Developing Telco Service Manager

V4.0
Document ID: TSGN-08-4.0-01
Date Published: 9.10.03

© 1997–2003 edocs® Inc. All rights reserved.

edocs, Inc., One Apple Hill Drive, Suite 301, Natick, MA 01760

The information contained in this document is the confidential and proprietary information of edocs, Inc. and is subject to change without notice.

This material is protected by U.S. and international copyright laws. edocs and eaPost are registered in the U.S. Patent and Trademark Office.

No part of this publication may be reproduced or transmitted in any form or by any means without the prior written permission of edocs, Inc.

eaSuite, eaDirect, eaPay, eaAssist, eaMarket, and eaXchange are trademarks of edocs, Inc.

Sun, Sun Microsystems, Solaris, Sun-Netscape Alliance, iPlanet, Java and JavaScript are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the United States and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

Netscape, Netscape Enterprise Server, Netscape Navigator, Netscape® Application Server and the Netscape N and Ship's Wheel logos are registered trademarks of Netscape Communications Corporation in the United States and other countries.

Microsoft, Windows, WindowsNT, Windows 2000, SQL Server and Microsoft Internet Information Server are registered trademarks of Microsoft Corporation in the United States and other countries.

Oracle, Oracle8, Oracle8i are registered trademarks of Oracle Corporation in the United States and other countries.

Adobe, Acrobat, and the Acrobat logo are trademarks of Adobe Systems Incorporated.

This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

Contains IBM Runtime Environment for AIX(R), Java(TM) 2 Technology Edition Runtime Modules (c) Copyright IBM Corporation 1999, 2000 All Rights Reserved.

This software contains Log4j Copyright (c) 1999 The Apache Software Foundation All Rights Reserved.

This software contains Jakarta-ORO regular expressions processing Copyright (c) 2000 The Apache Software Foundation All Rights Reserved.

This software contains Sun Multi-Schema XML Validator Copyright (c) 2001 Sun Microsystems All Rights Reserved.

All other product names and registered trademarks are the property of their respective holders. Any trademark name appearing in this guide is used for editorial purposes only, and to the benefit of the trademark owner, with no intention of infringing upon the trademark.

Federal Acquisitions: Commercial Software - Government users subject to standard license terms and conditions.

Preface

In This Section

Using this Manual iv
Finding the Information You Need x
If You Need Help xii

Using this Manual

Welcome to Developing Telco Service Mananger.

This manual covers building account management applications with TSM.

Before You Get Started

You should be familiar with the following:

- Your application architecture
- Programming Java and Java Server pages
- Designing or working with databases
- eXtended Markup Language (XML)

Who Should Read this Manual

This manual is for developers and project managers who are responsible for developing the user interface.

However, there are other topics covered in this manual that may interest other members of the project development team.

- Administrators

  You will find information about the different components that make up the user interface. You can learn the location of the different files which make up the user interface.

- Developers

  This manual is for building user interfaces for your solution. You learn how write JSPs that use the Presentation Manager JavaServer Page framework. You also learn how to group and program sets of JSPs. These sets, called channels, allow users to access the same solution by using different devices and protocols.

  You also learn how to use the framework to create new workflows, customize menus, and manage personalization information to create interactive and customizable user interfaces.

- Project Architect

  You can use the information in this manual to learn about channels and how they work. You can learn about the components and the flexibility of your solution when it is based on channels built on a common framework.

- Project Manager

  You will find information about channels and the Presentation Manager JavaServer Page framework important when developing user interfaces. You may also be interested in reading about menus and workflows as the their characteristics may influence how you go about developing the user interfaces of your solution.

How This Manual is Organized

This manual contains the following chapters:

- **Overview of Developing TSM**

  This chapter covers the basics of building your solution using TSM.

  It contains an introduction and overview of the following:

  - Configuring various components
  - Building a user interface
  - Deploying
  - Working with the CID

- **Extending the BLM Object Model**

  This chapter covers extending the definition of BLM objects.

  It contains information about:

  - Preparing your environment
  - Specifying the attribute
  - Extending the object definition
  - Working with the attribute

- **Working with Search Features**

  This chapter covers using the search feature in your solution.

  It contains information about:

  - Using the search feature
  - Working with search filters
  - Creating new search filters
  - Writing search JSPs

- **Changing the BLM Business Logic**

  This chapter covers changing the default business logic defined in the Business Logic Manager (BLM.)

  It contains information about:

  - Writing a new logic class
  - Compiling the new class
  - Integrating the new class

- **Managing Security**

  This chapter covers using the security features.

  It contains information about:

  - Configuring user authentication
  - Using the trust mode
  - Managing the access to BLM objects
  - Using explicit security

- **Accessing External Data Sources**

  This chapter covers using data sources other than the CID.

  It contains information about:

  - Configuring the components
  - Specifying the data binding
  - Programming access to the external data

- **Localizing Your Application**

  This chapter covers localizing your application.

  It contains information about:

  - Specifying the character set
  - Specifying the available languages
  - Specifying language-specific formats
  - Localizing CID entries
  - Localizing BLM error messages
  - Localizing JSPs

- **Managing Reference Data**

  This chapter covers managing Reference Data found in the CID.

  It contains information about:

  - The reference data
  - Returning the reference data
  - Reloading reference data

- **Managing Changes to BLM Objects**

  This chapter covers how to manage changes to objects.

  It contains information about:

  - Different ways of managing changes
  - Using the ActionManager to manage changes
  - Managing changes in synchronous mode

- **Working with Shopping Carts**

  This chapter covers using shopping carts in your solution.

  It contains information about:

  - Creating simple shopping carts
  - Managing complex shopping carts
  - Modifying items in a shopping cart
  - Displaying contents
  - Submitting the contents
  - Saving and restoring the content
  - Working with shopping cart templates

- **Using Bulk Ordering**

  This chapter covers using bulk ordering.

  It contains information about:

  - Modifying services of contracts
  - Removing services from contracts
  - Changing rateplans

- **Working with Approvals**

  This chapter covers using the approval feature to create an approval process for user requests.

  It contains information about:

  - Approval processes
  - Writing the approval class
  - Integrating the approval class

- **Managing Errors**

  This chapter covers managing BLM errors.

  It contains information about:

  - The different BLM exceptions
  - Customizing the messages

- **Logging Events**

  This chapter covers using the system logger.

  It contains information about:

  - Using the logger
  - The Logger API
  - Customizing the logger

- **Working with User Events**

  This chapter covers using user events.

  It contains information about:

  - Activating user events
  - Creating custom user events
  - Inserting events in JSPs

- **Deploying TSM**

  This chapter covers deploying TSM.

  It contains information about:

  - Configuring the environment

- Creating and deploying the WAR file
- Configuring deployed applications

What Typographical Changes and Symbols Mean

This manual uses the following conventions:

TYPEFACE	MEANING	EXAMPLE
Italics	Manuals, topics or other important items	Refer to *Developing Connectors*.
Small Capitals	Software and Component names	Your application uses a database called the CID.
Fixed Width	File names, commands, paths, and on screen commands	Go to `//home/my file`

Finding the Information You Need

The product suite comes with comprehensive documentation set that covers all aspects of building Account Management solutions. You should always read the release bulletin for late-breaking information.

Getting Started

If you are new to the edocs Telco Solutions, you should start by reading *Introducing Telco Service Manager*. This manual contains an overview of the various components along with a list of the available features. It introduces various concepts and components you must be familiar with before moving on to more specific documentation. Once you have finished, you can read the manual which covers different aspects of working with the application. At the beginning of each manual, you will find an introductory chapter which covers concepts and tasks.

Designing Your Solution

While reading *Introducing Telco Service Manager*, you should think about how the different components can address your Account Management Solution's needs.

You can refer to *Developing Telco Service Manager* for information about extending the object model, application security, and other design issues. The *CID Reference Guide* also gives you the information about how the information in your solution is managed and stored.

Installing Your Telco Service Manager

You should start by reading the Release Bulletin. For detailed installation and configuring information, refer to *Installing Telco Service Manager*. This manual covers installing TSM on one or more computers. It also contains the information you need to configure the different components you install. You might also refer to *Developing Telco Service Manager* and *Developing Connectors for Telco Service Manager* as these manuals contain information on customizing applications and working with other software.

Building Account Management Solutions

If you are designing and programming *Telco Service Manager*, you have several different sources of information. If you are programming the user interface of the solution, you should read *Developing User Interfaces for Telco Service Manager*. You also refer to the BLM Specification for detailed information about programming the user interface. For configuring the various components, you refer to *Installing Telco Service Manager*and sections in other documents which deal with the component to configure.

If you are working with the business logic of your solution, you should read *Developing Telco Service Manager*. You can also refer to the *BLM Reference Guide* for more information about the design and structure of the BLM object model. For information about how this information is stored, you should refer to the *CID Reference Guide* along with the CID Reference documentation for your database. In order to develop your application, you most likely will need to install and run the Loopback Connector. This component mimics back-end applications for development purposes. For information about installing and running this component, refer to *Using the Loopback Connector with Telco Service Manager*.

Integrating Account Management Solutions

If you are involved in configuring your solution to work with Operation Support Software (OSS), you should read *Developing Connectors with Telco Service Manager*. This manual helps you understand the integration architecture and shows you how to build connectors to connect to today's market-leading OSS software. You can also read *Using the Loopback Connector with Telco Service Manager* for information about a connector built for development purposes. Other manuals you can refer to for information about configuring your application include *Introducing Telco Service Manager* and *Developing Telco Service Manager*.

Managing Telco Service Manager (TSM)

If you are responsible for managing TSM, you should read *Installing Telco Service Manager* for information about configuring various components and information about working with different application servers. *Administrating Telco Service Manager* covers what you need to know about managing your solution at runtime. For information about OSS systems, you should read *Developing Connectors with Telco Service Manager*.

If You Need Help

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

To reach the U.S. Service Center, located in Natick, MA (Monday through Friday 8:00am to 8:00pm EST):

- Telephone: 508.652.8400
- Toll Free: 877.336.3362
- E-support: support.edocs.com (This requires a one-time online registration)
- E-mail: support@edocs.com

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

- What is your name and role in your organization?
- What is your company's name?
- What is your phone number and best times to call you?
- What is your e-mail address?
- In which edocs product did a problem occur?
- What is your Operating System version?
- What were you doing when the problem occurred?
- How did the system respond to the error?
- If the system generated a screen message, please send us that screen message.
- If the system wrote information to a log file, please send us that log file.

If the system crashed or hung, please tell us.

Contents

Preface iii

Overview of Developing TSM 19

About Developing Applications 20
Configuring the BLM 21
Building a User Interface 22
Developing Connectors 23
Deploying Your Solution 24

Extending the BLM Object Model 25

About Working with the CID 26
About Extending the BLM Object Model 27
Before You Start 28
Determining if the Object is Extensible 28
Adding Tables for Attribute Values 29
Declaring the Attributes in the DAL 30
Accessing Attribute Values 32
Modifying Attribute Values 32
Specifying the Attribute in the CID 35
Extending the Object Definition 37
Extending the Object Definition in the SmartLink Framework 39
Working With New Attributes 40
Returning the Object Description 40
Returning Values 40
Setting Values of Attributes 41
Setting Values of Objects Being Created or Modified 42

Working with Search Features 45

About Using the Search Feature 46
About the BLM Methods 46
About Search Filters 47
About Filter Criteria 48
Working with Search Filters 49
About Search Filters 49
Available Filters and Criteria 49
Configuring Search Filters 67
Customizing Filters 67
Customizing Criteria 68
Creating Search Filters 71
About Search Filters 71
About Filter Criteria 72
Overview of Creating Search Filters 73
Optimizing Oracle Databases For Searches 74

Declaring the New Filter in the CID 75
Declaring the New Search Method in the DAL 77
Writing the New Query 82
Writing a Search JSP 86
Getting the Filter 86
Getting Dynamic and Hidden Criteria 88
Displaying the Search Criteria 93
Executing the Search 95
Displaying the Results 98

Changing the BLM Business Logic 101

About Business Logic 102
Changing Business Logic 103

Managing Security 107

About Security 108
Understanding the CID Schema Security 109
Configuring Authentication 110
Using Trust Modes 111
Managing Access to BLM Objects 112
Using Explicit Security 113
About Explicit Security 113
Getting Managers of a Contract 113
Specifying a Manager 114
Removing a Manager 114

Accessing External Data Sources 115

About Accessing External Data Sources 116
Configuring a New DAL Instance 117
Creating the Configuration File 118
Specifying the Binding Properties 120
Programming the Data Access 121

Localizing Your Application 125

About Localizing an Application 126
Limitations of Localizing Applications 127
Specifying the Character Set 128
Specifying Application Languages 130
Specifying Language-specific Formats 131
Localizing Database Entries 133
Localizing BLM Error Messages 134
Localizing JSPs 136

Managing Reference Data 141

About Reference Data 142
Returning All Reference Data 143
Returning Only Certain Types of Reference Data 144
Reloading Reference Data 145
How the Internal BLM Cache Works 145
Updating Reference Objects in the Cache 145
Programming your Application for Reference Data Reloads 147
Example of JSPs Using the Reference Data Reload Feature 148

Managing Changes to BLM Objects 151

About Changes to BLM Objects 152
Managing Basic Changes to Objects 153
Managing Changes with the ActionManager 154
Managing Changes in Synchronous Mode 155

Working with Shopping Carts 157

About Shopping Carts 158
About the BLM Interfaces 159
Before Developing Shopping Carts 160
Action Manager Hierarchies 160
Action Manager Types 160
About the Presentation Layer 161
Creating a Simple Shopping Cart 162
Declaring and Retrieving the Shopping Cart 162
Listing Services in the Shopping Cart 163
Adding Services to the Shopping Cart 163
Submitting the Shopping Cart 163
Managing Complex Shopping Cart Contents 165
Creating a Complex Shopping Cart for Contracts 165

Creating Customers in the Shopping Cart 170
Modifying a Shopping Cart Item 174
Editing the Object 174
Modifying Additional Information 176
Modifying the Quantity 177
Modifying Service Parameters 177
Removing a Shopping Cart Item 178
Displaying the Contents of a Shopping Cart 180
Browsing the shopping cart 180
Displaying All Items 181
Working with core services 182
Retrieving the detail of an entry 182
Saving Shopping Carts 184
About Saving Shopping Carts 184
Saving a Shopping Cart 185
Create a Shopping Cart from a Saved Copy 186
Using Shopping Cart Templates 187
About Persistent Action Managers 187
About Shopping Cart Templates 188
Saving a Shopping Cart as a Template 188
Getting the List of Available Templates 189
Using Shopping Cart Templates 189
Deleting Shopping Cart Templates 190

Using Bulk Ordering 191

About Bulk Ordering 192
Adding a Service to Contracts 193
Modifying a Service of Contracts 194
Removing a Service from Contracts 195
Changing the Rate Plan of Contracts 196

Working with Approvals 197

About Approving Orders 198
Creating Approval Processes 199
Submitting Requests for Approval 200
Displaying Requests for Approval 201
Approving or Denying Requests 201
About Approval Processes Logic 203
Creating a New Approval Process Class 204
Deploying the Approval Class 209
Running the Approval Sequencer 210

Managing Errors 211

About the BLM Exceptions 212
BlmLogicException 212
BlmSecurityException 212
BlmBadValueException 212
PersistenceException 212

Customizing Error Messages 213

Logging Events 215

About Logging Events 216
Before You Get Started 216
Filtering Logs 218
About the Logger API 219
Available Events 219
Custom Event Codes 220
Logger API Object Model 221
Creating the Custom Event Code File 222
Creating the properties File 222
Example of the Custom Event Code File 222
Deploying the Custom Event Code File 223
Programming Custom Event Logs 224
Loading the Custom Event Code 224
Initalizing the Logger 225
Checking Severity Settings 226
Logging Standard Messages 227
Logging Debug Messages 229
Logging Messages During Development 231

Working with User Events 233

About User Events 234
About Creating Custom User Events 235

Deploying TSM 239

About Deploying 240
For WebSphere 4.x 241
Configuring Your Environment 241
Creating and Deploying a WAR File 242
Configuring Deployed Channels 245
Accessing a Deployed Channel 246
For WebLogic 6.x and 7.x 247
Configuring Your Environment 247
Creating and Deploying a WAR File 248
Accessing a Deployed Channel 251
For Oracle 9i Application Server 252
Configuring Your Environment 252
Creating and Deploying a WAR File 253
Configuring Deployed Channels 256
Accessing a Deployed Channel 258

Index 259

CHAPTER 1

Overview of Developing TSM

In This Section

About Developing Applications 20
Configuring the BLM 21
Building a User Interface 22
Developing Connectors 23
Deploying Your Solution 24

About Developing Applications

You use TSM to build your solution for your customer's needs in managing their relationship with communication service providers. The TSM comes with everything you need to build your solution straight out of the box. We have included all the tools and application frameworks you need to go online.

By their very nature, these applications are based on Internet standards and architectures. To manage the TSM and to ensure seamless integration, application servers manage basic application functions. The TSM is built using standard technologies, such as the powerful Java Server Page technology used to build user interfaces.

Before you start developing your solution, you should determine the scope along with the information you want to manage. You also need to determine the different kinds of users who use the application.

Developing TSM involves:

- Configuring the BLM
- Building the User Interface
- Developing Connectors
- Deploying your solution

Configuring the BLM

A key part of the CSS Engine, the Business Logic Manager (BLM) enables Communications Service Providers to rapidly implement an e-business solution tailored precisely to their operational requirements and business processes.

The BLM contains a set of Java packages that correspond to business objects you can manage in your application. The BLM also comes with a set of APIs you use in your JSPs to access and manipulate these objects.

To help you manage BLM objects, you configure:

- Access to BLM objects
- Authentication
- Display of rate plans
- Error handling

For detailed information about working with the BLM Engine, refer to the *BLM Reference Guide* and the *BLM API Reference Documentation.*

Building a User Interface

Developing any application from scratch is a long, tedious process that requires planning and lengthy development cycles. We have decided to help you reduce the time it takes to get your solution up and running.

The Java Server Page Framework (JSPF) is a set of JSP pages and Java classes that contain code you can use quickly and easily build TSM using the BLM. This framework presents a uniform and standardized way of writing and using JSPs. You can speed up your development of JSPs by using the framework to handle basic tasks and concentrate on coding the TSM's presentation layer.

The Personalization Manager comes with a set of channels that can be used as application templates:

- MyWeb - This template covers most of the essential features for users using the Internet.
- MyWAP - This Wireless Internet (WAP) Application template covers some core features for users of Wap-enabled handsets.
- MyIVR - This Interactive Voice Response (IVR) application template covers some core features of IVR-enabled applications.

You can also use these channels to help you write your own application. These fully documented sets of JSPs with reference workflows cover the basic features of any Account Management solution.

Building a user interface with the JSPF involves:

- Configuring name of your channel and location of its files
- Listing the JSP Pages that make up your channel
- Configuring security
- Localizing strings
- Specifying menu items
- Building Workflows

For detailed information about working with the Personalization Manager and the JSPF, refer to *Developing User Interfaces*.

Developing Connectors

The SmartLink Framework enables TSM to communicate, or integrate with the core CSP technology infrastructure, for example billing platforms, Customer Relationship Management (CRM) software, and other business and operational support systems (BSS/OSS). This in turn allows the CID to maintain a synchronized cache of semi-static customer and service data, as well as passing requests and responses between the various systems.

The SmartLink Framework can be considered to comprise:

- Connectors that move data between TSM and the transport layer. These connectors are called Synchronizers.
- Connectors that move data between BSS/OSS applications and the transport layer. These connectors are called BSS/OSS Connectors.

For more information, refer to *Developing Connectors*.

Deploying Your Solution

After developing your Account Management solution, you deploy the channels as a Web application. When you deploy your channel, you are telling the application server where to find your JSPs and components.

As each application server handles JSPs and other files differently, deploying your channel depends on the version and the editor of your application server. Deploying channels can be as easy as configuring your application server to look for the JSPs in a directory. Other application servers recommend that you deploy web applications as a J2EE Web Application aRchive (WAR) file.

For detailed information about configuring and deploying web applications, refer to your application server's documentation.

CHAPTER 2

Extending the BLM Object Model

In This Section

About Working with the CID 26
About Extending the BLM Object Model 27
Before You Start 28
Specifying the Attribute in the CID 35
Extending the Object Definition 37
Extending the Object Definition in the SmartLink Framework. 39
Working With New Attributes 40

About Working with the CID

When building and customizing TSM, you may have to change or add information in the CID. Most of the time, you work with product catalog and other reference data.

When working with the CID, please keep in mind that there are reserved ranges for object IDs. In general, your object IDs must be greater than 1000, but the reserved ranges vary from table to table.

Before adding any custom object to the CID and assigning it an ID, you should refer to the *Customizing the CID* section in the *CID Reference Guide* for more information about reserved ranges.

About Extending the BLM Object Model

The comprehensive BLM object model covers the fundamental features of customer self-service applications. Because it is, the object model may not fully cover the requirements of your solution and certainly not the evolving requirements for future enhancements.

However, you can customize the object model to meet your requirements for today and tomorrow.

You can extend the object model by adding attributes to the objects by using additional information. Additional information, or `add_info`, is an additional attribute of a core object.

Extending the BLM object model involves:

1. Before you start, you:
   - Determine if the object can be extended.
   - Determine if the CID has the required tables to store attribute values.
2. Specifying the attribute type in the CID. The attribute can be a string, boolean, date, and so on. You add the parameter descriptor in the CID.
3. Associate the attribute with the object description.
4. Extending the object definition in the SmartLink Framework Messages Schema Reference.
5. Programming access to the new attribute

Before You Start

1. Determining if the object can be extended. This involves verifying that the object can use additional information as object attributes.
2. Determining if the CID has the required tables to store attribute values. Most of the core BLM objects already have these tables.

   However, if the CID does not have the required tables, you must:

   - Create the tables
   - Configure the DAL to access and modify the information in these tables

Determining if the Object is Extensible

BLM objects that can be extended using additional information implement or inherit the BLM `AdditionalInfoMgrIF` interface.

The `RatePlanServiceIF` object is not extendable. The `getAdditionalParameters()` method returns additional attribute values of the related `ServiceIF` object.

To determine if the BLM object is extensible

1. Open the BLM API HTML documentation.
2. Click the `com.netonomy.blm.interfaces.util` link. The reference of this package opens.
3. In the *Interface Summary* list, click the `AdditionalInfoMgrIF` link. The reference of this interface opens.
4. Check the list of objects implementing or inheriting this interface under *All Known Subinterfaces.* If the object is listed, you can use additional information to extend the object.

Adding Tables for Attribute Values

Depending on the object you want to extend by using additional information, you may have to add tables to the CID to store the value of your object attributes. By default, most of the core objects already have the tables you need to store additional information.

The CID uses the following tables to store additional information.

- <OBJECT_NAME>_ADD_INFO
- <OBJECT_NAME>_ADD_INFO_VALUE
- <OBJECT_NAME>_ADD_INFO_LIST

Before you start, you need to verify if the object already has the tables in the CID. For more information, refer to the *CID Reference Guide* and the *CID Reference Documentation* for your database. If the object does not have these tables, you need to add the tables and configure the DAL.

Adding tables for attribute values involves:

- Verifying if the tables exist
- Creating the tables in the CID
- Configuring the access to these tables in the DAL

To create the tables in the CID

1 Create the following tables in the CID:

- <OBJECT_NAME>_ADD_INFO
- <OBJECT_NAME>_ADD_INFO_VALUE
- <OBJECT_NAME>_ADD_INFO_LIST

where <OBJECT_NAME> is the core object name

We suggest using the SQL already in crebas.sql as an example of how to add these tables. In this file , find the section where CONTRACT_ADD_INFO is created. This file is located in <home_dir>/data/cid/<db_name>/creation.

2 Do one of the following:

- For Oracle, create the SEQ_<OBJECT_NAME>_AI_VALUE sequence.
- For DB2, create the SEQ_<OBJECT_NAME>_AI_VALUE sequence. Be sure to use sequence objects.

  For more information about the sequence objects, refer to the DB2 release notes.
- For SQL Server, do the following:

1. Create a stored procedure.
2. Insert the name of the stored procedure in the KEYSTORAGE table.

Refer to the cre_sequences.sql file for more information about creating sequences for your database. This file is located in <home_dir>/data/cid/<db_name>/creation.

When you create the tables in the CID, you can define additional database structures for performance and integrity reasons. For example, you can create indexes and constraints.

Declaring the Attributes in the DAL

Declaring the attributes in the DAL involves:

- Declaring the new parameters in the DAL core_containers.xml file

  In this file, you enter the different elements used to find the appropriate data access methods to use.
- Declaring the object IDs in the DAL core_containers.xml and objectID_aliases.xml file

  In these files, you declare the internal ID of the object so your Account Management application can instantiate the object.

To declare the additional attributes in the DAL

1 Go to <home_dir>/classes/nmycfg/dal.

2 Open core_containers.xml.

3 Find the <object_name> element that defines the object to modify.

4 Under the <references> element, add an <addInfos> element. The following attributes you enter in this element also must be declared in the core_containers.xml file:

- internal_function the name of the element specifying the data access method
- type the declaration of the additional attributes

Example:

```
<references>

    <addInfos internal_function="getAddInfos" null="true"
type="DalContainer[]/objects.<OBJECT_NAME>ADDINFO" internal_name=""/>

    ...

</references>
```

5 Under the <internal_functions> element, add a <getAddinfos> element. This element is specified in the internal_function attribute of the <addInfos> element. This element has the query attribute which specifies the name of the SQL query in the core_queries.xml file.

Example:

```
<getAddInfos query="<OBJECT_NAME>.getAddInfos(this/object_ID)">
          <parameters/>
</getAddInfos>
```

6 6. Under the <objects> element, add the following child elements:

- <object_name>ADDINFO element. This element is specified in the type attribute of the <addInfos> element.
- <object_name>ADDINFOVALUE element

To create these elements and their content, you should use elements in core_containers.xml as a template. For your object copy the XML written for the <CONTRACTADDINFO> and <CONTRACTADDINFO> elements. You can then replace CONTRACT with the name of your object. You can then customize the content as required.

To declare the objectIDs of additional attributes in the DAL

1 Go to <home_dir>/classes/nmycfg/dal.

2 Open core_containers.xml.

3 Go to objects\ROOT\accessors.

4 Under the <data_accessors> element, add the following element:

```
<object_nameAddInfoValue internal_name="" type="DalContainer/objects.OBJECT_NAMEADDINFOVALUE" null="true"
internal_function="getObject_nameAddInfoValue"/>
```

5 Under the <internal_functions> element, add the following element:

```
<getObject_nameAddInfoValue query="object_nameaddinfovalue.get(parameter/id)">
  <parameters>
 <id internal_name="value_id" type="ObjectId/OBJECT_NAMEADDINFOVALUE" null="false" internal_function=""/>
  </parameters>
</getObject_nameAddInfoValue>
```

6 Save your changes and close the file.

7 Open objectID_aliases.xml.

8 Under the <XML_CONFIGURATOR> element, add the following element:

```
<OBJECT_NAMEADDINFOVALUE type="LONG" dal_name="objects.OBJECT_NAMEADDINFOVALUE"
name="object_nameAddInfoValue"/>
```

9 Save your changes.

Accessing Attribute Values

When declaring new parameters, you created the <getAddinfos> element as a child element of <internal_functions>. This query attribute of this element specified the name of the element containing the SQL query to use for this data access method.

You enter the elements and SQL in the DAL core_queries.xml file.

Accessing the attribute values involves:

- Entering the SQL of the declared get access method

To code the access to new attributes

1. Open the DAL core_queries.xml file corresponding to your database.
2. Find the <object_name> element that defines the queries for the object.
3. Under the <object_name>, create a <getAddInfos> child element which will contain the SQL to manage the additional information.
4. In the <getAddInfos> element, enter the SQL to access the attributes.
5. Under the <object_name>, create a <objectnameaddinfovalue> element then create the following child elements:
   - <get>
   - <getSubValues>
6. Enter the SQL in these elements.

   We suggest using the SQL already in the core_queries.xml file as a template for your SQL. For your object copy the SQL written for the CONTRACT <getAddInfos>,<get> and <getSubValues> elements. You can then replace CONTRACT with the name of your object. You can then customize the SQL if required.
7. Save your changes.

Modifying Attribute Values

Configuring the modification of attribute values involves:

- Declaring the data access methods to modify the attribute values
- Entering the SQL of these methods

To configure the DAL to modify attributes

1. Go to `<home_dir>/classes/nmycfg/dal`.
2. Open `core_container.xml`.
3. Find the tag `<object_name>` that defines the object to modify.
4. Add the `<updateAddInfos>` element.

   To create this element and its contents, you should use the element in `core_containers.xml` as a template. For your object, copy the XML written for the `<CONTRACT>` `<updateAddInfos>` element. You can then customize the content as required.
5. Find the `<object_nameADDINFO>` tag that defines the additional information to modify.
6. Add the following method accessors:

```
<data_accessors>

          <oldValue internal_name="" type="DalContainer/objects.OBJECT_NAMEADDINFO" null="true"
internal_function="getOld"/>

</data_accessors>

<method_accessors>

          <insert internal_name="" type="" internal_function="insert_cascade"/>

          <delete internal_name="" type="" internal_function="delete_cascade"/>

          <update internal_name="" type="" internal_function="update_cascade"/>

</method_accessors>
```

7. Add the internal functions.

   To add the internal functions, you should use the element in `core_containers.xml` as a template. For your object, copy the XML written for the `<CONTRACTADDINFO>` element. You can then customize the content as required.
8. Find the `<object_nameADDINFOVALUE>` tag that defines the additional information value to modify
9. Add the following method accessors:

```
<data_accessors>

          <newInternalID internal_name="seq_paramvalue" type="ObjectId/OBJECT_NAMEADDINFOVALUE"
null="true" internal_function="newInternalID"/>

</data_accessors>

<method_accessors>

          <insert internal_name="" type="" internal_function="insert_cascade"/>

          <insert_link internal_name="" type="" internal_function="insert_link"/>

          <delete internal_name="" type="" internal_function="delete_cascade"/>

          <delete_link internal_name="" type="" internal_function="delete_link"/>

</method_accessors>
```

10. Add the following internal functions:

11 To add the internal functions, you should use the element in `core_containers.xml` as a template. For your object, copy the XML written for the `<CONTRACTADDINFOVALUE>` element. You can then customize the content as required.

12 Save your changes.

To code the modification of the attributes

1 Open the DAL `core_queries.xml` file corresponding to your database.

2 Find the `<object_name>` element that defines the queries for the object.

3 Under the `<object_name>`, create an `<object_nameaddinfo>` element then create the following child elements:

- `<getOld>`
- <insert>
- <delete>

4 Enter the SQL in these elements.

5 Under the `<object_nameaddinfovalue>` element, create the following child elements:

- `<insert_link>`
- `<delete_link>`
- `<insert>`
- `<delete>`
- `<newInternal>`

6 Enter the SQL in these elements.

We suggest using the SQL already in the `core_queries.xml` file as a template for your SQL. For your object copy the SQL written for the corresponding `CONTRACT` elements. You can then replace `CONTRACT` with the name of your object. You can then customize the SQL if required.

---

7 Save your changes.

Specifying the Attribute in the CID

You need to specify the attribute in the CID. The PARAMETER table contains the attribute definition.

Specifying the attribute in the CID involves:

- Determining the type of parameter to use for the attribute.
- Add the attribute definition in the PARAMETER table.

For more information about the PARAMETER table, refer to the CID Reference Guide and the CID HTML reference documentation for your RDBMS.

To add an attribute to the CID

1. Refer to the CID Reference Guide to determine the type of parameter for your attribute.
2. In the PARAMETER table, add a record describing the new attribute. Use the syntax as shown in this example:

```
insert into PARAMETER ( PARAM_ID, PARAM_LEGACY_ID, PARAM_CODE,
PARAM_TYPE, PARAM_NAME, STRING_ID, PARAM_DESCRIPTION,
PARAM_DESC_STRING_ID, PARAM_SHORT_DESCRIPTION,
PARAM_SHORT_DESC_STRING_ID, PARAM_MIN_ITEMS, PARAM_MAX_ITEMS,
PARAM_MIN, PARAM_MAX, PARAM_PATTERN ) values ( PARAM_ID value,
PARAM_LEGACY_ID value, PARAM_CODE value... );
```

   - PARAM_ID: Internal ID of the parameter
   - PARAM_LEGACY_ID: Legacy ID of the parameter
   - PARAM_CODE: Code to manipulate the parameter
   - PARAM_NAME: Display name of the parameter
   - STRING_ID: String ID of the display name
   - PARAM_DESCRIPTION: Description of the parameter
   - PARAM_DESC_STRING_ID: String ID of the description name
   - PARAM_SHORT_DESCRIPTION: Short description of the parameter
   - PARAM_SHORT_DESC_STRING_ID:String ID of the short name
   - PARAM_MIN_ITEMS: Minimum number of items in case of a list
   - PARAM_MAX_ITEMS: Maximum number of items in case of a list
   - PARAM_MIN: Minimum value allowed for the parameter

- PARAM_MAX: Maximum value allowed for the parameter
- PARAM_PATTERN: The regular expression associated with the parameter

Extending the Object Definition

Once you have entered the attribute as a parameter in the CID, you need to extend the object definition of objects created using the BLM API.

The following objects can be created using the BLM API:

- `Organization`
- `Level`
- `Member`
- `Contract`
- `BillingAccount`

For these objects, extending the object definition involves:

- Defining extra parameters for requests creating the object (`create` requests)
- Defining the same extra parameters for requests modifying the object (`modify` requests)

After extending the object definitions, you can get the extended object definition dynamically. You can then display a page to enter all possible additional attributes of the object.

To define extra parameters in create requests

Use the `optParams` attribute of methods creating the object. You pass an array containing the parameter names and the corresponding values.

For example, the method for creating a contract:

```
LevelIF.createContract(... optParams,...)
```

For more information, refer to *Additional Parameters for a Request* in the *CID Reference Guide*.

To define extra parameters in modify requests

Use the `optParams` attribute of methods modifying the object. You pass an array containing the parameter names and the corresponding values.

For example, the method for modifying a contract:

```
ContractF.modifyContract(... optParams,...)
```

For more information, refer to *Additional Parameters for a Request* in the *CID Reference Guide*.

