



# **Developing User Interfaces with Telco Service Manager**

---

© 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 .....	vii
If You Need Help .....	ix

# Using this Manual

Welcome to Building User Interfaces.

This manual covers building user interfaces of account management applications built using 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 that 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 interest in reading about personalization data, 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 User Interfaces**

This chapter covers the basics of building User Interfaces for Account Management solutions.

It contains information about:

- User interfaces and Account Management solutions
- Developing user interfaces

- **Understanding the JSPF**

This chapter covers the JavaServer Page Framework (JSPF).

It contains information about:

- The JSPF and its components
- Properties and configuration files
- JSPF classes
- Framework JSPs

- **Working with Personalization Manager Channels**

This chapter covers the Personalization Manager.

It contains information about:

- How the Personalization Manager is organized
- Personalization Manager Channels
- Customizing the channels

## What Typographical Changes and Symbols Mean

This manual uses the following conventions:

TYPEFACE	MEANING	EXAMPLE
<i>Italics</i>	Manuals, topics or other important items	Refer to <i>Developing Connectors</i> .
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 <code>//home/my file</code>

# 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 that 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 the *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](http://support.edocs.com) (This requires a one-time online registration)
- E-mail: [support@edocs.com](mailto:support@edocs.com)

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

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

If the system crashed or hung, please tell us.



# Contents

<b>Preface</b>	<b>iii</b>
----------------	------------

<b>Overview of Developing User Interfaces</b>	<b>13</b>
---	-----------

About User Interfaces and the Personalization Manager	14
About the Personalization Manager Channels	15
MyWeb Channel	15
MyWAP Channel	20
MyIVR Channel	21
About Developing User Interfaces	23

<b>Understanding the JSPF</b>	<b>25</b>
-------------------------------	-----------

Overview of the JSPF	26
Features of the JSPF	26
JSPF Components	27
About the JSPF Properties file	28
About the JSPF Configuration File	30
Application Properties Settings	30
JSP Declarations and Properties	33
Application Menu Settings	36
Specifying Workflows	41
Application Properties Settings	45
About the JSPF Classes	48
About the Framework JSPs	49
Application Framework JSPs	50
Graphical Chart	52
Form Handlers	54
Logic Handlers	56

<b>Working with Personalization Manager Channels</b>	<b>57</b>
--	-----------

About Working with Channels	58
About the Sample Data	59
Contents of the Sample Data	59
Creating the CID Database	61
About the Contents of a Channel	63
Channel JSPs	63
Channel Menus	64
Channel Functional Steps	65
Adding a New Channel	66
Declaring the Channel Media	66
Example of Creating Support of a New Channel	66
Adding New Personalization Data	68
Adding Parameters to the CID	69
Associating a Parameter with a Parameter Group	71

Example of Adding A New Parameter	72
Modifying Menus	76
Example of Modifying Menus	78
Creating a Workflow	80
Understanding Workflows	80
Using Functional Steps	81
Using Display Pages	83
Using Form Handlers	87
Using Logic Handlers	89
Passing Data From One Page to Another	93
Sample Documented Workflow	95
Working with New JSPF JSPs	100
Writing a JSP	100
Modifying the JSPs	101
Examples of a JSPF JSP	102
Reloading <code>JSPF</code> Configuration Information	104

<b>Index</b>	<b>105</b>
--------------	------------

---

## CHAPTER 1

# Overview of Developing User Interfaces

### In This Section

About User Interfaces and the Personalization Manager .....	14
About the Personalization Manager Channels .....	15
About Developing User Interfaces .....	23

# About User Interfaces and the Personalization Manager

The Personalization Manager is the component that handles the user interface of TSM. This is the entry point for users of your application and is considered as the presentation layer of the application.

Because users access information with different devices, the Personalization Manager is built around channels. In general, a channel corresponds to one of the different types of devices users can use to access the application.

For each channel, the Personalization Manager has a corresponding set of JSPs. This way the Personalization Manager can tailor information and features for each type of device because they each have their own technical and practical constraints. For instance, the amount of information displayed by mobile Internet devices is different from what users see on the Web. Not only that, but the Internet is based on HTML whereas the mobile Internet devices use another language called WML. These sets of JSPs are referred to as application channels or channels for short.

No matter which channel you use for the user interface of your application, there are several basic application functions and features that are standard to all applications. The Personalization Manager comes with a JSP framework called the JSPF. This framework handles these basic functions and creates the foundation of any Personalization Manager-based Account Manager Application.

# About the Personalization Manager Channels

TSM comes with the following channels:

- **MyWeb**  
This set of JSPs builds the user interface for users using the Internet.
- **MyWAP**  
This set of JSPs builds the user interface for users using their WAP-enabled mobile hand-held devices.
- **MyIVR**  
This set of JSPs builds the user interface for users using voiceXML voice-recognition technology.

All of these channels use the JSPF for common application features.

---

In order for you channels to run correctly, you need to have information in the CID that the application can access and manage. The CID comes with sample data that you use while working with channels. The sample data are different from the reference data. Sample data cover users, their contracts, commercial offers, rate plans and so on.

For more information about the sample data, refer to *About the Sample Data* in this manual.

---

## MyWeb Channel

### Contents of the Channel

This channel includes:

- `MyWeb.xml` configuration file
- 
- Channel JSPs
- Set of web site graphics and files

## Location of the Channel

By default, the channel files are in the following directories:

- Files:
  - <home\_dir>/channels/WEB-INF/classes/nmycfg/jfn for the MyWeb.xml file
  - <home\_dir>/channels/common/fwkw for the JSPF files
  - <home\_dir>/channels/common/form\_handler for the form handler JSPs
- jfnApplication.properties configuration file
  - <home\_dir>/channels/WEB-INF/classes/nmycfg/jfn
- Channel JSPs and Configuration File
  - <home\_dir>/channels/MyWeb for the Web channel JSPs
  - <home\_dir>/channels/MyWeb/Include for the different web site graphics and files

## MyWeb Channel Features

This channel includes the following features:

### Managing Users

- Logging in
- Logging out
- Creating logins
- Changing passwords
- Changing languages

### Acquiring and Activating Customer Accounts

- Creating contracts
- Migrating contracts
- Creating new organizations
- Adding levels to organizations
- Adding billing information to organizations
- Adding members to organizations
- Creating Personalization Data

### Enabling Self-service Account Management

- Modifying payment information



- Adding legal contacts
- Adding billing contacts
- Modify billing contacts
- Modify legal contacts
- Modifying contracts
- Searching for organizations
- Searching organization hierarchies
- Browsing organization hierarchies
- Changing rate plans
- Listing services
- Adding services
- Changing parameters of a services
- Replacing services
- Removing services
- Listing rate plans
- Grouping contract modifications
- Ordering with a shopping cart
- Ordering documentation
- Viewing usage information
- Listing prepaid packages
- Recharging prepaid contracts
- Approving orders
- Managing contracts

#### Reporting and Resolving Problems

- Creating trouble tickets
- Searching for trouble tickets
- Viewing trouble ticket details
- Modifying trouble tickets
- Declaring of loss or theft

#### Viewing Account Information Online

- Searching for invoices
- Viewing invoice details
- Searching payments

- Viewing payments
- Creating organization views
- Modifying organization views

#### System Features

- Displaying requests
- Auditing
- Generating user events
- Listing user events

## Graphical Charts

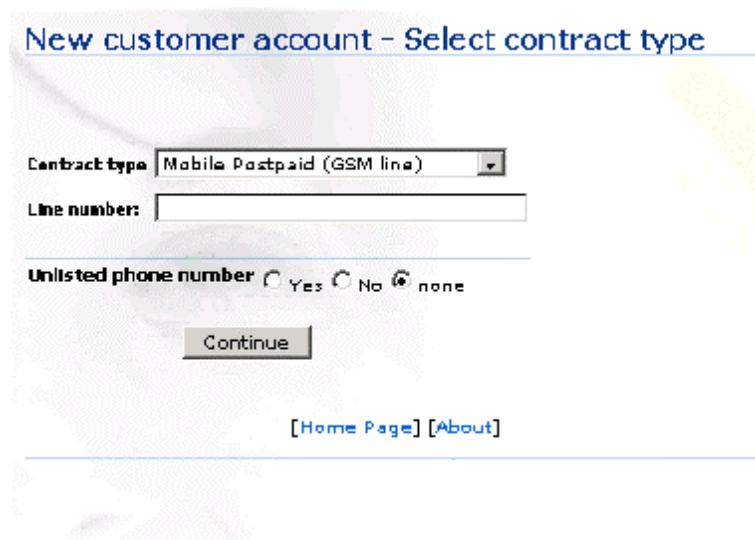
The MyWeb channel comes with the following basic graphical charts:

- Guest
- Normal
- Simple

### Guest Graphical Chart

When a user is a guest and signing up for an account, the application uses the Guest graphical chart.

An example of the Guest graphical chart:



The screenshot shows a web form titled "New customer account - Select contract type". The form includes a dropdown menu for "Contract type" with "Mobile Postpaid (GSM line)" selected, a text input field for "Line number:", and radio buttons for "Unlisted phone number" with options "Yes", "No", and "none" (selected). A "Continue" button is at the bottom. At the very bottom, there are links for "[Home Page]" and "[About]". The background of the form has a faint world map.

## Normal Graphical Chart

When a user has an account and logs in, MyWeb uses the Normal graphical chart.

An example of the Normal graphical chart:

The screenshot displays the 'Contracts' page for user John Dixon (ID: 0660100029). The page features a navigation bar at the top with links to Account, Contracts, Invoices, User Information, Problems, Requests, My Shopping Cart, and Home. Below the navigation bar, the user's name and ID are shown. The main content area is titled 'Contracts' and provides details for contract 0660100029, including its status (active), type (Mobile Postpaid), line type (GSM Line), and rate plan (Rate Plan 8h + 8h weekend). A table lists the included features, such as Telephony, Call waiting, Caller ID, Short Messages, Call forwarding, and VoiceMail, along with their respective categories, signup fees, and monthly costs. The page also includes a 'You can:' section with links to subscribe new services, order a product, change the rate plan, migrate a contract, view usage information, and view status history. A 'Go to top' link is located at the bottom right.

