAG | CHAR(1) | null, | Y / N to indicate whether or not the currency belongs to European Union. |

## currencyconversion

Table 9-4 decribes the data fields used in the `currencyconversion` table.

**Table 9-4** Data fields for the `currencyconversion` table.

| Contents | Convention Used | Null | Description |
|---|---|---|---|
| CONVID | NUMBER(15) | not null, | Sequence number, which is internally generated by the loader program. |
| FROMCUR | CHAR(3) | not null, | Currency Code that was converted *from*. |
| TOCUR | CHAR(3) | not null, | Currency Code that was converted *to*. |
| FACTOR | NUMBER(20,8) | not null, | Multiplication factor used for conversion. |
| FILEID | NUMBER(15) | not null, | Sequence number, which is internally generated. |

**NOTE** The load date refers to the date the data was loaded. The conversion date refers to the date the data became valid.

For more information, please refer to the *iPlanet BillerXpert B2B Edition API & Schema Reference Manual.*

# Loading Data

The loader program is used to load the currency conversion entries. The loader program takes in an XML file of conversion data and loads it into the conversion table. Updates to the currency conversion table can be performed by the administrator either manually, or via an external scheduling software program.

Typically, a biller would manually execute the loader on a daily basis; whereas a financial institution may choose to schedule updates every 30 minutes, or more frequently.

# Loading As A Standalone

**Code Example 9-1** Loading as a standalone file.

```
java com.iplanet.ecomm.currconv.CCLoad <InputXMLFile>
<dbpropertiesfile >
```

where

-`InputXMLFile` : Input XML file *(please refer to DTD Format and sample XML file)*

-`dbpropertiesFile` : properties file, which is used to make a connection to the database.

## Database Properties File

**Code Example 9-2** Database Properties File

```
currConv.dbUserName=USER_NAME_VAL
currConv.dbPassWord=PASSWORD_VAL
currConv.SID=SID_VAL
currConv.dburl=jdbc:oracle:oci8:
currConv.driverClass=oracle.jdbc.driver.OracleDriver
```

## DTD Format

**Code Example 9-3** DTD Format

```
<!ELEMENT CurrencyConversion (Source, ConvDate, Convert+)>
<!ELEMENT Source (#PCDATA)>
<!ELEMENT ConversionDate (#PCDATA)>
<!ELEMENT Convert (From, To, Factor)>
<!ELEMENT From (#PCDATA)>
<!ELEMENT To (#PCDATA)>
<!ELEMENT Factor (#PCDATA)>
```

The above format can be used to load any file that is in a non-Reuter's format. For example, you may want to use this to load fixed rates, as Reuters does not supply fixed rates.

**Code Example 9-4** Example of DTD file format

```
currConv.load.CCLoad.CCFFile=${BX_HOME}/currconv/euro_fixed_rates.x
ml
```

### Sample XML File

The following is a sample of an xml file for fixed rates.

**Code Example 9-5** Sample XML file for fixed rates.

```
<?xml version=”1.0” standalone = ‘no’?>

<CurrencyConversion>
<source>Reuters>/source>
>convDate>2000.11.29 11:48::34 GMT+05:30</convDate>

<Convert>
<From>EUR<//From>
<To>BEF</To>
<Factor>40.3399</Factor>
</Convert>

<Convert>
<From>EUR</From>
<To>DEM</To>
<Factor>1.95583</Factor>
</Convert>
</CurrencyConversion>
```

## Loading Within Your Code

**Code Example 9-6** Loading a File Within Your Code.

```
CCLoad ccLoad = new CCLoad(xmlFile, dbpropertiesFile);
```

or

```
CCLoad ccLoad = new CCLoad(xmlFile, con);

ccLoad.loadData();
```

where `con` is a jdbc Connection, `xmlFile` and `dbpropertiesFile` are String variables whose value is the `xmlFile` name and `dbpropertiesFile` name.