Extending the Object Definition in the SmartLink Framework

When extending objects with new attributes, you also need to add this information to messages used by the SmartLink Framework to communicate with OSS.

By default, the SmartLink Framework message schema supports the extension of all extendable objects. If you have added additional attributes to core BLM objects, the structure of these additional parameters is already defined. You do not have to define the structure, either for outbound messages or for inbound messages.

The additional attributes are located under the `<CUSTOM>` element of the message definition.

For more information about the SmartLink Framework Message Schema, refer to the *Working With the Message Schema Reference* section in *Developing Connectors.*

Working With New Attributes

After you extend the BLM object by configuring and entering additional information, you can now use a set of methods to work with the new object attributes in TSM.

Using the methods, you can:

- Return the object description
- Return the values of the attributes
- Set the value of the attributes using SQL or SmartLink Framework inbound messages
- Set the value of attributes of objects being created

Returning the Object Description

To return the extended object definition

Use the `ObjectRefMgr.getOptionalParameterDescriptors()` method and pass the ID of the action for creating the object (example: ID of create contract action) as the `action_id` parameter.

This method returns an array of `ParameterIF` corresponding to the extended object definition.

You can only use this method on objects which can be created using the BLM API. For more information about these objects, refer to *Extending the Object Definition* in this chapter.

Returning Values

You can return the attribute values of an object by using two different methods. For a specific object, you can return:

- All of the set attributes values
- The value of a specific set attribute
- All of the attributes and their corresponding values

To return all set additional attribute values

Use the `<objectname>.getAdditionalParameters()` method.

This method returns an array of `ParameterIF` corresponding to the additional parameters. Unset additional attributes are not returned by this method.

To return the value of a specific additional attribute

Use the `<objectname>.getAdditionalParameters()` method to return an array of corresponding to all of the set parameters of the object.

You can then find the parameter and its corresponding value in the array. If the attribute is not found, the value is not set.

To return all additional attributes and their values

1. Use the `<objectname>.getAdditionalParameters()` method to return an array of `ParameterIF` corresponding to the set additional parameters.
2. Use the `ObjectRefMgr.getOptionalParameterDescriptors()` method to return an array of `ParameterIF` corresponding to the additional parameters.
3. Merge both `ParameterIF` arrays to obtain the complete list of additional attributes and their values.

For more information about using the `getOptionalParameterDescriptors` method, refer to *Returning the Extended Object Description* in this chapter.

Setting Values of Attributes

You can set the values of attributes by using:

- The inbound SmartLink Framework message
- The appropriate BLM APIs

To set additional attributes using inbound SmartLink Framework messages

- Additional attributes values of customer data objects (Organization, Level, Member, Contract, Billing Account) are set in the <CUSTOM> section below the section for the customer data object being created or modified.

  Note about treatment of additional attributes in an inbound message: when setting the values related to additional attributes of an object being modified in the CID, only those defined in the message are changed. Other additional attributes linked to the object in the CID, but not found in the message, are not changed.

- The doSetAddInfo message contains modifications to the additional attributes of some of the extendable objects (to get the list, refer to section <setaddinfos> in the DoSetAddInfo message, in the SmartLink Framework schema reference).

  Although you can use this message to set additional attributes values of customer data objects, it is recommended to use the standard messages for creating/modifying them (DoAddContract, DoModifyContract…)

  This message is particularly useful if you need to modify additional attributes values of “Business Reference” objects (Service…), based on specific events at the back-end side.

  Ex: you can update the remaining numbers of handsets (additional attribute of “handset” service) from the DoAddService message.

- If you need the previous feature, but in the same transaction as the treatment of a standard inbound message:

  The inbound messages come with a special section (<setaddinfos> section) that deals with the additional attributes of some of the extendable objects (to get the list, refer to section <setaddinfos> in any DO message, in the SmartLink Framework schema reference). This dedicated section manages updating additional attributes of the object.

  The additional attributes in this section must not be related to the other objects being modified. This feature allows you to update additional attributes of unrelated objects when your application requires an update in a single transaction.

Setting Values of Objects Being Created or Modified

You can set the additional attribute values of objects being created.

Depending on how you are creating the object, you use:

- Standard methods
- `ActionItemIF` methods

  These methods are for setting attributes of objects in shopping carts.

To set additional information of an object being created

1. Get the additional attributes of the object. These additional attributes are stored in a `ParameterIF[]` array.

   Refer to *To get the extended object definition*.

2. Set the values for each additional attribute in this array.

   Refer to the BLM R*eference Docum*entation for information about setting the values of `ValueChoiceIF`, `ValueCompositeIF`, `ValueDynamicIF`, `ValueListIF`, `ValueSimpleIF`.

3. Pass this array as the `optParams` parameter of the method creating the object.

   For example, the method for creating a contract:

   ```
   LevelIF.createContract(... optParams,...)
   ```

To set additional information of an object being modified

1. Get the additional attributes of the object. These additional attributes are stored in a `ParameterIF[]` array.

   Refer to *To get the extended object definition*.

2. Set the values for each additional attribute in this array.

   Refer to the BLM R*eference Docum*entation for information about setting the values of `ValueChoiceIF`, `ValueCompositeIF`, `ValueDynamicIF`, `ValueListIF`, `ValueSimpleIF`.

3. Pass this array as the `optParams` parameter of the method modifying the object.

   For example, the method for modifying a contract:

   ```
   ContractF.modifyContract(... optParams,...)
   ```

To set additional information in a shopping cart

1. Use the `ActionItemIF.getOptParameters()` method to get the values of the additional attributes. These additional attributes are stored in a `ParameterIF[]` array.

2. Set the values for each additional attribute in this array.

   Refer to the BLM R*eference Docum*entation for information about setting the values of `ValueChoiceIF`, `ValueCompositeIF`, `ValueDynamicIF`, `ValueListIF`, `ValueSimpleIF`.

- Set the optional parameters by passing this array as the `optParams` parameter of the `ActionItemIF.setOptParameters(ParameterIF[] optParams)` method.

CHAPTER 3

Working with Search Features

In This Section

About Using the Search Feature 46
Working with Search Filters 49
Configuring Search Filters 67
Creating Search Filters 71
Writing a Search JSP 86

About Using the Search Feature

You can use the powerful search feature to find the information you are looking for easily and quickly.

This feature uses a set of specific APIs to program the search and display of results. There is also a large set of predefined search filters that let you specify the criteria to use and their default values.

This section covers the technical details of the search feature and the different APIs and filters you use to build search pages for your application. The example covers a simple JSP search page with features such as dynamic and hidden criteria and limiting the number of results returned. This document also covers modifying search filters.

About the BLM Methods

When creating searches, you use the following types of methods:

- `findBy<attributes>` You use this type of API for simple searches when you know the attribute to use as a filter or when search filters are too complicated to implement. For example, you can use the `findRatePlansByCode` method to quickly return a list of rate plans having a specific code pattern. You cannot customize the behavior of this type of search.
- `find<object>ByFilter` You use these methods to create multiple criteria searches for specific objects. For example, you can use the `findRatePlansByFilter(filter)` method to use the `RatePlanFilter` with the following search criteria:
  - Contract Level type
  - Contract Type
  - Rate plans for contracts only
  - Line Type
  - Offer Type
  - Organization Type
  - Scoring Value

  You cannot customize the behavior of this type of search.

- `<object>.search` You use this method to create powerful search features. For example, if you need to search members in an organization or use additional information as search criteria. To customize these searches, you create new search filters.

These methods return arrays of objects found matching the given search criteria. You can also use the search feature to limit the number of objects returned. This means that once the search returns the specified number of objects, you can warn users that the limit has been reached and that they should refine their search.

About Search Filters

There is a set of predefined search filters you can use to search and find specific objects. These filters allow you to dynamically determine the criteria of a search. Filters also help ease the development of search pages or other search features found in the presentation layer.

Filters are located in the `SEARCH_FILTER` table in the CID.

Filters have the following settings you can modify:

- Maximum number of objects to return
- `IS_DEFAULT` to specify if the filter is the default for the object type
- `START_DATE` of the filter
- `END_DATE` of the filter
- One or more filter criteria

About Filter Criteria

A filter is made up of one or more filter criteria. These criteria are parameters. This means you can manage search criteria the same way you manage other parameters.

The following basic types of criteria:

- Normal criteria associated with simple parameters(integer, string, and so on) or choice parameters
- Dynamic criteria associated with dynamic parameters.

Search criteria can be displayed for user input or hidden. Hidden criteria are not displayed but are required for the search. For example, your JSP may enter the user's level when searching for contracts by level. Users are not required to enter their own level, but the level is required by the filter.

Filter criteria are located in the `SEARCH_CRITERIA` table.

Filter criteria have the following settings you can modify:

- `IS_ACTIVE` to specify if the criteria is active
- `IS_MANDATORY` to specify if the criteria is mandatory
- `DEFAULT_VALUE` to specify the default value
- `DISPLAY_ORDER` to specify the display order

Working with Search Filters

About Search Filters

There is a set of predefined search filters you can use to search and find specific objects. These filters allow you to dynamically determine the criteria of a search. Filters also help ease the development of search pages or other search features found in the presentation layer.

You can specify default filters for the following objects:

- Organization
- Level
- Member
- Contract
- Billing Account
- User Event
- Persistent Action Manager
- Request
- Service

Filters are located in the `SEARCH_FILTER` table in the CID. Filters have the following settings:

- Maximum number of objects to return
- `IS_DEFAULT` to specify if the filter is the default for the object type
- `START_DATE` of the filter
- `END_DATE` of the filter
- One or more filter criteria

Available Filters and Criteria

Filter Criteria

CODE	TYPE	NAME	DESCRIPTION	PARAMETER VALUE MIN-MAX
CORE_C_REFNUMBER	String	Reference number	Customer reference number	0-100
CORE_C_LCONTACTFNAME	String	First name	Legal contact first name	0-40

CODE	TYPE	NAME	DESCRIPTION	PARAMETER VALUE MIN-MAX
CORE_C_LCONTACTLNAME	String	Last name	Legal contact last name	0-40
CORE_C_LCOMPANYNAME	String	Company name	Legal contact company name	0-50
CORE_C_LSTREETNUMBER	String	Street number	Legal contact street number	0-10
CORE_C_LSTREETADDRESS	String	Street Address	Legal contact street address	0-70
CORE_C_LZIP_CODE	String	Zip code	Legal contact Zip Code	0-15
CORE_C_LCITY	String	City	Legal contact city name	0-40
CORE_C_LSTATE	String	State	Legal contact state name	0-25
CORE_C_SCOREVALUEMIN	Integer	Scoring value min	Scoring value Min	0-
CORE_C_SCOREVALUEMAX	Integer	Scoring value max	Scoring value Max	0-
CORE_C_LINENUMBER	String	Line number	Contract line number	0-50
CORE_C_ORGTYPE	Dynamic	Organization type	Organization type	0-
CORE_C_ORGID	Dynamic	Organization	Organization id	0-1
CORE_C_CONTRACTTYPE	Dynamic	Contract type	Contract type id	0-
CORE_C_CONTRACTRP	Dynamic	Contract rate plans	Contract rate plans	0-10
CORE_C_CONTRACTSRV	Dynamic	Contract services	Contract services	0-10
CORE_C_CONTRACTSTATUS	Dynamic	Contract statuses	Contract statuses	0-
CORE_C_SEARCHINSUBLEVELS	Boolean	All levels below	Search in all sub levels below the current one	--

CODE	TYPE	NAME	DESCRIPTION	PARAMETER VALUE MIN-MAX
CORE_C_LOGIN	String	Login	Login	0-50
CORE_C_ROLES	Dynamic	Roles	Roles	0-
CORE_C_PAYMENTMETH	Dynamic	Payment method	Payment method	
CORE_C_LLEVELNAME	String	Level name	Legal level name	0-50
CORE_C_PAMGENERERATEDBY	Dynamic	Generated by	User who created the persistent action manager	0-1
CORE_C_PAMCATEGORY	Dynamic	Persistent Action Manager Category	Persistent Action Manager Category	0-
CORE_C_CONTRACTSID	Dynamic	Contracts list	List of contract ids	0-
CORE_C_UEVTTYPECATEGORY	Dynamic	User event type category	List of user event type category codes	0-
CORE_C_UEVTDATEMIN	Date	User event date min	User event min date	0-1
CORE_C_UEVTDATEMAX	Date	User event date max	User event max date	0-1
CORE_C_UEVTCREATEDOBJECTID	Dynamic	User event created Object	User events created object ID	0-1
CORE_C_UEVTCREATEDOBJECTTYPEID	Dynamic	User event created Object type	User events created object type ID	0-1
CORE_C_UEVTIMPACTEDOBJECTID	Dynamic	User event impacted object	User event impacted object ID	0-1
CORE_C_UEVTIMPACTEDOBJECTTYPEID	Dynamic	User event impacted object type	User event impacted object type ID	0-1
CORE_C_UEVTCREATEDOBJECTTYPEIDTOFILTER	Dynamic	User event created object type id to filter	User event created object type id to filter	0-
CORE_C_APPROVALMEMBERID	Dynamic	Member approver	Approval member approver	
CORE_C_APPROVALREFERENCE	String	Approval reference	Approval reference	0-255
CORE_C_INVOICENUMBER	String	Invoice Name	Invoice name	0-100

CODE	TYPE	NAME	DESCRIPTION	PARAMETER VALUE MIN-MAX
CORE_C_BILLPERIODIDS	Dynamic	Bill period list	List of bill period IDs	0-
CORE_C_CONTRACTID	Dynamic	Contract	Contract ID	0-1
CORE_C_INVOICEID	Dynamic	Invoice	Invoice ID	0-1
CORE_C_ORGVIEWNODEID	Dynamic	Organizationview ID	Organization view	0-1
CORE_C_SEARCHINSUBORGVIEWLEVELS	Boolean	All levels below	Search in all sub organization view levels below the current one	
CORE_C_MEMBERID	Dynamic	Member ID	Member	0-1
CORE_C_REQUESTDATEMIN	Date	Request date min	Minimum request creation date	
CORE_C_REQUESTDATEMAX	Date	Request date max	Maximum request creation date	
CORE_C_REQUESTSTATUS	Dynamic	Request statuses	Request statuses	0-

Bill Period Filters

All Bill Periods of Organization Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_INTERORG_BILLPERIOD	Bill Periods	Search bill periods associated with invoices below an organization or level	1	50	-Bill Cycle ID -Start Date

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_ORGID	1	1	-1	Equal	None
CORE_C_SEARCHINSUBLEVELS	1	1	-1	Equal	TRUE

Billing Account Filters

All Billing Accounts By Payment Method Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_ BACCOUNTBYPAYMETH	Billing accounts by payment method	Search billing account by their payment method in all organizations	0	20	

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_ORGTYPE	1	0	1	In	None
CORE_C_PAYMENTMETH	1	0	2	In	None

Billing Accounts By Payment Method in Organization Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_INTORG_ BACCOUNTBYPAYMETH	Billing accounts by payment method	Search billing account by their payment method in all organizations	0	20	

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_ORGID	1	0	1	In	None
CORE_C_PAYMENTMETH	1	0	2	In	None
CORE_C_SEARCHINSUBLEVELS	1	1	-1	Equal	TRUE

Bulk Ordering Filters

Modifiable Contracted Services by Contracts Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_SERVICEMODIFIABLE	Modifiable contracted services by contracts	Search modifiable services contracted in a list of contracts	0	30	

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_CONTRACTSID	1	1	1	In	None

Modifiable Contracted Services by Organization Contracts Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_INTORG_SERVICEMODIFIABLE	Modifiable contracted services by organization contracts	Search modifiable services contracted in a contracts below an organization or level	0	30	

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_ORGID	1	1	-1	Equal	None
CORE_C_SEARCHINSUBLEVELS	1	1	-1	Equal	None

Removable Contracted Services by Contracts Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_SERVICEREMOVABLE	Removable contracted services by contracts	Search removable services contracted in a list of contracts	0	30	

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_CONTRACTSID	1	1	-1	In	None

Removable Contracted Cervices by Organization Contracts Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_INTORG_SERVICEREMOVABLE	Removable contracted services by organization contracts	Search removable services contracted in a contracts below an organization or level	0	30	

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_ORGID	1	1	-1	Equal	None
CORE_C_SEARCHINSUBLEVELS	1	0	2	Equal	None

Contract Filters

Contract By Line Number Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_CONTRACTBYLINENUMBER	Contract by line number	Search contracts by their line number in all organizations	0	1	

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_LINENUMBER	1	1	1	Equal	None
CORE_C_SEARCHINSUBLEVELS	1	1	-1	Equal	None

Contract By Line Number in Organization Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_INTORG_CONTRACTBYLINENUMBER	Contract by line number in Organization filter	Search contracts by their line number in a specific company or level	0	1	

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_ORGID	1	1	-1	Equal	None
CORE_C_LINENUMBER	1	1	1	Equal	None
CORE_C_SEARCHINSUBLEVELS	1	1	-1	Equal	TRUE

Managed Contracts Advanced Search Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_MANAGEDBY_CONTRACTADVANCEDSEARCH	Managed Contracts advanced search	Search contracts managed by a specific user by contract type, status, rate plans, services	0	20	- Line Number

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_MEMBERID	1	1	-1	Equal	None
CORE_C_CONTRACTTYPE	1	0	1	in	None
CORE_C_CONTRACTRP	1	0	2	in	None
CORE_C_CONTRACTSRV	1	0	3	in	None
CORE_C_CONTRACTSTATUS	1	0	4	in	None

Contracts Advanced Search Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_INTORG_CONTRACTADVANCEDSEARCH	Contract advanced search	Search contracts by contract type, rate plans, services in a specific company or level	0	20	-Level - Line Number

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_ORGID	1	1	-1	Equal	None
CORE_C_CONTRACTTYPE	1	0	1	in	None
CORE_C_CONTRACTRP	1	0	2	in	None
CORE_C_CONTRACTSP	1	0	3	in	None
CORE_C_CONTRACTSTATUS	1	0	4	in	None
CORE_C_SEARCHINSLEVELS	1	1	5	Equal	TRUE

Billing Accounts by Payment Method in Organization Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_INTORGVIEW_CONTRACTBYLINENUMBER	Contracts in organization view by line number	Search contracts in a specific organization view by contracts line number	0	1	- Line Number

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_ORGVIEWNODEID	1	1	-1	Equal	None
CORE_C_LINENUMBER	1	1	1	Equal	None
CORE_C_SEARCHINSUBORGVIEWLEVELS	1	1	2	Equal	TRUE

Contracts in Organization View Advanced Search Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_INTORGVIEW_CONTRACTADVANCEDSEARCH	Contracts in organization view advanced search	Search contracts in a specific organization view by contract type, status, rate plans, services	0	20	- Line Number

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_ORGVIEWNODEID	1	1	-1	Equal	None
CORE_C_CONTRACTTYPE	1	0	1	in	None
CORE_C_CONTRACTRP	1	0	2	in	None
CORE_C_CONTRACTSRV	1	0	3	in	None
CORE_C_CONTRACTSTATUS	1	0	4	in	None
CORE_C_SEARCHINSUBORGVIEWLEVELS	1	1	5	Equal	TRUE

Managed Contracts By Line Number Search Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_MANAGEDBY_CONTRACTBYLINENUMBER	Managed Contracts by line number	Search contracts managed by a specific user by contract line number	0	1	- Line Number

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_MEMBERID	1	1	-1	Equal	None
CORE_C_LINENUMBER	1	1	1	Equal	None

Invoice Filters

Invoice By Number Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_INVOICEBYNUMBER	Invoice by number	Search invoice by its number	0	1	

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_INVOICENUMBER	1	1	1	Equal	None

Invoice By Bill Period Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_INTORG_INVOICEBYBILLPERIOD	Invoice by bill period	Search invoices by bill period in a specific organization or level	0	20	-MainInvoice creation date desc -Billing account ID

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_ORGID	1	1	-1	Equal	None
CORE_C_SEARCHINSUBLEVELS	1	1	-1	Equal	TRUE
CORE_C_BILLPERIODIDS	1	0	1	in	None

Level Filters

Level By Contact Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_INTORG_ LEVELBYCONTACT	Level by contact	Search levels by their legal contact in a specified organization or level	0	20	- Level - Name

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_ORGID	1	1	-1	Equal	None
CORE_C_REFNUMBER	1	1	1	Like	None
CORE_C_ISRECURSIVE (Case insensitive)	1	1	-1	Equal	TRUE

Level By Reference Number

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_INTORG_ LEVELBYREFNUMBER	Level by reference number	Search levels by their legal contact in a specified organization or level	0	20	- Level - Name

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_ORGID	1	1	-1	Equal	None
CORE_C_REFNUMNUMBER	1	1	-1	Equal	None
CORE_C_SEARCHINSLEVELS (Case insensitive)	1	1	1	Equal	TRUE

Member Filters

Members By Contact Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_MEMBERBYCONTRACT	Members by contact	Search members by their legal contact in all organizations	0	20	- Last Name - First Name

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_ORGTYPED	1	0	1	In	None
CORE_C_LCONTACTLNAME (case insensitive)	1	1	2	Like	None
CORE_C_LCONTACTFNAME (case insensitive)	1	0	3	Like	None

Members By Login Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_MEMBERBYLOGIN	Member by login	Search members by their login in all organizations	0	1	

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_LOGIN (case insensitive)	1	1	1	Equal	None

Member By Contact in Organization Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_INTORG_ MEMBERBYCONTACT	Member by contact	Search members by their legal contact in a specified organization or level	0	20	- Level - Last Name - First Name

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_ORGID	1	1	-1	Equal	None

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_LCONTACTLNAME (case insensitive)	1	1	1	Like	None
CORE_C_LCONTACTFNAME (case insensitive)	1	0	2	Like	TRUE
CORE_C_SEARCHINSUBLEVELS	1	0	-1	Equal	TRUE

Member by Login in Organization Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_INTORG_ MEMBERBYLOGIN	Member by login	Search members by their login in a specified organization or level	0	20	- Login

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_ORGID	1	1	-1	Equal	None
CORE_C_LOGIN	1	1	1	Equal	None
CORE_C_SEARCHINSLEVELS	1	1	-1	Equal	TRUE

Member by Role in Organization Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_INTORG_ MEMBERBYROLES	Member by roles	Search members by their roles in a specified organization or level	0	20	- Level - Last Name - First Name

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_ORGID	1	1	-1	Equal	None
CORE_C_ROLES	1	1	1	in	None
CORE_C_SEARCHINSLEVELS	1	1	2	Equal	TRUE

Organization Filters

Organization By Reference Number Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_ORGBYREFNUMBER	Organization by reference number	Search organizations by their reference number	0	1	

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_REFNUMBER (Case insensitive)	1	1	1	Equal	None

Company By Contact Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_BUORGBYCONTACT	Company by contact	Search companies by their legal contact attributes	0	20	- Company name

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_ORGTYPE	1	1	-1	In	None
CORE_C_LCOMPANYNAME (cace insensitive)	1	1	1	Like	None
CORE_C_LSTREETNUMBER (cace insensitive)	0	0	2	Equal	None
CORE_C_LSTREETADDRESS (cace insensitive)	0	0	3	Like	None
CORE_C_LZIPCODE (cace insensitive)	0	0	4	Like	None
CORE_C_LCITY (cace insensitive)	0	0	5	Like	None
CORE_C_LSTATE (cace insensitive)	0	0	6	Like	None
CORE_C_SCOREVALUEMIN	0	0	7	>	None
CORE_C_SCOREVALUEMAX	0	0	8	<	None

Consumer By Contact Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_RESORGBYCONTACT	Consumer by contact	Search consumers by their legal contact attributes	0	20	- Last Name - First Name

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_LCONTACTLNAME (cace insensitive)	1	1	-1	Like	None
CORE_C_LCONTACTFNAME (cace insensitive)	1	0	2	Like	None
CORE_C_LSTREETNUMBER (cace insensitive)	0	0	3	Equal	None
CORE_C_LSTREETADDRESS (cace insensitive)	0	0	4	Like	None
CORE_C_LZIPCODE (cace insensitive)	0	0	5	Like	None
CORE_C_LCITY (cace insensitive)	0	0	6	Like	None
CORE_C_LSTATE (cace insensitive)	0	0	7	Like	None
CORE_C_SCOREVALUEMIN	0	0	8	>	None
CORE_C_SCOREVALUEMAX	0	0	9	<	None

Customer By Line Number Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_ORGBYLINENUMBER	Customer by line number	Search customer by their contract line number	0	1	

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_LINENUMBER	1	1	1	Equal	None
CORE_C_SEARCHINSUBLEVELS	1	1	-1	Equal	True

Order Validation Filters

Requests By Approval Member Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_REQUESTBYAPPROVALMEMBER	Request by approval member	Search request having an approval process to approve for a specific member	0	1-10	-Request date (oldest date first)

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_APPROVALMEMBERID	1	1	-1	Equal	None

Requests By Approval Reference Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_REQUESTBYAPPROVALREFERENCE	Request by approval reference	Search request having an approval process to approve for a specific reference	0	1-10	- Request date (oldest date first)

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_APPROVALREFERENCE	1	1	-1	Equal	None

Persistent Action Manager Filters

Persistent Action Manager By Category Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_PAMBYCATEGORY	Action manager by category	Search action manager by category	1	1	- Name - Date

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_PAMGENERERATEDBY	1	1	-1	Equal	None
CORE_C_PAMCATEGORY	1	1	-1	In	

Request Filters

Requests Impacting Managed Contracts Search

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_REQUESTSIMPACTINGMANAGEDCONTRACTS	Requests impacting managed contracts search	Search requests impacting explicitly managed contracts	0	1-10	- Request creation date (asc)

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_MEMBERID	1	1	-1	Equal	None
CORE_C_REQUESTDATEMIN	1	1	-1	>=	None
CORE_C_REQUESTDATEMAX	1	0	2	<=	None
CORE_C_REQUESTSTATUS	1	0	3	In	None

Subinvoice Filters

Subinvoice By Contract Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_SUBINVOICE_BYCONTRACT	Subinvoice by contract	Search sub-invoices related to a specific contract	0	20	- Main Invoice - Creation Date

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_CONTRACTID	1	1	-1	Equal	None
CORE_C_INVOICEID	1	0	-1	Equal	None

User Event Filters

User Events By Object Filter

Filter Properties

FILTER CODE	NAME	DESCRIPTION	DEFAULT	ROW	ORDER
CORE_UEVTSBYIMPACTEDORCREATEDOBJECT	User events by impacted or created object	Search user events impacting or creating an object of a business organization	0	50	- Creation Date

Filter Criteria

PARAMETER CODE	A	M	O	OPERATOR	DEFAULT VALUE
CORE_C_UEVTIMPACTEDOBJECTTYPEID	1	0	-1	Equal	None
CORE_C_UEVTIMPACTEDOBJECTID	1	0	-1	Equal	None
CORE_C_UEVTCREATEDOBJECTTYPEID	1	0	-1	Equal	None
CORE_C_UEVTCREATEDOBJECTID	1	0	-1	Equal	None
CORE_C_UEVTTYPECATEGORY	1	0	-1	In	None
CORE_C_UEVTDATEMIN	1	0	1	<	None
CORE_C_UEVTDATEMAX	0	0	2	>	None
CORE_C_UEVTCREATEDOBJECTTYPEIDTOFILTER	1	0	-1	Not In	

`CORE_C_UEVTCREATEDOBJECTTYPEIDTOFILTER` can be used to search all user events impacting a level without the ones creating a contract, a member or a sublevel

Configuring Search Filters

You can configure the properties of search filters.

For each filter, you can:

- Declare the filter as the default filter
- Specify the maximum number of items returned
- Specify the following criteria parameter properties for each filter:
  - Active
  - Mandatory
  - Display order
  - Default value

For each Criteria Parameter, you can specify:

- The Min Max value of the parameter

Customizing Filters

To declare the filter as default

1. Use your database tool to connect to the CID.
2. In the `SEARCH_FILTER` table, find the search filter you want to modify.
3. In the `IS_DEFAULT` column, enter 1 to declare the filter as the default.
4. Set the other values in the `IS_DEFAULT` column to 0. This is only for filters on the same type of object.
5. Save your changes.

To set the number of items returned

1. Use your database tool to connect to the CID.
2. In the `SEARCH_FILTER` table, find the search filter you want to modify.
3. In the `ROW_COUNT` column, enter the maximum number of items to return.
4. Save your changes.

To activate a filter criteria parameter

1. Use your database tool to connect to the CID.
2. In the `SEARCH_CRITERIA` table, find the criteria parameter you want to modify.

3 In the IS_ACTIVE column, enter the one of the following:
   - 1 to activate the criteria parameter
   - 0 to deactivate the criteria parameter
4 Save your changes.

To declare the criteria as mandatory

1 Use your database tool to connect to the CID.
2 In the SEARCH_CRITERIA table, find the criteria parameter you want to modify.
3 In the IS_MANDATORY column, enter the one of the following:
   - 1 to make the criteria parameter mandatory
   - 0 to make the criteria parameter optional
4 Save your changes.

To set the display order of the filter criteria

1 Use your database tool to connect to the CID.
2 In the SEARCH_CRITERIA table, find the criteria of the filter.
3 For each criteria, do the following:
   - Enter the numbers corresponding to the order the criteria are displayed
   - Enter -1 to specify the criteria as a hidden system parameter
4 Save your changes.

To set the default value of the filter criteria

1 Use your database tool to connect to the CID.
2 In the SEARCH_CRITERIA table, find the criterion of the filter.
3 In the DEFAULT_ID_VALUE column, enter the ID of the default object value.
4 Save your changes.

Customizing Criteria

Filter Criteria are parameters. Like all parameters, filter criteria have different settings you can change. For search criteria, you can change the minimum and maximum value of the criteria parameter.

The Search Criteria parameters are in the FILTER_CRITERIA table.

To customize a criterion

1. Use your database tool to connect to the CID.
2. In the `PARAMETER` table, find the criteria parameter you want to modify.
3. In the `PARAM_MIN` column, enter the minimum number of values.
4. In the `PARAM_MAX` column, enter the maximum number of values
5. Save your changes.

CHAPTER 4

Creating Search Filters

In This Section

About Search Filters 47
About Filter Criteria.....72
Overview of Creating Search Filters.....73
Optimizing Oracle Databases For Searches 74
Declaring the New Filter in the CID 75
Declaring the New Search Method in the DAL 77
Writing the New Query.....82

About Search Filters

There is a set of predefined search filters you can use to search and find specific objects. These filters allow you to dynamically determine the criteria of a search. Filters also help ease the development of search pages or other search features found in the presentation layer.

Filters are located in the `SEARCH_FILTER` table in the CID.

Filters have the following settings you can modify:

- Maximum number of objects to return
- `IS_DEFAULT` to specify if the filter is the default for the object type
- `START_DATE` of the filter
- `END_DATE` of the filter
- One or more filter criteria

About Filter Criteria

A filter criterion is a parameter with the following characteristics:

- IS_ACTIVE Flag: specifies if the criteria must be used (allows customization without modifying the query)
- IS_MANDATORY flag: specifies if the criteria must be filled
- DISPLAY_ORDER: specifies the order to display criteria
- DEFAULT_VALUE: specifies the default value of the criteria
- DAL_PARAMETER_NAME: specifies the DAL parameter for mapping.

You must first declare you criteria as parameter.

For more information about declaring new parameters, refer to the *CID Reference Guide*.

Dynamic Criteria

If you want to use a criterion for which the value is depending on a reference table or on the user context, you must declare a dynamic parameter and set its value specifically in the JSP (example: contract statuses if you want to perform a search by contract status)

You can also use the core criteria if they match your requirements. The names of the parameters used as criteria in core filters begin with CORE_C.

Special Criteria

You can also use special criteria in your filters. These special criteria are for filters of objects that can be part of a hierarchy. For instance, you can use the organization id as a filter criterion for a level. You then use another criteria to specify if search also applies to the sublevels of the level. Because of this, these special criteria come in pairs and are used together.

The organization id criteria pair:

- CORE_C_ORGID: Use this dynamic criterion to search below a specific organization or level. In the JSP, you set the value of this criterion using the organization id or the level id.
- CORE_C_SEARCHINSUBLEVELS: Use this criterion to specify if the search must be done only inside or in all sub levels of the organization (or level) specified in the CORE_C_ORGID criteria.

The organization view criteria pair:

- CORE_C_ORGVIEWNODEID: Use this dynamic criteria to search below a specific organization view node. In the JSP, you set the value of this criterion using the organization view node id.
- CORE_C_SEARCHINSUBORGVIEWLEVELS: Use this criterion to specify if the search must be done only inside or in all sub organization view levels of the organization view node specified in the CORE_C_ORGVIEWNODEID criteria.

When working with these special criteria, you must use them together. For instance, when creating a filter for organization ids, you use the CORE_C_ORGID and CORE_C_SEARCHINSUBLEVELS criteria together.

Overview of Creating Search Filters

Creating a search filter involves:

- Optimizing your database for searches
- Declaring the new filter in the CID
- Declaring the new DAL search method
- Writing and declaring the new query to perform the search

The examples show you how to create a search function to find contracts by their SIM card number.

For Oracle, make sure your database is configured to use function based indexes. This helps optimize the response time of your searches. For more information, refer to *Installing Telco Service Manager*.

Optimizing Oracle Databases For Searches

Using the search feature with filter criteria may have a serious impact on your application performance.

The CID database model comes with indexes on some database columns, but not on all of them. Because indexes consume a lot of system resources, only data that is accessed by the application is indexed. When creating search filters and searching information in the CID, this lack of some indexes may cause unsatisfactory response times.

In order to improve response time, we strongly recommend defining indexes on the database column your filter criteria is based on.

With Oracle databases, you may have a problem creating indexes on columns that contain string values.

In the CID database model, string values are handled as `VARCHAR2(4000)`, but Oracle databases do not support creating indexes on `VARCHAR2(4000)` data. The maximum size the Oracle database indexes support (around 3000) is smaller that the maximum size for `VARCHAR2`. The exact maximum supported size for indexing depends on your Oracle instance and system.

For more information about Oracle and indexing, refer to your Oracle documentation.

To create indexes for searches

1. Make sure the column to index does not contain any data.

   This change must be done while designing your application and before you load any application or customer data.