Service	Category	Signup Fee	Monthly Cost	Options
Telephony	Telephony services	EUR 0.0	EUR 30.0	-
Call waiting	Telephony services	EUR 0.0	EUR 0.0	-
Caller ID	Telephony services	EUR 0.0	EUR 0.0	-
Short Messages	Telephony services	EUR 0.0	EUR 0.0	-
Call forwarding	Telephony services	EUR 0.0	EUR 0.0	-
VoiceMail	Telephony services	EUR 0.0	EUR 0.0	-
General Service	Value services	added	EUR 0.0	remove modify
Mms	Value services	added	EUR 0.0	EUR 10.0 remove modify

**You can:**  
[Subscribe new service](#)  
[Order a product](#)  
[Change the rate plan](#)  
[Migrate contract](#)  
[View the requests for this contract](#)  
[View usage information](#)  
[View status history](#)

[Go to top](#)

## Simple Graphical Chart

When specified, MyWeb can use a Simple graphical chart.

An example of the simple graphical chart:

---

[\[Account\]](#)[\[Contracts\]](#)[\[Invoices\]](#)[\[User Information\]](#)[\[Problems\]](#)[\[Requests\]](#)[\[My Shopping Cart\]](#)[\[Home\]](#)

- Manage contracts - Sign up for a new contract -

---

John DixonLogout

**Details for contract D660100029**

This contract is active  
The contract type is Mobile Postpaid  
The line type is GSM line  
The rate plan is Rate Plan 3h + 3h weekend

**Included features**

Service	Category	Signup Fee	Monthly Cost	Options
Telephony	Telephony services	EUR 0.0	EUR 300.0	-
Call Waiting	Telephony services	EUR 0.0	EUR 0.0	-
Caller ID	Telephony services	EUR 0.0	EUR 0.0	-
Short Messages	Telephony services	EUR 0.0	EUR 0.0	-
Call forwarding	Telephony services	EUR 0.0	EUR 0.0	-
VoiceMail	Telephony services	EUR 0.0	EUR 0.0	-
Generic Service	Value added services	EUR 0.0	EUR 65.0	remove   modify
News	Value added services	EUR 0.0	EUR 139.0	remove   modify

**You can:**  
Subscribe new service  
Order a product  
Change the rate plan  
Migrate contract  
View the requests for this contract  
View usage information  
View status history

---

## MyWAP Channel

### Contents of the Channel

This web channel includes:

- MyMobile.xml configuration file  
The IVR and WAP channels use this file.
- 
- Channel JSPs

## Location of the Channel

By default, the channel files are in the following directories:

- **Files:**
  - `<home_dir>/channels/WEB-INF/classes/nmycfg/jfn` for the `MyMobile.xml` file
  - `<home_dir>/channels/common/fw` for the JSPF files
  - `<home_dir>/channels/common/form_handler` for the form handler JSPs
- `jfnApplication.properties` configuration file
- `<home_dir>/channels/WEB-INF/classes/nmycfg/jfn`
- **Channel JSPs and Configuration File**
  - `<home_dir>/channels/MyWap` for the WAP channel JSPs

## MyWAP Channel Features

### WAP Channel Features

This channel includes the following features:

- Logging in
- Logging out
- Viewing contract details
- Adding service
- Changing parameters of a service
- Changing rate plans
- Viewing usage information
- Recharging prepaid contracts
- Viewing pending requests

## MyIVR Channel

### Contents of the Channel

This channel includes:

- `MyMobile.xml` configuration file
- The IVR and WAP channels use this file.
- - Channel JSPs

## Location of the Channel

By default, the channel files are in the following directories:

- **Files:**
  - `<home_dir>/channels/WEB-INF/classes/nmycfg/jfn` for the `MyMobile.xml` file
  - `<home_dir>/channels/common/fw` for the JSPF files
  - `<home_dir>/channels/common/form_handler` for the form handler JSPs
- `jfnApplication.properties` configuration file
  - `<home_dir>/channels/WEB-INF/classes/nmycfg/jfn`
- **Channel JSPs and Configuration File**
  - `<home_dir>/channels/MyIvr` for the IVR channel JSPs

## MyIVR Channel Features

This channel includes the following features:

- Logging in
- Logging out
- Getting contract details
- Adding service
- Changing parameters of a service
- Changing rate plans
- Viewing usage information
- Recharging prepaid contracts
- Viewing pending requests

# About Developing User Interfaces

Before you start developing your user interface, you need to become familiar with the JSPF. The JSPF uses JSPs to build an application framework that provides seamless integration of the application JSPs and the CSS Engine. The JSPF provides a number of key features:

- Easy customization of workflow, language, available features, and so on.
- Session management, form-handling and exception handling
- Separation of application code and user presentation, enabling parallel outsourcing of the operational and image aspects of site design

By using the JSPF as the foundation of your applications, you can separate the presentation logic from basic application tasks. The JSPF also reduces development time because you do not have to rewrite all of these basic tasks for the Personalization Manager JSPs.

Developing user interfaces involves:

- Becoming familiar with the JSPF
- Working with the Channels:
  - Adding a new channel
  - Creating new personalization data
  - Modifying menus
  - Creating workflows
  - Writing a new JSP





## CHAPTER 2

# Understanding the JSPF

### In This Section

Overview of the .....	26
About the JSPF Properties file.....	28
About the Configuration File .....	30
About the JSPF Classes .....	48
About the Framework JSPs .....	49

## Overview of the JSPF

One of the obvious drawbacks to developing JSPs is the mix of Java code and HTML. Not only does this represent a problem when trying to read and code JSPs, it also has an impact on performance and maintenance. Large-scale, robust applications can have a very large number of JSPs and this can lead to a lot of duplicated code. Each page has to carry out user authentication, error checking, and so on. When users connect to TSM, any JSP must carry out a number of tasks before it can begin working with the BLM. For example, users must be authenticated, their roles and scopes must be checked, and so on.

The Java Server Page Framework (JSPF) is a set of JSP pages and Java classes and configuration files that contain the code that is required by the sets of JSPs in the Personalization Manager. You can speed up your development of JSPs by using the framework to handle basic tasks and concentrate on coding the access to the BLM. The JSPF can be considered the foundation of your channels.

## Features of the JSPF

This open and customizable JSPF contains all the code you need to quickly and easily do the following:

FUNCTION	DESCRIPTION
Authentication	Checks at the start of the JSP if the user has been authenticated according to the JSPF configuration file.
Form handling	Calls the function that processes the form associated with the JSP, and validates the information entered against the 'regular expressions', as defined in the JSPF configuration file.
Logic handling	Calls to the BLM for processing are isolated in special JSP. By using these JSP, you can easily separate the logic processing and the presentation logic of your application.
Localization	Displays a character string in a given language.
Exception handling	When an error occurs, intercepts the error and displays a message using the JSP defined in the JSPF configuration file.
Role verification	Checks that the role of the user matches the role of the JSP (ex: CUSTADMIN or SUBSCRIBER).
Menus	For a given JSP, displays and manages the menu items.
Basic workflow management	Calls functional steps, this workflow management tool helps make the framework modular and customizable. As the user navigates through the pages, this function keeps track of the information stored as a variable and passes information between JSPs that determine the displayed contents..

## JSPF Components

The JSPF contains a set of JSPs and other files needed to configure and use the JSPF.

The JSPF components include:

- Configuration files:
  - `jfnApplication.properties` configuration file.  
This file sets the application media and location of the JSPF configuration file.
  - JSPF configuration file.  
This XML file manages the JSPF JSPs in the application.
- JSPF JSPs
  - Application Framework JSPs.  
These files are the foundation of the JSPF. They are located in:
    - `<home_dir>/channels/common/fwk.`
  - form handler JSPs  
These files contain the code that read the data to be displayed.
  - logic handler JSPs  
These files contain generic methods that channel JSPs use to interact with the BLM.
- JSPF Classes  
The JSPF uses these classes to read the configuration files, program the features of the framework, and manage other application tasks.

By default, the JSPF components are installed in the following directories:

- `<home_dir>/channels/WEB-INF/classes/nmycfg/jfn` contains the JSPF application configuration files
- `<home_dir>channels/common/` contains the shared JSPF JSPs
- `<home_dir>/channels/WEB-INF/lib/nmyjfn.jar` archive contains the JSPF classes.

---

The JSPF JSPs are JSP 1.0. Your JSPs must be based on the Java Server Pages Specification 1.0 or higher

---

## About the JSPF Properties file

The JSPF uses the `jfnApplication.properties` configuration file to set the location of the JSPF configuration file and which channel to use for the application. This file is located in `<home_dir>/channels/WEB-INF/classes/nmycfg/jfn`.

This file has the following sections:

- Character encoding settings
- Default channel settings
- Additional channel settings

For each channel, this file specifies:

- The location of the JSPF configuration file
- The associated URL path.

When this path is found in the URL requested by the browser, this media is selected.

The location of the JSPF configuration file has the syntax:

```
media.<channel_name>=<JSPF configuration file path>
```

This setting can be one of the following:

- Absolute path by entering the full path of the JSPF configuration file
- Classpath by using the following format:

```
media.<channel_name>=res:<classpath>
```

The associated URL path of the channel has the syntax:

```
media.<channel_name>.path1=<URL path of the application>
```

**EXAMPLE OF JSPFAPPLICATION.PROPERTIES**

```
# List of media installed on this application server
# default => JSP pages. It will be the default to be used when the user-agent is not recognized

# wap => Wap Media jsp files
# ivr => VoiceXML jsp files

media.default=<home_dir>/channels/WEB-INF/classes/nmycfg/jfn/MyWeb.xml
media.default.path1=/MyWeb

media.ivr=<home_dir>/channels/WEB-INF/classes/nmycfg/jfn/MyMobile.xml
media.ivr.path1=/MyIvr

media.wap=<home_dir>/channels/WEB-INF/classes/nmycfg/jfn/MyMobile.xml
media.wap.path1=/MyWap
```

## About the JSPF Configuration File

TSMs that use the JSPF have a JSPF configuration file. This XML file contains 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`.

Each channel media may have its own JSPF configuration file. You specify the location and name of the JSPF configuration file in the `jfnApplication.properties` configuration file. This file is located in `<home_dir>/channels/WEB-INF/classes/nmycfg/jfn`.

The JSPF uses the configuration file to set the following:

- Application properties
- JSP Pages and the following properties:
  - Security rules
  - Localization strings
  - Forms and input constraints
- Menu items
- Workflow information

---

For detailed information about the JSPF configuration file, refer to the *JSPF Configuration File Reference Documentation*.