# Configuring Load Properties

## To Load Reuters Data

BillerXpert provides out of the box functionality to load data from Reuters. Please refer to the *BillerXpert B2B Edition Integration Guide* for more details on setting up feeds from Reuters.

The `CCLoad.properties` file is read while pulling data from Reuters. The following properties need to be configured:

1. `currConv.load.PullData.servername=PROXY_HOSTNAME_NULL_IF_DIRECT_INTERNET_CONNECTION`
   - sets the proxyserver from where connection will be made to Reuters. Null (Empty String) if direct connection to internet is available.
   - gets data from external sites, if the application is hosted within a firewall, this would be the proxy server. (host.domain)
2. `currConv.load.PullData.serverPort=PROXY_PORT_NULL_IF_DIRECT_INTERNET_CONNECTION`
   - sets the proxy port from where connection will be made to Reuters. Null (Empty String) if direct connection to internet is available.
3. `currConv.load.PullData.reutersHostname=REUTERS_HOSTNAME`
   - This is Reuters host name e.g. "ri2.rois.com"
4. `currConv.load.PullData.reutersPort=REUTERS_PORT`
   - http port on Reuters host (80)
5. `currConv.load.PullData.reutersSession=SESSION_ID_FROM_REUTERS`
   - session id string as given by Reuters
6. `currConv.load.PullData.reutersFormat=XML`
   - Reuters data format (XML or HTML). This should remain as XML
7. `currConv.load.PullData.reutersRICs=REUTER_RICSs_SPACE_SEPERATED`
   - Reuters RICS that need to be pulled (eg : FX JPYX )
8. `currConv.load.PullData.outDir=DIR_TO_PULL_THE_RICs`
   - Output dir where the data will be pulled to. The pull program creates a directory `.Pulldata` under this directory.

### To Load Non-Reuters Data

1. `currConv.load.CCLoad.CCFFile=${BX_HOME}/currconv/euro_fixed_rates.xml`
    - This is an optional property which allows you to specify an XML file that will be loaded with each load. The purpose is to load certain fixed currency rates (e.g., the European Union rates). It is recommended to load this file if conversion requires EURO currencies. Since these are fixed rates, Reuters feed will not have these rates.

## Configuring getData.csh

`getData.csh` defines the following properties. It is recommended to use these values.

1. `setenv DATABASEPROPFILE  $BX_HOME/currconv/database.properties`

- The file contains the database id and password. This file is created at time of installation.

2. `setenv REUTERS_PROP  $BX_HOME/currconv/CCLoad.properties`

- Reuters properties file. For more detail, refer to the section *Configuring Load.properties*

## Enabling Live Currency Updates

Customers interested in purchasing the Reuters Investor Product to enable Live currency updates with the iPlanet currency converter should contact their local Reuters sales representative at `http://www.reuters.com/onlinemedia`. Click on CONTACT US.

# Sample Code

## API Usage

```
CurrencyIface a = (CurrencyIface) new CurrencyDBImpl(con);
Currency fromCur = a.getCurrency ("USD");
Currency toCur = a.getCurrency ("ESP");
```

```
CurrencyConvertor cc = a.getCurrencyConvertor (fromCur, toCur);
outval1 = cc.convert (100, 3, 2);
outval2 = cc.convert (200, 2, 2);
outval2 = cc.convert (500);
```

where con is a `java.sql`.Connection

and USDCur, and ESPCur are com.iplanet.ecomm.currconv.Currency

## To get conversion for a conversion rate as of a past Date:

```
Date dt ; // Initialize the Date
...
CurrencyIface a = (CurrencyIface) new CurrencyDBImpl(con);
LoadTrack lt = a.getLastLoadTrack (dt);
Currency fromCur = a.getCurrency ("USD");
Currency toCur = a.getCurrency ("ESP");
CurrencyConvertor cc = new a.getCurrencyConvertor (fromCur,
toCur, d1t);
```

For more information, please refer to the *BillerXpert B2B Edition API & Schema Reference Manual.*