   ---

2. Define the maximum length of your string data.
3. Connect to the CID database model as the database schema owner.
4. Use the `ALTER TABLE` SQL statement to modify the column definition. Use the syntax:

   ```
   ALTER TABLE <table> MODIFY <column> VARCHAR2(2000);
   ```

   Example:

   For the `CONTRACT_ADD_INFO_VALUE` table, the syntax is:

   ```
   ALTER TABLE CONTRACT_ADD_INFO_VALUE MODIFY VALUE_STRING
   VARCHAR2(1000);
   ```

Declaring the New Filter in the CID

Declaring the new filter in the CID involves:

- Inserting the filter information in the `SEARCH_FILTER` table
- Declaring the associated search criteria in the `SEARCH_CRITERIA` table

To declare the filter object

1. Use your database tool to connect to the CID.
2. In the `SEARCH_FILTER` table, add a record corresponding to your new filter.

COLUMN NAME	CONTENT	NOTES
SEARCH_FILTER_ID	ID of the new filter	Must be equal to or greater than 1000. IDs smaller than this are reserved.
SEARCH_FILTER_CODE	Code of the filter you use when programming JSPs.	Every default system filter begins with CORE_. Do not use this reserved prefix.
OBJECT_TYPE_ID	Object type of the searched object	One of the allowed object types.
NAME	Name of the filter displayed in JSP	
STRING_ID	ID of the localization string to localize the filter name	
DESCRIPTION	Description of the filter	
DESCRIPTION_STRING_ID	ID of the localization string to localize the filter description	
DAL_FUNCTION_NAME	Name of the DAL accessor that will call the query to perform the search.	
IS_DEFAULT	Boolean to specify if this filter will be used as default to search this type of object	
ROW_COUNT	Default maximum number of records returned by this filter (you can override this value in your JSPs)	
START_DATE	Start date of the filter	
END_DATE	End date of the filter	

List of object types:

OBJECT TYPE ID	OBJECT TYPE NAME
1	Level
3	Member

OBJECT TYPE ID	OBJECT TYPE NAME
4	Contract
5	Contact
6	Invoice
7	Prepaid package
8	Billing account
9	Trouble ticket
10	User
11	Persistent action manager
12	User event
13	Service
14	Request
15	Bill period
16	Contract subinvoice
17	Organization view
18	Organization view level

Example of Declaring New Filter

This example shows declaring the filter for searching contracts by SIM card number:

```
INSERT INTO SEARCH_FILTER

( SEARCH_FILTER_ID, SEARCH_FILTER_CODE, OBJECT_TYPE_ID, NAME, STRING_ID, DESCRIPTION, DESCRIPTION_STRING_ID,
DAL_FUNCTION_NAME, IS_DEFAULT, ROW_COUNT, START_DATE, END_DATE )

VALUES

( 1000,  'CONTRACTBYSIMCARDNUMBER', 4, 'Contract by SIM card number', NULL, 'Search contract by its SIM card
number', NULL, 'contractByFilterBYSIMCARDNUMBER', 0, 1, NULL, NULL);
```

To declare filter criteria

1. Use your database tool to connect to the CID.
2. For each criteria of the filter, add a record to the `SEARCH_CRITERIA` table.

COLUMN NAME	VALUE
SEARCH_FILTER_ID	Id of the new filter.
PARAM_ID	Id of the parameter used as criteria
DAL_PARAMETER_NAME	Name of the parameter used by the DAL to send the criteria value to the query.
IS_ACTIVE	Boolean to specify if the criteria is active in the filter

COLUMN NAME	VALUE
IS_MANDATORY	Boolean to specify if the criteria is mandatory for this filter
DEFAULT_VALUE_ID	Default value of the criteria
DISPLAY_ORDER	Order to display the parameter. –1 if the parameter must not be set by the user.

Example of Declaring Filter Criteria

This example shows the declaration of filter criteria for searching contracts by SIM card number:

Declare `CORE_C_ORGID` criteria	`INSERT INTO SEARCH_CRITERIA ( SEARCH_FILTER_ID, PARAM_ID, DAL_PARAMETER_NAME, IS_ACTIVE, IS_MANDATORY, DEFAULT_VALUE_ID, DISPLAY_ORDER ) VALUES ( 1000, 158, null, 1, 1, NULL, -1);`
Declare `CORE_C_SEARCHINSUBLEVELS` criteria	`INSERT INTO SEARCH_CRITERIA ( SEARCH_FILTER_ID, PARAM_ID, DAL_PARAMETER_NAME, IS_ACTIVE, IS_MANDATORY, DEFAULT_VALUE_ID, DISPLAY_ORDER ) VALUES ( 1000, 163, Null, 1, 1, 1, 2);`
Declare `simCardNumber` criteria	`INSERT INTO SEARCH_CRITERIA ( SEARCH_FILTER_ID, PARAM_ID, DAL_PARAMETER_NAME, IS_ACTIVE, IS_MANDATORY, DEFAULT_VALUE_ID, DISPLAY_ORDER ) VALUES ( 1000, 1006, 'simCardNumber', 1, 1, NULL, 1);`

Declaring the New Search Method in the DAL

Declare the new search method in the DAL involves:

- Declaring a new DAL internal function to call the query
- Declaring a new DAL accessor to access the new DAL internal function

To declare the new DAL internal function

1. Go to `<home_dir>/classes/nmycfg/dal`.
2. Open `core_containers.xml`.
3. Find the ROOT element (`<ROOT name="root" internal_name="root" type="DalContainer::DalObject">`)
4. Under the `<internal_functions>` element, add the following element. Use the syntax:

```
<Internal function name
query="SearchObjectName.queryName(
parameter/DAL Parameter name 1,
parameter/DAL Parameter name 2,
parameter/HIERARCHY_LEVELS,
parameter/SCOPEMEMBER_MEMBERID,
parameter/SCOPEEXPLICIT_MEMBERID,
parameter/SCOPEHIERARCHY_LEVELS,
parameter/SCOPEEXTERNALORG_ORGANIZATIONID,
parameter/SCOPEEXTERNALORG_ORGTYPES,
parameter/SCOPEORGANIZATIONMANAGED_ORGANIZATIONID,
parameter/SCOPEORGANIZATIONMANAGED_ORGTYPES,
parameter/SCOPELEVELMANAGED_LEVELID,
parameter/SCOPELEVELMANAGED_ORGTYPES,
parameter/SCOPEMEMBERMANAGED_MEMBERID,
parameter/SCOPEMEMBERMANAGED_ORGTYPES,
parameter/ROWCOUNT)">
            <parameters>
< DAL Parameter name 1 internal_name="" type="Parameter type" null="true" internal_function=""/>
< DAL Parameter name 2 internal_name="" type="Parameter type" null="true" internal_function=""/>
<HIERARCHY_LEVELS internal_name="" type="ObjectId[]/ORGANIZATION" null="true" internal_function=""/>
<SCOPEMEMBER_MEMBERID internal_name="" type="ObjectId/MEMBER" null="true" internal_function=""/>
<SCOPEEXPLICIT_MEMBERID internal_name="" type="ObjectId/MEMBER" null="true" internal_function=""/>
<SCOPEHIERARCHY_LEVELS internal_name="" type="ObjectId[]/ORGANIZATION" null="true" internal_function=""/>
<SCOPEEXTERNALORG_ORGANIZATIONID internal_name="" type="ObjectId/ORGANIZATION" null="true"
internal_function=""/>
<SCOPEEXTERNALORG_ORGTYPES internal_name="" type="ObjectId[]/ORGANIZATIONTYPE" null="true"
internal_function=""/>
<SCOPEORGANIZATIONMANAGED_ORGANIZATIONID internal_name="" type="ObjectId/ORGANIZATION" null="true"
internal_function=""/>
<SCOPEORGANIZATIONMANAGED_ORGTYPES internal_name="" type="ObjectId[]/ORGANIZATIONTYPE" null="true"
internal_function=""/>
<SCOPELEVELMANAGED_LEVELID internal_name="" type="ObjectId/ORGANIZATION" null="true"
internal_function=""/>
<SCOPELEVELMANAGED_ORGTYPES internal_name="" type="ObjectId[]/ORGANIZATIONTYPE" null="true"
internal_function=""/>
<SCOPEMEMBERMANAGED_MEMBERID internal_name="" type="ObjectId/MEMBER" null="true" internal_function=""/>
<SCOPEMEMBERMANAGED_ORGTYPES internal_name="" type="ObjectId[]/ORGANIZATIONTYPE" null="true"
internal_function=""/>
<ROWCOUNT internal_name="" type="Long" null="true" internal_function=""/>
    </parameters>
</ Internal function name >
```

5 Save your changes.

ELEMENT	NOTES
Internal function name	Name of the internal function, use a name consistent with the goal of the function

ELEMENT	NOTES
SearchObjectName.queryName	The search object name must be the name of the object that you want to search as declared in the DAL core containers. Supported objects for the search are: - billingaccount - contract - contract subinvoice - invoice - member - organization (also used when searching levels) - persistentactionmgr - request - service - userevent The query name is the name of the query that will be executed to perform the search
parameter/DAL Parameter name	Name of the DAL parameter associated with every filter criteria and declared in the parameters section. This name must be the same as the one declared in the associated filter criteria.
parameter/HIERARCHY_LEVELS	Reserved parameter. Mandatory if you use the `CORE_C_ORGID` and `CORE_C_ SEARCHINSUBLEVELS` criteria in your filter.
parameter/SCOPEMEMBER_MEMBERID to parameter/SCOPEMEMBERMANAGED_ORGTYPES	Reserved system parameters. Mandatory if you want to apply security on your search. To declare these parameters, you can copy the parameters declared for the search function of the same object type.
DAL Parameter name 1	Name of the DAL parameter associated with every filter criteria. This name must be the same as the one declared in the associated filter criteria.
Parameter type	Type of the parameter. It must be consistent with the criteria parameter type declared in the CID. Refer to the DAL Parameter Type Table.

The table below shows the list of supported types and the compatibility with CID parameter types

DAL PARAMETER TYPE	CID PARAMETER TYPE
String	String
ObjectId/ObjectName	Dynamic
Long	Integer
Date	Date, Time and Timestamp
Float	Decimal
ObjectId/CHOICEITEM	Choice

Note: if you want to use a list instead of a simple value, add brackets to the parameter type (example Long[], ObjectId[]/ObjectName). The associated criteria parameter must also be declared as a list. For more information, refer to *PARAMETER - About this Package* in the *CID Reference Guide*).

Example of declaring a DAL internal function

This example shows the declaration of the DAL internal function for searching contracts by SIM card number:

```
<getContractByFilterBYSIMCARDNUMBER query="contract.getByFilterBYSIMCARDNUMBER(

parameter/simCardNumber,

parameter/HIERARCHY_LEVELS,

parameter/SCOPEMEMBER_MEMBERID,

parameter/SCOPEEXPLICIT_MEMBERID,

parameter/SCOPEHIERARCHY_LEVELS,

parameter/SCOPEEXTERNALORG_ORGANIZATIONID,

parameter/SCOPEEXTERNALORG_ORGTYPES,

parameter/SCOPEORGANIZATIONMANAGED_ORGANIZATIONID,

parameter/SCOPEORGANIZATIONMANAGED_ORGTYPES,

parameter/SCOPELEVELMANAGED_LEVELID,

parameter/SCOPELEVELMANAGED_ORGTYPES,

parameter/SCOPEMEMBERMANAGED_MEMBERID,

parameter/SCOPEMEMBERMANAGED_ORGTYPES,

parameter/ROWCOUNT)">

<parameters>

          <simCardNumber internal_name="" type="String" null="true" internal_function=""/>

          <HIERARCHY_LEVELS internal_name="" type="ObjectId[]/ORGANIZATION" null="true" internal_function=""/>

          <SCOPEMEMBER_MEMBERID internal_name="" type="ObjectId/MEMBER" null="true" internal_function=""/>

          <SCOPEEXPLICIT_MEMBERID internal_name="" type="ObjectId/MEMBER" null="true" internal_function=""/>

          <SCOPEHIERARCHY_LEVELS internal_name="" type="ObjectId[]/ORGANIZATION" null="true"
internal_function=""/>

          <SCOPEEXTERNALORG_ORGANIZATIONID internal_name="" type="ObjectId/ORGANIZATION" null="true"
internal_function=""/>

          <SCOPEEXTERNALORG_ORGTYPES internal_name="" type="ObjectId[]/ORGANIZATIONTYPE" null="true"
internal_function=""/>

          <SCOPEORGANIZATIONMANAGED_ORGANIZATIONID internal_name="" type="ObjectId/ORGANIZATION" null="true"
internal_function=""/>

          <SCOPEORGANIZATIONMANAGED_ORGTYPES internal_name="" type="ObjectId[]/ORGANIZATIONTYPE" null="true"
internal_function=""/>

          <SCOPELEVELMANAGED_LEVELID internal_name="" type="ObjectId/ORGANIZATION" null="true"
internal_function=""/>

          <SCOPELEVELMANAGED_ORGTYPES internal_name="" type="ObjectId[]/ORGANIZATIONTYPE" null="true"
internal_function=""/>

          <SCOPEMEMBERMANAGED_MEMBERID internal_name="" type="ObjectId/MEMBER" null="true"
internal_function=""/>

          <SCOPEMEMBERMANAGED_ORGTYPES internal_name="" type="ObjectId[]/ORGANIZATIONTYPE" null="true"
internal_function=""/>

          <ROWCOUNT internal_name="" type="Long" null="true" internal_function=""/>

          </parameters>

</getContractByFilterBYSIMCARDNUMBER>
```

To declare the new DAL accessor

1. Go to `<home_dir>/classes/nmycfg/dal`
2. Open `core_containers.xml`.
3. Find the ROOT object (`<ROOT name="root" internal_name="root" type="DalContainer::DalObject">`)
4. Under the `<data_accessors>` element, add the following element. Use the syntax:

```
<DAL accessor name internal_name=""
type="DalContainer[]/objects.Searched Object"
null="true"
internal_function="DAL Internal function"/>
```

ELEMENT	NOTES
DAL accessor name	Name of the accessor. Use a name consistent with the goal of the search. The name must be the same as the one declared in the associated filter.
objects.Searched Object	Supported objects: • billingaccount • contract • contract subinvoice • invoice • member • organization (also used when searching levels) • persistentactionmgr • request • service • userevent
DAL Internal function	Name of the associated internal function declared above

Example of Declaring a New DAL Accessor

This example shows the declaration of the DAL accessor for searching contracts by SIM card number:

```
<contractByFilterBYSIMCARDNUMBER internal_name=""
 type="DalContainer[]/objects.CONTRACT"
null="true"
internal_function="getContractByFilterBYSIMCARDNUMBER"/>
```

Writing the New Query

Writing the new search query involves:

- Writing the query using the DAL syntax
- Declaring the query

About DAL Query Syntax

The DAL queries are preprocessed before being executed by a standard SQL engine. The syntax of the DAL queries is very similar to standard SQL with specific behavior for criteria evaluation and security management.

The structure of a search query is:

- SELECT

  List of table columns storing searched object data (Standard SQL). The simplest way to write this section is to copy the one provided by core search queries for the same object

- FROM

  List of tables involved in the search (Standard SQL)

- WHERE

  Criteria Evaluation

  Security Management

  (ROWNUM <= NVL(?, 1000))

About Criteria Evaluation

This section is written in SQL. Every criteria value is replaced by a '?'.

At run time, the '?' is replaced with parameter values in the same order as they are declared in the DAL internal function call to the query.

Syntax Rules:

- IGNORE IF NULL:
  When a criterion is optional, it must be always considered as TRUE if its value is NULL. In this case, you must use the following syntax:
  TABLE.COLUMN_NAME [SQL logical operator ? OR IGNORE IF NULL].
  At run time, if the parameter value is NULL, the condition will always be True.
- IGNOREALL IF NULL:
  In some cases, you may want to exclude part of the evaluation section from being taken in account during runtime if a parameter value is null. In this case, you must use the following syntax:
  [Beginning of the section to take in account only if the parameter value is not null….
  TABLE.COLUMN_NAME [SQL logical operator ? OR IGNOREALL IF NULL]…End of the section to take in account only if the parameter value is not null]
  At run time, if the parameter value is NULL, all SQL statements between the two brackets are ignored.

About Security Management

This section is used to apply security and only select objects that the user is authorized to view.

You must write this section only if the searched object is protected. Use the syntax:

```
/******* SECURITY SCOPES MANAGEMENT ********/
(
   [C.CONTRACT_ID [IN CONTRACTSMANAGEDBY ?] /* EXPLICITSCOPE */   OR ]
   [C.MEMBER_ID [=? OR IGNOREALL IF NULL] /* MEMBERSCOPE */ OR ]
   [C.ORGANIZATION_ID [IN (?) OR IGNOREALL IF NULL] /* HIERARCHY SCOPES */  OR ]
   (C.ORGANIZATION_ID [IN INTERORG SCOPES ?,?,?,?,?,?,?,?])
)
```

The simplest way to write a section on security is to copy the security section declared in core search queries for the same object.

To declare a new query

1 Go to `<home_dir>/classes/nmycfg/dal`.

2 Open `core_queries.xml`.

3 Find the object you want to search

4 Insert the new query. Use the syntax:

```
 <Query name>
Query
</Query name>
```

Where Query name is the one you previously declared in the DAL internal function.

Example of a DAL Query

This example shows the DAL query for searching contracts by SIM card number:

```
SELECT
C.CONTRACT_ID,
C.ORGANIZATION_ID,
C.RATEPLAN_ID,
C.CONTRACT_TYPE_ID,
C.BILLING_ACCOUNT_ID,
C.MEMBER_ID,
C.CONTRACT_PARENT_ID,
C.CONTRACT_LEVEL_TYPE,
C.CONTRACT_STATUS_ID,
C.CONTRACT_STATUS_DATE,
C.LINE_ID,
L.LINE_NUMBER,
L.LISTED,
L.LINE_TYPE_ID
FROM
CONTRACT C,LINE L
WHERE
C.LINE_ID = L.LINE_ID(+) AND
/******* CRITERIA *******/
[C.CONTRACT_ID IN
     (SELECT CAI.CONTRACT_ID
      FROM CONTRACT_ADD_INFO CAI, CONTRACT_ADD_INFO_VALUE CAIV
      WHERE
      CAI.PARAM_ID=1006 AND
      CAI.VALUE_ID=CAIV.VALUE_ID AND
      CAIV.VALUE_STRING [=? OR IGNOREALL IF NULL] /* SIMCARD NUMBER */
     )
AND ]
(C.ORGANIZATION_ID [IN (?) OR IGNORE IF NULL]  /* HIERARCHY */ ) AND
/******* SECURITY SCOPES MANAGEMENT ********/
(
   [C.CONTRACT_ID [IN CONTRACTSMANAGEDBY ?] /* EXPLICITSCOPE */   OR ]
   [C.MEMBER_ID [=? OR IGNOREALL IF NULL] /* MEMBERSCOPE */ OR ]
   [C.ORGANIZATION_ID [IN (?) OR IGNOREALL IF NULL] /* HIERARCHY SCOPES */  OR ]
   (C.ORGANIZATION_ID [IN INTERORG SCOPES ?,?,?,?,?,?,?,?])
) AND
/*********************************/
(ROWNUM &lt;= NVL(?, 1000))
)
```

Writing a Search JSP

To help you understand how to create a search, you can use the example of a simple search workflow for contracts. This example shows you how to write a search that uses search filters to determine which criteria to display.

In your application, you can use this example to create several search features easily by using the search filters and sample code to automatically display criteria based on the search filter instead of being programmed in the presentation logic of each JSP.

Writing a search JSP involves:

1. Getting the filter
2. Getting dynamic and internal criteria
3. Display the search criteria
4. Getting the values of the search criteria the user submits
5. Executing the search
6. Displaying the results

Getting the Filter

BLM search methods use search filters. Search filters are sets of search criteria located in the CID. These filters help narrow the search and determine default criteria and required criteria.

You use the methods in the `ObjectRefMgr` to search and retrieve filters.The BLM `FilterIF` interface contains methods to manage search filters.

This sample code shows how to:

- Return the specified filter
- Test the filter to make sure it is a valid filter
- Send all of the criteria to the display page because there are no dynamic or hidden criteria

Return the filter	<%@ page import = "com.netonomy.blm.interfaces.search.*" %>
	<%!
	SIMPLE SEARCH
	String []listOfFilters = new String[]{"CORE_CONTRACTBYLINENUMBER"};
	
	//
	
	// Get all the filters described in the array above.
	FilterIF [] filters = new FilterIF[listOfFilters.length];
	//Sets the filter in a an array that will contains, for each filter:
	// filter at [0], ids and values to display at [1] and [2].
	// This array will be sent to the displaying page
	
	Object [][] dynamicFilters = new Object[filters.length][5];

Test the filter	`Date curDate = new Date ();`
	`for (int i=0;i<listOfFilters.length;i++)`
	`{`
	`filters[i] = ObjectRefMgr.getFilterByCode (listOfFilters[i]);`
	`// Test if the filter is still valid`
	`if (((filters[i].getStartDate()!=null)&&(curDate.compareTo (filters[i].getStartDate ())<0)) ||`
	`(((filters[i].getEndDate()!=null) && curDate.compareTo (filters[i].getEndDate ())>0)))`
	`{`
	`dynamicFilters[i]=null;`
	`}`
	`else`
	`{`
	`dynamicFilters[i] = new Object[5];`
	`dynamicFilters[i][0] = filters[i];`
	`dynamicFilters[i][1] = null; // dynamic parameters values will be filled below`
	`dynamicFilters[i][2] = null;`
	`dynamicFilters[i][3] = null;`
	`dynamicFilters[i][4] = null;`
	`}`
	`}`

Send the criteria to the display page	`//////////////////////////////////////////////`
	`// No dynamic data to get`
	`//////////////////////////////////////////////`
	`// No dynamic default values to get`
	`//////////////////////////////////////////////`
	`// Sends all criteria and values to the display page.`
	`request.setAttribute ("criteria", dynamicFilters);`
	`}`
	`%>`

Some searches may have required dynamic or hidden criteria that you need to get values for before sending information to the display page.

Getting Dynamic and Hidden Criteria

When writing searches, you may be required to use criteria that can change and other criteria that you need to program the search, but that the user should not see. You can use the following:

- **Dynamic Criteria** These criteria are values criteria that may change from one search to another. An example of this kind of search criteria is a contract service. Contract services may be limited to only a specific type of contract, it may expire, and it may be incompatible with the rate plan of the contract. If the search filter contains this kind of criteria, you must write the code to obtain the possible values for these criteria for display.
- **Hidden Criteria** These criteria are hidden criteria that you may need to use but that the user should not see. Like dynamic criteria, you must write the code to obtain values for these criteria.

This sample code shows how to:

- Return the specified filter
- Test the filter to make sure it is a valid filter
- Get the values of dynamic criteria to display
- Get the values of hidden criteria
- Send all of the criteria to the display page

Return the specified filter	<%@ page import = "com.netonomy.blm.interfaces.search.*" %>
	
	<%!
	
	String []listOfFilters = new String[]{"CORE_INTORG_CONTRACTBYLINENUMBER",
	"CORE_INTORG_CONTRACTADVANCEDSEARCH"};
	//
	
	// Get all the filters described in the array above.
	
	FilterIF [] filters = new FilterIF[listOfFilters.length];
	
	//Sets the filter in a an array that will contains, for each filter:
	// filter at [0], ids, values and their coded types to display at [1], [2]
	//and [3], hidden field coded type at [4].
	// This array will be sent to the displaying page
	Object [][] dynamicFilters = new Object[filters.length][5];

Test the filters	Date curDate = new Date ();
	for (int i=0;i<listOfFilters.length;i++)
	{
	filters[i] = ObjectRefMgr.getFilterByCode (listOfFilters[i]);
	// Test if the filter is still valid
	if (((filters[i].getStartDate()!=null) && (curDate.compareTo (filters[i].getStartDate
	())<0)) ||
	(((filters[i].getEndDate()!=null) && curDate.compareTo (filters[i].getEndDate ())>0)))
	{
	dynamicFilters[i]=null;
	}
	else
	{
	dynamicFilters[i] = new Object[5];
	dynamicFilters[i][0] = filters[i];
	dynamicFilters[i][1] = null; // dynamic parameters values will be filled below
	dynamicFilters[i][2] = null;
	dynamicFilters[i][3] = null;
	dynamicFilters[i][4] = null;
	}
	}

Browse filters and criteria

```
  ////////////////////////////////////////////////

  // Get the data for all dynamic values

  // We must tell the type of the dynamic parameters so that it can be correctly encoded

  //  and retrieved by fillParameterWithRequest function. Codes are inserted in the types array

  //  Codes are : ID for objectIds, STR for strings, DATE for dates, BOOL for booleans, INT for
integers, FLOAT for double

  ////////////////////////////////////////////////

  // and

  ////////////////////////////////////////////////

  // Get the dynamic default values

  // We must tell the type of the dynamic parameters so that it can be correctly encoded

  //  and retrieved by fillParameterWithRequest function.

  //  Codes are : ID for objectIds, STR for strings, DATE for dates, BOOL for booleans, INT for
integers, FLOAT for double


  // Values of dynamic or hidden data are kept in this cache if another filter needs it.

  HashMap cache = new HashMap ();


  String[] ids;

  String[] displays;

  String   type;

  Object   hiddenValue;


  Object [][] cachedValue;

  String  criteriaCode;

  ParameterDescriptorIF descriptor;

  FilterIF  filter;

  ParameterIF[] allParameters;

    // Temp arrays that will eventually be copied to dynamicFilters.

  ArrayList idsArray = new ArrayList();

  ArrayList displaysArray = new ArrayList();

  ArrayList typesArray = new ArrayList();

  ArrayList hiddenTypesArray = new ArrayList();


    // Loop through all criteria of all filters.

  for (int j=0;j<dynamicFilters.length;j++)

    {

    filter = (FilterIF)dynamicFilters[j][0];

    allParameters = filter.getCriteria (FilterIF.ALL);

    idsArray.clear ();

    displaysArray.clear ();

    typesArray.clear ();

    hiddenTypesArray.clear ();

    for (int iParam=0;iParam<allParameters.length;iParam++)

      {

      descriptor = allParameters[iParam].getParameterDescriptor ();

      criteriaCode = descriptor.getCode();

      cachedValue = (Object[][])cache.get (criteriaCode);

        // If it's in the cache, just fills in the value to the main structure

      if (cachedValue == null)

        {

          // Reset the values that will be filled by the custom code

        ids = null;

        displays = null;

        type = null;

        hiddenValue = null;
```

Get the values of dynamic criteria	```
	//
	// You can customize here
	if (criteriaCode.equals ("CORE_C_CONTRACTTYPE"))
	{
	// It's a dynamic filter. We fill ids, displays, type
	OrganizationTypeIF orgType = org.getType ();
	ContractTypeIF contractTypes[] = orgType.getCompatibleContractTypes ();
	ids = new String[contractTypes.length];
	displays = new String [contractTypes.length];
	
	for (int i=0;i<contractTypes.length;i++)
	{
	ids[i] = contractTypes[i].getIdentifier().toString();
	displays[i] = contractTypes[i].getName ();
	}
	type = "ID";
	}
	``

Get the values of hidden criteria	```
	else if (criteriaCode.equals ("CORE_C_ORGID"))
	{
	// It's a hidden criteria. We fill hiddenValue, type
	hiddenValue = curLevelId;
	type="ID";
	}
	``

End browse of filter and criteria

```
        if (ids != null)
          {
            // Adds the dynamic criteria values to the cache.
          cache.put (criteriaCode, new String[][]{ids, displays, new String[]{type}});
          }
        else if (type != null)
          {
            // Adds the hidden criteria values to the cache.
          cache.put (criteriaCode, new Object[][]{new Object[]{hiddenValue}, new
Object[]{type}});
          }
        }

        // Maybe the value has been filled by the former code, so we try again to get it
      cachedValue = (Object[][])cache.get (criteriaCode);
        // If it's in the cache, just fills in the value to the main structure
      if (cachedValue != null)
        {
          // Is it a hidden or a dynamicvalue ?
        if (cachedValue.length==3)    // It"s a dynamic
          {
          idsArray.add (cachedValue[0]);
          displaysArray.add (cachedValue[1]);
          typesArray.add (cachedValue[2][0]);
          }
        else
          {
          switch (descriptor.getType())
           {
             case  ParameterDescriptorIF.DYNAMIC:
             // set the value of the hidden criteria with the value stored in the hashtable
              ((ValueDynamicIF)allParameters[iParam]).setDynamicValue(cachedValue[0][0]);
                 break;
              default:
              ((ValueSimpleIF)allParameters[iParam]).setValue(cachedValue[0][0].toString());
                 break;
            }

          hiddenTypesArray.add (cachedValue[1][0]);
          }
        }
      }

      // Now that all parameters of a filter has been seen, we will aggregate all values on
it's dynamicFilters array
      if (idsArray.size()>0)
        {
         dynamicFilters[j][1] = idsArray.toArray(new String [idsArray.size()][]);
         dynamicFilters[j][2]= displaysArray.toArray(new String[displaysArray.size()][]);
         dynamicFilters[j][3]= typesArray.toArray( new String[typesArray.size()]);
        }

      if (hiddenTypesArray.size()>0)
        {
        dynamicFilters[j][4]= hiddenTypesArray.toArray( new String[hiddenTypesArray.size()]);
        }
```

Send the criteria to the display page	`  // Sends all criteria and values to the display page.`
	`  request.setAttribute ("criteria", dynamicFilters);`
	`}`
	`%>`

Now that the values of the different criteria have been set, send the results to a JSP for display.

Displaying the Search Criteria

Once you have retrieved the filter and set values for dynamic and hidden criteria, you display the search criteria in a search page.

The TSM Web channel comes with a search page for each of the main objects in the BLM. The search pages are located in `<home_dir>/channels/MyWeb`. The search pages are named `<object>_search.jsp`. In these JSPs, the code creates an HTML form and form button for each filter. In this form, the JSP creates a form element for each criteria in the filter. If your search uses more than one filter, your search page also has more than one search form.

These search pages are also used to display the results of your search. The upper section of the page displays the search forms for each filter you use. The lower section displays the results of the search and an associated message.

This sample code shows how to:

- Get the criteria sent from the preceding JSP
- Create the search form
- Display the criteria sent from the preceding JSP
- Display the default values of the criteria (if the first time the form is displayed)
- Display an image next to mandatory criteria

Get the criteria

```
<%@ include file="components/parameter_modify.jsp" %>
 if (request.getAttribute ("criteria")!=null)
  {
  Object [][]dynFilters = (Object[][])request.getAttribute ("criteria");
  FilterIF  curFilter;
  String [][] allIds;
  String [][] allDisplays;
  String [] types;
  String [] hiddenTypes;
  boolean oneParamMandatory=false;
    // Display parameters of all filters
  for (int i=0;i<dynFilters.length;i++)
    {
    curFilter = (FilterIF)dynFilters[i][0];
    allIds = (String[][])dynFilters[i][1];
    allDisplays = (String[][])dynFilters[i][2];
    types = (String[])dynFilters[i][3];
    hiddenTypes = (String[])dynFilters[i][4];
    if (curFilter != null)
      {
  %>


  <%
    // Now display search criteria using code in components/parameter_modify.jsp
    if (curFilter.getIdentifier().toString().equals (request.getParameter("filter")))
      {
     //if displaying the values along with the results, display the values as entered by
the user
      oneParamMandatory |= displayEditableParameters (curFilter.getCriteria
(FilterIF.VISIBLE), allIds, allDisplays, types, request, jspHelper, out,
curFilter.getIdentifier().toString(), true);
      }
    else
      {
      //if displaying the search form the first time, display default values
      oneParamMandatory |= displayEditableParameters (curFilter.getCriteria
(FilterIF.VISIBLE), allIds, allDisplays, types, request, jspHelper, out,
curFilter.getIdentifier().toString(), false);
      }
      // Enter the hidden system criteria
   displayHiddenParameters (curFilter.getCriteria (FilterIF.HIDDEN), hiddenTypes,
jspHelper, out, curFilter.getIdentifier().toString());
  %>
    </td></tr>
    <tr>
     // Form button with localized text
    <td align="center"><input type="submit" value="<%=jspHelper.localize
("search_button","default_text")%>"><input type="reset" value="<%=jspHelper.localize
("button_reset","default text")%>"></td>
    </tr>
    <tr height="1">
    <td height="1" class="linebottom" width="30%"><img height="2" width="2"
src="<%=jspHelper.getJFNAppSkinUrl()%>gif/varSpacerTransp1.gif"><img height="1" width="1"
src="<%=jspHelper.getJFNAppSkinUrl()%>gif/varSpacerTransp1.gif"></td>
    </tr>
    </form>
<%
      }
    }
    if (oneParamMandatory)
    {
    // For each mandatory search criteria, display a visual symbol (dot, gif, or whatever)
    %>
     <tr><td>
      <table cellspacing="0" cellpadding="2" border="0">
        <tr>
          <td>
            <img height="6" width="12" src="<%=jspHelper.getJFNAppSkinUrl()%>gif/dot3.gif"
align="absmiddle"> : 
            <%
            String text = jspHelper.localize("mandatory_field","default_text");
            out.println(text);
            %>
          </td>
        </tr>
      </table>
      </td></tr>
    <%
    }
  }
%>
```

Create the search form

```
// Create the HTML for for the search criteria
    <form action="<%=jspHelper.getUrl("CONTRACT_SEARCH")%>" method="post">
    <tr><td>
    <%=jspHelper.generateAllParametersAsHiddenFields ()%>
    <input type="hidden" name="filter" value="<%=curFilter.getIdentifier().toString()%>">
  <%
```

Display the criteria	`// Now display search criteria using code in components/parameter_modify.jsp`
	`if (curFilter.getIdentifier().toString().equals (request.getParameter("filter")))`
	`{`
	`//if displaying the values along with the results, display the values as entered by the user`
	`oneParamMandatory |= displayEditableParameters (curFilter.getCriteria (FilterIF.VISIBLE), allIds, allDisplays, types, request, jspHelper, out, curFilter.getIdentifier().toString(), true);`
	`}`
	`else`
	`{`
	`//if displaying the search form the first time, display default values`
	`oneParamMandatory |= displayEditableParameters (curFilter.getCriteria (FilterIF.VISIBLE), allIds, allDisplays, types, request, jspHelper, out, curFilter.getIdentifier().toString(), false);`
	`}`
	`// Enter the hidden system criteria`
	`displayHiddenParameters (curFilter.getCriteria (FilterIF.HIDDEN), hiddenTypes, jspHelper, out, curFilter.getIdentifier().toString());`
	`%>`
	`</td></tr>`
	`<tr>`
	`// Form button with localized text`
	`<td align="center"><input type="submit" value="<%=jspHelper.localize ("search_button","default_text")%>"><input type="reset" value="<%=jspHelper.localize ("button_reset","default text")%>"></td>`
	`</tr>`
	`<tr height="1">`
	`<td height="1" class="linebottom" width="30%"><img height="2" width="2" src="<%=jspHelper.getJFNAppSkinUrl()%>gif/varSpacerTransp1.gif"><img height="1" width="1" src="<%=jspHelper.getJFNAppSkinUrl()%>gif/varSpacerTransp1.gif"></td>`
	`</tr>`
	`</form>`

Display an image next to mandatory criteria	`<%`
	`}`
	`}`
	`if (oneParamMandatory)`
	`{`
	`// For each mandatory search criteria, display a visual symbol (dot, gif, or whatever)`
	`%>`
	`<tr><td>`
	`<table cellspacing="0" cellpadding="2" border="0">`
	`<tr>`
	`<td>`
	`<img height="6" width="12" src="<%=jspHelper.getJFNAppSkinUrl()%>gif/dot3.gif" align="absmiddle"> : `
	`<%`
	`String text = jspHelper.localize("mandatory_field","default_text");`
	`out.println(text);`
	`%>`
	`</td>`
	`</tr>`
	`</table>`
	`</td></tr>`
	`<%`
	`}`
	`}`

Executing the Search

After the user enters the search criteria and clicks *Submit*, the JSP:

1. Creates a request containing the name of the filter and the list of search criteria with their corresponding values.
2. Calls the associated logic handler to search for matching objects.