---

## Application Properties Settings

The `<config>` section of the JSPF configuration file contains information on the following:

- Application settings
- JSPF settings
- Search result settings
- Layout information of complex parameters

APPLICATION SETTINGS	
APP_SKIN_URL	Location of graphics, style sheets, and other interface elements
APP_TITLE_NAME	Name of the application
APP_LANG_CODE	Default application language

JSPF SETTINGS	
FILE_CONFIGURATION	Name of the JSPF configuration file.
FILE_LOGIN	Name of the JSP to which the user is redirected if an authentication check fails
FILE_LOGOUT	Name of the JSP to which the user is redirected after logout.
FILE_MESSAGE	Name of the file to display messages when: <ul style="list-style-type: none"> <li>the role checking fails</li> <li>a session expires</li> </ul>
FILE_LOGIC_ERROR	Name of the file to display BLM business logic messages
FILE_INTERNAL_ERROR	Name of the file to display BLM internal error messages
JSP_BIND_HTTPSESSION	True or False Saves the contents of the shopping cart when the session times out and log an event.
GETURL_CALL_ENCODE_URL	True or False Encodes the URL when the application server is in URL rewriting mode to manage sessions without cookies. Used when calling the URL using JSPFjspHelper.getURL.
URL_ROOT	The URL root for generated URLs. Some app servers (for example IAS 6.0) use /NASApp/MyWeb in URLs but pass only /MyWeb to the java code In this case, you must enter "NASApp" in the property below. The entry will be added to the URL generated in redirect calls.

SEARCH SETTINGS	
MAX_ELEMENTS_SEARCH_INVOICE	Maximum number of elements returned by the database when a search invoice is executed
MAX_ELEMENTS_SEARCH_PAYMENT	Maximum number of elements returned by the database when a search payment is executed
MAX_ELEMENTS_SEARCH_TICKET	Maximum number of elements returned by the database when a search ticket is executed
MAX_ELEMENTS_SEARCH_REQUEST	Maximum number of elements returned by the database when a search request is executed

LAYOUT SETTINGS	
PARAM_ITEM	Switch between list of multiple choices and multi-value choice
PARAM_LINES_IN_LIST	Number of items to display in a multi-value list
PARAM_LINES_IN_TEXTAREA	Number of lines in a text area
PARAM_CHARS	Number of chars to switch between textfield and textarea
PARAM_LIST_ITEMS	Max Number of items to be displayed horizontally, if superior display them vertically (for a list of items)

LAYOUT SETTINGS	
PARAM_COMPOSITE_ITEMS	Max Numbers of items to be displayed horizontally, if superior display them vertically (for a composite item)
PARAM_SIZE_STRING	Size of the field for an input text string
PARAM_SIZE_INTEGER	Size of the field for an input text of integer
PARAM_SIZE_DATE	Size of the field for an input text of date
PARAM_SIZE_TIME	Size of the field for an input text of time
PARAM_SIZE_DATETIME	Size of the field for an input text of datetime
PARAM_SIZE_FLOAT	Size of the field for an input text of decimal

EXAMPLE OF APPLICATION SETTINGS	
Application settings	<pre>&lt;config&gt;   &lt;property name="APP_SKIN_URL"&gt;/MyWeb/include/&lt;/property&gt;   &lt;property name="APP_LANG_CODE"&gt;en&lt;/property&gt;</pre>
JSPF settings	<pre>&lt;property name="FILE_CONFIGURATION"&gt;MyWeb.xml&lt;/property&gt; &lt;property name="FILE_LOGIN"&gt;login.jsp&lt;/property&gt; &lt;property name="FILE_MESSAGE"&gt;message.jsp&lt;/property&gt; &lt;property name="FILE_LOGIC_ERROR"&gt;logic_error.jsp&lt;/property&gt; &lt;property name="FILE_INTERNAL_ERROR"&gt;internal_error.jsp&lt;/property&gt; &lt;property name="JSP_BIND_HTTPSESSION"&gt;true&lt;/property&gt; property name="GETURL_CALL_ENCODE_URL"&gt;&gt;false&lt;/property&gt;</pre>
Search Settings	<pre>&lt;property name="MAX_ELEMENTS_SEARCH_INVOICE"&gt;7&lt;/property&gt; &lt;property name="MAX_ELEMENTS_SEARCH_PAYMENT"&gt;7&lt;/property&gt; &lt;property name="MAX_ELEMENTS_SEARCH_TICKET"&gt;7&lt;/property&gt; &lt;property name="MAX_ELEMENTS_SEARCH_REQUEST"&gt;50&lt;/property&gt;</pre>
Layout settings	<pre>&lt;property name="PARAM_ITEM"&gt;10&lt;/property&gt; &lt;property name="PARAM_LINES_IN_LIST"&gt;10&lt;/property&gt; &lt;property name="PARAM_LINES_IN_TEXTAREA"&gt;10&lt;/property&gt; &lt;property name="PARAM_CHARS"&gt;64&lt;/property&gt; &lt;property name="PARAM_LIST_ITEMS"&gt;4&lt;/property&gt; &lt;property name="PARAM_COMPOSITE_ITEMS"&gt;4&lt;/property&gt; &lt;property name="PARAM_SIZE_STRING"&gt;20&lt;/property&gt; &lt;property name="PARAM_SIZE_INTEGER"&gt;10&lt;/property&gt; &lt;property name="PARAM_SIZE_DATE"&gt;10&lt;/property&gt; &lt;property name="PARAM_SIZE_TIME"&gt;8&lt;/property&gt; &lt;property name="PARAM_SIZE_DATETIME"&gt;20&lt;/property&gt; &lt;property name="PARAM_SIZE_FLOAT"&gt;14&lt;/property&gt;  &lt;/config&gt;</pre>



## JSP Declarations and Properties

Each JSP of an application must have an entry in the JSPF configuration file. This entry contains information about the following:

- Name of the JSP, Authentication, and access rules
- Forms and input constraints
- String localization

### JSP Information and Access Rules

The `<jsp>` element contains information on the JSP and access rules. The syntax is:

```
<jsp name="name.jsp" authenticate="true/false"  
roles="ROLE1;ROLE2">
```

It contains:

- One or more `<form>` elements
- One or more `<string>` elements

<JSP> ELEMENT ATTRIBUTES	
ATTRIBUTES	DESCRIPTION
name	The name of the JSP (used to locate the configuration of a JSP within the configuration file)
authenticate	True if the user must be authenticated to access this JSP, otherwise False.
role	<p>The code of the user roles allowed to access this JSP. A sequence of ";" separated names of the roles allowed to enter the JSP (such as "CUSTADMIN;SUBSCRIBER"). String, optional.</p> <p>Do not leave any spaces between the role names and the separator.</p>

## Form Information and Input Constraints

The `<form>` element contains information on the forms of the JSP and access rules. The syntax is:

```
<form name="form_name">
```

It contains one or more `<input>` elements.

---

The declaration of forms in the JSPF configuration file is optional. For instance, if your presentation logic controls the format of the user input, you do not need to declare it again in this file.

---

<FORM> ELEMENT	
ATTRIBUTES	DESCRIPTION
name	The name of the form. This name is used when calling the <code>checkForm</code> function of the JSPF API

The `<input>` element contains the regular expression that the application uses to check the information submitted by the form.

<INPUT> ELEMENT	
ATTRIBUTES	DESCRIPTION
name	The name of the entry field. Must correspond to the value of the attribute name of an element <code>&lt;input&gt;</code> , <code>&lt;select&gt;</code> , or <code>&lt;textarea&gt;</code> in the HTML form.
regexp	The regular expression. The user input must match the regular expression in order to be valid.

## String Localization

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_to_localize">  
  <value name="language_code">String to use</value>
```

```
</string>
```

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

You declare the default language in the JSPF configuration file.

<STRING> ELEMENT	
ATTRIBUTES	DESCRIPTION
name	The name of the character string

<VALUE> ELEMENT	
ATTRIBUTES	DESCRIPTION
name	The language code of the string. This code corresponds to the APP_LANG_CODE property declared in the JSPF configuration file and the code entered in the CID.

## Example of a JSP Declaration

This example shows the entry of a login JSP in the JSPF configuration file. This JSP has the following characteristics:

- User does not need to be authenticated to access this page and no roles are required
- The JSP contains the login form
- The entry fields have entry constraints
- Three strings in English and French

EXAMPLE OF A JSP DECLARATION	
JSP information and Access Rules	<jsp name="login.jsp" authenticate="false">
Form and input constraints	<form name="login"> <input name="login" regexp=".*"/> <input name="password" regexp=".*"/> </form>

**EXAMPLE OF A JSP DECLARATION**

String localization

```
<string name="login_user_name_field">
  <value name="en">User name</value>
  <value name="fr">Identifiant</value>
</string>
<string name="login_password_field">
  <value name="en">Password</value>
  <value name="fr">Mot de passe</value>
</string>
<string name="login">
  <value name="en">Login</value>
  <value name="fr">Valider</value>
</string>
</jsp>
```

## Application Menu Settings

The `<menu>` section of the JSPF configuration file contains information on the following:

- Menu names and use
- Menu item localization
- Menu structure

The menus used by the application also depend on the workflow of your application. The functional step feature of the JSPF manages workflow between pages.

In menus, the functional step specified the functional step entry in the JSPF configuration file. This functional step entry contains the target URL of the menu item when activated.

## Menu Declaration

The `<menu>` element contains information about the users who have access to the menu along with workflow information. The syntax is:

```
<menu name="menu_name" funct="functionalstep" orgtypes=""
roles="">
```

It contains one or more `<string>` elements

<MENU> ELEMENT	
ATTRIBUTES	DESCRIPTION
name	The name of the Menu element
funct	The functional step associated with the link
orgtypes	The type of organization that can see the menu
roles	Allowed user roles that can see the menu

## Menu String Localization

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

```
<string name="label">
    <value name="language_code">String to use</value>
</string>
```

It contains one or more `<value>` elements

<STRING> ELEMENT	
ATTRIBUTES	DESCRIPTION
name	The string name must be "label" for menu items

<VALUE> ELEMENT	
ATTRIBUTES	DESCRIPTION
name	The language code of the string. This code corresponds to one of the language codes entered in the CID.

## Menu Structure

Menus in applications have different levels. For example, you might have three menus with several sub menus in each menu. And all of these sub menus contain menu items users can activate.

To create this kind of structure for your menus, you use the XML tree structure of the JSPF configuration file. You create a menu entry that is the top level entry and then create XML menu nodes to create the physical structure of your menu.

In TSM, the context of the workflow along with the logged user's role and organization type determine which menus are displayed. In the , the `framework_start.jsp` is responsible for determining which menus to display.

## Example of a Menu Declaration

This example shows the entry of a menu that looks like this:

```
My Account
    My Contracts
    Sign up for a New Contract
My Bills
    View my Bill
```

<b>MENU EXAMPLE</b>
---------------------

**MENU EXAMPLE**

```
<menu name="menu_example" funct="" orgtypes="" roles="">
    <string name="label">
        <value name="en">Menu Example</value>
        <value name="fr">Exemple</value>
    </string>
    <menu name="account_example" funct="" orgtypes="" roles="">
        <string name="label">
            <value name="en">My Account</value>
            <value name="fr">Mon compte</value>
        </string>
        <menu name="VIEW_CONTRACT" funct="" orgtypes="" roles="">
            <string name="label">
                <value name="en">My Contracts</value>
                <value name="fr">Mes contracts</value>
            </string>
        </menu>
        <menu name="ADD" funct="" orgtypes="" roles="">
            <string name="label">
                <value name="en">Sign up for a New Contract</value>
                <value name="fr">Souscrire un nouveau contrat</value>
            </string>
        </menu>
    </menu>
    <menu name="bill_example" funct="" orgtypes="" roles="">
        <string name="label">
            <value name="en">My Bills</value>
            <value name="fr">Mes factures</value>
        </string>
        <menu name="VIEW_BILL" funct="" orgtypes="" roles="">
            <string name="label">
                <value name="en">View My Bill</value>
                <value name="fr">Voir ma facture</value>
            </string>
        </menu>
    </menu>
</menu>
```



## Specifying Workflows

Each JSP of an application has links to other JSPs. A group of links that carry out a specific task is referred to as a workflow. The JSPF functional step feature helps you create and manage workflow between your pages and manages not only the link from one JSP to another, but also lets you determine different workflows for different users.

Instead of hard-coding the target of your links, you link to a functional step. This functional step contains all the information the JSP page needs in order to continue the workflow. Not only does it contain the link, it can contain information on the associated form or logic handler. A logic handler is a stand-alone JSP that uses the BLM to modify information. A form handler is JSP included in a display page to retrieve data for display. Functional steps can also redirect users to other pages depending on how you program the JSP. You can even use the functional steps to override the localization strings in the target JSP.

By using this feature, you can create standard pages that carry out basic functions. You use the functional steps to customize the page. For instance, you can create a standard JSP that searches for contracts in the CID and use information in the functional steps to determine what the JSP can do in a specific workflow.

The `<functionalstep>` section of the JSPF configuration file contains information on the following:

- Name of the workflow
- The target URL
- The form or logic handler to use
- Functional step redirects if required
- Localization for the functional step

## Declaring the Functional Step

The `<functionalstep>` element contains information on the name, target, and forms. The syntax is:

```
<functionalstep name="functionalstep_name" url="name.jsp"
form="form_handler">
```

It contains:

- One or more `<redirect>` elements
- One or more `<string>` elements

The name of the functional step is the name you use in creating menus or links in your application.

<b>&lt;FUNCTIONALSTEP&gt; ELEMENT</b>	
Attributes	Description
name	The name of the functional step element
url	The URL of the target JSP
form or logic	The name of the form handler or logic handler called before the JSP.

## Declaring the Functional Step Redirects

The `<redirect>` element contains the name of the link as it is coded in the JSP. It contains information on the name in the JSP and the functional step containing the target JSP. The syntax is:

```
<redirect originalname="link_name" funct="functional_step"/>
```

<b>&lt;REDIRECT&gt; ELEMENT</b>	
ATTRIBUTES	DESCRIPTION
originalname	The name of the link as coded in the JSP
funct	The name of the functional step to call

## Declaring Functional Step String Localization

The `<string>` element in functional steps overrides the string localizations declared in the target JSP 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="label">

    <value name="language_code">String to use</value>

</string>
```

It contains one or more `<value>` elements.

<STRING> ELEMENT	
ATTRIBUTES	DESCRIPTION
name	The string name must be "label" for menu items

<VALUE> ELEMENT	
ATTRIBUTES	DESCRIPTION
name	The language code of the string. This code corresponds to one of the codes entered in the CID.

## Example of a Functional Step

This example shows a functional step in the JSPF configuration file. This functional step has the following characteristics:

- Functional step is for customer users adding a service
- Uses the generic `service_modify.jsp`
- The JSP contains two links `MODIFY_SERVICE` and `CANCEL`
- Overrides the default localization of the form button string

EXAMPLE OF A FUNCTIONAL STEP DECLARATION	
Functional step name and attributes	<pre>&lt;functionalstep name="CUSTOMER.CONTRACT.ADD.SERVICE" url="service_modify.jsp" form="getServiceForParameters"&gt;</pre>

EXAMPLE OF A FUNCTIONAL STEP DECLARATION	
Redirects	<pre>&lt;redirect originalname="MODIFY_SERVICE" funct="I_SUBSCRIBER.CONTRACT.ADD.SERVICE_QUANTITY"/&gt;  &lt;redirect originalname="CANCEL" funct="CUSTOMER.CONTRACT.HOME"/&gt;</pre>
String localization overrides	<pre>&lt;string name="service_modify_button"&gt;     &lt;value name="en"&gt;Add This Service&lt;/value&gt;     &lt;value name="fr"&gt;Ajouter ce service&lt;/value&gt; &lt;/string&gt;  &lt;/functionalstep&gt;</pre>

## Workflows for Guest Users

Guest users have special workflows in order to limit their access to different features. Guest users have the following roles:

- `GUEST_FIRSTLOGIN`

These guests are users who begin the sign up process and have minimum access to the objects they need to create a login.

- `GUEST_NEWCUSTOMER`

These guests are users who can create a new organization and associated login, contracts, levels, and so on. They do not have access to any other existing objects.

In order to restrict their access to features, the workflows have specific functional steps for these roles. The name of the functional steps begin with the guest login name.

For example, `guest_newcustomer.NEW.SERVICE` is the functional step for the user having the `GUEST_NEWCUSTOMER` role.

---

If a user having a guest role tries to access any other functional steps, an authorization error occurs.

---

## Application Properties Settings

The <config> section of the JSPF configuration file contains information on the following:

- Application settings
- settings
- Search result settings
- Layout information of complex parameters

APPLICATION SETTINGS	
APP_SKIN_URL	Location of graphics, style sheets, and other interface elements
APP_TITLE_NAME	Name of the application
APP_LANG_CODE	Default application language

SETTINGS	
FILE_CONFIGURATION	Name of the JSPF configuration file.
FILE_LOGIN	Name of the JSP to which the user is redirected if an authentication check fails
FILE_LOGOUT	Name of the JSP to which the user is redirected after logout.
FILE_MESSAGE	Name of the file to display messages when: <ul style="list-style-type: none"> <li>• the role checking fails</li> <li>• a session expires</li> </ul>
FILE_LOGIC_ERROR	Name of the file to display BLM business logic messages
FILE_INTERNAL_ERROR	Name of the file to display BLM internal error messages
JSP_BIND_HTTPSESSION	True or False  Saves the contents of the shopping cart when the session times out and log an event.
GETURL_CALL_ENCODE_URL	True or False  Encodes the URL when the application server is in URL rewriting mode to manage sessions without cookies. Used when calling the URL using <code>jspHelper.getURL</code> .
URL_ROOT	The URL root for generated URLs.  Some app servers (for example IAS 6.0) use <code>/NASApp/MyWeb</code> in URLs but pass only <code>/MyWeb</code> to the java code  In this case, you must enter "NASApp" in the property below. The entry will be added to the URL generated in redirect calls.

SEARCH SETTINGS	
MAX_ELEMENTS_SEARCH_INVOICE	Maximum number of elements returned by the database when a search invoice is executed
MAX_ELEMENTS_SEARCH_PAYMENT	Maximum number of elements returned by the database when a search payment is executed
MAX_ELEMENTS_SEARCH_TICKET	Maximum number of elements returned by the database when a search ticket is executed
MAX_ELEMENTS_SEARCH_REQUEST	Maximum number of elements returned by the database when a search request is executed

LAYOUT SETTINGS	
PARAM_ITEM	Switch between list of multiple choices and multi-value choice
PARAM_LINES_IN_LIST	Number of items to display in a multi-value list
PARAM_LINES_IN_TEXTAREA	Number of lines in a text area
PARAM_CHARS	Number of chars to switch between textfield and textarea
PARAM_LIST_ITEMS	Max Number of items to be displayed horizontally, if superior display them vertically (for a list of items)
PARAM_COMPOSITE_ITEMS	Max Numbers of items to be displayed horizontally, if superior display them vertically (for a composite item)
PARAM_SIZE_STRING	Size of the field for an input text string
PARAM_SIZE_INTEGER	Size of the field for an input text of integer
PARAM_SIZE_DATE	Size of the field for an input text of date
PARAM_SIZE_TIME	Size of the field for an input text of time
PARAM_SIZE_DATETIME	Size of the field for an input text of datetime
PARAM_SIZE_FLOAT	Size of the field for an input text of decimal

EXAMPLE OF APPLICATION SETTINGS	
Application settings	<pre>&lt;config&gt;   &lt;property name="APP_SKIN_URL"&gt;/MyWeb/include/&lt;/property&gt;   &lt;property name="APP_LANG_CODE"&gt;en&lt;/property&gt;</pre>
settings	<pre>&lt;property name="FILE_CONFIGURATION"&gt;MyWeb.xml&lt;/property&gt; &lt;property name="FILE_LOGIN"&gt;login.jsp&lt;/property&gt; &lt;property name="FILE_MESSAGE"&gt;message.jsp&lt;/property&gt; &lt;property name="FILE_LOGIC_ERROR"&gt;logic_error.jsp&lt;/property&gt; &lt;property name="FILE_INTERNAL_ERROR"&gt;internal_error.jsp&lt;/property&gt; &lt;property name="JSP_BIND_HTTPSESSION"&gt;true&lt;/property&gt; property name="GETURL_CALL_ENCODE_URL"&gt;false&lt;/property&gt;</pre>
Search Settings	<pre>&lt;property name="MAX_ELEMENTS_SEARCH_INVOICE"&gt;7&lt;/property&gt; &lt;property name="MAX_ELEMENTS_SEARCH_PAYMENT"&gt;7&lt;/property&gt; &lt;property name="MAX_ELEMENTS_SEARCH_TICKET"&gt;7&lt;/property&gt; &lt;property name="MAX_ELEMENTS_SEARCH_REQUEST"&gt;50&lt;/property&gt;</pre>