The logic handler contains code that manages the business logic of the channel. Having the logic code in separate JSPs helps separate the business logic from the presentation logic. Like the search pages, the MyWeb channel comes with a logic page for each of the main objects in the BLM. The pages are located in `<home_dir>/channels/MyWeb` and are named `logic_<object>.jsp`.

This sample code shows how to:

- Get the name of the filter from the request
- Get the filter from the CID
- Fill the filter using the criteria values from the request
- Search the corresponding objects
- Call the page to display the results

Get and prepare filter	<%!include file= "../common/form_handler/setParameters.jsp"%>
	
	<%!
	/**
	 * Logic handler for finding contracts
	 *
	 * @param session The current HTTP session
	 * @param httpRequest The current HTTP request
	 * @param httpResponse The current HTTP response
	 * @param jfnJsp The JSP configuration
	 * @param results Hashtable of returned objects
	 * @param errors Hashtable of error objects
	 */
	
	public void logic_searchContract (HttpSession session,
	  HttpServletRequest request,
	  HttpServletResponse response,
	  JFNJspHelper jspHelper,
	  Hashtable errors
	  ) throws Throwable
	{
	
	  SessionF blmSession;
	  ContractF[] contracts;
	  FilterIF filter;
	  //'*** Get the BLM session ***
	  blmSession = jspHelper.getBlmSession ();
	  // Get the filter from the request
	  filter = ObjectRefMgr.getFilter (ObjectId.instantiate (request.getParameter("filter")));
	  int maxCount = filter.getRowCount(); // The max # of items declared in the database.
	  // Get one more than the max number of returned items declared for the filter
	  filter.setRowCount (maxCount+1);
	  jspHelper.doNotSendBeginningWith("p-");
	  jspHelper.doNotSend("filter");

Fill with criteria	try
	  {
	  // Fill the criteria values of the filter with the values from the request
	  fillParametersWithRequest (filter.getCriteria (FilterIF.ALL), request, jspHelper,
	request.getParameter ("filter"));
	  }
	  // If an error occurs, throw exception
	  catch (BadValueException ex)
	  {
	  String strValue = jspHelper.tryToGetParameterValue(ex.getBadValue ());
	  ParameterDescriptorIF descriptor = ex.getBadValue().getParameterDescriptor();
	  Hashtable cgiparams = new Hashtable();
	  cgiparams.put("param_value", strValue);
	  cgiparams.put("param_desc", descriptor.getIdentifier().toString());
	  cgiparams.put("param_code", String.valueOf(ex.getType()));
	  response.sendRedirect(jspHelper.encodeURLFunct ("GLOBAL.PARAMETER_ERROR",
	cgiparams, true));
	  jspHelper.exitJSP();
	  }

Execute search	// Search for contracts
	contracts = ContractF.search(filter);

Send contracts found to next page and inform newt page if more than the maximum number of items was returned	`// We asked for max+1, did we get all of them?`
	`if (contracts!=null)`
	`{`
	`if (contracts.length>=maxCount+1)`
	`{`
	`// Limit of the filter`
	`// if more objects are returned, set the more attribute to display message`
	`ContractF[] ret = new ContractF[maxCount];`
	`System.arraycopy(contracts, 0, ret, 0, maxCount);`
	`request.setAttribute ("contracts", ret);`
	`request.setAttribute ("more", String.valueOf(maxCount));`
	`}`
	`else`
	`{`
	`request.setAttribute("contracts", contracts);`
	`}`
	`}`
	`request.setAttribute ("display", Boolean.TRUE);`
	`request.getRequestDispatcher(jspHelper.getUrlAndMoveToPage ("ON_OK")).forward(request, response);`
	`}`
	`%>`

Displaying the Results

After you carry out the search, the logic handler sends the result to the display JSP.

The MyWeb channel uses the search pages to display the results of your search. The upper section of the page displays the search forms for each filter you use. The lower section displays the results of the search and a message.

This sample code shows how to:

- Get the result from the form handler
- Display the result message
- Display the list of contracts found

Get the result from the form handler

```
<%
    ////////////////////////////////////////////////////////////////////////////
    // Display the result.
if (results.get ("display")!=null)
{  //' if custadmin with form or subscriber then display search
ContractF[] contracts = (ContractF[])request.getAttribute ("contracts");
%>
  <tr><td>
<%  if (jspHelper.isFunctionalStepValid ("CONTRACT_SEARCH"))
      {
%>
  // result section heading
  <br/><div class="datatitle"><%=jspHelper.localize
("search_result_heading","default_text")%></div><br/>
<%
      }
%>
```

Display the result message

```
// result message
  <br/><div class="errorTitleText">
<%
    if ((contracts==null) || (contracts.length==0))
        {%>
      <%=jspHelper.localize ("no_contract_found_message","default_text")%>
<%
     }
      else if (request.getAttribute ("more")!=null)
        {%>
        <%=jspHelper.localize ("more_contracts_found_message",new
Object[]{request.getAttribute ("more")})%>
<%
      }
      else if ((contracts != null) && (contracts.length>0))
        {%>
        <%=jspHelper.localize ("contracts_found_message",new Object[]{new
Integer(contracts.length)})%>
<%
        }
%>
```

Display the list of contracts found

```
 </div></td></tr>
 <% // The list of contracts found
 if ((contracts!=null) && (contracts.length>0))
    {
%>
<tr><td>
<table cellspacing="0" cellpadding="2" border="0" width="70%">
 <tr class="inverseHeaderText">
  <td><%=jspHelper.localize ("lineNumber","default text")%></td>
  <td><%=jspHelper.localize ("lineType","default text")%></td>
  <td><%=jspHelper.localize ("status","default text")%></td>
  </tr>
  <%
  for (int i=0; i<contracts.length; i++)
    {
        ContractF contract  = contracts[i];µ
  <tr>
  <td><%=contract.getPhoneNumber()%></td>
  <td><%=contract.getLineType().getName()%></td>
  <td><%=contract.getStatus().getName()%></td>
  </tr>
  <%
    }
  %>
</table>
</td></tr>
<%
    }
  }
%>
</table>
```

CHAPTER 5

Changing the BLM Business Logic

In This Section

About Business Logic 102
Changing Business Logic 103

About Business Logic

In the BLM, the business logic you can modify is implemented in BLM external classes. You can extend the business logic to meet your needs or even replace it if you must apply complex rules and processing to your BLM.

The external classes are declared in the BLM `external_custom.xml` customization file and located in the `com.netonomy.blm.external` package. This file is located in `<home_dir>/classes/nmycfg/blm`.

The set of default BLM external classes include:

- Check Classes

  These classes check the validity of an action. For example, the `CheckAddBillingAccount` class contains the code that validates the AddBillingAccount action.

- List Classes

  These classes return lists of objects that are allowed for a specific action. For example, the `ListAddableServices` class returns a list of services that can be added to the specified contract.

- Business Logic Classes

  These classes implement specific business logic for specific features. For example, the `EvaluateApprovalProcess` class contains the business logic for approval processes.

Changing Business Logic

Changing the business logic involves:

- Creating a custom class extending the default BLM external class
- Overriding the method processing the business logic
- Compiling the custom class
- Deploy the custom class
- Declare the custom class in the `external_custom.xml` customization file

For more information about the BLM external classes and their methods, refer to the *BLM API Reference Documentation* under the `com.netonomy.blm.external` package.

To create a custom class extending a BLM class

When creating your Java class, we suggest declaring a Java package to implement your class.

For example, `com.<yourclasspackage>.netonomy.blm.external` where `<yourclasspackage>` is the name of your company or the name of your customer.

Example of extending a default BLM external class:

```
package com.<yourclasspackage>.netonomy.blm.external;


import com.netonomy.blm.external.<DefaultClassName>;


public class <CustomClassName> extends <DefaultClassName>
{ }
```

To override the method processing the business logic

1. Refer to the *BLM API Reference Documentation* to find the signature of the method to override. The BLM external classes are under `com.netonomy.blm.external`.
2. In your class, enter the signature then write your own business logic.

   Example:

```
package com.<yourclasspackage>.netonomy.blm.external;

import com.netonomy.blm.external.<DefaultClassName>;

public class <CustomClassName> extends <DefaultClassName>
{
public <ReturnedParameter> <methodName>(<Parameters list>) {

// enter your code here

return <ReturnedValue>;
}
}
```

If you want to extend the default business logic instead of replacing it, use the standard Java inheritance mechanism by calling the core method (`super.methodName`)

To compile your custom class

When compiling your class, you need to make sure the following `jar` files are in the `CLASSPATH`:

- `<home_dir>/lib/nmycore.jar`
- `<home_dir>/lib/nmyutil.jar`

To deploy your custom class

1. Create sub folders consistent with your package name. For example, create a folder called `classes/com/<yourclasspackage>/netonomy/blm/external`.
2. Copy your compiled class to this folder.

To declare your custom class

1. Go to `<home_dir>/classes/nmycfg/blm`.
2. Open the `external_custom.xml` customization file.
3. Find the `class` element with `default` attribute equal to the default BLM external class name to override.
4. Enter the name of your custom class as the value of the `custom` attribute.

   Example:

```
<class default="com.netonomy.blm.external.<DefaultClassName>" custom="
com.<yourclasspackage>.netonomy.blm.external.<CustomClassName>"/>
```

5 Save your changes.

CHAPTER 6

Managing Security

In This Section

About Security 108
Understanding the CID Schema Security 109
Configuring Authentication 110
Using Trust Modes 111
Managing Access to BLM Objects 112
Using Explicit Security 113

About Security

Security of the Account Management application can be divided into the following categories:

- TSM security

  This category is the basic system security settings and its different components.

- Host Environment security

  This category involves security settings for the host environment (application server, OSS, database and other components.)

For security concerning these components, refer to your product documentation.

Managing security involves:

- Understanding CID Users
- Configuring authentication
- Configuring trust modes
- Securing BLM Objects
- Using explicit security
- Securing JSPs

Understanding the CID Schema Security

The CID database has two different users. The two users are:

- CID_ADMIN
- CID_USER

The CID_ADMIN user is the owner of the CID schema. This administration user can create, modify and administer the CID. The CID_ADMIN also grants permission to the CID_USER_ROLE.

The CID_USER user is the application user. Applicative processes 9application server, Synchronizer, Approval Sequencer, and so on) use the CID_USER to connect to and manage information in the CID.

Configuring Authentication

You can use either the CID or LDAP authentication method to grant users access to your application. By default, the CID is used for authentication.

For more information about configuring authentication, refer to the *BLM Reference Guide*.

Using Trust Modes

The BLM manages the authentication of users. This means that the BLM checks to see if the user account exists and if the password is valid. If your TSM is part of a large Internet site or if users are using handsets, they may already be authenticated and you do not want them to have to reenter logon information. In this case, the BLM uses the trust mode to authenticate users.

For more information about trust modes, refer to the *BLM Reference Guide*.

Managing Access to BLM Objects

When developing applications that have several users and organizations, your TSM has to be able to limit access to certain features. For example, your TSM may want to allow some users to change their billing address or rate plans. At the same time, you may also want some users to be able to view such information but not be allowed to change it.

You can easily manage the access to the BLM and the CID. By using the built-in security, you can assign access rights for any user of the application.

For more information about managing access to BLM objects, refer to the *BLM Reference Guide*.

Using Explicit Security

About Explicit Security

The access rights to objects are defined by roles and scopes. This security can be enhanced by using the explicit security feature.

You can use this feature to specify that a user in an organization has the rights to manage:

- Other users of the same organization
- Organization contracts

By using this type of security, your applications can have managers. For instance, you can manage users who are contract managers. Theses users have access to different sets of contracts and can manage them as if the contracts belonged to them. And you can also have a manager who assigns the contracts to different contract managers. Using explicit security allows you to enhance the features of your applications and improve security at the same time.

Explicit security has the following limits:

- A manager and the managed users and contracts must belong to the same organization
- The rights of a manager may be different than the rights of an owner

Using Explicit security involves:

- Listing the managers of a contract or user.
- Adding a user to the list of managers of a user or contract
- Removing a user from the list of managers of a user or contract

The examples show you how to use explicit security to have an organization member manage a contract.

Getting Managers of a Contract

Getting all of the managers of a contract involves:

1 Getting the contract

2 Getting the managers of the contract

Get the contract	SessionF blmSession = jspHelper.getBlmSession(); ObjectId contractId = ObjectId.instantiate(request.getParameter ("contractId")); ContractF contract = new ContractF (blmSession,contractId);
Get the managers	UserF[] users = ((ContractF)contract).getManagers();

Specifying a Manager

Adding a manager to a contract manager list involves:

1. Getting the contract
2. Getting the manager
3. Adding the contract to the manager list

Get the contract	SessionF blmSession = jspHelper.getBlmSession(); ObjectId contractId = ObjectId.instantiate(request.getParameter ("contractId")); ContractF contract = new ContractF (blmSession,contractId);
Get the manager	UserF manager = blmSession.getUserF();
Add the contract	manager.doChangeManagedContracts(ChangeMode.ADD, new Contract[]{contract});

Removing a Manager

Removing a manager from a contract manager list involves:

1. Getting the contract
2. Getting the manager
3. Removing the contract from the manager's list

Get the contract	SessionF blmSession = jspHelper.getBlmSession(); ObjectId contractId = ObjectId.instantiate(request.getParameter ("contractId")); ContractF contract = new ContractF (blmSession,contractId);
Get the manager	UserF manager = blmSession.getUserF();
Remove the contract	manager.doChangeManagedContracts(ChangeMode.REMOVE, new Contract[]{contract});

CHAPTER 7

Accessing External Data Sources

In This Section

About Accessing External Data Sources 116
Configuring a New DAL Instance 117
Creating the Configuration File 118
Specifying the Binding Properties 120
Programming the Data Access 121

About Accessing External Data Sources

The DAL is responsible for managing the access to data to and from the CID and external sources. Most likely your TSM uses various sources of data on different computers and maybe even using different databases.

For several reasons, you might want your application to use data from an external data source thus bypassing the CID. Designed with this in mind, you can use the DAL to access SQL-compatible databases or external Java methods.

Accessing external data sources using the DAL involves:

- Declaring a new DAL instance
- Creating configuration files for the new instance
- Specifying the binding properties
- Programming the data access
  - If using SQL to access the data, fill in the queries file
  - If using Java, implement the external function
- Redirect the DAL to the new instance

Configuring a New DAL Instance

When redirecting the data source to manage external data, you must declare an instance of the DAL.

You declare DAL instances in the `instances.properties` customization file. This file is located in `<home_dir>/classes/nmycfg/dal/instances`.

Once you have done this, you create a properties file and enter information such as the database driver to use and its unique instance ID.

To declare a new DAL Instance

1. Go to `<home_dir>/classes/nmycfg/dal/instances`.
2. Open `instances.properties`.
3. Enter a new instance. Use the syntax:

   `instance_name=path of the instance properties file.`

   Example:

   ```
   other=nmycfg.dal.instances.instance_other
   jsource=nmycfg.dal.instances.instance_jsource
   ```

4. Save your changes.

Creating the Configuration File

You create a configuration file for the new DAL instance. This configuration file contains general information about the instance.

To create a DAL instance properties file

1. Go to `<home_dir>/classes/nmycfg/dal/instances`.
2. Create a text file with the name of the instance declared in the `instances.properties` configuration file.

   Example: `instance_other.properties`
3. Open the `.properties` file and enter the following mandatory settings:

SETTINGS	DESCRIPTION
`instance_id`	Internal id used for this instance Must be different than the default id of 0.

`dal_driver`	Driver used for this instance:
	Driver for J2EE application servers:
	`com.netonomy.dal.drivers.impl.sql.jndi.JNDIDatasourceInstance`
	Driver for standalone connection pooling:
	`com.netonomy.dal.drivers.impl.sql.jdbc2x.PooledInstance`
	Driver for standalone generic JDBC:
	`com.netonomy.dal.drivers.impl.sql.basic.BasicInstance`
	Driver to map queries to java code:
	`com.netonomy.dal.drivers.impl.generic.object.ObjectInstance`

`description`	Description of the instance
`queries_file`	DAL configuration file for queries used with this instance

4. Enter the following internal settings and their values:

```
source_name=N/A
login=N/A
max_string_size=1900
native_driver=N/A
password=N/A
debug_driver=N/A
```

5. Save your changes.

Example of a DAL instance properties file:

```
#Internal id used for this instance
instance_id=2
#Driver used for this instance
dal_driver=com.netonomy.dal.drivers.impl.generic.object.ObjectInstance
description=custom java database description
#path to the DAL configuration file for queries used with this instance
queries_file=nmycfg.dal.instances.core_queries_other
# INTERNAL SETTINGS DO NOT MODIFY
source_name=N/A
login=N/A
max_string_size=1900
native_driver=N/A
password=N/A
debug_driver=N/A
```

Specifying the Binding Properties

You specify the binding properties in the `functions_routing.properties` configuration file. This file is located in `<home_dir>/classes/nmycfg/dal/instances`.

To specify the binding properties

1. Go to `<home_dir>/classes/nmycfg/dal`.
2. Open `functions_routing.properties`.
3. For each `object.request`, enter the binding properties.

   Example:

   ```
   contract.getContractedServices=jsource
   ```

4. Save your changes.

Programming the Data Access

You have to program the access to the data. To access the data, you create a DAL queries XML file. This XML file contains:

- The SQL statements if using the JDBC driver to execute SQL queries
- The external Java method if using an object driver to call a method

To create a DAL queries XML file

1. Go to `<home_dir>/classes/nmycfg/dal/instances`.
2. Create an XML file with the name corresponding to the file specified in the `queries_file` setting in the DAL `instance_other.properties` configuration file.

   Example: `core_queries_other.xml`
3. Open the SQL queries file. Add your queries to this file. Use the syntax:

   ```
   <object_name>
              <query_name>SQL statement or external java method</query_name>
   </object_name>
   ```
4. Save your changes.

To implement an external Java method

1. In your Java file, import the following:

   ```
   import com.netonomy.dal.api.DalObjectFactory;
   import com.netonomy.dal.api.DalObjectIF;
   import com.netonomy.dal.util.DalException;
   import com.netonomy.util.ObjectId;
   import com.netonomy.util.StringId;
   import com.netonomy.util.LongId;
   import java.util.ArrayList;
   ```
2. Code the access to the object.

   Example of a test class:

```
package com.netonomy.dal.api.test;
import com.netonomy.dal.api.DalObjectFactory;
import com.netonomy.dal.api.DalObjectIF;
import com.netonomy.dal.util.DalException;
import com.netonomy.util.ObjectId;
import com.netonomy.util.StringId;
import com.netonomy.util.LongId;
import java.sql.*;
import java.util.ArrayList;
public class route
{
public static Object routecs(Object[] params) throws DalException
    {
        String driver_jsource   = "oracle.jdbc.driver.OracleDriver";
        String url_jsource      = "jdbc:oracle:oci8:@cid";
        String login_jsource    = "cid_admin";
        String password_jsource = "cidadmin";
        Statement stmt          = null;
        ResultSet rs            = null;
        Connection con          = null;
        ObjectId contract_id    = null;
        System.out.println("###### JFK1 ########");
        if (params.length > 0)
        {
            contract_id = (ObjectId)params[0];
        }
        System.out.println("###### JFK2 ########");
        String query = "SELECT  CONTRACT_ID,SERVICE_ID,SUBSCRIPTION_DATE FROM "
            + "CONTRACTED_SERVICES WHERE CONTRACT_ID="
            + ObjectId.convertToLong(contract_id);
        System.out.println("###### query :" + query + "########");
        ArrayList list = new ArrayList();// Declare an array list to return ContractedServices
        try
        {
            // select JDBC driver
            Class.forName(driver_jsource);
            // open JDBC connection
            con = DriverManager.getConnection(url_jsource, login_jsource, password_jsource);
            stmt = con.createStatement();
            rs = stmt.executeQuery(query);
            while (rs.next())
            {
                //Create a DalObjectIF to store a contracted service
                DalObjectIF dalobject = DalObjectFactory.createDalObjectIF( "objects.CONTRACTEDSERVICE");
                //Fill the DalObjectIF with the attribute values of the contracted services
                dalobject.setItem("contractID",       new Long(rs.getLong("CONTRACT_ID")));
                dalobject.setItem("serviceID",        new LongId(rs.getLong("SERVICE_ID")));
                dalobject.setItem("subscriptionDate", rs.getDate("SUBSCRIPTION_DATE"));
                list.add(dalobject);
```

```
            }
            // close result set
            rs.close();
            // close statement
            stmt.close();
            // close connection
            con.close();
        }
        // handle JDBC SQL exceptions
        catch (SQLException ex)
        {
            System.out.println("\n*** route.routecs : SQLException caught ***\n");
            while (ex != null)
            {
                //error message
            }
        }
        catch (Exception ex)
        {
            ex.printStackTrace();
        }
        //Create an array of DalObjectIF to return the result of the method
        DalObjectIF[] ret = new DalObjectIF[list.size()];
        list.toArray(ret);
        return ret;
    }
}
```

3 Save your changes.

For more information about the `DalObjectIF` API, refer to the *BLM HTML Reference Documentation*.

You can modify the parameters of the `DalInternalFunction` in the DAL `core_containers.xml` file using the syntax in this manual. By modifying these parameters, you can use your system's legacy ID instead of the internal ID.

CHAPTER 8

Localizing Your Application

In This Section

About Localizing an Application 126
Limitations of Localizing Applications 127
Specifying the Character Set 128
Specifying Application Languages 130
Specifying Language-specific Formats 131
Localizing Database Entries 133
Localizing BLM Error Messages 134
Localizing JSPs 136

About Localizing an Application

In order to adapt your channel to your users, you need to make sure the application uses a language that your users can understand. You may also need to have an application that requires more than one language.

Localizing an application involves:

- Specifying the character set to use
- Specifying the application languages
- Specify the language-specific formats
- Localizing the database entries
- Localizing the BLM error messages
- Localizing the JSPs

When localizing an application, sometimes it is not clear where to go when an element has not been properly localized. If you have problems with dynamic data, you should check the localization of your database entries. If your error messages are not localized, you should check the JSPF configuration file and the BLM error message file. All user interface elements including menus are localized in the JSPF configuration file.

Limitations of Localizing Applications

You can create completely localized applications for users. However, there are some limitations on what you can localize.

The following are not localized:

- System or host names
- Shell and environment variables
- Administration tools and their syntaxes
- Log messages
- Some user interface components such as keyboard shortcuts

Most of these limitations are imposed by Information Technology systems in use today. As these limitations are mostly limited to TSM and database administration, only technical staff see this part of the product.

Along with this part of the application, when you localize your application, you must:

- Ensure that the character encoding supports ASCII along with all of the other characters your application uses
- Ensure that all of the software you use is fully compliant with the character set you use
- Use only one character set in the entire CID
- Use only one encoding for the entire set of JSPs (JSPF and presentation)
- XML file encoding must be 8-bit safe (recommended default is UTF-8)

Specifying the Character Set

Before localizing the interface of your application, you need to make sure your application can handle the appropriate character set.

The default character encoding is ISO-8859-1. This standard covers Latin character based languages widely used in Western Europe, Africa, and the Americas. But if you have to localize your application in a language that is not based on Latin characters, you need to use a Double-Byte Character Set (DBCS). One of the most common examples of this kind of problem is an application that has a Chinese or Japanese and English interface.

If you use a DBCS, your entire system has to be compatible with this kind of character encoding.

Specifying the character set involves:

- Verifying the compatibility of your system. For more information, refer to the release notes on the CD-ROM.
- Declaring the character encoding for the JSPF

Once you declare the DBCS to use in these components, you can localize the rest of your application using the instruction in this section.

The Synchronizer and loopback connector are already configured to use the UTF-8 DBCS. If you need to use another DBCS, refer to *Developing Connectors* for more information about using other character sets.

Be sure to check the limitations and other constraints when using DBCS. For example, HTML allows passwords in UTF-8, but automatically disables passwords being entered using a DBCS.

To declare the character encoding for the JSPF

1. Go to `<home_dir>/channels/WEB-INF/classes/nmycfg/jfn`.
2. Open `jfnApplication.properties`.
3. Change the media.encoding setting to your DBCS. Use the syntax:

   `media.encoding=your_dbcs_name`

   `media.encoding` is the encoding standard to be used when exchanging information with TSM.

When working with HTTP and Java, the standard encoding for exchanging information is UTF8. Application servers written in Java assume that all information sent by browsers is encoded in UTF8. Not all browsers send information in UTF8. This setting informs Java to process all messages as if they are encoded in the specified character set.

4 Go to `<home_dir>/channels/common/fwk`.

5 Open `framework_start.jsp`.

6 Clear the comments in front of the following line of code:

```
//  request = new i18nRequest ("your_dbcs_name",request);
```

7 Replace `your_dbcs_name` with the name of your DBCS.

This value of `your_dbcs_name` must be the same as the encoding you declare in `jfnApplication.properties`.

8 Save your changes.

Specifying Application Languages

Each application has a specific set of available languages. You have to declare the application languages in the CID. In the CID, all of the tables that can be localized have a STRING_ID column. This column contains the ID of the string that is translated. If this column is empty, by default the application returns the string in the NAME column.

This STRING_ID corresponds to entries in the UNIVERSAL_STRING table. The UNIVERSAL_STRING table has the following columns:

- STRING_ID containing the STRING_ID in the table with information to translate
- LANG_ID containing the ID of the language
- STRING_NAME contains the localization of the string

For each string to translate, you insert a row for each language.

To specify application languages

1. In the AVL_APP_LANG table, add a record describing the available language. Use the syntax:

   ```
   insert into AVL_APP_LANG ( APP_ID, LANG_ID ) values (APP_ID value, LANG_ID value);
   ```

   - APP_ID corresponds to the application ID in the APP table
   - LANG_ID corresponds to the Language ID in the LANGUAGE table

2. Save your changes

EXAMPLE OF SPECIFYING APPLICATION LANGUAGES	

Default languages available for MyWeb channel in the CID	`/*For English LANG_ID is 25 */`
	`insert into AVL_APP_LANG (APP_ID, LANG_ID) values (1, 25);`
	`/*For  French LANG_ID is  34 */`
	`insert into AVL_APP_LANG (APP_ID, LANG_ID) values (1, 34);`

Specifying Language-specific Formats

The different languages of your application might require different formats to display information. For example, in North America, dates are usually displayed in a month/day/year format. And in Europe, dates are displayed in a day/month/year format.

You can customize these formats for each language. You use the `jsp_parameters.xml` file to specify the different formats to use in your application. This file is located in `<home_dir>/classes/nmycfg/util/formatter`.

Customizing the formats involves specifying the following:

- Time format
- Date format
- Timestamp format
- Number formats (decimals and separators)

Java uses this XML file to set these formats. The information in this XML file must use the Java standards for format and locale.

For more information, refer to:

- `java.util.Locale` for the contents of language and country
- `java.text.SimpleDateFormat` for the syntax of date formats
- `java.text.DecimalFormat` for the syntax of decimal formats

To specify language specific formats

1. Go to `<home_dir>/classes/nmycfg/util/formatter`.
2. Open `jsp_parameters.xml`.
3. Under `<FORMATS>`, add the `<LANGUAGE.LANG_CODE>` element. Use the syntax:

```
<LANGUAGE.LANG_CODE>
          <language>java_locale_language_code</language>
          <country>java_locale_country_code</country>
          <time_format>time_format</time_format>
          <date_format>time_format</date_format>

<timestamp_format>timestamp_format</timestamp_format>
          <int_format>int_format</int_format>
```

```
            <float_format>float_format</float_format>
 </LANGUAGE.LANG_CODE>
```

The `LANGUAGE.LANG_CODE` must be one of the languages declared as an application language in the `AVL_APP_LANG` table.

4 Under the root element, enter the default language to use. Use the format:

`<DEFAULT>LANGUAGE.LANG_CODE</DEFAULT>`

The default language must correspond to a language declared in this file.

5 Save your changes.

EXAMPLE OF A JSP_PARAMETERS.XML FILE

Properties for English (UK) formats

```
<XML_CONFIGURATOR __IS_HASH__="true">
       <formats>
                <en>
                        <language>en</language>
                        <country>UK</country>
                        <time_format>HH:mm:ss</time_format>
                        <date_format>MM/dd/yyyy</date_format>
                        <timestamp_format>MM/dd/yyyy HH:mm:ss</timestamp_format>
                        <int_format>#,###,###,###</int_format>
                        <float_format>#,###,###,###.##</float_format>
                </en>
```

Properties for French (France) formats

```
<fr>
                        <language>fr</language>
                        <country>FR</country>
                        <date_format>dd/MM/yyyy</date_format>
                        <time_format>HH:mm:ss</time_format>
                        <timestamp_format>dd/MM/yyyy HH:mm:ss</timestamp_format>
                        <int_format>#,###,###,###</int_format>
                        <float_format>#,###,###,###.##</float_format>
                </fr>
```

Default format to use

```
       <default>en</default>
</XML_CONFIGURATOR>
```

Localizing Database Entries

In the CID, all of the tables that can be localized have a STRING_ID column. This column contains the ID of the string to translate. If this column is empty, by default the application returns the string in the NAME column.

This STRING_ID corresponds to entries in the UNIVERSAL_STRING table. The UNIVERSAL_STRING table has the following columns:

- STRING_ID containing the STRING_ID in the table with information to translate
- LANG_ID containing the ID of the language
- STRING_NAME contains the localization of the string

For each string to translate, you add a record for each language.

Some of your database entries may have more than one string to localize. For instance, you have to provide the localization of a service name as well as its description. In this case, the columns of additional strings to localize end in STRING_ID.

To localize the database entries

1. In the table containing strings to translate, enter a unique STRING_ID in each record to localize.
2. In the UNIVERSAL_STRING table, add a record containing the following for each language:
   - STRING_ID of the record to localize
   - LANG_ID of the language
   - STRING_NAME the localized string
3. Save your changes

EXAMPLE OF CID LOCALIZATION OF A DOCUMENT	
Create a new document called MyDocument	`insert into DOC ( DOC_ID, DOC_LEGACY_ID, DOC_NAME, STRING_ID, DOC_DESCRIPTION, DOC_DESC_STRING_ID, DOC_CATEGORY_ID, DOC_START_DATE, DOC_END_DATE ) values ( 5, 'LEG_5', 'MyDocument', 47005, 'A document describing something', 47305, 1, Null, Null );`

Localize DOC_NAME and DOC_DESCRIPTION in French	`insert into UNIVERSAL_STRING(STRING_ID, LANG_ID, STRING_NAME) values(47005, 34, 'MonDocument');`
	
	`insert into UNIVERSAL_STRING(STRING_ID, LANG_ID, STRING_NAME) values(47305, 34, 'Un document decrivant quelquechose');`

Localizing BLM Error Messages

The BLM has a set of files that the BLM uses to localize internal BLM messages. The internal BLM messages include business logic and security messages.

You use the following files for the BLM error messages:

- `<home_dir>/classes/nmycfg/util/translator.properties` contains the location of the error settings for each language
- `<home_dir>/classes/nmycfg/errors/<language_filename>` contains the location of the localized error message file
- `<home_dir>/classes/nmycfg/errors/<localization_file>` contains the error messages

Localizing the BLM error messages involves:

- Declaring the language and location of the error settings
- Creating the error settings
- Creating the localization file containing the localized messages

To localize BLM error messages

1. Go to `<home_dir>/classes/nmycfg/util`.
2. Open `translator.properties`.
3. Enter the message file to use. Use the format:

   `<language_code>=nmycfg.errors.<language_filename>`

   - `language_code` is the two letter language code of one of the available application languages
   - `language_filename` is the name of the .properties configuration filename containing the location of the localization file.

   The language_code must be one of the languages declared as an application language in the `AVL_APP_LANG` table.

4. Go to `<home_dir>/classes/nmycfg/errors`.
5. Open `<language_filename>` and enter the following:

```
__INCLUDES_EXTERNAL_BUNDLES__=1
__BUNDLE_1_add_method=ADD
__BUNDLE_1_extension=EXTENSION_DEFAULT
__BUNDLE_1_format=TYPE_PROPS_NORMAL
__BUNDLE_1_file=nmycfg.errors.<localization_file>
```

- `localization_file` is the `.properties` configuration file containing the localized error messages.

Enter only the name of the file. The BLM automatically appends the .properties extension to this name.

6 Open `<localization_file>.properties` and enter the message. Use the format:

`ERROR_NUMBER=Message`

When adding a new language, you can copy and rename `core_english.properties`. You can then localize the message in the corresponding language.

7 Save your changes.

EXAMPLE OF BLM MESSAGE LOCALIZATION	

Contents of `translator.properties`	`Default=en`
	`en=nmycfg.errors.english`
	`fr=nmycfg.errors.french`

Contents of `errors.english.properties`	`__INCLUDES_EXTERNAL_BUNDLES__=1`
	`__BUNDLE_1_add_method=ADD`
	`__BUNDLE_1_extension=EXTENSION_DEFAULT`
	`__BUNDLE_1_format=TYPE_PROPS_NORMAL`
	`__BUNDLE_1_file=nmycfg.errors.core_english`

Contents of `errors.french.properties`	`__INCLUDES_EXTERNAL_BUNDLES__=1`
	`__BUNDLE_1_add_method=ADD`
	`__BUNDLE_1_extension=EXTENSION_DEFAULT`
	`__BUNDLE_1_format=TYPE_PROPS_NORMAL`
	`__BUNDLE_1_file=nmycfg.errors.core_french`

Contents of `errors.core_english.properties`	`# ------------------------------------------------------------------------`
	`# BLM English Business Logic Messages.`
	`# ------------------------------------------------------------------------`
	`0={0} Access Denied`
	`1=Access Denied. The session is not valid nor authenticated.`
	`2={0} Access Denied`

Contents of `errors.core_french.properties`	`# ------------------------------------------------------------------------`
	`# Messages de logique applicative en français.`
	`# ------------------------------------------------------------------------`
	`0={0} accès refusé!`
	`1=Session invalide ou non authentifiée, accès refusé!`
	`2={0} accès refusé!`

Localizing JSPs

You use the following to localize framework JSPs:

- The JSPF configuration file
- The BLM API

The JSPF configuration file is an XML file containing application properties and various JSPF settings. By default, the channels use the `MyWeb.xml` JSPF configuration file. This file is located in `<home_dir>/channels/WEB-INF/classes/nmycfg/jfn`.

You declare the default language in this file. The `<property name="APP_LANG_CODE">` element specifies the code of the default language.

The JSPF configuration file also contains the strings to use for each language. The `<string>` element contains the strings the application uses. Each string in the JSP to localize has an entry in the JSPF configuration file. For each string, one or more value elements contain the localization of the string. The syntax is:

```
<string name="string_name">
        <value name="language_code1">String1</value>
        <value name="language_code2">String2</value>
  </string>
```

The `name` attribute corresponds the one of the application language codes entered in the CID.

Localizing strings in JSPs involves:

- Specifying your character set (if required)
- Localizing simple strings in the JSP
- Localizing strings using Java's `java.text.MessageFormat`

If your text is easy to translate and you do not need to change the order of information contained in the string, you can use the simple localize function.

But if your localization requires information in your JSP to be displayed in a different order, or if you need to introduce some dynamic information into your strings, you can use `java.text.MessageFormat` to help you do this. For example, your JSP displays a confirmation page and you want to display a message asking users to confirm their choice. Localizing this kind of question might require you to display information in a different order.

Functional steps can also override the localization in the JSPF configuration file.

To set the default language of the JSPF

1. Open the JSPF configuration file.
2. Under `<config>` find the `<property name="APP_LANG_CODE">` element.
3. In this element, enter the language code of the default language.

   Example of the declaration of English as the default language:

   ```
   <property name="APP_LANG_CODE">en</property>
   ```
4. Save your changes.

To specify the encoding in the JSPF configuration file

1. Open the JSPF configuration file.
2. In the XML declaration, enter your encoding. Use the syntax:

   ```
   <?xml version="1.0" encoding="your_dbcs_name"?>
   ```

   Note: XML file encoding must be 8-bit safe (recommended default is UTF-8)
3. Save your changes.

To localize simple strings in JSPs

1. Open the JSP you want to localize.
2. For each string, use the JSPF localize function. Use the syntax:

   ```
   <%=jspHelper.localize("string_name", "default_string")%>
   ```

   - `string_name` is the name of the string in the JSPF configuration file.
   - `default_string` is the string to display if the `string_name` is not found in the JSPF configuration file.
3. Open the JSPF configuration file and locate the entry of the JSP.
4. Under the `<jsp>` tag, enter the strings to display. Use the format:

   ```
   <string name="string_name">
         <value name="language_code1">String1</value>
         <value name="language_code2">String2</value>
   </string>
   ```
5. Save your changes

EXAMPLE OF SIMPLE JSP LOCALIZATION	
In the login.jsp	`<td><%=jspHelper.localize("login_user_name_field","default_text")%></td>`

EXAMPLE OF SIMPLE JSP LOCALIZATION	
In the MyWeb.xml JSPF configuration file	<jsp name="login.jsp" authenticate="false"> <string name="login_user_name_field"> <value name="en">User name:</value> <value name="fr">Identifiant :</value> </string> ... </jsp>

To localize strings using java.text.MessageFormat

1. Open the JSP you want to localize.
2. For each string, use the JSPF localize function. Use the syntax:

   ```
   <%=jspHelper.localize("string_name", "default_string", new
   Object[]{your code to get arguments for
   java.text.MessageFormat})%>
   ```

   - `string_name` is the name of the string in the JSPF configuration file.
   - `default_string` is the string to display if the `string_name` is not found in the JSPF configuration file.
   - `new` is an array of attributes for the java formatter.
3. Open the JSPF configuration file and locate the entry of the JSP.
4. Under the <jsp> tag, enter the strings to display. Use the syntax:

   ```
   <string name="string_name">
         <value
   name="language_code1">String{0}string{1}</value>
         <value
   name="language_code2">{0}Stringstring{1}</value>
   </string>
   ```
5. Save your changes.

For more information about using `java.text.MessageFormat`, refer to your copy of the Java API Specification.

EXAMPLE OF USING JAVA.TEXT.MESSAGEFORMAT

In the `service_change_confirm.jsp`

```
<td><%=jspHelper.localize("service_replace_confirm_quantity_text","default_text",newObject[]
{
                request.getParameter("quantity"),
                objService.getName (),
                objContract.getPhoneNumber ()
                        })%></td>
```

In the MyWeb.xml JSPF configuration file

```
<jsp name="login.jsp" authenticate="false">
...
<string name="service_replace_confirm_quantity_text">
        <value name="en">Are you sure you want to add {0} {1} to contract {2}.</value>
        <value name="fr">Confirmez l'ajout de {0} {1} au contrat {2}.</value>
</string>
...
</jsp>
```

CHAPTER 9

Managing Reference Data

In This Section

About Reference Data 142
Returning All Reference Data 143
Returning Only Certain Types of Reference Data 144
Reloading Reference Data 145

About Reference Data

The CID contains a large set of reference data for TSM. Reference data includes things like country names, currencies, User Roles, and so on. The BLM comes with the `objectRefMgr` that is a functional interface to easily access this kind of information.

You can use the `objectRefMgr` to access objects without having mandatory prerequisites. By using this interface, you can access reference information available to all objects in the BLM. You can use the `objectRefMgr` to return lists of reference information or specific items of a list.

For more information about the `objectRefMgr`, refer to the *BLM API Reference* Documentation.

Returning All Reference Data

This class has a set of methods to return the entire list of reference objects such as rate plans, payment methods and so on. These methods have the following syntax:

```
getAll<Object_Name>()
```

Some of the methods include:

- `getAllRateplans()`
- `getAllPaymentMethods()`

Returning Only Certain Types of Reference Data

For several reasons, you may not want to return the entire list of reference data for a specified object. You may just want to use the objects that match specific criteria or have a certain code.

This class also has a set of methods to filter the results using your criteria. These methods have the following syntax:

- `get<Object_Name>ByFilter(filter)` to return objects matching the specified criteria
- `get<Object_Name>ByCode(code)` to return objects matching the specified code

Reloading Reference Data

In general, your reference data should not change frequently. But on occasion, you may need to change this data. You can use the reference data reload feature to program your application to reload reference data without having to take your application off line.

You cannot use this feature to remove reference data and it cannot be used to add new types of reference data. This feature is for reloading modified and new reference information only. If you modify the data structure or remove reference information, you must stop and restart your application server.

How the Internal BLM Cache Works

The BLM cache is a global cache for all user sessions. All of the cached BLM objects are one of following types:

- `GLOBAL` The object is cached the entire life of the BLM host process.
- `RELOADABLE` The object is cached and can be updated using the reference data reload feature.
- `HTTP_REQUEST` The object is cached and updated for each HTTP request.

The `policy.properties` configuration file contains the list of objects and their assigned type. This file is located in `<home_dir>/classes/nmycfg/blm/util`.

The `VERSIONS` table contains information about the reference data. This table contains the version of the reference data and its associated timestamp.

Updating Reference Objects in the Cache

By using the session management features, the BLM can check the CID to see if the reference data has been modified.

You use the `session.Add()` to attach the session to a thread that performs the HTTP requests. Every time you call the `session.Add()` method, the BLM automatically checks the `VERSIONS` table to see if the timestamp has changed. If the reference data has changed, the BLM updates all of the RELOADABLE objects in the cache and starts the session.

Updating reference objects in the cache involves:

- Specifying the caching policy
- Creating batch files to reload the reference data

To specify the caching policy of BLM objects

1. Go to `<home_dir>/classes/nmycfg/blm/util`.
2. Open `policy.properties`.
3. Find the BLM object to modify.
4. Do one of the following:
   - To cache object the entire life of the BLM host process, change the setting to `GLOBAL`
   - To update the object using the reference data reload feature, change the setting to `RELOADABLE`.
   - To update the object for each HTTP request, change the setting to `HTTP_REQUEST`
5. Save your changes.
6. Restart your application server.

Do not change the Reference Data declared in this file. This data and its caching policies are for internal use only and must not be modified.
To activate the reference data reload feature, you must change ALL of the `GLOBAL` objects to `RELOADABLE`. You **cannot** mix `GLOBAL` and `RELOADABLE` cache settings.

To create a batch file to reload reference data

1. Create a batch file to extract and load information into the CID.
2. At the end of the transaction, do the following in the `VERSIONS` table:

   When `ITEM_CODE='REFERENCE_DATA'`, do the following:
   - Replace `ITEM_VERSION with the version of your reference data`
   - Replace `ITEM_TIMESTAMP with the current date`

Do not insert or change the information in the `STRUCTURE` data. This data is for internal use only and must not be modified.

Programming your Application for Reference Data Reloads

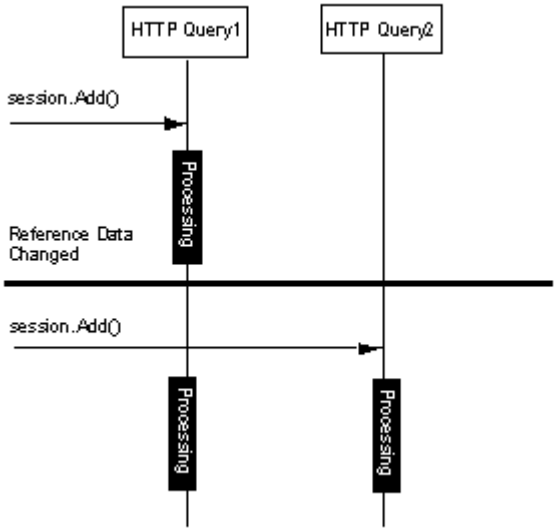
When programming your application to use the reload feature, you must keep in mind that it is the application logic that:

- Checks the timestamp
- Manages the workflow changes

As you know which reference data may change, you can program your application to take changes into account. Your application JSP should do the following:

1. Open and validate the session.
2. Use the `ObjectRefMgr.getReferenceDataTimestamp()` to return the time stamp of the current reference data.
3. Store the timestamp in the HTTP session.
4. When the thread handles an HTTP request within the current session, retrieve the timestamp and compare it with the stored timestamp.
   - If the timestamps differ, the reference data has changed and the BLM cache has been updated. You can change the workflow (display a message, reset the workflow, and so on).
   - If the timestamps are identical, the application continues normally.

Concurrent user sessions may cause some problems and you should design your application to take the following sequence of events:



In this sequence of events, the HTTP Query2 causes the BLM cache to be updated because the reference data changed. In this case, the HTTP Query 1 cannot be notified and the BLM cache might not contain data needed by the query. This is especially important if your data reload contains modified reference information. This situation does not cause problems when you reload only new reference information, because the threads do not need to know about new information.

In order to avoid this situation, you can:

- Use this feature only to load modified or new information in existing reference tables
- Block access to the impacted service during reloads (redirect to another JSP, display a message)
- Limit the frequency of reloads
- Reload your data during off-peak hours

For more information about the reload feature, refer to *Administrating Telco Service Manager*.

Example of JSPs Using the Reference Data Reload Feature

The following code is an example of the JSPF JSPs of the MyWeb channel.

Each time the application calls `blmSession.add`, the code extracts the timestamp and compares them. If they are different, the JSP displays a message. You can also customize it to call another JSP, change the workflow, end the session, or whatever your application requires.

Using the reference data reload sample involves:

- Changing the properties of BLM objects
- Entering the code in the JSPF
- Entering the message strings in the JSPF configuration file
- Running your application

To customize the application to reload reference data

1. Go to `<home_dir>/classes/nmycfg/blm/util`.
2. Open `policy.properties`.
3. Change all of the reference data objects to `RELOADABLE` and save your changes. The objects in the BLM cache can now be reloaded.

4 Go to <home_dir>/channels/common/fwk.

5 Open framework_start.jsp.

6 Go to the blmSession.add () function.

7 Under the blmSession.add () function, enter the following:

```
   // We get the CID Ref Data Timestamp and the last saved  Reference Data Time
 Date current = ObjectRefMgr.getReferenceDataTimestamp ();

 Date oldDate = (Date)session.getValue ("REF_DATA_TIME");

   if (!current.equals (oldDate))
      {
      // The date has changed, save the new one
      session.putValue ("REF_DATA_TIME", current);
      // If we have already saved an old date
      if (oldDate != null)
        {
// The date of the data reference has changed. So we display an error
          Hashtable parameters = new Hashtable ();
          parameters.put ("title",jspHelper.localize("jfn_warning","Warning"));

          parameters.put ("section",jspHelper.localize("jfn_data_ref","Refence Data"));

          parameters.put ("message",jspHelper.localize("jfn_data_ref_reload","Reference data change"));

          response.sendRedirect (jspHelper.encodeURLFunct("GLOBAL.MESSAGE", parameters, true));
          exitJSP ();
        }
      }
```

This code tells the JSPF to compare reference data timestamps. If the timestamps are different, the JSPF displays a message.

8 Save your changes.

9 Go to <home_dir>/channels/WEB-INF/classes/nmycfg/jfn.

10 Open MyWeb.xml.

11 Create a framework_start.jsp strings section, then add the following:

```
<!-- Strings for framework_start.jsp -->

<!-- message strings -->

<!-- text strings -->

<string name="jfn_data_ref">

   <value name="en">Configuration Error</value>

     <value name="fr">Erreur de configuration</value>

</string>

<string name="jfn_data_ref_reload">

   <value name="en">The reference data has been changed.</value>

   <value name="fr">Les données de référence ont été changées.</value>

</string>
```

12 Save your changes.

MyWeb can now reload reference data and display a message telling users when it happens.

To run the Reference Data Reload sample

1 Start your application server and open the MyWeb index.jsp. The login page appears.

2 Log in using the following:

- UserName: tammy
- Password: tammy

The home page appears.

3. Click *change your rateplan*. The *Manage Contracts* page appears.
4. Open your database administration tool and change the reference data timestamp.
5. Commit your changes. You have changed the reference data timestamp. The next time the application calls `session.Add`, the BLM updates its cache.
6. Go back to MyWeb and click *View and Modify Contracts*. MyWeb does the following:
   - Compares the timestamps and finds them different
   - Empties the internal BLM cache
   - Displays a message

CHAPTER 10

Managing Changes to BLM Objects

In This Section

About Changes to BLM Objects 152
Managing Basic Changes to Objects..... 153
Managing Changes with the ActionManager 154
Managing Changes in Synchronous Mode 155

About Changes to BLM Objects

When users use TSM, they can modify BLM Objects. With simple modifications, you can manage these changes directly with the BLM. For more complex modifications, you use the ActionManager to manage how your application manages these changes.

You use the ActionManager if your application requires:

- Users creating complex requests such as composite requests.
- Users need to be able to change or remove requests before submitting them.
- User requests must be submitted in synchronous mode.

Managing Basic Changes to Objects

This example is a simple add service request. This sample sends the modification to the BLM which then inserts the request in the CID

MANAGING SIMPLE CHANGES TO OBJECTS

```
UserF     user     = session.getUserF();
ContractF contract = user.getContracts()[0];
ListAddableServiceHelper lash = null;
AddableServiceIF         svc  = null;
ParameterIF[]            params = null;
RequestIF                rqst;
/*
*  Sample to directly insert an add service request into CID
*/
lash = contract.getSubscriptableServices(null);
svc  = lash.get(0);
params = svc.getDefaultParameters();
// Add Service request is directly inserted into CID
rqst = contract.addService(null, svc, params, null);
```

Managing Changes with the ActionManager

This example shows how to use the ActionManager to create a holder for requests. The `ActionMgr` holds the requests until told to send the requests to the BLM for insertion in the CID.

You use this feature to create shopping carts because the requests are held in the action manager itself. As they are not sent to the BLM, users can modify all kinds of information. Once they are finished, they validate the changes and the action manager sends the information to the BLM for processing.

MANAGING CHANGES TO OBJECTS WITH AN ACTION MANAGER

```
UserF     user     = session.getUserF();
ContractF contract = user.getContracts()[0];
ListAddableServiceHelper lash = null;
AddableServiceIF         svc  = null;
ParameterIF[]            params = null;
RequestIF                rqst;


/*
*  Sample to create a shopping cart with an add service request in it
*/
ActionMgrIF order = BlmFacade.createOrder(null);
lash = contract.getSubscriptableServices(order);
svc  = lash.get(0);
params = svc.getDefaultParameters();
// Add Service request is stored in product cart, but not inserted in CID
rqst = contract.addService(order, svc, params, null);
// Product cart request is inserted into CID
//   (-> add service rqst is now inserted as well)
RequestIF rqst_order = order.submit(RequestIF.SUBMIT_MODE_NORMAL);
```

For more information, refer to *Working with Shopping Carts* in this manual.

Managing Changes in Synchronous Mode

This example shows how to send changes to the BLM for processing in synchronous mode. This example sends changes directly to the BLM which then inserts the request directly in the CID.

MANAGING CHANGES TO OBJECTS IN SYNCHRONOUS MODE

```
UserF     user     = session.getUserF();
ContractF contract = user.getContracts()[0];
ListAddableServiceHelper lash = null;
AddableServiceIF         svc  = null;
ParameterIF[]            params = null;
RequestIF                rqst;
/*
*  Sample to insert an add service request in CID with "synchronous" mode
*/
ActionMgrIF actionmgr = BlmFacade.createActionManager();
lash = contract.getSubscriptableServices(actionmgr);
svc  = lash.get(0);
params = svc.getDefaultParameters();
// Add Service is stored in action manager, but not inserted in CID
rqst = contract.addService(actionmgr, svc, params, null);
// Add service request is inserted into CID with "synchronous" mode
actionmgr.submit(RequestIF.SUBMIT_MODE_SYNCHRONOUS_NORMAL_ON_ERROR);
```

CHAPTER 11

Working with Shopping Carts

In This Section

About Shopping Carts 158
About the BLM Interfaces 159
Before Developing Shopping Carts 160
Creating a Simple Shopping Cart 162
Managing Complex Shopping Cart Contents 165
Modifying a Shopping Cart Item 174
Displaying the Contents of a Shopping Cart 180
Saving Shopping Carts 184
Using Shopping Cart Templates 187

About Shopping Carts

You can use the shopping cart feature to greatly enhance TSM. This feature is available in the MyWeb channel and involves both the Presentation and Business Logic layers.

This feature is the key to building an application where end-users are able to group, check and modify their orders before submitting them in a single click.

This chapter covers the technical details of the shopping cart and the entities you handle while developing shopping carts for your application. It covers shopping carts from a simple implementation that handles a single order to a fully-featured shopping cart where complex objects are handled. You also see how to save the contents of shopping carts and use the saved shopping cart contents.

About the BLM Interfaces

When creating a shopping cart, you use the following BLM API interfaces:

- ActionItem (`ActionItemIF`)
- ActionManager (`ActionMgrIF`)

These two interfaces are used to manage the persistence of modifications made to BLM objects. This means that the shopping cart feature is technically the use of Action Managers.

The principle of a shopping cart is to associate an Action Manager or an Action Item to BLM objects. Once you do this, you can manage the changes made to these objects and decide when to submit the associated requests.

The Action Manager interface inherits the Action Item interface. That means Action Manager and Action Item interfaces are used for similar actions.

The Action Manager interface offers the capability of creating children items. This capability is very helpful when you have to group actions together as a major one.

Before Developing Shopping Carts

Before you start developing your shopping carts, you should read this section. This section covers important information you need to keep in mind while developing your shopping cart. It also gives an overview of the samples and methodology covered in the examples.

Action Manager Hierarchies

Action Managers support hierarchies.

You can use this feature to:

- Handle several order type actions
- Handle actions on objects that depend on each other

Action Manager Types

The BLM handles two kinds of Action Managers:

- Action Managers for Orders

  With this kind of action manager, you can handle several actions within the same action manager. You can use this action manager for almost any action except creating an organization, because the BLM comes with a special type of action manager for creating single objects such as organizations. For example, you can handle the creation of:

  - Several new contracts
  - Several new members
  - Several new services

  You use the `BlmFacade.createOrder()` method to handle this kind of shopping cart.
- Action Managers for Creating Single Objects

  With this kind of action manager, you can handle the creation of a new complex object:

  - New organization
  - New member
  - New contract

  You use the `BlmFacade.createActionManager()` method to handle this kind of shopping cart.

About the Presentation Layer

The MyWeb channel has a presentation layer that:

- Handles only one shopping cart in a given HTTP session.

  Because there are two distinct kinds of shopping carts, not all of the actions can be mixed and be handled as persistent in shopping carts within the same HTTP session.
- Provides helpers in the JSPF to handle the shopping cart objects.

  By using these helpers, you do not have to interact with the BLM facade.

Creating a Simple Shopping Cart

To help you understand how to handle the basic workflow of a shopping cart, you can use the example of a simple shopping cart. The examples in this section show you how to manage Add Service workflows. In your application, instead of directly submitting an `AddService` action, you put it into a shopping cart and when you submit the shopping cart, the `AddService` action is processed and all of the required requests are generated.

Creating a simple shopping cart involves:

1. Declaring and retrieving the shopping cart
2. Listing the services the user can add
3. Adding the selected services to the shopping cart
4. Submitting the shopping cart

Declaring and Retrieving the Shopping Cart

The shopping cart is stored in the HTTP session. In each form handler page you use to manage your shopping cart, retrieve the reference of the shopping cart object.

The JSPF `JFNJspHelper` class contains a method that helps you Declare and retrieve a Shopping Cart. The `getCurrentShoppingCart()` method returns a shopping cart object handle for the current user session:

- If the shopping cart object has already been created, the method returns the object reference.
- If the shopping cart object does not exist, the method creates the object and returns its object reference.

This sample code shows how to retrieve the shopping cart object reference:

```
ActionMgrIF shoppingCart = null;

shoppingCart = jspHelper.getCurrentShoppingCart();
```

The `shoppingCart` variable now contains the shopping cart for the current user.

Listing Services in the Shopping Cart

When you list the services, you need to make sure you do not list services that have already been added into the current shopping cart.

The `getAddableServices()` method has a filtering feature to do this. You call the method and pass the current shopping cart object reference as an input parameter:

```
ActionMgrIF shoppingCart = null;
shoppingCart = jspHelper.getCurrentShoppingCart();
AddableServiceIF[] addableServices;
addableServices = contract.getAddableServices (shoppingCart, null, …);
```

Adding Services to the Shopping Cart

To add a service to your shopping cart, you put an add service action into the shopping cart. You call the `addService()` method and pass the current shopping cart object as the first input parameter.

This sample code puts an add service action into the shopping cart. This code assumes that you have already retrieved the reference of the Contract object that the service will be added to.

In this example, the first argument is the shopping cart that was just returned. The other parameters come from the form in the JSP.

The add service action is now handled by the shopping cart

```
ActionMgrIF shoppingCart = null;
shoppingCart = jspHelper.getCurrentShoppingCart();
contract.addService (shoppingCart, ratePlanService, parameters, quantity, …);
```

Submitting the Shopping Cart

When finished, your workflow may have a summary page with a Submit button. This button submits the contents of the shopping cart.

This sample code submits the contents of the shopping cart and removed the shopping cart object handler from the user session. This code is located in the `logic_handler` page of the workflow.

```
shoppingCart = jspHelper.getCurrentShoppingCart();
shoppingCart.submit(RequestIF.SUBMIT_MODE_NORMAL);
jspHelper.destroyCurrentShoppingCart();
```

The `submit()` method throws an exception if a problem occurs when submitting the contents of the cart.

Once the contents of the cart has been submitted, you should use the `destroyCurrentShoppingCart()` method to destroy the shopping cart. This method removes the shopping cart object reference from the HTTP session.

Managing Complex Shopping Cart Contents

Your application may require shopping carts that are more complex than the simple shopping cart. For example, your application may let users change services that are in a shopping cart.

These examples show you how to handle several levels of actions inside your shopping cart, taking advantage of the hierarchy of Action Manager objects. You can also see how to retrieve an object from the shopping cart, complete it, and then update it in the shopping cart.

The following examples show you how to create:

- A shopping cart that allows existing users to add contracts and services to these contracts
- A shopping cart that allows first-time users to sign up and add contracts and services to the contracts

Creating a Complex Shopping Cart for Contracts

To help you understand how to handle users adding contracts then adding services to this contract, you can use the example of a complex shopping cart.

Creating an Add Contract / Add Service shopping cart involves:

1. Declaring and retrieving the shopping cart
2. Creating a child action manager
3. Adding the contract to the shopping cart
4. Adding a service using the child action manager
5. Submitting the shopping cart

Declaring and Retrieving the Shopping Cart

The shopping cart is stored in the HTTP session. In each form handler page you use to manage your shopping cart, retrieve the reference of the shopping cart object.

The JSPF `JFNJspHelper` class contains a method that helps you Declare and retrieve a Shopping Cart. The `getCurrentShoppingCart()` method returns a shopping cart object handle for the current user session:

- If the shopping cart object has already been created, the method returns the object reference.
- If the shopping cart object does not exist, the method creates the object and returns its object reference.

This sample code shows how to retrieve the shopping cart object reference:

```
ActionMgrIF shoppingCart = null;
shoppingCart = jspHelper.getCurrentShoppingCart();
```

The `shoppingCart` variable now contains the shopping cart for the current user.

Creating a Child Action Manager

You use Child Action Managers to manage Action Manager hierarchies.

You use them to manage actions and sub actions to objects in the shopping cart.

In the case of managing complex shopping carts, we create new contract, add a service to this new contract and submit all the requests. In this case, the Add Service action is considered as a sub action of the Create Contract action.

When doing this, the Create Contract action should be handled through a child Action Manager to ensure that a hierarchy of Action Manager objects is created.

This table contains a list of Action/Subactions you may have to process and which require a hierarchy of Action Manager objects to be handled:

ACTION	SUBACTIONS
Add Organization	Add Member Add Contract Migrate contract Add legal contact Add billing contact Add billing account Declare payment responsible • Add level • Add manager

ACTION	SUBACTIONS
Add Member	Add contract Migrate contract Add legal contact Add billing contact Add billing account Declare payment responsible • Set language • Set persnalization data • Add login
Add Contract	Add service
Migrate Contract	Add service
Add level	Add legal contact • Add billing contact • Add billing account • Declare payment responsible • Add member • Add contract • Migrate contract

This sample code shows you how to create a child Action Manager in the current shopping cart:

```
ActionMgrIF shoppingCart = null;
shoppingCart = jspHelper.getCurrentShoppingCart();
ActionMgrIF actionMgrChild;
actionMgrChild = shoppingCart.createChild();
```

Adding a Contract

Once the child Action Manager is available, you post the Create Contract action to the shopping cart using this child Action Manager object.

This sample code shows you how to post the Create Contract action:

```
ActionMgrIF shoppingCart = null;
shoppingCart = jspHelper.getCurrentShoppingCart();
ActionMgrIF actionMgrChild;
ActionMgrChild = shoppingCart.createChild();
ContractIF contract;
contract = organization.createContract (actionMgrChild, null, ratePlan, contractType,…);
```

The Create Contract action is linked to the `actionMgrChild` object. As the `actionMgrChild` object is one of the shopping cart Action Manager child objects, the Create Contract action is handled by the shopping cart.

You now need to retrieve the Contract object that is created by the Create Contract action.

Adding a Service to the Contract

Once you have created the contract in the shopping cart, you can get a reference to this object, handle it, and perform actions such as Add Service.

To add a service to a contract in a shopping cart involves

1. Getting the Action Manager object related to the newly created contract (this is actually the Action Manager child `<actionMgrChild>` object).
2. Retrieving handle to the contract object.
3. Adding a service to this contract by using the `addService()` method.

The sample code below assumes that:

You implement the Add Service action in a logic handler and the Create Contract action in another.

You pass the Id of the Action Manager that handled the contract creation to the second form handler through the HTTP request.

You get Action Manager ID by using the `ActionManagerIF.getIdentifier()` method.

The sample code shows you how to retrieve the action manager object reference through the HTTP request parameters.

For this example, this code retrieves the Action Manager child object used for the Create Contract action.

```
ActionMgrIF shoppingCart = null;
shoppingCart = jspHelper.getCurrentShoppingCart();
ActionItemIF item;
ActionMgrIF actionMgrChild;
item = shoppingCart.getActionItem (ObjectId.instantiate(request.getParameter("action_id")),true);
actionMgrChild = (ActionMgrIF) item;
```

Once you retrieve the Action Manager child object, you can retrieve the Contract object it handles.

The sample code shows you how to retrieve the Contract object:

```
ContractIF contract;

contract = (ContractIF)actionMgrChild.getObject();
```

The getObject method always returns the created object.

Now you can add the service to the contract.

Adding a service to the contract is the same standard way described above, but in this workflow you work with the child Action Manager and not the shopping cart object.

```
contract.addService (actionMgrChild, ratePlanService, parameters, quantity, …);
```

The multi-form_handler workflow we have described is based on the capability to pass action manager Ids through HTTP requests and knowing the parameters coming from the form handler that handles the Add Service action.

The `getObject` Java call casts the results to a `ContractIF` type to ensure you retrieve the object from the Action Manager as a Contract.

When implementing the second form handler, the action manager you get from the HTTP request handles a contract object.

Submitting the Shopping Cart

When finished, your workflow may have a summary page with a Submit button. This button submits the contents of the shopping cart.

This sample code submits the contents of the shopping cart and removed the shopping cart object handler from the user session. This code is located in the `logic_handler` page of the workflow.

```
shoppingCart = jspHelper.getCurrentShoppingCart();

shoppingCart.submit(RequestIF.SUBMIT_MODE_NORMAL);

jspHelper.destroyCurrentShoppingCart();
```

The `submit()` method throws an exception if a problem occurs when submitting the contents of the cart.

Once the contents of the cart has been submitted, you should use the `destroyCurrentShoppingCart()` method to destroy the shopping cart. This method removes the shopping cart object reference from the HTTP session.

Creating Customers in the Shopping Cart

To help you understand how to handle creating a new customer and then adding a new contract, you can use this example of a complex shopping cart.

Creating an Add Customer shopping cart involves:

1. Specifying the correct action manager
2. Adding a customer
3. Adding a contract
4. Submitting the shopping cart

Specifying the Action Manager

There are two kinds of Action Managers:

- Action Managers for Order requests
- Action Managers for Create Organization requests

When creating customers, you have to explicitly create an ActionManager object from the BLM facade. When you do this, you get an Action Manager that supports Create Organization actions.

The sample code shows you how to return a Create Organization action manager:

```
ActionMgrIF shoppingCart = null;
shoppingCart = BlmFacade.createActionManager();
jspHelper.setCurrentShoppingCart(shoppingCart);
```

Adding a Customer

You handle the Add Member action the same way you did with the Create Contract action when managing complex shopping cart contents.

Adding a member involves a hierarchy of actions and then processing the workflow using child Action Manager objects.

After creating an organization in the shopping cart, the sample code sample shows you how to retrieve the shopping cart object and post the Add Member action:

```
ActionMgrIF shoppingCart = null;
shoppingCart = jspHelper.getCurrentShoppingCart();
ActionMgrIF actionMgrChild;
actionMgrChild = shoppingCart.createChild();

OrganizationIF customer;
customer = (OrganizationIF)shoppingCart.getObject();
MemberIF member;
member = customer.addMember (actionMgrChild,…);
```

Adding a Contract

You can add a contract to the member you just created.

Because the Contract is to be linked to this member, you have to create the contract using a new Action Manager child object.

This Action Manager object should be a child of the Action Manager object that handles the Add Member action.

To do this, you:

1. Retrieve the Action Manager child object that handles the AddMember action.
2. Create a new ActionManager child object, as a sub-action manager for the just retrieved action manager object.
3. Add the Contract through this new Action Manager child object.

We assume you want to add a contract using a different form handler than the one used to create the member. We also assume that you pass the action manager's id to this page with the request (returned by calling `'actionMgrChild.getIdentifier()'`).

The sample code sample assumes that:

- You implement the Add Contract action using another logic handler than the one which handled the Add Member action.
- You pass the Id of the Action Manager that handled the Add member creation to the second form handler by using the HTTP request.

The sample code shows you how to retrieve the action manager object reference in the HTTP request parameters.

This code retrieves the Action Manager child object you used for the Add Member action.

```
ActionMgrIF shoppingCart = null;
shoppingCart = jspHelper.getCurrentShoppingCart();
ActionItemIF item;
ActionMgrIF actionMgrMember;
item = shoppingCart.getActionItem (ObjectId.instantiate(request.getParameter("action_id")),true);
actionMgrMember = (ActionMgrIF) item;
```

After retrieving the Action Manager object reference, create a child Action Manager object:

```
ActionMgrIF actionMgrContract;
actionMgrContract = actionMgrMember.createChild();
```

Now you can add the contract using this new Action Manager child object

Because of the business model, a contract is not directly linked to a member but to an organization.

Consequently, you add the Contract to the organization of the newly created member.