EXAMPLE OF APPLICATION SETTINGS	
Layout settings	<pre>&lt;property name="PARAM_ITEM"&gt;10&lt;/property&gt; &lt;property name="PARAM_LINES_IN_LIST"&gt;10&lt;/property&gt; &lt;property name="PARAM_LINES_IN_TEXTAREA"&gt;10&lt;/property&gt; &lt;property name="PARAM_CHARS"&gt;64&lt;/property&gt; &lt;property name="PARAM_LIST_ITEMS"&gt;4&lt;/property&gt; &lt;property name="PARAM_COMPOSITE_ITEMS"&gt;4&lt;/property&gt; &lt;property name="PARAM_SIZE_STRING"&gt;20&lt;/property&gt; &lt;property name="PARAM_SIZE_INTEGER"&gt;10&lt;/property&gt; &lt;property name="PARAM_SIZE_DATE"&gt;10&lt;/property&gt; &lt;property name="PARAM_SIZE_TIME"&gt;8&lt;/property&gt; &lt;property name="PARAM_SIZE_DATETIME"&gt;20&lt;/property&gt; &lt;property name="PARAM_SIZE_FLOAT"&gt;14&lt;/property&gt;  &lt;/config&gt;</pre>

## About the JSPF Classes

To help you program and build applications based on the JSPF, you use the extensive set of Java classes. These classes help you manage any JSPF-based TSM.

The `com.netonomy.jfn` package manages the JSPF configuration file and your application.

This package is in the `<home_dir>/channels/WEB-INF/lib/nmyjfn.jar` archive.

---

For more information about the classes and the methods in this package, refer to the *JSPF API Reference Documentation*.

---



## About the Framework JSPs

The Personalization Manager channels use a set of framework files to create its JSP pages. These framework files are divided into the following categories:

- **Framework** They manage the layout and look and feel of the application. By default, the following graphical charts are provided:
  - `guest` for the guest graphical chart
  - `rich` for the rich graphical chart
  - `simple` for the simple graphical chart
- **Form handler** They manage the extraction of information for display
- **Logic handler** They manage the business logic and interaction between the application and the CSS Engine. They also handle workflows depending on the results of processing business logic.

The Personalization Manager has its JSP files in the directories corresponding to the different channels. By default, they are installed in the following directories:

- Framework JSPs are in `<home_dir>/channels/common/fw` and the graphical chart subdirectories
- Form handler JSPs are in `<home_dir>/channels/common/form_handler`
- Logic handler JSPs are in `<home_dir>/channels/<channel_name>`

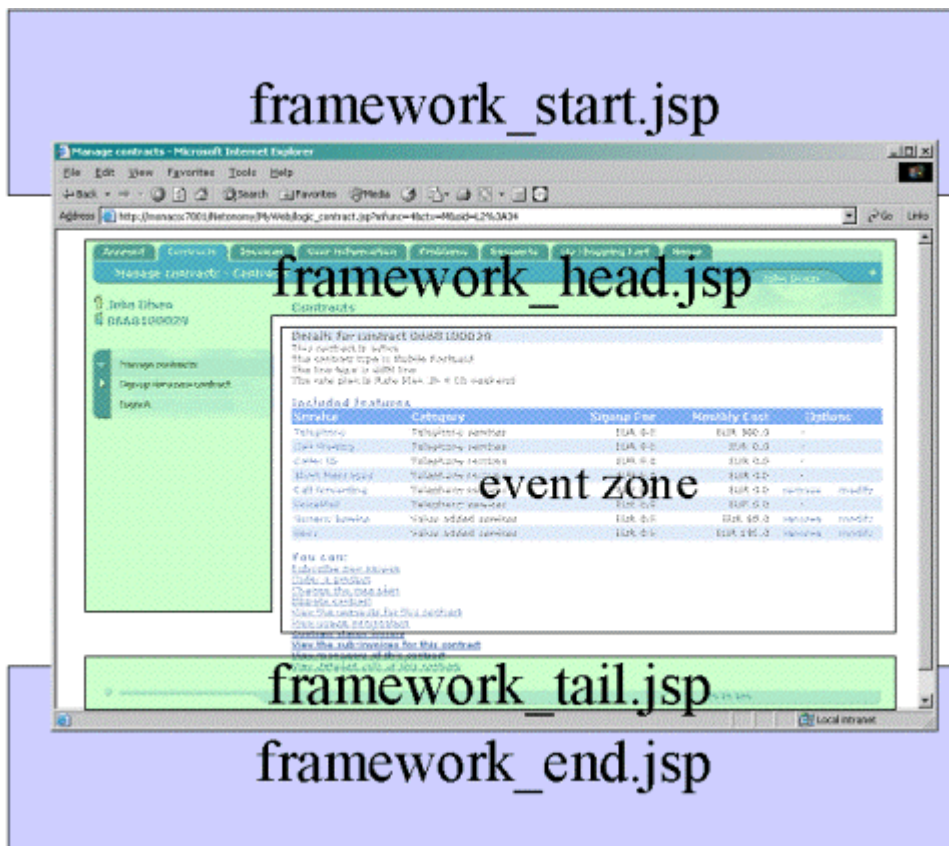
## Application Framework JSPs

These JSPs contain the code to carry out most of the basic functions of the framework so you can concentrate on programming the access to information in the CID. The Personalization Manager JSPs that make up the channels (MyWeb, MyWAP and MyIVR) also use the framework files.

The Personalization Manager Channel JSPs are installed in the following directories:

- `<home_dir>/channels/myweb` for the Web channel
- `<home_dir>/channels/mywap` for the WAP channel
- `<home_dir>/channels/myivr` for the IVR channel

This is a diagram of the different JSPs and how they relate to a MyWeb page.



## framework\_start.jsp and framework\_end.jsp

This JSP is the first part of the main framework code for JSPs.

You must declare a string `strJSP` variable before you include this code in your own JSP, and use this variable to store the JSP file name (the value of the `name` attribute of the `<jsp>` element in the configuration file).

You **must** include `framework_end.jsp` at the end of your JSP. Refer to `login.jsp` as an example.

This JSP carries out the following functions:

- Declares the variables for the JSP
- Gets the session object
- Gets the JSP configuration
- Checks if the session is new. If yes, redirects the user to the login page if they do not come from this page
- Retrieves the BLM session and adds it to the session manager
- Checks authentication and redirects user to the login page if it fails
- Checks role and redirects to a message if this fails
- Creates a repository object for errors
- Creates a repository object for results
- Calls the form or logic handler declared in the functional step
- Finds the name of the menu to display

## framework\_head.jsp and framework\_tail.jsp

These JSPs start and end the static HTML code of the application page.

The root `framework_start` JSP contains the test of the graphical chart to use and the `include` statement of the corresponding `framework_start`.

These files are located in subdirectories corresponding to the graphical chart.

The `framework_head` contains simple html tags. You can add a banner ad rotator here.

The `framework_tail` contains the closing tags of the html.

## event zone

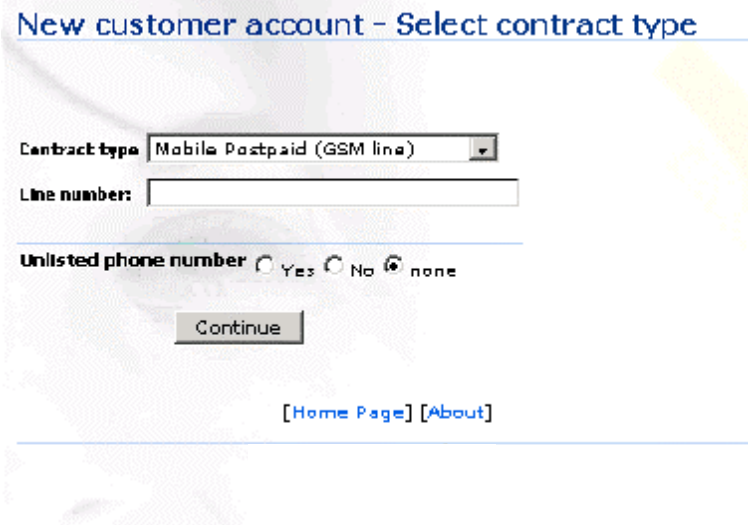
This is your JSP that contains the display of information or the application's context. The heart of the system, you can easily write JSP to interact with the framework files and the BLM to build TSMs. These JSPs are referred to as display pages.

## Graphical Chart

The following images are examples of the different default graphical charts used in the MyWeb TSM.

- Guest Graphical chart

This is the default for guest users.



The screenshot shows a web form titled "New customer account - Select contract type". The form includes a dropdown menu for "Contract type" with "Mobile Postpaid (GSM line)" selected, a text input field for "Line number:", and radio buttons for "Unlisted phone number" with options "Yes", "No", and "none" (selected). A "Continue" button is at the bottom. At the very bottom, there are links for "[Home Page]" and "[About]".

- Simple Graphical Chart

This is the default for Business Subscriber, residential subscriber and supplier users.

---

[\[Account\]](#)
[\[Contracts\]](#)
[\[Invoices\]](#)
[\[User Information\]](#)
[\[Problems\]](#)
[\[Requests\]](#)
[\[My Shopping Cart\]](#)
[\[Home\]](#)

-Manage contracts-- Sign up for a new contract -

---

John Dixon Logout

**Details for contract D660100029**

This contract is active  
 The contract type is Mobile Postpaid  
 The line type is GSM line  
 The rate plan is Rate Plan 3h + 3h weekend

**Included features**

Service	Category	Signup Fee	Monthly Cost	Options
Telephony	Telephony services	EUR 0.0	EUR 300.0	-
Call Waiting	Telephony services	EUR 0.0	EUR 0.0	-
Caller ID	Telephony services	EUR 0.0	EUR 0.0	-
Short Messages	Telephony services	EUR 0.0	EUR 0.0	-
Call forwarding	Telephony services	EUR 0.0	EUR 0.0	-
VoiceMail	Telephony services	EUR 0.0	EUR 0.0	-
Generic Service	Value added services	EUR 0.0	EUR 65.0	remove modify
News	Value added services	EUR 0.0	EUR 133.0	remove modify

**You can:**  
[Subscribe new service](#)  
[Order a product](#)  
[Change the rate plan](#)  
[Migrate contract](#)  
[View the requests for this contract](#)  
[View usage information](#)  
[View status history](#)

---

- Rich Graphical Chart

This is the default for Business Contract administrator, Business Customer Administrators, Telco users and dealer users.