- Both member and contract objects you create through the shopping cart (and child action managers) are linked to the current Organization objects.
- The Action Manager child objects hierarchy structure handles the association between the contract and the member that uses it.

```
OrganizationIF customer;
customer = (OrganizationIF)shoppingCart.getObject();
ContractIF contract;
contract = customer.createContract (actionMgrContract, null, ratePlan, contractType,…);
```

You can now submit the shopping cart. This submits all of the actions handled by the hierarchy of Action Manager objects hierarchy.

Submitting the Shopping Cart

When finished, your workflow may have a summary page with a Submit button. This button submits the contents of the shopping cart.

This sample code submits the contents of the shopping cart and removed the shopping cart object handler from the user session. This code is located in the `logic_handler` page of the workflow.

```
shoppingCart = jspHelper.getCurrentShoppingCart();

shoppingCart.submit(RequestIF.SUBMIT_MODE_NORMAL);

jspHelper.destroyCurrentShoppingCart();
```

The `submit()` method throws an exception if a problem occurs when submitting the contents of the cart.

Once the contents of the cart has been submitted, you should use the `destroyCurrentShoppingCart()` method to destroy the shopping cart. This method removes the shopping cart object reference from the HTTP session.

Modifying a Shopping Cart Item

Your application may let users modify the contents of the different items in the shopping cart.

This section helps you understand how to implement a workflow to modify an object in a shopping cart.

Modifying an item in the shopping cart involves:

1. Creating a link to a modify page
2. Modifying the entry using one of the following methods:
   - Editing the object
   - Modifying an object's additional information
   - Modifying the quantity ordered
   - Modifying parameters of an object (only supported for service parameters)

Editing the Object

When editing an item in a shopping cart, you:

1. Create the link to a detail page
2. Get the object from the shopping cart
3. Modify it
4. Send it back

In general, you modify shopping cart items in a separate page. You must pass the entry id to the modify page by using the HTTP request. The sample code shows how to modify the add legal contact entry.

```
ActionItemIF addMemberEntry;

Hashtable parameters = new Hashtable ();

parameters.put ("action_id", entry.getIdentifier().toString());

<a href="<%=jspHelper.encodeURLFunct("MODIFY_PAGE_STEP",parameters, false)%>">Modify entry</a>
```

This code illustrates the complete sequence required to modify the legal contact:

```
ContactIF legalContact;
// write the code to fill the legalContact with the request parameters.
ActionMgrIF shoppingCart = null;
shoppingCart = jspHelper.getCurrentShoppingCart ();
ActionItemIF item;
ActionMgrIF actionMgr;
item = shoppingCart.getActionItem (ObjectId.instantiate(request.getParameter("action_id")),true);
actionMgr = (ActionMgrIF) item;
MemberIF member;
member = (MemberIF)actionMgr.getObject();
member.modifyLegalContact (actionMgr, legalContact, null);
```

You can use the same logic handler to modify or create the entry:

```
ContactIF legalContact;
ActionMgrIF shoppingCart = null;
shoppingCart = jspHelper.getCurrentShoppingCart ();
ActionItemIF item;
ActionMgrIF actionMgr;
if (request.getParameter("parent_id")!= null)
 {
   // You need to specify the id of the parent entry to know where to add the new one.
   // the parent_id is set to the member action manager when creating the contract
 item = shoppingCart.getActionItem (ObjectId.instantiate(request.getParameter("parent_id")),true);
 actionMgr = (ActionMgrIF) item;
 MemberIF member;
 member = (MemberIF)actionMgr.getObject();
   // Write the code here to fill the contactMgr with the request parameters.
 member.createContact (actionMgr, legalContact, null);
 }
else
 {
  // here the action_id is set to addlegal contact action item
  // when modifying an add legal contract entry
  item = shoppingCart.getActionItem (ObjectId.instantiate(request.getParameter("action_id")),true);
 actionMgr = (ActionMgrIF) item;
 MemberIF member;
 member = (MemberIF)actionMgr.getActionMgr().getObject();
   // Write the code to fill the contactMgr with the request parameters.
 member.modifyLegalContact (actionMgr, legalContact, null);
 }
```

Modifying Additional Information

You can also modify the additional information of objects in the shopping cart.

To change additional parameter values, you:

1. Create the link to a "modify" page
2. Get the entry to be modified
3. Call the modify additional information method for the entry

In general, you modify shopping cart items in a separate page. You must pass the entry id to the modify page by using the HTTP request. The sample code shows how to modify the additional information of the object related to an action manager.

```
ActionItemIF entry;
ParameterIF[] criteria=new ParameterIF[0];
if ((entry.getOptParameters (criteria)!=null) && (entry.getOptParameters (criteria).length>0))
  {
  Hashtable parameters = new Hashtable ();
  parameters.put (“action_id”, entry.getIdentifier().toString());
  <a href=”<%=jspHelper.encodeURLFunct("MODIFY_PAGE_STEP",parameters, false)%>”>Modify entry</a>
  }
```

In the modify form handler, you get the entry to modify:

```
ActionItemIF item;
ActionMgrIF actionMgr;
item = shoppingCart.getActionItem (ObjectId.instantiate(request.getParameter("action_id")),true);
actionMgr = (ActionMgrIF) item;
```

This code illustrates the complete sequence required to change additional information.

```
ActionMgrIF shoppingCart = null;
shoppingCart = jspHelper.getCurrentShoppingCart ();
ActionItemIF item;
item = shoppingCart.getActionItem (ObjectId.instantiate(request.getParameter("action_id")),true);
ParameterIF []newParameters;
 // Fill the newParameters array with the request data
 // Additional information are part of the request data
item.setOptParameters (newParameters);
```

Modifying the Quantity

You can only modify the quantity of services.

Modifying the quantity of services involves:

1. Creating a link to a "modify" page if the service quantity is modifiable
2. Getting the service to modify
3. Calling the modify entry quantity method

In general, you modify shopping cart items in a separate page. You must pass the entry id to the modify page by using the HTTP request. The sample code shows how to modify the service quantity.

```
ActionItemIF currentService;

If ((currentService.getMaxQuantity ()>=2) && (currentService.getMaxQuantity ()!=currentService.getMinQuantity
()))

 {

 Hashtable parameters = new Hashtable ();

 parameters.put (“action_id”, currentService.getIdentifier().toString());

 <a href=”<%=jspHelper.encodeURLFunct("MODIFY_QUANTITY_STEP", parameters, false)%>”>Modify service quantity</a>

 }
```

The modify quantity entry method to call:

```
ActionMgrIF shoppingCart = null;

shoppingCart = jspHelper.getCurrentShoppingCart ();

ActionItemIF item;

item = shoppingCart.getActionItem (ObjectId.instantiate(request.getParameter("action_id")),true);

int quantity;

 // Fill the quantity object with the request data

item.setQuantity(quantity);
```

Modifying Service Parameters

Modifying service parameters involves:

1. Creating a link to a "modify" page if the service is modifiable
2. Getting the service to modify
3. Calling the modify entry parameters method

In general, you modify shopping cart items in a separate page. You must pass the entry id to the modify page by using the HTTP request. The sample code shows how to modify the service parameters.

```
ActionItemIF currentService;
If (currentService.isModifiable ())
 {
 Hashtable parameters = new Hashtable ();
 parameters.put (“action_id”, currentService.getIdentifier().toString());
 <a href=”<%=jspHelper.encodeURLFunct("MODIFY_SERVICE_STEP",parameters, false)%>”>Modify service parameters</a>
 }
```

This code illustrates the complete sequence required to change set new parameter value.

```
ActionMgrIF shoppingCart = null;
shoppingCart = jspHelper.getCurrentShoppingCart ();
ActionItemIF item;
item = shoppingCart.getActionItem (ObjectId.instantiate(request.getParameter("action_id")),true);
ParameterIF []newParameters;
 // Fill the newParameters array with the request data
item.setModifiableParameters (newParameters);
```

Removing a Shopping Cart Item

Removing an item from the shopping cart involves:

1. Create a link to a "remove" page if the entry can be removed
2. Get the entry to remove
3. Call the remove entry method on its parent

The following code shows how to create the link.

```
ActionItemIF entry;
If (entry.isRemovable ())
 {
 Hashtable parameters = new Hashtable ();
 parameters.put (“action_id”, entry.getIdentifier().toString());
 <a href=”<%=jspHelper.encodeURLFunct("REMOVE_ENTRY_STEP",parameters, false)%>”>Remove entry</a>
 }
```

The following code illustrates the complete sequence required to remove the entry

```
ActionMgrIF shoppingCart = null;
shoppingCart = jspHelper.getCurrentShoppingCart ();
ActionItemIF item;
item = shoppingCart.getActionItem (ObjectId.instantiate(request.getParameter("action_id")),true);
item.getActionMgr ().removeActionItem (item);
```

Displaying the Contents of a Shopping Cart

Your application may display the contents of a shopping cart to users so they can look over their order before submitting them.

These examples show you how to browse and display the contents of a shopping cart. This example is based on a shopping cart creating a customer.

Browsing the shopping cart

Browsing the shopping cart involves:

1. Getting the created customer
2. Getting the customer contact and payment information
3. Getting the members created
4. Getting the member contacts and logins
5. Getting the created contracts of each member
6. Getting the contract services

You get the created customer by retrieving the shopping cart:

```
ActionMgrIF shoppingCart = null;
shoppingCart = jspHelper.getCurrentShoppingCart ();
```

You get the customer contact and payment information:

```
ActionItemIF[] legalContacts;
legalContacts = shoppingCart.findActionItemsByAction (DescriptorOidIF.ACTION_ADD_LEGAL_CONTACT, false);
ActionItemIF[] billingContacts;
billingContacts = shoppingCart.findActionItemsByAction(DescriptorOidIF.ACTION_ADD_BILLING_CONTACT, false);
ActionItemIF[] billingAccounts;
billingAccounts = shoppingCart.findActionItemsByAction (DescriptorOidIF.ACTION_ADD_BILLING_ACCT, false);
```

Get the created members:

```
ActionItemIF[] members;
members = shoppingCart.findActionItemsByAction (DescriptorOidIF.ACTION_ADD_MEMBER, false);
```

For each member, you get the contacts, logins and contracts:

```
ActionMgrIF currentMember;
for (int intMembers=0; intMembers < members.length; intMembers++)
  {
  currentMember = (ActionMgrIF) members[intMembers];
  ActionItemIF[] legalContacts;
  legalContacts = currentMember.findActionItemsByAction (DescriptorOidIF.ACTION_ADD_LEGAL_CONTACT,   false);
  ActionItemIF[] billingContacts;
  billingContacts = currentMember.findActionItemsByAction(DescriptorOidIF.ACTION_ADD_BILLING_CONTACT, false);
  ActionItemIF[] logins;
  logins = currentMember.findActionItemsByAction (DescriptorOidIF.ACTION_ADD_LOGIN, false);
  ActionItemIF[] contracts;
  contracts = currentMember.findActionItemsByAction (DescriptorOidIF.ACTION_ADD_CONTRACT, false);
  }
```

For each contract, you get its contracted services:

```
ActionMgrIF currentContract;
For (int intContracts=0; intContracts < contracts.length; intContracts++)
  {
  currentContract = (ActionMgrIF) contracts[intContracts];
  ActionItemIF[] services;
  services = currentContract.findActionItemsByAction (DescriptorOidIF.ACTION_ADD_SERVICE
,   false);
  }
```

The sample code shows detailed information (actually objects parameters and attributes) that is handled by the shopping cart through Action Item objects.

You should be familiar with the types of action items you can work with.

Displaying All Items

To display shopping cart entries, you use the generic methods of Action Item objects.

The following generic methods are available for every type of entry in your shopping cart:

METHOD	COMMENTS
ActionItemIF.getAction().getItemName()	Get the localized name of the entry type. (ex. Contract, Member, …)
ActionItemIF.getLabel ()	Get useful information specifying the entry. (ex. service name in case of an add service entry)

METHOD	COMMENTS
ActionItemIF.getQuantity ()	Get the ordered quantity if applicable, returns –1 if not
ActionItemIF. getAmount(ActionItemIF.FEE_ACCESS) Or ActionItemIF. GetAmount(ActionItemIF.FEE_SUBSCRIPTION)	Get the two possible costs of an entry if applicable, or return Double.NaN if not.

If you want to display the total cost of the contents of your shopping cart, use the following method:

```
double totalFeeAccess = shoppingCart.getTotalAmount(ActionItemIF.FEE_ACCESS);
double totalFeeSubscription = shoppingCart.getTotalAmount(ActionItemIF.FEE_SUBSCRIPTION);
```

Working with core services

When working with the shopping cart, remember the following when dealing with core services.

- While creating a contract, core services are automatically included when submitted.
- When creating a contract using a shopping cart, the core services included with the new contract object are not available as shopping cart action items.

Therefore, you cannot handle core services through generic action item APIs.

If you need to display them for a contract being created in the shopping cart, you can use the following code:

```
ContractIF contract;
Contract = (ContractIF) currentContract.getObject ();
ContractedServiceIF[] coreServices;
coreServices = contract.getContractedServices(null,null); // null because we want only core services
```

Retrieving the detail of an entry

To get more details of an item:

- If the item is itself an action manager or is an add billing account action, use the following code:

```
MemberIF member;
member = (MemberIF)currentMember.getObject();
// Insert code to display any attribute of the member
```

- You must get the parent object then retrieve the object to display. The following is an example of getting the added legal contact of a created member.

```
ActionItemIF currentLegalContact;

ActionMgrIF parentActionMgr;

MemberIF member;

parentActionMgr = currentLegalContact.getActionMgr();

member = (MemberIF)parentActionMgr.getObject();

ContactIF legalContact;

legalContact = member.getLegalContact();
```

Saving Shopping Carts

Your application may need to create a backup copy of the contents of a shopping cart for several reasons. These reasons include:

- Users losing their connections and the sessions time out
- Remember the contents of a shopping cart for users when reconnect to the application

These examples show you how to backup the contents of the shopping cart and display the contents of a saved shopping cart. Not only can this feature be used for backup copies of the shopping cart, you can also create content templates in order to create shopping carts with predefined items.

About Saving Shopping Carts

When saving shopping carts and their contents, you should keep in mind the following:

- You use the `PersistentActionManager` interface to manage the persistence of an action manager
- There is no limit to the number of saved shopping carts per user
- There is no history of saved shopping carts

You use `com.netonomy.blm.interfaces.util.PersistantActionManagerIF` to manage shopping carts saved in the CID. When saving an action manager, you can set the following:

- Name to identify the persistent shopping cart if required
- Description to describe it if required
- Category to specify the future use
- Additional information to be used as criteria to retrieve it if required

When saving PersistentActionManagers in the CID, you use the `CORE_BACKUP` category for backup copies. You can also create your own categories in the CID to save Persistent Action Managers. When retrieving saved PersistentActionManagers from the CID, you use the `ObjectRefMgr.getPersistentActionMgrCategoryByCode` method to retrieve a list of the saved PersistentActionManagers. You can then look for the optional name, description, or additional parameters.

The Shopping Cart Package of the CID contains the tables used to save the information concerning the shopping cart. The tables include:

- `PERSISTENT_ACTIONMGR_CATEGORY` Table
- `PERSISTENT_ACTIONMGR` Table

Saving a Shopping Cart

Your application may need to save the shopping cart while the user adds and modifies the contents of the shopping cart.

Saving the contents of a shopping cart involves:

1. Creating a Persistent Action Manager
2. Saving the action manager using the Persistent Action Manager
3. If required, update the attributes of a saved copy. For saved copies, you can only update the attributes. If you need to save other changes, use the standard method to save the shopping cart.

The sample code shows how to create and update a Persistent Action Manager:

```
PersistentActionMgrIF persistentActionMgr=ObjectMgr.createPersistentActionManager();
PersistentActionMgrCategoryIF persistentActionMgrCat= null;
persistentActionMgrCat= ObjectRefMgr.getPersistentActionMgrCategoryByCode("CORE_BACKUP");
// get current shopping cart from the session to save
ActionMgrIF actionMgr = JFNJspHelper.getCurrentShoppingCart();
                if ( (actionMgr != null) && (actionMgr.getActionItems().length > 0) )
                  {
                    //Set the name of the persistent action manager
                    //for your application if required
                    persistentActionMgr.setName("PAM name");

                    //Set the description of the persistent action manager
                    //for your application if required
                    persistentActionMgr.setDescription("PAM description");

                    //Set the category of the persistent action manager
                    //to one of the categories in the PERSISTENT_ACTIONMGR_CATEGORY table
                    //in this example, it is the CORE_BACKUP.
                    persistentActionMgr.setCategory(persistentActionMgrCat);

                    //Set the additional parameters of the
                    //persistent action manager if required
                    persistentActionMgr.setAdditionalParameters(yourAdditionalParameters[]);

                    //Save to the database
                    persistentActionMgr.persist(actionMgr);
                  }
//code your application
//Update if required
persistentActionMgr.update();
```

Create a Shopping Cart from a Saved Copy

Your application may save the contents of the shopping cart for workflows or security reasons. You may also have templates you want to use to create content templates to ease navigation and create ready-to-purchase shopping cart contents.

To create a shopping cart from a saved Persistent Action Manager, you:

1. Find the Persistent Action Manager to use.
2. Fill the Action Manager with the contents of the correct Persistent Action Manager.
3. Verify the contents of the Action Manager.
4. Declare this Action Manager as the current shopping cart.
5. Remove old persistent shopping carts.

This example contains code to create a shopping cart from backup copies when the user logs in.

```
PersistentActionMgrIF persistentActMgr;
ActionMgrIF action;
ParameterIF[] criteria;
FilterIF filter;
// get filter, fill mandatory criteria

filter = ObjectRefMgr.getFilterByCode("CORE_PAMBYCATEGORY");
criteria = filter.getCriteria(FilterIF.ALL);

// get the Persistent Action Manager created by the specified user and category

((ValueDynamicIF) ParameterHelper.getParameterByCode(criteria,
"CORE_C_PAMGENERATEDBY")).setDynamicValue(user.getLoginId());

((ValueDynamicIF) ParameterHelper.getParameterByCode(criteria,
"CORE_C_PAMCATEGORY")).setDynamicValue(ObjectRefMgr.getPersistentActionMgrCategoryByCode("CORE_BACKUP").getIden
tifier());

PersistentActionMgrIF[] actMgrs = ObjectMgr.searchPersistentActionManagers(filter);
    if (actMgrs.length > 0)

      {
      // Fill the Action ManagerIF with the first Persistent Action Manager found (the newest one)
        action = actMgrs[0].instantiateActionMgr();
        if (action != null)
          {
          // Test Action Manager to make sure the contents are valid
               BLMErrorIF[] actionMgrError = action.verify();
            if (actionMgrError.length == 0)
               {
                    // Declare this new Action Manager as the current shopping cart for this session
                    jspHelper.setCurrentShoppingCart(action);
                    // Delete all returned persistent shopping carts
                    for (int j=0;j<actMgrs.length;j++)

                    {
                    actMgrs[j].delete();
                    }
             }
             else {
             // Code error management if content is not valid
                  }
          }
      }
```

Using Shopping Cart Templates

About Persistent Action Managers

Shopping Carts correspond to BLM Action Managers. These action managers do exactly what their name implies, they manage the actions in the current context. An action manager is considered a shopping cart when it holds a set of actions to submit at the same time. The types of action managers are:

- Action Managers
- Persistent Action Managers

The context of your application determines which type of action manager you use. In general, you use Action Managers to manage the actions in a specific context or workflow. However these action managers and their contents cannot be saved. If your application requires saving the contents of an action manager, you use Persistent Action Managers. The saved Persistent Action Managers can be backup copies of the shopping cart or used as a shopping cart template.

When saving persistent action managers as Shopping Carts and templates, you should keep in mind the following:

- There is no limit to the number of saved persistent action managers
- There is no history of saved persistent action managers

Shopping Cart templates have the following:

- Name
- Description
- Category
- Additional information to be used as criteria to retrieve it

The shopping carts are saved in the same table as the backup shopping carts. This information is in the Shopping Cart Package tables in the CID:

- `PERSISTENT_ACTIONMGR_CATEGORY` Table
- `PERSISTENT_ACTIONMGR` Table

About Shopping Cart Templates

You can create and manage shopping cart templates. One of the primary uses of a shopping cart template is to implement quick buy features where your application presents pre-configured shopping carts to users. The templates may correspond to contracts with a specific rate plan and selection of services. Once these templates are saved, your application can use them to fill a shopping cart with all the options and services needed to let users order quickly and without having to go through an entire workflow to sign up to for a contract.

Technically, a shopping cart template is a saved Persistent Action Manager. There is also a tool to help you manage the templates you create.

For more information about managing saved templates, refer to *Administrating Telco Service Manager.*

Using shopping cart templates involves:

- Saving shopping carts as templates
- Viewing the list of available templates
- Using a template to quickly create a new order
- Deleting shopping cart templates

The examples show you how to save the contents of the shopping cart as a template. You can then view the template, use it in a shopping cart. Once it is no longer needed, an example shows you how to delete it.

This feature is restricted to creating and managing contract templates.

Saving a Shopping Cart as a Template

Saving the contents of a shopping cart as a template involves:

1 Creating a Persistent Action Manager.

2 Setting the category of the Persistent Action Manager

3 Saving the Persistent Action Manager as a template.

The sample code shows how to save an Action Manager as a template:

Get the current Action Manager and create the Persistent Action Manager	`ActionMgrIF actionMgr;`
	`//The code below gets the current action manager.`
	`actionMgr = getCurrentActionMgr (request, response, jspHelper);`
	`PersistentActionMgrIF persistentActionMgr = ObjectMgr.createPersistentActionManager();`

Set the category, name and description	//The code below initializes the category and sets the category of the
	// Persistent Action Manager to save.
	
	PersistentActionMgrCategoryIF persistentActionMgrCat =
	ObjectRefMgr.getPersistentActionMgrCategoryByCode("CORE_CONTRACTTEMPLATE");
	persistentActionMgr.setName (request.getParameter ("template_name"));
	persistentActionMgr.setDescription (request.getParameter ("template_description"));
	persistentActionMgr.setCategory (persistentActionMgrCat);

Save the Persistent Action Manager	persistentActionMgr.persist (actionMgr);

Getting the List of Available Templates

Displaying the list of shopping cart templates involves:

- Getting the list of available templates

The sample code shows how to display the list of templates:

Declare variables	PersistentActionMgrIF []templates; ParameterIF[] criteria; FilterIF filter;

Get default filter and fill the PAM category criteria	// get filter, fill mandatory criteria
	
	filter = ObjectRefMgr.getDefaultFilter(DescriptorOidIF.OBJECTTYPE_PERSISTENTACTIONMGR);
	
	criteria = filter.getCriteria(FilterIF.ALL);
	
	((ValueDynamicIF) ParameterHelper.getParameterByCode(criteria,
	"CORE_C_PAMCATEGORY")).setDynamicValue(ObjectRefMgr.getPersistentActionMgrCategoryByCode("C
	ORE_CONTRACTTEMPLATE").getIdentifier());

Search for templates and put matching templates in array	templates = ObjectMgr.searchPersistentActionManagers(filter);

Using Shopping Cart Templates

Using a shopping cart template in a shopping cart involves:

1. Get the template
2. Reset level and member of the contract template
3. Insert the template contents into the shopping cart
4. Validate the template
5. Submit the shopping cart

The sample code shows how to add a shopping cart template to a shopping cart:

Instantiate the template	actionMgrIF actionMgr; actionMgr = template.instantiateActionMgr();
Get the shopping cart	ActionMgrIF shoppingCart=getCurrentActionMgr (request, response, jspHelper);

Reset level and member then add the template to the shopping cart	`//here you should reset the level and member of the contract template`
	`//`
	`shoppingCart.addChild (actionMgr);`

Check for errors	`BlmErrorIF[] errors = actionMgr.verify ();`
	`if ((errors == null) || (errors.length==0))`
	`{`

Submit the shopping cart	`shoppingCart.submit (RequestIF.SUBMIT_MODE_NORMAL);` `}`

Deleting Shopping Cart Templates

Deleting shopping cart templates involves:

1. Getting the template to delete
2. Deleting the selected templates

The sample code shows how to retrieve and delete a previously saved template:

Get the template	`PersistentActionMgrIF template;`
	`ObjectId templateId = ObjectId.instantiate (request.getParameter ("templateId"), true);`
	`template = ObjectMgr.getPersistentActionManager (templateId);`

Delete the template	`if (template != null)`
	`{`
	`template.delete ();`
	`}`

CHAPTER 12

Using Bulk Ordering

In This Section

About Bulk Ordering 192
Adding a Service to Contracts 193
Modifying a Service of Contracts 194
Removing a Service from Contracts 195
Changing the Rate Plan of Contracts 196

About Bulk Ordering

When you create an application to manage several users and contracts, sometimes you may need to group together changes to a specific set of contracts. Instead of changing the contracts individually, you can use the bulk ordering feature to allow administrators to change sets of contracts.

Using bulk ordering involves:

- Adding a service
- Modifying a service
- Removing a service
- Changing a rate plan

The examples show you how to use this feature in your application.

Adding a Service to Contracts

Adding a service to a set of contracts involves:

1. Getting the selected contracts to modify
2. Getting the service selected by the user
3. Adding the service to the contracts
4. Getting any errors that occurred for further processing

The sample code shows how to add a service to a set of contracts:

Getting the contracts	`// we get selected contracts from the request. The selected contracts identifier`
	`// are represented by the contractId request parameters`
	
	`SessionF blmSession = jspHelper.getBlmSession ();`
	`String[] values = request.getParameterValues("contractId");`
	`ContractF[] contracts = new ContractF[values.length];`
	`for (int i=0;i<values.length;i++) {`
	`contracts[i] = new ContractF(blmSession, ObjectId.instantiate(values[i]));`
	`}`

Get the list of available services	`OrganizationF org = jspHelper.getLevel().getOrganization();`
	
	`RatePlanServiceIF[] rateplanServices = org.getAllowedRatePlanServices(null, null, CommercialOfferIF.TYPE_PRIORITY_TO_DEDICATED, new Boolean(false), new Boolean(true), null, null);`
	
	`ListServiceHelper listServiceHelper = new ListServiceHelper(rateplanServices);`
	
	`ServiceIF[] services = listServiceHelper.getServices();`

Get the service selected by the user	`// we get selected service from the request. The selected service identifier`
	`// is represented by the service request parameters`
	
	`ServiceIF service;`
	
	`// Instantiate the service using its identifier`
	`service =ObjectRefMgr.getService(ObjectId.instantiate(request.getParameter("service")));`

Add the service and get the errors	`BlmErrorIF[] blmError = null;`
	`ParameterIF[] parameters = null;`
	
	`parameters = service.getParameters ();`
	
	`// use this method from the setParameters.jsp form handler`
	
	`fillParametersWithRequest (parameters, request);`
	
	`// for a quick buy bulk order, pass Null as the ActionMgr parameter,`
	`// otherwise pass the shopping cart action manager.`
	
	`blmError = ContractF.addServiceToContracts(actionMgr, contracts, service, parameters, true, null);`
	
	`// blmError is an array of business logic errors that occurred when submitting changes`
	`//(incompatible services, expired services or contracts, and so on)`
	
	`//You can use the blmError.getObject method to get the object that caused the error`
	`// (in this case, it should be the contract that caused the error)`

Modifying a Service of Contracts

Modifying a service to a set of contracts involves:

1. Getting the selected contracts to modify
2. Getting the service selected by the user
3. Modifying the service to the contracts
4. Getting any errors that occurred for further processing

The sample code shows how to modify a service of a set of contracts:

Getting the contracts	`// we get selected contracts from the request. The selected contracts identifier`
	`// are represented by the contractId request parameters`
	
	`SessionF blmSession = jspHelper.getBlmSession ();`
	`String[] values = request.getParameterValues("contractId");`
	`ContractF[] contracts = new ContractF[values.length];`
	
	`ObjectId[] objectId = new ObjectId[values.length];`
	`for (int i=0;i<values.length;i++) {`
	`contracts[i] = new ContractF(blmSession, ObjectId.instantiate(values[i]));`
	
	`objectId[i] = ObjectId.instantiate(values[i]);`
	`}`

Get the list of modifiable services	`// get filter, fill mandatory criteria`
	
	`filter = ObjectRefMgr.getFilterByCode("CORE_SERVICEMODIFIABLE");`
	
	`criteria = filter.getCriteria(FilterIF.ALL);`
	
	`((ValueDynamicIF) ParameterHelper.getParameterByCode(criteria,`
	`"CORE_C_CONTRACTSID")).setDynamicValue(objectId);`
	
	`services = ObjectRefMgr.searchServices(filter);`

Get the service selected by the user	`// we get selected service from the request. The selected service identifier`
	`// is represented by the service request parameters`
	
	`ServiceIF service;`
	
	`// Instantiate the service using its identifier`
	`service =ObjectRefMgr.getService(ObjectId.instantiate(request.getParameter("service")));`

Modify the service and get the errors	`// MODIFY YOUR SERVICE HERE`
	
	`BlmErrorIF[] blmError = null;`
	`ParameterIF[] parameters = null;`
	
	`parameters = service.getParameters ();`
	
	`// use this method from the setParameters.jsp form handler`
	
	`fillParametersWithRequest (parameters, request);`
	
	`// for a quick buy bulk order, pass Null as the ActionMgr parameter,`
	`// otherwise pass the shopping cart action manager.`
	
	`blmError = ContractF.modifyServiceOfContracts(actionMgr, contracts, service, parameters,`
	`true, null);`
	
	`// blmError is an array of business logic errors that occurred when submitting changes`
	`//(incompatible services, expired services or contracts, and so on)`
	
	`//You can use the blmError.getObject method to get the object that caused the error`
	`// (in this case, it should be the contract that caused the error)`

Removing a Service from Contracts

Removing a service to a set of contracts involves:

1. Getting the selected contracts to modify
2. Getting the service selected by the user
3. Removing the service from the contracts
4. Getting any errors that occurred for further processing

The sample code shows how to remove a service from a set of contracts:

Getting the contracts	`// we get selected contracts from the request. The selected contracts identifier`
	`// are represented by the contractId request parameters`
	
	`SessionF blmSession = jspHelper.getBlmSession ();`
	`String[] values = request.getParameterValues("contractId");`
	`ContractF[] contracts = new ContractF[values.length];`
	`ObjectId[] objectId = new ObjectId[values.length];`
	`for (int i=0;i<values.length;i++) {`
	`contracts[i] = new ContractF(blmSession, ObjectId.instantiate(values[i]));`
	
	`objectId[i] = ObjectId.instantiate(values[i]);`
	`}`

Get the list of services to remove	`// get filter, fill mandatory criteria`
	
	`filter = ObjectRefMgr.getFilterByCode("CORE_SERVICEREMOVABLE");`
	
	`criteria = filter.getCriteria(FilterIF.ALL);`
	
	`((ValueDynamicIF) ParameterHelper.getParameterByCode(criteria,`
	`"CORE_C_CONTRACTSID")).setDynamicValue(objectId);`
	
	`services = ObjectRefMgr.searchServices(filter);`

Get the service selected by the user	`// we get selected service from the request. The selected service identifier`
	`// is represented by the service request parameters`
	
	`ServiceIF service;`
	
	`// Instantiate the service using its identifier`
	`service =ObjectRefMgr.getService(ObjectId.instantiate(request.getParameter("service")));`

Removing the service and get the errors	`BlmErrorIF[] blmError = null;`
	
	`// for a quick buy bulk order, pass Null as the ActionMgr parameter,`
	`// otherwise pass the shopping cart action manager.`
	
	`blmError = ContractF.removeServiceFromContracts(actionMgr, contracts, service, parameters,`
	`true, null);`
	
	`// blmError is an array of business logic errors that occurred when submitting changes`
	`//(incompatible services, expired services or contracts, and so on)`
	
	`//You can use the blmError.getObject method to get the object that caused the error`
	`// (in this case, it should be the contract that caused the error)`

Changing the Rate Plan of Contracts

Changing the rate plan of a set of contracts involves:

1. Getting the selected contracts
2. Getting the selected commercial offer (if required)
3. Getting the list of available rate plans
4. Changing the rate plan of the selected contracts
5. Getting any errors that occurred for further processing

The sample code shows how to change the rate plan of a set of contracts:

Get the contracts	`// we get selected contracts from the request. The selected contracts identifier`
	`// are represented by the contractId request parameters`
	
	`SessionF blmSession = jspHelper.getBlmSession ();`
	`String[] values = request.getParameterValues("contractId");`
	`ContractF[] contracts = new ContractF[values.length];`
	`for (int i=0;i<values.length;i++)`
	`{`
	`contracts[i] = new ContractF(blmSession, ObjectId.instantiate(values[i]));`
	`}`

Get the commercial offer	`// we get the selected commercial offer from the request. The selected offer identifier`
	`// is represented by the offerId request parameter`
	
	`ObjectId offerId = ObjectId.instantiate (request.getParameter ("offerId"),true);`

Get the list of available rate plans	`RatePlanIF[] ratePlans;`
	`CommercialOfferIF offer = null;`
	
	`// if at least one offerId is available, rate plans are selected using this offerId`
	`// if not, we get all the authorized rate plans allowed for the current organization`
	`if (offerId!= null)`
	`{`
	`offer = ObjectRefMgr.getCommercialOffer(offerId);`
	`ratePlans = offer.getAllowedRatePlans(new Boolean(true), null, null);`
	`}`
	`else`

Get the rate plan selected by the user	`{`
	`ratePlans = jspHelper.getLevel().getOrganization().getAllowedRatePlans(null, null,`
	`CommercialOfferIF.TYPE_PRIORITY_TO_DEDICATED, new Boolean(true), null);`
	`}`

Change rate plans and get the errors	`BlmErrorIF[] blmError = null;`
	
	`// for a quick buy bulk order, pass Null as the ActionMgr parameter,`
	`// otherwise pass the shopping cart action manager.`
	
	`blmError = ContractF.changeRatePlanOfContracts(null, contracts, rateplan, null);`
	
	`// blmError is an array of business logic errors that occurred when submitting changes`
	`//(expired rate plans or contracts, and so on)`
	
	`//You can use the blmError.getObject method to get the object that caused the error`
	`// (in this case, it should be the contract that caused the error)`

CHAPTER 13

Working with Approvals

In This Section

About Approving Orders 198
About Approval Processes Logic 203
Creating a New Approval Process Class 204
Deploying the Approval Class 209
Running the Approval Sequencer 210

About Approving Orders

Submitting a request can be subject to approval by a specified user. For instance, users of your application can modify their contracts but the changes are not taken into account until approved by an administrator. Your application can also send a request for approval by the OSS.

When users submit a request and after checking security and business logic rules, a request is inserted in the CID for processing by the Synchronizer connector. You can program the request to obtain approval by specified users before being processed by the Synchronizer connector. Obtaining approval for a specific request is called the Approval Process.

The approval process manages:

- Order requests
- Create customer requests
- Create level requests
- Create member requests
- Create contract requests

In the process of approving requests, there are users and OSS systems which are part of the process. Their role is to approve or deny a request and your approval process may have more than one user or OSS system needed to approve a request. In the approval process, an actor which approves or denies requests is referred to as an Approval. For instance, in order to change the rate plan of a contract, your approval process may require both the manager and the billing system to approve changes to rate plans of a contract. This approval process has two approvals before the request considered as approved.

Using the Approval Process involves:

- Creating an Approval Process
- Submitting Requests for Approval
- Displaying the Requests for Approval
- Approving or Denying Requests
- Creating a new Approval logic class
- Running the Approval Sequencer

When dealing with simple modifications, such as AddService, you put the simple modification in an Order request to be validated.

Creating Approval Processes

When you create an approval process, you specify which users must approve the request before the request status changes from `To be approved` to `Not yet submitted`.

Creating an approval process involves:

1. Creating an ApprovalProcess
2. Determining which users approve the request
3. Adding the user to the process

The sample JSP code shows how to create an approval process that requires:

- The hierarchical superior to approve the request:
  - If a business subscriber, their administrator must approve the request
  - If a level manager, their superior approves the request
- The OSS to approve the request

Create the approval process	`// Now creates the approval process` `ApprovalProcessIF process = ObjectMgr.createApprovalProcess ();`

If the user is a business subscriber, the request must be approved by an administrator	`// A business subscriber must be approved by an administrator`
	`if (jspHelper.checkRole (new String []{"SUBSCRIBER"}) && (!jspHelper.checkRole (new String []{"CUSTADMIN"})) &&(jspHelper.getLevel().getHierarchyRoot ().getOrganization ().getType().getCode ().equals ("BUSINESS")))`
	`{`
	`// Gets the first custadmin`
	`LevelF level = jspHelper.getLevel ();`
	`FilterIF filter = ObjectRefMgr.getFilterByCode ("CORE_INTORG_MEMBERBYROLES");`
	`RoleIF adminRole = ObjectRefMgr.getRoleByCode ("CUSTADMIN");`
	`ValueDynamicIF levelParam = (ValueDynamicIF)ParameterHelper.getParameterByCode (filter.getCriteria(FilterIF.HIDDEN), "CORE_C_ORGID");`
	`levelParam.setDynamicValue (level.getIdentifier());`
	`ValueDynamicIF roleParam = (ValueDynamicIF)ParameterHelper.getParameterByCode (filter.getCriteria(FilterIF.ALL), "CORE_C_ROLES");`
	`roleParam.setDynamicValue (adminRole.getIdentifier());`
	`UserF admins[] = UserF.search (filter);`
	`if ((admins!=null) && (admins.length>0))`
	`{`
	`process.addMemberApproval (admins[0], null, null);`
	`}`
	`}`

If the user is an administrator, the request must be validated by the administrator's superior	else if (jspHelper.checkRole (new String []{"CUSTADMIN"}))
	{
	// Adds an approval for the parent administrator
	LevelIF level = jspHelper.getBlmSession().getUserF().getLevel ().getParentLevel();
	if (level != null)
	{
	FilterIF filter = ObjectRefMgr.getFilterByCode ("CORE_INTORG_MEMBERBYROLES");
	RoleIF adminRole = ObjectRefMgr.getRoleByCode ("CUSTADMIN");
	ValueDynamicIF levelParam = (ValueDynamicIF)ParameterHelper.getParameterByCode (filter.getCriteria(FilterIF.HIDDEN), "CORE_C_ORGID");
	levelParam.setDynamicValue (level.getIdentifier());
	ValueDynamicIF roleParam = (ValueDynamicIF)ParameterHelper.getParameterByCode (filter.getCriteria(FilterIF.ALL), "CORE_C_ROLES");
	roleParam.setDynamicValue (adminRole.getIdentifier());
	UserF admins[] = UserF.search (filter);
	if ((admins!=null) && (admins.length>0))
	{
	process.addMemberApproval (admins[0], null, null);
	}
	}
	}
	process.addOSSApproval ("OSS_APPROVER_SAMPLE", null);

The OSS must also approve this request	process.addOSSApproval ("OSS_APPROVER_SAMPLE", null);

Submitting Requests for Approval

Once the approval process has been written, you submit the action manager to be processed.

Submitting a request for approval involves:

- Using the `actionMgr.submitForApproval` method

The sample JSP code shows how to submit the action manager for approval:

Create the approval process	ApprovalProcessIF process = ObjectMgr.createApprovalProcess (); // Code you approval process here
Submit for approval	actionMgr.submitForApproval(process);

Displaying Requests for Approval

To display the requests that a user must approve, you use the search feature to return a list of requests to approve.

Retrieving the requests involves:

1 Declaring the user

2 Using the search filter to return the requests to approve

The sample JSP code shows how to return an array of requests to approve for the current user:

Declare user and filter	`ParameterIF[] criteria;`
	`FilterIF filter;`
	
	`SessionF  blmSession = jspHelper.getBlmSession();`
	`UserF user = blmSession.getUserF();`
	`// get filter, fill mandatory criteria`
	`filter = ObjectRefMgr.getFilterByCode("CORE_REQUESTBYAPPROVALMEMBER");`
	`criteria = filter.getCriteria(FilterIF.ALL);`
	
	`final int maxCount = filter.getRowCount();`
	`filter.setRowCount (maxCount+1);`
	`((ValueDynamicIF) ParameterHelper.getParameterByCode(criteria, "CORE_C_APPROVALMEMBERID")).setDynamicValue(new ObjectId[] {user.getIdentifier()});`

Find requests	`RequestIF[] requests = ObjectMgr.searchRequests(filter);`

Approving or Denying Requests

To approve or deny a request, you change the status of the approval related to the user to `APPROVED` or `DENIED`.

Approving or denying the requests involves:

1 Getting the request and its approval process

2 Getting the list of approvals associated with the user

3 Changing the status of the approval

4 Applying changes

The sample JSP code shows how to approve or deny requests of the current user:

Get the request and its approval process	`//Get the requestId and the reasonpassed as parameters of the request`
	
	`ObjectId requestId = ObjectId.instantiate (request.getParameter("request"));`
	
	`ApprovalProcessIF process = ObjectMgr.getApprovalProcess (requestId);`

Get the approvals associated with the user	`ApprovalIF[] toApprove = process.findApprovalsByMember (jspHelper.getBlmSession().getUserF());`
	
	`jspHelper.doNotSend ("reason");`
	
	`jspHelper.doNotSend ("request");`

Change the status of the approval to APPROVED or DENIED	`String reason = request.getParameter ("reason");`
	
	`if ((toApprove != null) && (toApprove.length>0))`
	
	`{`
	
	`for (int i=0;i<toApprove.length;i++)`
	
	`{`
	
	`if (toApprove[i].isNext() && toApprove[i].getStatus ().getIdentifier().equals (ApprovalStatusIF.TO_BE_APPROVED))`
	
	`{`
	
	`toApprove[i].setStatus (ObjectRefMgr.getApprovalStatus (ApprovalStatusIF.APPROVED));`
	
	`//to deny the request`
	
	`//toApprove[i].setStatus (ObjectRefMgr.getApprovalStatus (ApprovalStatusIF.DENIED));`
	
	`if (reason != null)`
	
	`toApprove[i].setReason (reason);`
	
	`}`
	
	`}`
	
	`}`

Update the approval process	`process.update (true);`

About Approval Processes Logic

An approval process is when another user must approve a request before it can be submitted.

For example, an approval process can designate a certain user in an organization who is responsible for approving changes to contracts and services made by other users of the organization.Not only can you designate who is responsible for approving requests, you can write your own code that carries out your approval process logic to fit your needs.

The approval process evaluation logic is in the BLM external class `com.netonomy.blm.external.EvaluateApprovalProcess`.

The default approval process evaluation logic is:

- A request is denied if one approver has denied it
- A request is approved if every approver has approved it
- The approval process is sequential. Approvers approve the request one after the other in the same order that the approval wew added to the approval process.

Implementing your own approval process evaluation logic involves:

- Writing a new class with your approval process evaluation logic that extends the existing class
- Deploying and declaring your class

Creating a New Approval Process Class

Writing a new class implementing your own approval process evaluation logic involves:

- Defining a new package for your custom class
- Creating the new class extending the core class
- Redefining the evaluate method in your class
- Writing your approval process logic:
  - To evaluate the next status of the request
  - To set the next approver(s)
  - To send notification (if required)
- Compiling your class

Once you have written and compiled your class, you deploy it and declare it in the BLM.

When creating your java class, we suggest declaring a Java Package to implement your class.

For example, `com.<yourclasspackage>.netonomy.blm.external` where `<yourclasspackage>` is the name of your company or the name of your customer.

To create a new class extending the core class

Example of extending the core class:

```
package com.<yourclasspackage>.netonomy.blm.external;
import com.netonomy.blm.interfaces.validation.ApprovalProcessIF;
import com.netonomy.blm.interfaces.validation.ApprovalStatusIF;
import com.netonomy.blm.util.BlmLogicException;
import com.netonomy.blm.api.utils.ObjectRefMgr;
import com.netonomy.blm.interfaces.validation.ApprovalIF;
import com.netonomy.blm.external.EvaluateApprovalProcess;

public class CustomEvaluateApprovalProcess extends EvaluateApprovalProcess
{


}
```

To redefine the evaluate method in your class

The approval process evaluation logic is implemented in the `evaluate` method.

The evaluate method has the following parameters:

- `ApprovalProcess`: approval process to evaluate
- `SendNotifications`: Boolean to enable or disable notifications

This flag is used by Analtical Applications. If you use Analytical Applications, refer to *Developing Analytical Applications*. If not, ignore this parameter.

The evaluate method returns an `ApprovalStatusIF` which corresponds to the global status of the approval process and the next status of the request. The status is one of the following:

- `APPROVED`: the request status will be set to `not yet submitted`
- `DENIED`: the request status will be set to `denied`
- `TO BE APPROVED`: the request status will stay unchanged.

Example of the evaluate method in your class:

```
package com.<yourclasspackage>.netonomy.blm.external;


import com.netonomy.blm.interfaces.validation.ApprovalProcessIF;
import com.netonomy.blm.interfaces.validation.ApprovalStatusIF;
import com.netonomy.blm.util.BlmLogicException;
import com.netonomy.blm.api.utils.ObjectRefMgr;
import com.netonomy.blm.interfaces.validation.ApprovalIF;
import com.netonomy.blm.external.EvaluateApprovalProcess;



public class CustomEvaluateApprovalProcess extends EvaluateApprovalProcess
{
public ApprovalStatusIF evaluate(ApprovalProcessIF approvalProcess, boolean sendNotifications) {
                ApprovalStatusIF globalStatus = null;


          return globalStatus;
          }
}
```

To code your approval process logic

You can now implement your own logic.

Here is an example of the location of approval process logic in the evaluate method:

```
package com.<yourclasspackage>.netonomy.blm.external;


import com.netonomy.blm.interfaces.validation.ApprovalProcessIF;
import com.netonomy.blm.interfaces.validation.ApprovalStatusIF;
import com.netonomy.blm.util.BlmLogicException;
import com.netonomy.blm.api.utils.ObjectRefMgr;
import com.netonomy.blm.interfaces.validation.ApprovalIF;
import com.netonomy.blm.external.EvaluateApprovalProcess;


public class CustomEvaluateApprovalProcess extends EvaluateApprovalProcess
{
public ApprovalStatusIF evaluate(ApprovalProcessIF approvalProcess, boolean sendNotifications) {
     ApprovalStatusIF globalStatus = null;
     // Type your own code to evaluate the approval process and calculate the global status to return
      if (globalStatus.equals(TO_BE_APPROVED)) {
               // Type your code to set the next approver(s)
               if (sendNotifications) {
                    // Type your own code if you want to send notifications
                    }
               }
          return globalStatus;
          }
}
```

Example of the default evaluate method code:

```
public ApprovalStatusIF evaluate(ApprovalProcessIF approvalProcess, boolean sendNotifications) {
        ApprovalStatusIF TO_BE_APPROVED = ObjectRefMgr.getApprovalStatus(ApprovalStatusIF.TO_BE_APPROVED);
        ApprovalStatusIF DENIED = ObjectRefMgr.getApprovalStatus(ApprovalStatusIF.DENIED);
        ApprovalStatusIF APPROVED = ObjectRefMgr.getApprovalStatus(ApprovalStatusIF.APPROVED);
        ApprovalIF[] approvals = approvalProcess.getApprovals();
        // Evaluate the current global status of the approval process
        boolean hasOneDeniedOrMore = false;
        boolean hasOneToBeApprovedOrMore = false;
        for (int i=0; i<approvals.length; i++) {
            if (approvals[i].getStatus().equals(TO_BE_APPROVED)) {
                hasOneToBeApprovedOrMore = true;
            }
            if (approvals[i].getStatus().equals(DENIED)) {
                hasOneDeniedOrMore = true;
            }
        }
        ApprovalStatusIF globalStatus = null;
        if (hasOneDeniedOrMore) {
            globalStatus = DENIED;
        } else {
            if (hasOneToBeApprovedOrMore) {
                globalStatus = TO_BE_APPROVED;
            } else {
                globalStatus = APPROVED;
            }
        }
        if (globalStatus.equals(TO_BE_APPROVED)) {
            // define the next to be approved
            ApprovalIF[] nextPendingApprovals = approvalProcess.getNextPendingApprovals();
            if (nextPendingApprovals.length == 0) {
                for (int i=0; i<approvals.length; i++) {
                    if (approvals[i].getStatus().equals(TO_BE_APPROVED)) {
                        approvals[i].setNext(true);
                        // update the approval process (for the setNext change)
                        // the flag stateChanged is set to false because
                        // the approval sequencer shouldn't treat this approval process again
                        // until another change
                        approvalProcess.update(false);
                        break;
                    }
                }
            }
        }
        return globalStatus;
    }
```

To compile your class

To compile your class, you need to make sure the following jar files are in your classpath:

- `lib/nmycore.jar`
- `lib/nmyutil.jar`

Deploying the Approval Class

To deploy your class

1. Create sub folders consistent with your package name. For example create a folder called `classes/com/<yourclasspackage>/netonomy/blm/external`.
2. Copy your compiled class to this folder.

To declare your class

1. Go to `<home_dir>/classes/nmycfg/blm`.
2. Open the `external_custom.xml` customization file.
3. Find the `class` element with `default` attribute equal to `com.netonomy.blm.external.EvaluateApprovalProcess`.
4. Enter the name of your custom class as the value of the `custom` attribute.

   Example:

```
<class default=" com.netonomy.blm.external.EvaluateApprovalProcess" custom="
com.<yourclasspackage>.netonomy.blm.external.CustomEvaluateApprovalProcess"/>
```

5. Save your changes.

Running the Approval Sequencer

The Approval Sequencer is an agent which evaluates requests requiring approval.

The Approval Sequencer applies the Approval Process logic whenever the information in a request requiring approval changes.

For more information about running the Approval Sequencer, refer to *Administrating Telco Service Manager*.

CHAPTER 14

Managing Errors

In This Section

About the BLM Exceptions 212
Customizing Error Messages..... 213

About the BLM Exceptions

When errors occur in the BLM, you can use the following exceptions to display customized messages to users. These chained exceptions help you display meaningful messages to users and help you manage workflow and handle security problems.

You should use or expand these exceptions instead of using generic Java exceptions.

BlmLogicException

BLM Logic Exceptions occur when an error occurs in the processing of business logic. For example, a `BlmLogicException` is thrown if you try to change the rate plan of a contract that cannot change rate plans.

BlmSecurityException

BLM Security Exceptions occur when a security violation occurs when processing business logic. For example, a `BlmSecurityException` is thrown if a user tries to view an invoice of another user.

BlmBadValueException

BLM Security Exceptions occur when the value of a parameter does not meet the specified format. For example, a `BlmBadValueException` occurs when a user tries to enter parameter value of 110 when the parameter is limited to numbers between 10 and 100.

PersistenceException

Persistence exceptions occur when an error occurs in saving the instance of the BLM object.

Customizing Error Messages

The core_<language>.properties configuration file contains the BLM error messages. The files are located in <home_dir>/classes/nmycfg/errors.

The BLM has one message file per language.

You can use one of the following language properties files:

- core_english.properties
- core_french.properties

You can customize these files to create your own error messages or use them as a template to create a new language file.

If your application does not specify a language, the BLM uses the core_english.properties configuration file.

An example of BLM error messages in the core_english.properties configuration file

BLM ERROR MESSAGES

```
# ------------------------------------------------------------------

# Contract error messages

# ------------------------------------------------------------------


2000=We are sorry. We cannot process your request. The pending declaration of loss or theft has not been processed.


# Change RatePlan

#

2010=We are sorry. The {0} rate plan is not allowed.

2011=We are sorry. You cannot change your {0} rate plan. The pending request to change this rate plan has not been
processed.

2012=We are sorry. You cannot change your current rate plan {1} to {0}.
```

CHAPTER 15

Logging Events

In This Section

About Logging Events 216
About the Logger API 219
Creating the Custom Event Code File 222
Programming Custom Event Logs 224

About Logging Events

The system logger generates standarized logs to track important system events for development and supervision. No matter which component generates a message, you can be sure that it corresponds to a set format and content. This also allows you to use supervision tools to manage and administrate your TSM.

When you extend components and create custom modules, you can also use this logger to log your own events. The BLM comes with a logger utility package that contains all the tools you need to create your own log messages.

Logging events from your code involves:

- Creating your event codes
- Using the logger to log your events

Before You Get Started

Before you get started, you need to understand how logs are created.

The basis of the logger is to create a record for events. An event is when something specific occurs while the application is running. For example, events include initializing components, loading configurations, opening connections, and so on. When an event occurs, the logger creates a structured record with attributes.

The attributes of an event record (or log entry) include:

- An associated type
- A severity level
- The module that generated the event

Logger Events

ATTRIBUTE	DESCRIPTION
Date/Time	ISO-8601 date string to identify the time
Thread id	To identify the thread that hosts the event generator
Unique id	To identify the event
Session id	To identify the user session
Type	To classify the impact of event
Severity	To classify the issue level
Module	To identify the event technical source
Code	To classify and describe what occurs
Description	To classify and describe what occurs – linked to code
Debug information	To provide additional technical information

Event Types

EVENT TYPE	DESCRIPTION
INIT	Covers application/module initialization processes
STATE	Covers application/module state changes
EXCEPTION	Covers internal exceptions
SESSION	Covers user session life cycle
REQUEST	Covers request life cycles
MESSAGE	Covers lifecycle in the integration framework
OBJECT	Covers object handling
RESOURCE	Covers component events
DATA	Covers customer data handling in the CID
NONE	Unclassified events

Severity Levels

SEVERITY LEVEL	DESCRIPTION
FATAL	Events that have an impact on the availability of the application
ERROR	Events that cause a given application workflow to not work properly
WARN	Events that may impact the behavior of the application
INFO	Events that record successful basic system actions for supervision
DEBUG	Level 0: no debug information. Level 3: for activation in a production environment within a working day. Level 5: for activation in a production environment for a limited period. Level 7: for activation in a very limited way with one or two concurrent users.

Event Modules

EVENT MODULE	DESCRIPTION
AGT	Any agent – synchronizer, connector, sequencer
BLM	Business Logic Manager
DAL	Data Access Layer
SmartLink Framework	SmartLink Framework and all of its sub modules
JSPF	JSP Framework

EVENT MODULE	DESCRIPTION
JSP	Java Server Pages
LOG	The logger platform
NIL	No module involved
UTL	Internal utility components

Event Code/description:

- Events are coded to ensure the description, for a given case, is always the same whatever the event source is.
- A code is associated with a description, which is the actual event message.
- The code is the reference to detect what occurs.

Filtering Logs

You can configure the logger to filter events by `type` and `severity`. It is also possible to filter events by `module` for given event types.

About the Logger API

The BLM comes with the `com.netonomy.util.api.logger` package you use to create your logs.

For detailed information about the classes and their methods, refer to the *UTIL API Reference Documentation*.

Available Events

You use the logger to create log entries with the following attributes:

- **Event Types**
  - INIT
  - STATE
  - EXCEPTION
  - SESSION
  - REQUEST
  - MESSAGE
  - OBJECT
  - DATA
  - NONE
- **Event Severity**
  - FATAL
  - ERROR
  - WARN
  - INFO
  - DEBUG
  - DEBUG LEVEL (used when severity is DEBUG)
- **Debug Information**

  Standard logger debug output. This technical information is a set of lines.
- **Event Code**

  The code that sets the event description.

You can only use your custom event codes with an event code greater than `2000000`. Messages with codes that are less than `2000000` are system events. You cannot use system event codes with this logger.

- **Event Description Parameters**

  A list of strings that are value parameters for event description – these parameters depend on the event code (through its associated description)

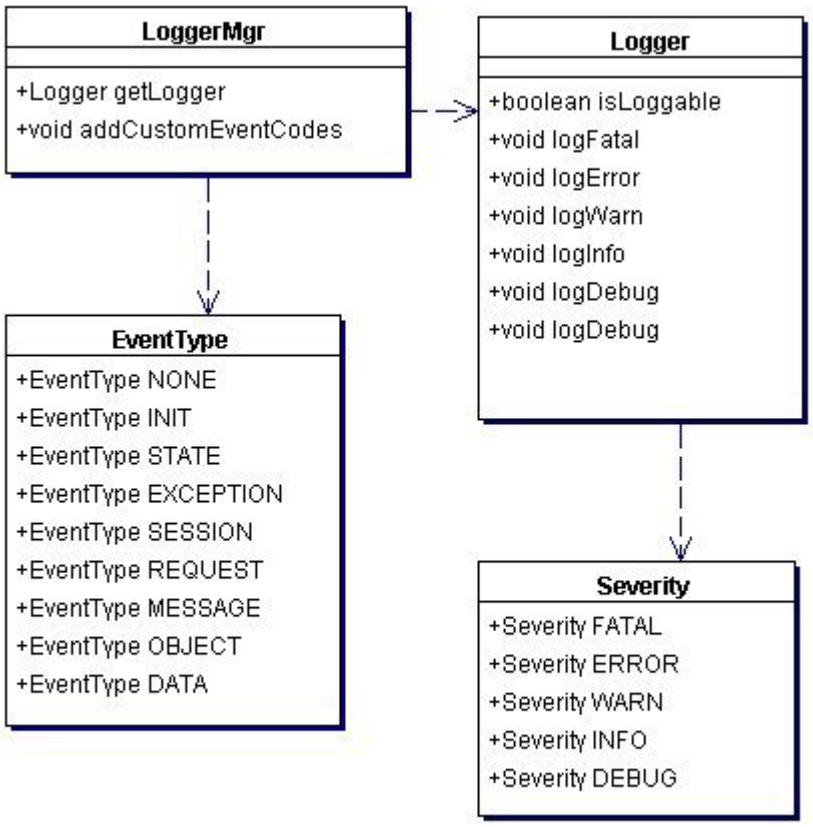
When using this logger, the Module attribute of the log is `CUS` (custom).

Custom Event Codes

You specify your event codes in a standard `properties` file. The logger comes with the methods you need to access this file via the `CLASSPATH`.

You can load as many Custom Even Code `properties` files as are required by your application.

Logger API Object Model



Creating the Custom Event Code File

When you program your application to use the logger, you need to give the logger an event code. To assure efficient and coherent integration of supervision platforms, you cannot use the standard set of system event codes. For your custom code, you create your own set of codes. This way, your event codes match the different events in your code you want to track.

The custom event codes are specified in a standard `properties` file.

Creating the custom event code file involves:

- Creating the `properties` file
- Entering your custom event codes

Creating the properties File

Using a text editor, create a `properties` file. You declare the event codes in this file. Use the syntax:

```
<code> = <description>
```

- `<code>` your event code
- `<description>` description of the event

Your description can contain dynamic information. Use the standard Java-formatting syntax to mark the dynamic sections of your description:

```
[<tag name>="{<parameter #>}"]
```

Example of the Custom Event Code File

In this example, this custom event code `properties` file declares the following event codes:

EVENT CODE	DESCRIPTION
3003100	Creating my new thread.
3003101	Creating my new thread failed.
3003102	Creating my new thread [thread id={thread_id}] succeeded.
3050000	New custom message to process in the request queue [message type id={message_type_id}], [request id={request_id}].
3050001	Retrieving custom message request from the request queue. [message type id={message_type_id}], [request id={request_id}].

The properties file contains the following:

```
#
3003100 = Creating my new thread.
3003101 = Creating my new thread failed.
3003102 = Creating my new thread [thread id={0}] succeeded.
#
3050000 = New custom message to process in the request queue [message type id={0}], [request id={1}].
3050001 = Retrieving custom message request from the request queue. [message type id={0}], [request id={1}].
```

In this sample above, the event descriptions have a mixture of static and dynamic parameters.

When logging an event using a code, the logger requires a table of parameters to be set as input parameters.

Your event codes must be greater than 2000000.

Deploying the Custom Event Code File

The TSM uses the CLASSPATH to find the custom event code properties file that contains your codes and their associated descriptions. As the logger is instantiated several times by different components such as the Synchronizer, Connectors, or BLM, this file must be available to all of these components.

To deploy the custom event code properties file, you place the file in a location accessible via the CLASSPATH.

Programming Custom Event Logs

After you have created and deployed your custom event code `.properties` file, you can use the system logger to log your custom events.

When logging your events, keep in mind that your log events are used to monitor a specific business process of your application. You must be careful when setting event attributes because these attributes are used by others to filter log files and generate alerts.

Programming your custom event logs involves:

- Loading the custom event code `.properties` file
- Initializing the `Logger` object
- Logging the event using the appropriate method

When using the logger in your JSPs, you should implement a static java class that initializes the logger objects and loads the `properties` file. Otherwise, you have to initialize the logger object and load the custom event code `properties` file in each JSP.

Loading the Custom Event Code

Before using the logger in your custom code, you must import the logger classes. This class is located in `com.netonomy.util.api.logger`.

You use the `LoggerMgr.addCustomEventCodes` method to load the custom event code `.properties` file.

Use the syntax:

```
addCustomEventCodes(String fileName)
```

- `fileName` the name of the `.properties` file containing your custom event codes

Example of Loading the Custom Event Code File

This example shows you how to:

- Import the `logger` package
- Load the `customCodes.properties` Custom Event Code file.

Import the logger package	`import com.netonomy.util.api.logger.*;`

Load the Custom Event Code file	`public class myClass {`
	`public void myClass() {`
	`LoggerMgr.addCustomEventCodes("custom/customCodes.properties");`
	`…`
	`…`
	`}`
	`}`

Initalizing the Logger

You use the `LoggerMgr.getLogger` method to initialize a Logger object. You initialize one Logger object per event type to be used.

Use the syntax:

```
getLogger(eventType)
```

You can use one of the following event types:

- `INIT` events cover application/module initialization processes
- `STATE` events cover application/module state changes
- `EXCEPTION` events cover internal exceptions
- `SESSION` events cover user session life cycle
- `REQUEST` events cover request life cycles
- `MESSAGE` events cover lifecycle in the integration framework
- `OBJECT` events cover object handling
- `DATA` events cover customer data handling in the CID
- `EXCEPTION` events cover internal exceptions
- `NONE` events are unclassified events

Example of Logging INIT and MESSAGE Events

This example shows you how to:

- Import the logger package
- Initalize the following loggers:
  - `INIT`
  - `MESSAGE`

Import the logger package	`import com.netonomy.util.api.logger.*;`

Initalize an INIT logger	`public class myClass {`
	`public void myClass() {`
	`LoggerMgr myLogger_initEvents = LoggerMgr.getLogger(EventType.INIT);`

Initalize a MESSAGE logger	`LoggerMgr myLogger_messageEvents = LoggerMgr.getLogger(EventType.MESSAGE);`
	`…`
	`…`
	`}`
	`}`

Checking Severity Settings

You can use the Logger API to see if a given severity has been activated. This means your custom code can check to see if a given severity level has been activated before you carry our different actions required to obtain information for the logger output.

For instance, before collecting information about various system settings and taking a snapshot of your system configuration while debugging, your code can check to see if this information is to be put in the log message. This way you can keep from using system resources to generate information that will not be logged in the current logger configuration.

You use the `Logger.isLoggable` method to verify if the severity has been activated for logging. Use the syntax:

```
isLoggable(severity)
```

- `severity` the severity to check

Example of Severity Checking Before Logging

This example shows you how to:

- Import the logger package
- Load the Custom Event Code `.properties` file
- Test the DEBUG severity

Import the logger package	`import com.netonomy.util.api.logger.*;`

Load the Custom Event Code file	`public class myClass {`
	` public void myClass() {`
	
	`// Load event codes`
	`LoggerMgr.addCustomEventCodes("custom/customCodes.properties");`

Get a Logger object for INIT event type	`//Get logger` `LoggerMgr myLogger_initEvents = LoggerMgr.getLogger(EventType.INIT);`
Test the DEBUG severity	`// Test using the DEBUG severity is relevant` `if (myLogger_initEvents.isLoggable(Severity.DEBUG)) {`

If TRUE, build the debug info and log the event	`// Build the debug info`
	`String debugInfo;`
	`….`
	`// Log the event`
	`myLogger_initEvents.logDebug("03100420", null, debugInfo, 7);`
	
	`}`
	`else {`
	`// no event logged`
	
	`}`
	
	`}`
	`}`

Logging Standard Messages

You use the Logger object method that corresponds to the severity of the message you want to log.

Use the syntax:

```
log<Severity>(String eventCode, Object[] params, String
debugInfo)
```

- <severity> is one of the following:

FATAL

ERROR

WARN

INFO

- `eventCode` one of your custom codes.
- `params` a one-dimension table that handles the parameters to be used to build the description. This table must have as many elements as the number of parameters declared in the description associated with the code.
- `debugInfo` a string that contains technical information to be logged as `DEBUG_INFO`. For instance, items of a list, HTTP stream, XML stream, and so on. This string may be a block of strings

Example of a Simple Event Log

This example shows you how to:

- Import the logger package
- Load the custom event code `.properties` file
- Log an `INIT FATAL` event having:
  - Event Code of 03100420
  - A static message

Import the logger package	`import com.netonomy.util.api.logger.*;`

Load the Custom Event Code file	`public class myClass {`
	` public void myClass() {`
	
	`// Load event codes`
	`LoggerMgr.addCustomEventCodes("custom/customCodes.properties");`

Get a Logger object for INIT event type	`//Get logger` `LoggerMgr myLogger_initEvents = LoggerMgr.getLogger(EventType.INIT);`

Log the event	`// Log the event`
	`myLogger_initEvents.logFatal("03100420", null, null);`
	
	`}`
	`}`

Example of a Dynamic Event Log

This example shows you how to:

- Import the logger package
- Load the custom event code `.properties` file
- Log an `REQUEST INFO` event having:
  - Event Code of 02600360
  - A dynamic message

Import the logger package	`import com.netonomy.util.api.logger.*;`

Load the Custom Event Code file	`public class myClass {`
	` public void myClass() {`
	
	`// Load event codes`
	`LoggerMgr.addCustomEventCodes("custom/customCodes.properties");`

Get a Logger object for REQUEST event type	`// Get a logger` `LoggerMgr myLogger_requestEvents = LoggerMgr.getLogger(EventType.REQUEST);`
Build the description parameters table	`// Build the description parameters table` `Object [] eventParameters = new Object [] {"#1 parameter"};`

Log the event	`// Log the event`
	`myLogger_ requestEvents.logInfo("02600360", eventParameters, null);`
	
	`}`
	`}`

Logging Debug Messages

You use the Logger object `logDebug` method to log debug information.

Use the syntax:

```
logDebug(String eventCode, Object[] params, String debugInfo,
int debugLevel)
```

- `eventCode` one of your custom codes.
- `params` a one-dimension table that handles the parameters to be used to build the description. This table must have as many elements as the number of parameters declared in the description associated with the code.

- `debugInfo` a string that contains technical information to be logged as `DEBUG_INFO`. For instance, items of a list, HTTP stream, XML stream, and so on. This string may be a block of strings
- `debugLevel` the level of logged information and the usage:

  Level 0: no debug information

  Level 3: for activation in a production environment within a working day

  Level 5: for activation in a production environment for a limited period

  Level 7: for activation in a very limited way with one or two concurrent users

Example of Debug

This example shows you how to:

- Import the logger package
- Load the custom event code `.properties` file
- Log an `INIT DEBUG` event:
  - Having Event Code of 04200440
  - A dynamic message
  - Some DEBUG info
  - DEBUG level set to 7

Import the logger package	`import com.netonomy.util.api.logger.*;`

Load the Custom Event Code file	`public class myClass {`
	` public void myClass() {`
	
	`// Load event codes`
	`LoggerMgr.addCustomEventCodes("custom/customCodes.properties");`

Get a Logger object for INIT event type	`//Get logger` `LoggerMgr myLogger_initEvents = LoggerMgr.getLogger(EventType.INIT);`
Build the description parameters table	`// Build the description parameters table` `Object [] eventParameters = new Object [] {"#1 parameter"};`
Build the debug info	`// Build the debug info` `String debugInfoString = "Debug line 1\n" + "Debug line 2\n" + "Debug line 3\n";`

Log the event	`// Log the event`
	`// The debug level is set to 7 because of the context:`
	`// advanced debugging while the application is offline.`
	`myLogger_ sessionEvents.logDebug("04200440", eventParameters, debugInfoString, 7);`
	
	`}`
	`}`

This debug event is to track a very specific workflow result in an advanced debugging context. For performance reasons, the application is not available to end-users.

Logging Messages During Development

The system logger has been designed to ease application supervision and create standard messages for supervision platforms. The logger API uses this system logger, which requires that each event have a specific code before the event can be logged.

This is not very practical when debugging an application. You cannot code all of the possible event codes before debugging. And when debugging, you may need to quickly log some information to pinpoint a specific problem that does not require an event code.