**Account**   **Contracts**   **Invoices**   **User Information**   **Problems**   **Requests**   **My Shopping Cart**   **Home**

**Manage contracts - Contracts**   John Dixon

**John Dixon**  
0660100029

Manage contracts  
Sign up for a new contract  
Logout

**Contracts**

**Details for contract 0660100029**  
This contract is active  
The contract type is Mobile Postpaid  
The line type is GSM line  
The rate plan is Rate Plan 3h + 3h weekend

**Included features**

Service	Category	Signup Fee	Monthly Cost	Options
Telephony	Telephony services	EUR 0.0	EUR 300.0	-
Call Waiting	Telephony services	EUR 0.0	EUR 0.0	-
Caller ID	Telephony services	EUR 0.0	EUR 0.0	-
Short Messages	Telephony services	EUR 0.0	EUR 0.0	-
Call forwarding	Telephony services	EUR 0.0	EUR 0.0	-
VoiceMail	Telephony services	EUR 0.0	EUR 0.0	-
Generic Service	Value services	added EUR 0.0	EUR 65.0	remove modify
News	Value services	added EUR 0.0	EUR 133.0	remove modify

**You can:**  
[Subscribe new service](#)  
[Order a product](#)  
[Change the rate plan](#)  
[Migrate contract](#)  
[View the requests for this contract](#)  
[View usage information](#)  
[View status history](#)

Go to top

## Form Handlers

A form handler is a JSP dedicated for preprocessing information to be sent to a JSP for display.

A form handler is simply a method declared in the JSP. The name begins with `formHandler_`.

The form handler contains a set of preprogrammed functions you use to:

- Speed up development

This code can be used by all the JSPs that make up your application. And as it is shared code, you can easily modify the behavior of your application without having to change several JSP.

- Isolate display logic from display processing

This code handle extracting information from the CID and preprocessing it. You then send the results to the display page.

For example, the `form_handler/getContract.jsp` includes the following form handlers:

- `formHandler_bulkAddContractsToManagerReport`
- `formHandler_bulkRemoveContractsFromManagerReport`
- `formHandler_getContract`
- `formHandler_getContractAndServices`
- `formHandler_getContracts`
- `formHandler_getContractsToRemoveFromManager`
- `formHandler_getContractsToRemoveFromOrgView`
- `formHandler_getNewContract`
- `formHandler_prepareUnassignContractConfirmation`

These formhandlers are executed by the just before the page start to display.

---

You can create your own form handlers.

---

## Logic Handlers

logic handler is a JSP dedicated to workflow and business rules processing. A logic handler processes and calls the next logic handler to execute or redirect to a page.

When a logic handler calls a page, you can also call a form handler.

logic handlers are functions grouped in a file which act as a library located in the channel directory. The logic handler JSP names begin with `logic_` to make them easy to identify and program. The same applies to the methods found inside the JSPs.

For example, in `logic_member.jsp` you will find the following methods:

- `logic_childCreateMemberForNext`
- `logic_searchMembers`

These logic handlers are executed by the JSPF at the beginning of the page before display. These pages do not display anything, they handle the logic. Once they finish processing, they call a page via a JSP server redirect or can call another process if needed.

---

You can create your own logic handlers.

---



## CHAPTER 3

# Working with Personalization Manager Channels

### In This Section

About Working with Channels.....	58
About the Sample Data.....	59
About the Contents of a Channel.....	63
Adding a New Channel .....	66
Adding New Personalization Data .....	68
Modifying Menus.....	76
Creating a Workflow .....	80
Working with New JSPs.....	100
Reloading Configuration Information .....	104

## About Working with Channels

The default channels are working applications that cover most of the functions of TSM.

Instead of building your application from scratch, you can use TSM channels as an application template. You do not have to re-code your application, you can just adapt one of the channels to meet your needs.

Working with Personalization Manager Channels involves:

- Adding a new channel
- Creating new personalization data
- Modifying menus
- Creating workflows
- Writing a new JSP

When developing, you can use the JSPF configuration file dynamically without having to restart your Application server.

---

This section is based on examples to help you understand how channels work.

Remember, you must install the sample data to use the channels that come with the Presentation Manager.

---

## About the Sample Data

In order for you channels to run correctly, you need to have information in the CID that the application can access and manage. The CID comes with sample data that you use while working with channels. Sample data include reference data (countries, and so on), a product catalog, customer/dealer/customer service representatives (CSR) data and user data.

The comprehensive sample data help you build and test your application quickly and easily as you do not need to create test data.

When installing the CID for development or demonstrations, you can use the `cidAdminTool` to:

- Install the CID with system data only  
When you install the CID with system data only, you can then populate it with your own sample data.
- Install the CID with system and sample data  
When installing the CID with both types of data, the CID is ready to be used for development or demonstrations.

## Contents of the Sample Data

### Reference Data

Along with the system data, the sample data includes reference data. In the CID, the reference data includes lists of languages, countries, and other general data.

### Product Catalog Data

Corresponding to a Communication Service Providers catalog, the sample data includes service data that includes services, commercial offers, rate plans and other data.

## User and Associated Customer/Dealer/CSR Information

The sample data includes a set of users and user types. This set of data includes the following logins:

The password is the same as the login.

---

When logging in, you must enter the login and password in lowercase.

---

LOGIN	ACT	NOTES	CHANNELS
joe	Consumer administrator	Has 1 contract	WEB WAP
tammy	Consumer administrator	Has more than 1 contract	WEB WAP
0660100032	Consumer administrator	Prepaid subscriber	WEB WAP IVR
0660100034	Consumer administrator		WEB WAP IVR
admadm	Business administrator	Administrator of Acme corporation	WEB
adm1, adm2, michel	Business administrator	Administrators of sub-levels of Acme corporation	WEB
contractadm	Contract administrator	Manages a set of Acme corporation contracts	WEB
bigboss	Business user	User at the top level of Acme corporation Has more than 1 contract	WEB
véro, jean, paul, victor	Business user	User at a sub-level of Acme corporation	WEB
jack	Business administrator	Administrator of Jack and Co	WEB
jim	Business user	User at the top level of Jack and Co	WEB
tpb	Dealer user	User at the top level of dealer organization	WEB
herve	Dealer user	User at a sub-level of dealer organization	WEB
eur	Telco user		WEB
will	Supplier user		WEB

LOGIN	ACT	NOTES	CHANNELS
acctmgr	Telco business account manager	Initially in charge of Acme corporation (list of managed business customers can be changed by senior_acctmgr)	WEB
acctmgr2	Telco business account manager	Initially in charge of Jack and Co (list of managed customers can be changed by senior_acctmgr)	WEB
senior_acctmgr	Telco business senior account manager	Manages all business accounts	WEB

## Creating the CID Database

To create the CID and populate it with sample data for the Personalization Manager channels, you use the cidAdmin tool.

---

When working with the demo CID Database, server components can only interact with the database using the <CID\_ADMIN> account.

---

### To create the CID with system and sample data

- 1 Go to <home\_dir>/bin.
- 2 Run the CID Administration tool. Use the syntax:
 

```
cidAdminTool create_demo_cid_test <CID> <CID_ADMIN login>
<CID_ADMIN password>
```

 where <CID>:
  - Oracle: <instance alias>
  - DB2: <database alias>
  - SQL Server: <database host> [:<port>] If no port is specified, the tool uses the default SQL server port

When finished, the CID Administration tool displays a message.

### To remove the CID

- 1 Go to <home\_dir>/bin.
- 2 Run the CID Administration tool. Use the syntax:

```
cidAdminTool drop_demo_cid_structure <CID> <CID_ADMIN login>  
<CID_ADMIN password>
```

where <CID>:

- Oracle: <instance alias>
- DB2: <database alias>
- SQL Server: <database host> [ :<port>] If no port is specified, the tool uses the default SQL server port

When finished, the CID Administration tool displays a message.

# About the Contents of a Channel

The components of a channel include:

- JSPF files
- Set of channel JSPs

By default, the components are in the following directories:

- JSPF files:
  - <home\_dir>/channels/common/fw for the files
  - <home\_dir>/channels/common/form\_handler for the form handler JSPs
- jfnApplication.properties configuration file
  - <home\_dir>/channels/WEB-INF/classes/nmycfg/jfn
- Channel JSPs
  - <home\_dir>/channels/MyWeb for the Web channel
  - <home\_dir>/channels/MyWap for the WAP channel
  - <home\_dir>/channels/MyIvr for the IVR channel

## Channel JSPs

Each individual JSP has a name that describes the feature it handles. For example, the `password_modify.jsp` is the page for changing user's passwords. And the `password_modify_done.jsp` displays a message after the user's password has been changed.

Each channel contains a set of different types of JSPs. The types of JSPs that make up a channel are:

- **Display** These JSPs contain the presentation logic of your application and are clearly named.
- **Handler** These JSPs contain sets of methods that interact directly with the BLM allowing you to separate your application Logic from the display logic. The types include:
  - **Form Handler JSPs**

These JSPs preprocess information for display.
  - **Logic Handler JSP**

These JSPs contain executable code used for creating or modifying information. The name of these files begin with `logic_`. For instance, `logic_service` contains a set of methods your display JSPs can use to manage contract services.

## Channel Menus

In the channel applications, JSPF menus change according to the role of the user that logs into the system. For instance, residential users and business users may have different menus because their needs are different. Certain features that are required for a specific set of users might not be needed by another set of users.

USER ROLE	FRAMEWORK MENU
Individual Subscriber	I_SUBSCRIBER
Business Subscriber	B_SUBSCRIBER
Business Administrator for Levels	B_CUSTADMIN_L
Business Administrator for Members	B_CUSTADMIN_M
Dealer	DEALER
Supplier	SUPPLIER
Business Contract Administrator	B_CONTADMIN_M
Telco	DEALER
Telco Account Manager	DEALER
Telco Senior Account Manager	DEALER

---

Menus are selected in the `framework_start.jsp`

---

## MyWeb Channel Menus

The following menus are available in the MyWeb channel:

- I\_SUBSCRIBER
- B\_SUBSCRIBER
- B\_CUSTADMIN\_L
- B\_CUSTADMIN\_M
- DEALER
- SUPPLIER
- B\_CONTADMIN\_M



## MyWAP Channel Menus

The following menus are available in the MyWAP channel:

- I\_SUBSCRIBER
- B\_SUBSCRIBER

## MyIVR Channel Menus

The following menus are available in the MyIVR channel:

- I\_SUBSCRIBER
- B\_SUBSCRIBER

## Channel Functional Steps

The functional steps of the channel applications depend on the user that logs into the application.