The logger comes with a special method for DEBUG events that does not require an Event Code. Use the syntax:

```
logDebug(String description, String debugInfo, int debugLevel)
```

- `description` free form description
- `debugInfo` optional free form, multi-line debug information
- `debugLevel` verbosity (3=low frequency, 5=medium frequency, 7=heavy frequency)

This should be used for development debugging only. All other DEBUG log messages should use an Custom Event code in order to ensure efficient supervision of your application.

All other log messages (`INFO`, `WARN`, and so on) require a Event Code in the Custom Event Code `.properties` file.

Example of Logging for Development

This example shows you how to:

- Import the logger package
- Load the custom event code `.properties` file
- Log an `INIT DEBUG` event using the development `logDebug` method

Import the logger package	`import com.netonomy.util.api.logger.*;`

Load the Custom Event Code file	public class myClass {
	public void myClass() {
	
	// Load event codes
	LoggerMgr.addCustomEventCodes("custom/customCodes.properties");

Get a Logger object for INIT event type	//Get logger LoggerMgr myLogger_initEvents = LoggerMgr.getLogger(EventType.INIT);
Test a condition	// Test using the DEBUG severity is relevant if (<condition>) {

Log the event	// Log the debug event
	myLogger_initEvents.logDebug("my message 1", null, 7);
	
	}
	else {
	
	// Log the debug event
	myLogger_initEvents.logDebug("my message 2", null, 7);
	
	}
	
	}
	}

This debug method is for development purposes only.

CHAPTER 16

Working with User Events

In This Section

About User Events 234
About Creating Custom User Events 235

About User Events

User events are log entries which you use to trace certain application events.

User events belong to one of the following categories:

- Session events (login, logout, session expiration)
- Execution features (creation of requests)
- Custom user events
- DO events generated by an OSS

For information about purging User Events in the CID, refer to *Administrating Telco Service Manager*.

About Creating Custom User Events

You can easily modify the user event tracking to meet your needs. You can extend the list of user events to include your own user events.

Customizing user events involves:

- Creating your own type of user event
- Inserting the call to a user event at a specific point of JSP processing

To create a custom user event type

1. Use your database tool to connect to the CID.
2. In the `USER_TYPE_EVENT` table, create a row and enter the following required information:
   - `USER_TYPE_EVENT_ID`

     The value should be greater than 10000
   - `USER_TYPE_EVENT_CODE`

     This code must begin with `CS_`
   - `USER_TYPE_CATEGORY_ID`

     The value to enter is `4` corresponding to Custom User Events in the `USER_TYPE_EVENT_CATEGORY` table.
3. If required, enter the following:
   - `USER_TYPE_EVENT_NAME`
   - `STRING_ID`
   - `USER_TYPE_EVENT_DESCRIPTION`
   - `USER_TYPE_EVENT_DESCRIPTION_STRING_ID`
4. In the `ACTIVATION_FLAG` column, enter 1.
5. Save your changes.
6. Restart your application server.

To program a user event in a JSP

1. Open the JSP.
2. Use the `UserEventIF` methods to insert a user event.

Example of a Login User Event

```
UserEventIF toSave = ObjectMgr.createUserEvent ();
toSave.setStatusCode("LOGIN_SUCCESS");
toSave.setUserEventType(ObjectRefMgr.getUserEventTypeFromCode("LOGIN"));
toSave.setLoginID (blmSession.getUserF().getIdentifier());
ParameterIF[] params = new ParameterIF[3];
params[0] = ObjectRefMgr.getParameterByCode ("APPSERVSID");
params[0].setValue (session.getId());
params[1] = ObjectRefMgr.getParameterByCode ("USERAGENT");
params[1].setValue (jspHelper.nonNullString(request.getHeader("User-Agent")));params[2] =
ObjectRefMgr.getParameterByCode ("ACCESCHAN");
params[2].setValue ("0");
toSave.setParameters (params);
UserEventF.saveUserEvent (toSave);
```

Example of a User Event

In the `ListEventHistory form_handler` JSP, this code searches the DO user evens impacting a contract.

```
<%!

/**
 * Form handler for a list of events on contract administered by a telco or dealer
 *
 * @param session    The current HTTP session
 * @param request    The current HTTP request
 * @param response   The current HTTP response
 * @param objHelper  The JSP helper class
 * @param results        Hashtable of returned objects
 * @param errors         Hashtable of error objects
 */

public void formHandler_listContractHistory (HttpSession session,
                                        HttpServletRequest request,
                                        HttpServletResponse response,
                                        JFNJspHelper jspHelper,
                                        Hashtable results,
                                        Hashtable errors) throws Throwable
{
    ParameterIF[] criteria;
    FilterIF filter;
    SessionF  blmSession = jspHelper.getBlmSession();
    UserF user = jspHelper.getUser();
    ContractF contract = new ContractF (blmSession, ObjectId.instantiate(request.getParameter ("contract")));
    // get filter, fill mandatory criteria
    filter = ObjectRefMgr.getFilterByCode("CORE_UEVTSBYIMPACTEDORCREATEDOBJECT");
    criteria = filter.getCriteria(FilterIF.ALL);

    ((ValueDynamicIF) ParameterHelper.getParameterByCode(criteria,
"CORE_C_UEVTIMPACTEDOBJECTTYPEID")).setDynamicValue(new ObjectId[]
{contract.getContractType().getIdentifier()});

    ((ValueDynamicIF) ParameterHelper.getParameterByCode(criteria,
"CORE_C_UEVTIMPACTEDOBJECTID")).setDynamicValue(new ObjectId[] {contract.getIdentifier()});

    ((ValueDynamicIF) ParameterHelper.getParameterByCode(criteria,
"CORE_C_UEVTCREATEDOBJECTTYPEID")).setDynamicValue(new ObjectId[]
{contract.getContractType().getIdentifier()});

    ((ValueDynamicIF) ParameterHelper.getParameterByCode(criteria,
"CORE_C_UEVTCREATEDOBJECTID")).setDynamicValue(new ObjectId[] {contract.getIdentifier()});

    ((ValueDynamicIF) ParameterHelper.getParameterByCode(criteria,
"CORE_C_UEVTTYPECATEGORY")).setDynamicValue(new ObjectId[]
{ObjectRefMgr.getUserEventTypeCategoryByCode("CORE_UEVT_0003").getIdentifier()});

    UserEventIF[]   userEvents = ObjectMgr.searchUserEvents(filter);
    request.setAttribute("userevents", userEvents);
    request.setAttribute("type", "contract");
    request.setAttribute ("display", Boolean.TRUE);
}
%>
```

In the `view_history` JSP, this code displays the user events found.

```
<table cellspacing="0" cellpadding="2">

<tr>
  <td class="headerText" width="80"><%=jspHelper.localize ("date_header")%></td>
  <td class="headerText"><%=jspHelper.localize ("event_name_header")%></td>
  <% if (request.getAttribute("type") != null) { %>
  <td class="headerText"><%=jspHelper.localize ("object_header")%></td>
  <% } %>
</tr>
<%
  //'*** now we can display history ***
  int n=0;
  boolean isFirst;
  isFirst=true;

  //' Display requests

  for (int intInfo = 0;intInfo != result.length; intInfo ++)
    {
      n++;
      %>
<tr><%-- for alternate background color --%>
  <td class="<%= (n % 2)==0 ? "listAText" : "listBText" %>">
    <%= jspHelper.convertToDisplayString(result[intInfo].getUserEventDate()) %>
  </td>
  <td class="<%= (n % 2)==0 ? "listAText" : "listBText" %>"><%=
result[intInfo].getUserEventType().getName()%></td>
  <% if (request.getAttribute("type") != null) { %>
  <td class="<%= (n % 2)==0 ? "listAText" : "listBText" %>"><%=
result[intInfo].getUserEventType().getDescription()%></td>
  <% } %>
</tr>
<%
  }
%>
</table>
```

CHAPTER 17

Deploying TSM

In This Section

About Deploying 240
For WebSphere 4.x..... 241
For WebLogic 6.x and 7.x..... 247
For Oracle 9i Application Server..... 252

About Deploying

After installing and configuring, you deploy the TSM channels as a web application. When you deploy your channel, you are telling the application server where to find your TSM's JSPs and components. As each application server handles JSPs and other files differently, deploying your channel depends on the version and the editor of your application server.

This section covers deploying your application on different application servers. Deploying channels can be as easy as configuring your application server to look for the JSPs in a directory. Other application servers recommend that you deploy web applications as a J2EE Web Application aRchive (WAR) file.

For detailed information about configuring and deploying web applications, refer to your application server's documentation and *Installing Telco Service Manager*.

For WebSphere 4.x

Deploying Channels on WebSphere 4.x involves:

- Configuring your environment
- Creating and deploying a WAR file
- Configuring the deployed channel

Configuring Your Environment

Depending on your application server and environment, you may have to carry out certain tasks before you can deploy your channel.

For this application server, preparing your environment involves:

- Creating the data source
- Specifying Java memory settings

To create a data source

1. Start the WebSphere Administration Server.
2. Start the Administration Console.
3. Choose *Console>Wizards>Create Data Source*. The wizard opens.
4. Enter the following information:

   Name: `cidDatasource`

   Database name: your database name
5. Choose *Next.*
6. Choose *Create a new JDBC Driver.*
7. Enter the following information:
   - *Name:* `cidDatasource`
   - *Implementation class*

   For the name of your implementation class, refer to your application server documentation.
8. Choose *Next*. A summary window appears.
9. Choose *Finish*.
10. On the console tree, go to the *Resources>JDBC Providers>cidDatasource/Data Sources* node.
11. On the *General* tab, enter the following:
    - *User ID:* your CID user name

- *Password*: the associated password
- Add the required Custom Properties for your database.

  For more information about custom properties, select the *Help* button on the tab.

12 Choose *Apply*.

To specify Java memory settings

1 Start the WebSphere Administration Server.

2 Start the Administration Console.

3 Go to the *Nodes>Node Name>Application Servers/Default Server* node on the console tree.

4 Change the Command Line Parameter to increase the amount of allocated memory. By default, WebSphere does not allocate enough memory.

For example, enter `-Xms196m -Xmx196m` to increase the allocated memory to 196 MB.

Creating and Deploying a WAR File

In Java2EE, Sun Microsystems published the specification and tools to create Web Application Archive files (WAR) files. A WAR file is a JAR file containing java class files and other files required by Web applications (utility classes, HTML files, applets, and so on.) When using WAR files, you create a single file containing all of the required files for easy deployment on all Java2EE-compliant application servers.

A Web application can be run from directly from the WAR file or a directory that conforms to the WAR specification.

The `WAR` file for TSM contains the following:

- Personalization Manager channel JSPs
- `WEB-INF` directory containing:
  - `web.xml` file describing the application

- `WEB-INF/lib` directory containing jar files
- `WEB-INF/classes` directory containing configuration files

After you generate your WAR file, you install this file using the application server's administration console.

Building a WAR file involves:

- Configuring the `web.xml` file
- Moving configuration files
- Generating the WAR file
- Deploying the WAR file

The java command for building WAR files (`jar.exe`) is located in your application server's copy of the Java Development Kit (JDK)

To configure the web.xml File

1. Go to `<app_dir>/WEB-INF` where `<app_dir>` is the location of your channel files.
2. Open `web.xml`.
3. Do the following:
   - In the `<display-name>` element, enter the name of your TSM application.
   - In the `<description>` element, enter a description of the application.

   The name and description you enter here are for the application server only.

   ---
4. Save your changes.

Example of web.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <display-name>Welcome to MyWeb</display-name>
  <description>Welcome to MyWeb</description>
</web-app>
```

To prepare your files

During installation, there are two identical sets of TSM configuration files. This is done to respect the requirements of the J2EE Web Application aRchive (WAR) file specifications and to help you easily deploy your TSM application.

The sets of configuration files are:

- Core configuration files

  These configuration files are in `<home_dir>/classes/nmycfg`.

- Channel configuration files

  These configuration files are in `<app_dir>/WEB-INF/classes/nmycfg` where `<app_dir>` is the location of your channel files. By default, they are installed in `<home_dir>/Channels`.

However, when you develop your TSM application, you may need to modify some of the core configuration files. When deploying your TSM application, you must make sure that your modifications are also in the Channel configuration files. If you do not, your deployed application will not behave as expected.

1 Copy the entire directory structure (`nmycfg/...`) of core configuration files located under `<home_dir>/classes` to `<app_dir>/WEB-INF/classes`.

  Your directory structure is now a deployable directory structure that conforms to the WAR specification. The structure should look like this:

DIRECTORIES			CONTENTS
<Channels>/	common/		Contents of <home_dir>/channels/common
	MyWeb/		Contents of <home_dir>/channels/MyWeb
	MyWap/		Contents of <home_dir>/channels/MyWap
	MyIvr/		Contents of <home_dir>/channels/MyIvr
	WEB-INF/		The web.xml file
		lib/	Copy of the required jar files in <home_dir>/lib
		classes/	Copy of <home_dir>/classes

To generate the WAR file

1 Go to `<home_dir>/Channels`.

2 Generate the WAR file. Use the syntax:

```
jar -cvf <home_dir>/myweb.war .
```

Do not forget the final period at the end of this command.

This Java command generates a WAR file called `myweb.war` in `<home_dir>`. You can generate this file in another directory if required. You use your application server's administration console to locate and deploy the generated WAR file.

To deploy the WAR file

1. Start the WebSphere Administration Server.
2. Start the Administration Console.
3. Choose *Console>Wizards>Install Enterprise Application*. The wizard opens.
4. Choose *Install stand-alone module.*
5. Enter the following information:

   Path: `<home_dir>/myweb.war`

   Name: `MyWeb`

   Root: `/`
6. Choose *Next.* Accept the default values and choose *Next* until you exit the wizard.

Configuring Deployed Channels

During installation, the installer creates directories and copies files in standard default locations.

There is also a set of configuration files that have default values that you may have to change for your environment.

After deploying your channels, you need to configure some configuration files. You modify the files found in your `<DEPLOY_DIR>`.

By default, WebSphere deploys the war files in `<WEBSPHERE_HOME>/AppServer/installedApps/MyWeb.ear/myweb.war`.

Configuring deployed channels involves:

- Changing path to log files in `log4j.properties`

To change logger paths

1. Go to `<DEPLOY_DIR>/WEB-INF/classes/nmycfg/util`.
2. Open `log4j.properties`.
3. Do the following:
   - Set `log4j.appender.ROL.File` to `/tmp/nmy_application.log`
   - Set `log4j.appender.DAY.File` to `/tmp/nmy_daily_application.log`

4 Save your changes.

Accessing a Deployed Channel

Before you can access your application, you need to restart your WebSphere server along with any other required components.

After you restart your application components, you can access your application using the following URL:

`http://host:port/<instance_name>/MyWeb/index.jsp`

where

`http://host:port/<instance_name>` corresponds to your WebSphere Server instance

For WebLogic 6.x and 7.x

Deploying Channels on WebLogic 6.x and 7.x involves:

- Configuring your environment
- Creating and deploying a WAR file

Configuring Your Environment

Depending on your application server and environment, you may have to carry out certain tasks before you can deploy your channel.

For this application server, preparing your environment involves:

- Creating the connection pool
- Creating the data source

To create a connection pool

Before creating your data source, you must create and configure a connection pool.

Refer to your application server documentation for more information about creating connection pool and activating the database connectivity.

For SQL Server, you must change the default `SelectMethod` in the JDBC connection string properties. The default `SelectMethod` is `direct`. The `SelectMethod` must be set to `cursor`.

To create a data source

1. Start the Weblogic Server.
2. Open the Weblogic Server Console.
3. Under *JDBC*, click *Data Sources*. The JDBC Data Sources page appears.
4. Click *Configure a new JDBC Data Source*. The Configure JDBC Data Sources page appears.
5. On the *Configuration* tab, enter the following:

FIELD	VALUE
Name	`cidDatasource`
JNDI Name	`jdbc/cidDatasource`

6. Click *Create*. The data source appears on the top of the page.
7. Click the home icon to return to the console home page.

Your WebLogic Server now has a declared data source corresponding to the CID.

Creating and Deploying a WAR File

In Java2EE, Sun Microsystems published the specification and tools to create Web Application Archive files (WAR) files. A WAR file is a JAR file containing java class files and other files required by Web applications (utility classes, HTML files, applets, and so on.) When using WAR files, you create a single file containing all of the required files for easy deployment on all Java2EE-compliant application servers.

A Web application can be run from directly from the WAR file or a directory that conforms to the WAR specification.

The `WAR` file for TSM contains the following:

- Personalization Manager channel JSPs
- `WEB-INF` directory containing:
  - `web.xml` file describing the application
- `WEB-INF/lib` directory containing jar files
- `WEB-INF/classes` directory containing configuration files

After you generate your WAR file, you install this file using the application server's administration console.

Building a WAR file involves:

- Configuring the `web.xml` file
- Moving configuration files
- Generating the WAR file
- Deploying the WAR file

The java command for building WAR files (`jar.exe`) is located in your application server's copy of the Java Development Kit (JDK)

To configure the web.xml File

1. Go to `<app_dir>/WEB-INF` where `<app_dir>` is the location of your channel files.
2. Open `web.xml`.
3. Do the following:
   - In the `<display-name>` element, enter the name of your TSM application.
   - In the `<description>` element, enter a description of the application.

The name and description you enter here are for the application server only.

---

4 Save your changes.

Example of web.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <display-name>Welcome to MyWeb</display-name>
  <description>Welcome to MyWeb</description>
</web-app>
```

To prepare your files

During installation, there are two identical sets of TSM configuration files. This is done to respect the requirements of the J2EE Web Application aRchive (WAR) file specifications and to help you easily deploy your TSM application.

The sets of configuration files are:

- Core configuration files

  These configuration files are in `<home_dir>/classes/nmycfg`.
- Channel configuration files

  These configuration files are in `<app_dir>/WEB-INF/classes/nmycfg` where `<app_dir>` is the location of your channel files. By default, they are installed in `<home_dir>/Channels`.

However, when you develop your TSM application, you may need to modify some of the core configuration files. When deploying your TSM application, you must make sure that your modifications are also in the Channel configuration files. If you do not, your deployed application will not behave as expected.

1 Copy the entire directory structure (`nmycfg/...`) of core configuration files located under `<home_dir>/classes` to `<app_dir>/WEB-INF/classes`.

Your directory structure is now a deployable directory structure that conforms to the WAR specification. The structure should look like this:

DIRECTORIES			CONTENTS
<Channels>/	common/		Contents of <home_dir>/channels/common
	MyWeb/		Contents of <home_dir>/channels/MyWeb

DIRECTORIES			CONTENTS
	MyWap/		Contents of <home_dir>/channels/MyWap
	MyIvr/		Contents of <home_dir>/channels/MyIvr
	WEB-INF/		The web.xml file
		lib/	Copy of the required jar files in <home_dir>/lib
		classes/	Copy of <home_dir>/classes

To generate the WAR file

1. Go to `<home_dir>/Channels`.
2. Generate the WAR file. Use the syntax:

   ```
   jar -cvf <home_dir>/myweb.war .
   ```

   Do not forget the final period at the end of this command.

   This Java command generates a WAR file called `myweb.war` in `<home_dir>`. You can generate this file in another directory if required. You use your application server's administration console to locate and deploy the generated WAR file.

To deploy your WAR file

1. Start the WebLogic Server.
2. Open the WebLogic Server Console.
3. Under *Deployments*, click Web Applications. The *Web Applications* page appears.
4. Click *Install a new Web Application*. The Upload and Install an Application page appears.
5. Click *Browse* to locate your `myweb.war` file. After locating the file, you return to the *Upload and Install an Application* page.
6. Click *Upload* to begin installation. The *Upload and Install an Application* page displays a message when the installation is finished.

Accessing a Deployed Channel

Before you can access your application, you need to restart your WebLogic server along with any other required components.

After you restart your application components, you can access your application using the following URL:

`http://host:port/<instance_name>/MyWeb/index.jsp`

where

`http://host:port/<instance_name>` corresponds to your WebLogic Server instance

For Oracle 9i Application Server

Deploying Channels on WebLogic 6.x and 7.x involves:

- Configuring your environment
- Creating and deploying a WAR file

Configuring Your Environment

Depending on your application server and environment, you may have to carry out certain tasks before you can deploy your channel.

For this application server, preparing your environment involves:

- Creating the data source

To create a data source

1. Start the Oracle 9i Application Server Administration Server.
2. Open the Web Oracle 9i Application Server Administration Console.
3. Under *Applications*, select the application to create a data source for.
4. Under *Administration* > *Application Defaults*, click *Data Sources*. The *Data Sources* page appears showing the available data sources for this application.
5. Click *Create Data Source*. The *Create Data Source* page appears.
6. Under *General*, enter the following:

FIELD	VALUE
Name	cidDatasource
Description	A description of the data source
Data Source Class	com.evermind.sql.DriverManagerDataSource
Schema	leave blank
Username	your CID user name
Password	associated password
JDBC URL	refer to your application server's documentation
JDBC Driver	refer to your application server's documentation

7. Under *JNDI Locations*, enter the following:

FIELD	VALUE
Location	jdbc/cidDatasource
Transactional (XA) Location	jdbc/XA/cidDatasource
EJB Location	jdbc/ejb/cidDatasource

8 Click *Create*. The confirmation page appears.

9 Click *Yes* to restart the instance. You must restart the instance to take into account your changes.

Creating and Deploying a WAR File

In Java2EE, Sun Microsystems published the specification and tools to create Web Application Archive files (WAR) files. A WAR file is a JAR file containing java class files and other files required by Web applications (utility classes, HTML files, applets, and so on.) When using WAR files, you create a single file containing all of the required files for easy deployment on all Java2EE-compliant application servers.

A Web application can be run from directly from the WAR file or a directory that conforms to the WAR specification.

The `WAR` file for TSM contains the following:

- Personalization Manager channel JSPs
- `WEB-INF` directory containing:
  - `web.xml` file describing the application
- `WEB-INF/lib` directory containing jar files
- `WEB-INF/classes` directory containing configuration files

After you generate your WAR file, you install this file using the application server's administration console.

Building a WAR file involves:

- Configuring the `web.xml` file
- Moving configuration files
- Generating the WAR file
- Deploying the WAR file

The java command for building WAR files (`jar.exe`) is located in your application server's copy of the Java Development Kit (JDK)

To configure the web.xml File

1. Go to <app_dir>/WEB-INF where <app_dir> is the location of your channel files.
2. Open web.xml.
3. Do the following:
   - In the <display-name> element, enter the name of your TSM application.
   - In the <description> element, enter a description of the application.

   The name and description you enter here are for the application server only.
4. Save your changes.

Example of web.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <display-name>Welcome to MyWeb</display-name>
  <description>Welcome to MyWeb</description>
</web-app>
```

To prepare your files

During installation, there are two identical sets of TSM configuration files. This is done to respect the requirements of the J2EE Web Application aRchive (WAR) file specifications and to help you easily deploy your TSM application.

The sets of configuration files are:

- Core configuration files

  These configuration files are in <home_dir>/classes/nmycfg.
- Channel configuration files

  These configuration files are in <app_dir>/WEB-INF/classes/nmycfg where <app_dir> is the location of your channel files. By default, they are installed in <home_dir>/Channels.

However, when you develop your TSM application, you may need to modify some of the core configuration files. When deploying your TSM application, you must make sure that your modifications are also in the Channel configuration files. If you do not, your deployed application will not behave as expected.

1 Copy the entire directory structure (`nmycfg/...`) of core configuration files located under `<home_dir>/classes` to `<app_dir>/WEB-INF/classes`.

Your directory structure is now a deployable directory structure that conforms to the WAR specification. The structure should look like this:

DIRECTORIES			CONTENTS
<Channels>/	common/		Contents of <home_dir>/channels/common
	MyWeb/		Contents of <home_dir>/channels/MyWeb
	MyWap/		Contents of <home_dir>/channels/MyWap
	MyIvr/		Contents of <home_dir>/channels/MyIvr
	WEB-INF/		The web.xml file
		lib/	Copy of the required jar files in <home_dir>/lib
		classes/	Copy of <home_dir>/classes

To generate the WAR file

1 Go to `<home_dir>/Channels`.

2 Generate the WAR file. Use the syntax:

```
jar -cvf <home_dir>/myweb.war .
```

Do not forget the final period at the end of this command.

This Java command generates a WAR file called `myweb.war` in `<home_dir>`. You can generate this file in another directory if required. You use your application server's administration console to locate and deploy the generated WAR file.

To deploy your WAR file

1 Start the Oracle 9i Application Server Administration Server.

2 Open the Web Oracle 9i Application Server Administration Console.

3 Select the OC4J instance to use for your TSM application.

4 Under *Deployed Applications* > *Applications*, click *Deploy WAR file*. The *Deploy Web Application* page appears.

5 Do the following:

- In *Web Application*, enter the full path of the channel WAR file to deploy.

- In *Application Name*, enter the channel name. This is the <APPLICATION_NAME> directory.
- In *Map to URL*, enter the absolute path corresponding to the URL of the MyWeb channel. This is the <MAP_TO_URL> directory. For instance /myweb.

6 Click *Deploy*. The confirmation page appears.

When Oracle 9i Application Server deploys your WAR file, it does the following:

- Copies the WAR file to <ORACLE_HOME>/j2ee/<your_OC4J_instance>/applications/<APPLICATION NAME>
- Extracts the directories and files in the WAR file under <ORACLE_HOME>/j2ee/<your_OC4J_instance>/applications/<APPLICATION_NAME>/<APPLICATION_NAME>/

  This is the <DEPLOY_DIR> directory.

Configuring Deployed Channels

During installation, the installer creates directories and copies files in standard default locations.

There is also a set of configuration files that have default values that you may have to change for your environment.

After deploying your channels, you need to configure some configuration files. You modify the files found in your <DEPLOY_DIR>.

Configuring deployed channels involves:

- Changing path to log files in log4j.properties
- Precompiling your channel JSPs

After modifying any of these settings, you must restart your OC4J instance.

To change logger paths

1 Go to <DEPLOY_DIR>/WEB-INF/classes/nmycfg/util.

2 Open log4j.properties.

3 Do the following:

- Set log4j.appender.ROL.File to /tmp/nmy_application.log
- Set log4j.appender.DAY.File to /tmp/nmy_daily_application.log

4 Save your changes.

To precompile your channel JSPs

When the Oracle 9i Application Server deploys your WAR file, it unpackages the files and moves them to their required locations.

The Oracle 9i Application Server comes with a JSP pre-compiler called `ojspc`. You can use this precompiler to compile your deployed JSPF channel JSPs. Depending on the size of your application, precompiling your JSP may take a few minutes.

When running, the Oracle 9i Application Server compiles the JSPs and places the resulting class files in `<ORACLE_HOME>/j2ee/<your_OC4J_instance>/application-deployments/<APPLICATION_NAME>/<APPLICATION_NAME>/persistence/_pages`. This is the `<COMPILE_DIR>`.

Before running this command, you need to make sure your JVM is set to use at least 256MB.

---

1. Go to `<ORACLE_HOME>/bin`.
2. Open the `ojspc` file and find following JVM command `$JAVA_HOME/bin/java`
3. Change the `-mx` parameter to `256` (or greater) and save your changes.

   Your `ojspc` file should now have the following JVM command line:

   ```
   $JAVA_HOME/bin/java  -ms64m -mx256m -classpath...
   ```
4. Go to the `<DEPLOY_DIR>` directory.
5. Run the `ojspc` compiler. Use the syntax:

   ```
   ojspc -d <COMPILE_DIR>/WEB-INF/MyWeb/*.jsp
   ```

   When finished, the precompiler does not display a message. If an error occurs while compiling, the compiler displays all corresponding messages in the application console.
6. Repeat this command for each channel.

Accessing a Deployed Channel

Before you can access your application, you need to restart your OC4J instance and your application server along with any other required components.

After you restart your application components, you can access your application using the following URL:

`http://host:port/<MAP_TO_URL>/MyWeb/index.jsp`

where

`http://host:port` corresponds to your 9iApplication Server instance

`MyWeb` the channel name

`<MAP_TO_URL>/MyWeb/index.jsp` is the path to your channel.

Index

A

accessors_custom.properties
  declaring search methods • 77, 81
ActionManager
  about • 152
  and persistent ActionManagers • 187
  and shopping carts • 159, 160
  approval processes • 200
  changes to BLM Objects • 154
  hierarchy • 165, 166
  saving • 184
  types • 160, 170
Approval Processes
  about • 198, 203
  compiling • 208
  creating • 199
  customizing • 203, 204, 205, 206, 209
  declaring approval process • 199
  displaying approval requests • 201
  submitting requests • 200
  using • 201, 209
Auditing
  customizing User Events • 235

B

BLM (Business Logic Manager)
  and external approval processes • 203, 209
  configuring • 21
  customizing • 101
  errors • 211
  reference data • 142, 145
  search features • 86
BLM Objects
  managing changes • 152, 153, 154, 155
  refreshing • 145, 146, 147, 148, 149
Bulk Ordering
  about • 192
  limits • 192
  using • 193, 194, 195, 196
Business Logic
  customizing • 101
  using custom classes • 101

C

Channels
  accessing deployed channels • 246, 251, 258
  deploying • 241, 247, 252
CID (Customer Interaction Datastore)
  about • 26
  accessing external data sources • 116
  adding search filters • 75, 76, 77
  and BLM Object changes • 152, 153, 154, 155
  approval processes • 198
  configuring data access • 241, 247, 252
  customizing • 26
  optimizing for searches • 74
  users • 109
Configuration Files
  and WAR files • 242
  core_english.properties • 213
  core_french.properties • 213
Configuring
  Configuring - environment for application servers • 241, 247, 252
core_english.properties Configuration File
  about • 213
  location • 213
  using • 134, 213
core_french.properties Configuration File
  about • 213
  location • 213
  using • 134
core_queries.xml Customization File
  and search filters • 83
  creating custom queries file • 121
  query syntax • 82, 85
CSS Applications
  about • 20
  deploying • 240
Custom Classes

about • 101
compiling • 208
creating • 103
creating approval processes • 204, 205, 206
declaring • 104
defining package • 204
deploying • 104
developing • 103
extending core class • 204
extending the BLM with • 101
Customizing
Approval Processes • 203
Business Logic • 101
CID • 26
logger • 222, 224

D

Data Access Layer (DAL)
accessing external data sources • 116, 117
creating new search queries • 82, 83, 85
declaring new instance • 117
declaring search methods • 77, 80, 81
query syntax • 82, 85
Data Sources
configuring for Oracle Application Server • 252
configuring for WebLogic • 247
configuring for WebSphere • 241
external data sources • 116
Deploying
about • 240
Oracle 9i Application Server • 252
using WAR files • 240
WebLogic • 247
WebSphere • 241

E

Errors
BLM exceptions • 212, 213
localizing • 213
Event Codes
about • 219
creating custom event codes • 222, 223
example of custom event codes • 222
using • 224
Explicit Security
about • 113
using • 113, 114
external_custom.xml Customization File
about • 101
location • 101
using • 103, 104

F

Filter Criteria
about • 72
and search filters • 47
customizing • 68
Form Handler JSPs
and shopping carts • 162
functions_routing.properties Configuration File • 120
about • 120
location • 120
using • 120

H

Help
technical support • xii

I

instances.properties Customization File
about • 117
creating • 118
location • 117
using • 117

J

jfnApplication.properties Configuration File
using • 128
JSPF
character encoding • 128
search pages • 46, 86
jsp_parameters.properties Configuration File
about • 131
location • 131
using • 131

L

Languages

about • 126, 127, 130, 131
declaring languages • 130
localizing • 137
localizing the BLM • 134, 212, 213
localizing the CID • 133
specifying formats • 131
specifying the character set • 128
specifying the default language • 130, 131
Localizing
about • 126, 127
BLM error messages • 134
character sets • 128
CID data • 133
formats • 131
JSPs • 136, 137, 138
languages • 130
limits • 127
strings • 137, 138
log4j.properties Configuration File
and WebSphere • 245
Logger
and the logger API • 219, 221
configuring events to log • 217, 220
customizing • 222, 224
modules • 217
severity levels • 217
using in custom code • 216
Logic Handler JSPs
search pages • 95
Logs
components • 216, 217
contents of • 216
custom event codes • 220, 222, 223
custom log messages • 227, 228, 229, 230, 231
event types • 216, 217, 219
severity levels • 217

N

Notification Logic Class
creating new class • 204

O

Oracle Application Server
accessing deployed applications • 258
creating WAR file for • 242
deploying WAR file • 255
post deployment configuration • 245, 256, 257

P

Parameters
and search filters • 77
Persistent Action Managers
about • 187
and shopping cart templates • 187
and Shopping Carts • 187
in the CID • 187
Personalization Manager
customizing • 22
policy.properties Configuration File
using • 146, 148

R

Reference Data
reloading • 145, 146, 147, 148, 149

S

Search Filters
about • 47
activating criteria • 67
and filter criteria • 48
available filters • 49
creating • 49, 73
declaring as default • 67
declaring criteria • 76, 77, 88
declaring criteria as mandatory • 68
declaring in the CID • 75, 76
declaring search methods • 77, 80
default value • 68
display order • 68
Security
about • 108
and CID users • 109
explicit security • 113
Shopping Cart Templates
about • 187, 188
and persistent action managers • 187
listing • 189
removing • 190
saving • 188
using • 189
Shopping Carts

about • 158
adding contents • 163, 167, 168
and Action Managers • 160, 161
and Persistent Action Managers • 187
complex shopping cart • 162, 163, 165, 166, 167, 168, 170, 171
displaying contents • 163, 180, 181, 182
limits in MyWeb channel • 161
modifying contents • 174, 176, 177, 178
multiple ActionManagers • 162, 165, 166
removing contents from • 178
retrieving saved contents • 186
saving content of • 184, 185, 186
simple shopping cart • 162, 163
submitting • 163
templates • 188

T

translator.properties Configuration File
about • 134
location • 134
using • 134

U

User Events
creating • 235
examples • 236, 237

W

WAR File
about • 240, 242
deploying • 240, 242, 245, 250, 255
generating • 242, 244
installing • 245, 250, 255
WebLogic
accessing deployed application • 251
configuring • 247
creating WAR file for • 242, 243, 244
data source • 247
deploying WAR file • 250
recommended deployment • 247
WebSphere
accessing deployed application • 246
configuring • 242, 243, 245
creating WAR file for • 242, 243, 244
data source • 241
deploying WAR file • 245
memory settings • 242
post deployment configuration • 245