The functional Step workflows include workflows for:

- GLOBAL
- GUEST\_FIRSTLOGIN
- GUEST\_NEWCUSTOMER
- I\_SUBSCRIBER
- B\_SUBSCRIBER
- B\_CUSTADMIN\_L
- B\_CUSTADMIN\_M
- DEALER
- SUPPLIER

## Adding a New Channel

One of the ways you can modify the channels is to use the existing channels to create a new channel. A channel is associated with the channel media. The TSM has a Web, WAP and IVR channel. However, your solution may require other support for other media.

Adding a new media involves:

- Declaring the media in the `jfnapplication.properties` configuration file
- Create your TSM application

---

This section includes an example of creating a new channel for a Palm hand held personal digital assistant.

---

### Declaring the Channel Media

- 1 Go to `<home_dir>/channels/WEB-INF/classes/nmycfg/jfn.`
- 2 Open `jfnApplication.properties`.
- 3 Declare the new media. Use the syntax:

`media.<media_name>=full path of the media the JSPF Configuration File or the res:/ resource path`

`media.<media_name>.path1= string of the media to use that should appear in the request URL. This allows the to find the right JSPF configuration file to use for this media's client.`

Example:

`media.wap.path1= /MyWap`

`http://host/MyWap/login.jsp` is a Wap site URL.

- 4 Save your changes.

### Example of Creating Support of a New Channel

- 1 Go to `<home_dir>/channels/WEB-INF/classes/nmycfg/jfn.`
- 2 Open `jfnApplication.properties`.
- 3 Add the following:  
`media.palm=res:nmycfg/jfn/MyPalm.xml`  
`media.palm.path1=/MyPalm`
- 4 Copy the `/channels/MyWeb` folder and its contents and rename it `MyPalm`.

- 5 Edit `<home_dir>/nmycfg/jfn/MyPalm.xml` and change the following:  
`<property name="APP_SKIN_URL">/palm/include/</property>`
- 6 Add your specific workflows.

## Adding New Personalization Data

The CID is designed to easily add and extend the information it stores. To do this, you use parameters. One of the uses of these parameters is to create new personalization data.

You may want your application to use the new parameters in your application JSPs. For instance, you may want to add new contract options or allow users to change and save their preferred interface settings.

For information that commonly needs parameters, the CID is preconfigured to handle parameters. For example, the CID is ready for service, payment, and request search parameters.

---

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

---

Adding a new parameter to your JSP involves:

- Creating the parameter
- Associating the new parameter with a personalization data parameter group

---

This section includes an example of adding a parameter that allows users to set their preferred user interface menus.

---

## Adding Parameters to the CID

When you add a parameter, insert the information in the CID.

Adding a new parameter to the CID involves:

- Inserting the parameter in the `PARAMETER` table.
- Setting the possible values of the parameter
- Defining the association between the values and the parameter
- Set the default values

When the parameter is in the CID, you program your application to use the parameter.

### To insert a parameter

- 1 Use your database tool to connect to the CID.
- 2 In the `PARAMETER` table, add a record and enter the required information. Use the syntax:

```
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... );
```

- 3 If required, localize the parameter name and descriptions. Use the syntax:

```
insert into UNIVERSAL_STRING ( STRING_ID, LANG_ID, STRING_NAME
) values ( STRING_ID value, LANG_ID value, STRING_NAME value
);
```

- 4 Save your changes.

### To set the values of the parameter

- 1 Use your database tool to connect to the CID.
- 2 In the `CHOICE_ITEM` table, add a record and enter the required information. Use the syntax:

```
insert into CHOICE_ITEM ( CHOICE_ITEM_ID, STRING_ID,
CHOICE_TYPE, DISPLAY_VALUE, LEGACY_VALUE, CHOICE_PARAM_ID,
START_DATE, END_DATE ) values ( CHOICE_ITEM_ID value,
STRING_ID value,...);
```

- 3 If required, localize the value name and descriptions. Use the syntax:

```
insert into CHOICE_ITEM ( CHOICE_ITEM_ID, STRING_ID,
CHOICE_TYPE, DISPLAY_VALUE, LEGACY_VALUE, CHOICE_PARAM_ID,
START_DATE, END_DATE ) values ( CHOICE_ITEM_ID value,
STRING_ID value,... );
```

- 4 Save your changes.

### **To define the association between values and the parameter**

- 1 Use your database tool to connect to the CID.
- 2 In the `PARAMETER_CHOICE_ITEM_LINK` table, add a record specifying the relationship between the parameter and its possible values. Use the syntax:

```
insert into PARAMETER_CHOICE_ITEM_LINK ( CHOICE_ITEM_ID,
PARAM_ID, DISPLAY_ORDER ) values ( CHOICE_ITEM_ID value,
PARAM_ID value, DISPLAY_ORDER value );
```

- 3 Save your changes.

### **To set the default values**

- 1 Use your database tool to connect to the CID.
- 2 In the `DEFAULT_VALUE` table, add a record specifying the default values of the parameter. Use the syntax:
  - `insert into DEFAULT_VALUE ( DEFAULT_VALUE_ID, VALUE_TYPE, VALUE_STRING, VALUE_FLOAT, VALUE_INTEGER, VALUE_BOOLEAN, VALUE_DATETIME, CHOICE_ITEM_ID, PARAM_ID) values ( DEFAULT_VALUE_ID value, VALUE_TYPE value,...);`
- 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 localize the parameter and its values**

- 1 Use your database tool to connect to the CID.

- 2 For each language, enter a record in the `UNIVERSAL_STRING` table for:

- Parameter name
- Parameter description
- Parameter short description
- Value

Use the syntax:

```
insert into UNIVERSAL_STRING ( STRING_ID, LANG_ID, STRING_NAME
) values ( STRING_ID value, LANG_ID value, STRING_NAME value
);
```

- 3 Save your changes.

## Associating a Parameter with a Parameter Group

Once you enter a parameter in the CID, you associate it with a parameter group. A parameter group is a set of parameters that have something in common or that go together for programming reasons.

### To associate the parameter

- 1 Use your database tool to connect to the CID.
- 2 In the `PDATA_GROUP_PARAM_LINK` table, add a record associating the new parameter with a personalization data group. Use the syntax:

```
insert into PDATA_GROUP_PARAM_LINK ( PDATA_GROUP_ID, PARAM_ID,
DEFAULT_VALUE_ID, MANDATORY_FLAG, DISPLAY_ORDER ) values (
PDATA_GROUP_ID value, PARAM_ID value, DEFAULT_VALUE_ID value,
MANDATORY_FLAG value, DISPLAY_ORDER value );
```

- 3 Save your changes.

## Example of Adding A New Parameter

This example covers how you can add a new option for a user. This new option is for user to set their preferences of menu styles. They can choose from:

- Rich
- Simple

The MyWeb channel displays the choice on the User Information/Options page. This new option will be in French and English.

Adding the new option involves:

- Adding the parameter to the CID. You add the parameter and its values in the CID. Then you associate this parameter with the user display options and specify that it is the second setting and that it is a default setting.
- Program the channel to use the parameter. You modify the `logic_init` logic handler and the `framework_start` JSPs to get the default value when initializing the application. Then you code the `logic_perso_data` logic handler to display the new parameter.

### Adding a Parameter

- 1 Use your database tool to connect to the CID.
- 2 Declare the new parameter. The parameter has the following characteristics:
  - Name: Style
  - Type: Choice
  - Values: Rich, Light

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 ( 101050, 'LEG_STYLE',
'STYLE', 'C', 'Preferred style', 1028050, 'Graphical chart style', 1028350, 'Style', 1028650, 1, 1, '1',
'1', NULL);
```

- 3 Localize the parameter name, description and short description.

Example:

```
insert into UNIVERSAL_STRING ( STRING_ID, LANG_ID, STRING_NAME ) values ( 1028050, 25, 'Preferred style');
insert into UNIVERSAL_STRING ( STRING_ID, LANG_ID, STRING_NAME ) values ( 1028350, 25, 'Graphical chart
style');
insert into UNIVERSAL_STRING ( STRING_ID, LANG_ID, STRING_NAME ) values ( 1028650, 25, 'Style');
```

- 4 For each choice, enter the value of the parameter: Rich and Light.

Example:

```
insert into CHOICE_ITEM ( CHOICE_ITEM_ID, STRING_ID, CHOICE_TYPE, DISPLAY_VALUE, LEGACY_VALUE,
CHOICE_PARAM_ID, START_DATE, END_DATE ) values ( 101026, 1029026, 'C', 'Rich', '', NULL, NULL, NULL);
insert into CHOICE_ITEM ( CHOICE_ITEM_ID, STRING_ID, CHOICE_TYPE, DISPLAY_VALUE, LEGACY_VALUE,
CHOICE_PARAM_ID, START_DATE, END_DATE ) values ( 101027, 1029027, 'C', 'Light', 'txt=1', NULL, NULL,
NULL);
```



## 5 Localize the values.

Example:

```
insert into UNIVERSAL_STRING ( STRING_ID, LANG_ID, STRING_NAME ) values ( 1029026, 25, 'Rich');
insert into UNIVERSAL_STRING ( STRING_ID, LANG_ID, STRING_NAME ) values ( 1029027, 25, 'Simple');
insert into UNIVERSAL_STRING ( STRING_ID, LANG_ID, STRING_NAME ) values ( 1029026, 34, 'Riche');
insert into UNIVERSAL_STRING ( STRING_ID, LANG_ID, STRING_NAME ) values ( 1029027, 34, 'Simple');
```

## 6 Define 2 parameter choices attached to the parameter.

Example:

```
insert into PARAMETER_CHOICE_ITEM_LINK ( CHOICE_ITEM_ID, PARAM_ID, DISPLAY_ORDER ) values ( 101026,
101050, 0 );
insert into PARAMETER_CHOICE_ITEM_LINK ( CHOICE_ITEM_ID, PARAM_ID, DISPLAY_ORDER ) values ( 101027,
101050, 1 );
```

## 7 Set the default value to Rich.

Example:

```
insert into DEFAULT_VALUE ( DEFAULT_VALUE_ID, VALUE_TYPE, VALUE_STRING, VALUE_FLOAT, VALUE_INTEGER,
VALUE_BOOLEAN, VALUE_DATETIME, CHOICE_ITEM_ID, PARAM_ID) values (10300, 'S', Null, Null, Null, Null, Null,
101027, Null);
```

## 8 Attach the new parameter to the same parameter group (1) as the “Skin” parameter. You also put it as the second element. This parameter is mandatory.

Example:

```
insert into PDATA_GROUP_PARAM_LINK ( PDATA_GROUP_ID, PARAM_ID, DEFAULT_VALUE_ID, MANDATORY_FLAG,
DISPLAY_ORDER ) values ( 1, 101050, 10300, 1, 2 );
```

## 9 Save your changes.

## 10 Restart your application and go to the page. Your new parameter should be displayed.

# Modifying the JSP

Your application should take into account the default menu style when the user logs in. You need to code the processing that tests for the default style.

- 1 Edit `logic_init.jsp` and find the `SKIN` statement  
(`pdfif.getCode().equals("SKIN")`).
- 2 Add the following after the statement:

```
else if (pdfif.getCode().equals("STYLE"))
{
    test = false;
    for (int j=0;j<chItem.length;j++)
    {
        if (chItem[j].isFilled())
        {
            test = true;
            item = chItem[j].getChoiceItem();
            String strValue = item.getLegacyValue();
            if (strValue != null && strValue.equals("txt=1"))
            {
                session.putValue("txt","1"); //'set session value if not already set
            }
            break;
        }
    }
    if (test == false)
    {
        session.removeValue("txt");
    }
} //end else if
```

**3 Save your changes.**

**4 Edit framework\_start.jsp and add the following before the statement `if (tmptxt != null && tmptxt.equals("1") ) :`**

```
if( tmptxt == null )
{
    tmptxt = (String)session.getValue("txt");
}
```

**5 To take changes into account when the user changes the setting, add the following to `logic_perso_data.jsp` just after the if statement `if (pdfif.getCode().equals("SKIN")):`**

```
else if (pdif.getCode().equals("STYLE"))
{
    test = false;
    for (int j=0;j<chItem.length;j++)
    {
        if (chItem[j].isFilled())
        {
            test = true;
            item = chItem[j].getChoiceItem();
            String strValue = item.getLegacyValue();
            if (strValue != null && strValue.equals("txt=1"))
            {
                session.putValue("txt","1"); //'set session value if not already set
            }
            break;
        }
    }
    if (test == false)
    {
        session.removeValue("txt");
    }
} //end else if
```

## 6 Save your changes.

# Modifying Menus

The MyWeb channel comes with a set of menus that you can configure.

The menu and its structure are located in the JSPF configuration file. This file contains information on the:

- **Item text** The text displayed in the menu.
- **Item action** The action performed when a user clicks or otherwise activates a menu option.
- **Structure** The different menu layers are configured using the natural hierarchy of the JSPF configuration file's XML format.
- **Authorized users** Because it is declared in the JSPF configuration file, you can use the security rules to create different menus for different roles all in the same application.

You can:

- Add a menu item
- Modify a menu item
- Remove a menu item

---

This section includes an example of modifying the Search menu.

---

## To add a menu item

- 1 Go to `<home_dir>/channels/WEB-INF/classes/nmycfg/jfn`.
- 2 Edit `MyWeb.xml`.
- 3 Find the menu to modify.
- 4 Enter the menu item using the following syntax:

```
<menu name="MENU_NAME" funct="FUNCTIONAL.STEP" orgtypes="org_types" roles="authorized_roles">
  <string name="label">
    <value name="language_code">MenuItemText</value>
    <value name="language_code">MenuItemText</value>
    ...
  </string>
```

- 5 Save your changes.

## To modify a menu item

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

- 2 Open `MyWeb.xml`.
- 3 Find the menu to modify.
- 4 Find the menu item to modify. You can change the following menu item attributes:
  - `func` the destination Functional Step when the menu item is activated
  - `orgtypes` the organization types that use this menu
  - `roles` the roles allowed to use this menu
  - `label` the text to display
- 5 You can add, remove and modify the `<value>` elements. Use the syntax:
 

```
<value name="language_code">MenuItemText</value>
```
- 6 Save your changes.

### To remove a menu item

- 1 Go to `<home_dir>/channels/WEB-INF/classes/nmycfg/jfn`.
- 2 Open `MyWeb.xml`.
- 3 Find the menu to modify.
- 4 Find the menu item to remove.
- 5 Delete the menu item and all of its value elements. The example below shows the elements to remove to delete a menu item.

Menu name and static label	<pre>&lt;menu name="CONTRACT" func="I_SUBSCRIBER.CONTRACT.LIST" orgtypes="" roles=""&gt;   &lt;string name="label"&gt;     &lt;value name="en"&gt;Contracts&lt;/value&gt;     &lt;value name="fr"&gt;Contrats&lt;/value&gt;   &lt;/string&gt; &lt;/menu&gt;</pre>
Entire menu item you can select to remove	<pre>&lt;menu name="VIEW" func="I_SUBSCRIBER.CONTRACT.LIST" orgtypes="" roles=""&gt;   &lt;string name="label"&gt;     &lt;value name="en"&gt;Manage contracts&lt;/value&gt;     &lt;value name="fr"&gt;Gérer les contrats&lt;/value&gt;   &lt;/string&gt; &lt;/menu&gt;</pre>
Another menu item	<pre>&lt;menu name="ADD" func="I_SUBSCRIBER.CONTRACT.ADD" orgtypes="" roles=""&gt;   &lt;string name="label"&gt;     &lt;value name="en"&gt;Sign up for a new contract&lt;/value&gt;     &lt;value name="fr"&gt;Souscrire un nouveau contrat&lt;/value&gt;   &lt;/string&gt; &lt;/menu&gt; &lt;/menu&gt;</pre>

- 6 Save your changes.

### To modify the menu structure

- 1 Go to `<home_dir>/channels/WEB-INF/classes/nmycfg/jfn`.
- 2 Open `MyWeb.xml`.
- 3 Find the menu to modify.
- 4 Find the menu item to modify.
- 5 Do one of the following:
  - To move the item up one level, move the entire menu item and its elements up one node.
  - To move the item down one level, move the menu item and its elements down one node.
- 6 Save your changes.

## Example of Modifying Menus

This sample explains how to modify the search menu for a dealer. The example shows how to add a menu, rename items, and reorganize the menu structure.

### Adding a New Customer Menu

- 1 Go to `<home_dir>/channels/WEB-INF/classes/nmycfg/jfn`.
- 2 Open `MyWeb.xml`.
- 3 Find the `DEALER` menu.
- 4 Before the `CUSTOMER` menu item, enter the following:

```
<menu name="NEWCUST" funct="DEALER.CUSTOMER.NEW" orgtypes="" roles="">
  <string name="label">
    <value name="en">New Customer</value>
    <value name="fr">Nouveau Client</value>
  </string>
</menu>
```

- 5 Save your changes.

### Removing the New Customer menu item

- 1 Go to `<home_dir>/channels/WEB-INF/classes/nmycfg/jfn`.
- 2 Edit `MyWeb.xml`.
- 3 Find the `DEALER` menu.

#### 4 Remove the NEWCUST menu item:

```
<menu name="NEWCUST" funct="DEALER.CUSTOMER.NEW" orgtypes="" roles="">

    <string name="label">

        <value name="en">New customer</value>

        <value name="fr">Nouveau client</value>

    </string>

</menu>
```

Save your changes.

## Renaming the Customer menu

- 1 Go to `<home_dir>/channels/WEB-INF/classes/nmycfg/jfn`.
- 2 Edit `MyWeb.xml`.
- 3 Find the DEALER menu.
- 4 In the CUSTOMER menu, change the two `<value>` items to the following:  
`<value name="en">Search Customer</value>`  
`<value name="fr">Chercher un Client</value>`
- 5 Save your changes.

# Creating a Workflow

## Understanding Workflows

A workflow is a series of functional steps. A workflow usually begins with a menu and ends with a done or confirmation page.

For example, one of the MyWeb workflows is:

- 1 View legal contact page
- 2 Modify legal contact page
- 3 Confirm modification page
- 4 Modification done page

A workflow can have several paths, depending on several factors (user's role, options selected during the page flow, and so on.) Most of the time however, they all lead to the same end page.

You use the following components to make a workflow:

- Functional Step
- Display Page
- Form Handler
- Logic Handler



## Using Functional Steps

A Functional Step is a step in a workflow. A functional step tells your application where to go when certain conditions are met. To do this, the functional step calls a JSP.

The JSP can be a:

- Logic handler
- Display page

When calling the JSP, you can specify which to run first:

- A Form handler method called prior to executing the JSP display page
- A Logic handler method located in the JSP Logic handler

The name of a functional step is referred to as an ALIAS. This is because the functional step does not contain a URL directly linking the functional step to a JSP. The functional step alias is an element in the JSPF configuration file. This element contains information such as the JSP to call, access rules, localization and so on.

## Defining the Functional Steps

You must define the next possible steps of a functional step.

When you declare the next possible functional step, you define the alias using the `<redirect>` tag.

Example of a Functional Step:

```
<functionalstep name="I_SUBSCRIBER.CONTRACT.VIEW" url="contract_view.jsp" form="getContractAndServices">
  <redirect originalname="CHANGE_LINE_NUMBER" funct="DEALER.CONTRACT.CHANGE_LINE_NUMBER"/>
  <redirect originalname="UNASSIGN_CONTRACT" funct="B_CUSTADMIN_M.CONTRACT.UNASSIGN"/>
  <redirect originalname="SAVE_AS_TEMPLATE" funct="B_CUSTADMIN_M.CONTRACT.SAVE_TEMPLATE"/>
  <redirect originalname="CHANGE_CONTRACT_STATUS" funct="DEALER.CONTRACT.CHANGE_CONTRACT_STATUS"/>
  <redirect originalname="SERVICE_ADD" funct="I_SUBSCRIBER.CONTRACT.SERVICE.ADD"/>
  <redirect originalname="VIEW_SUBINVOICE" funct="I_SUBSCRIBER.CONTRACT.SUBINVOICE_SEARCH.LIST"/>
  <redirect originalname="BALANCE_VIEW" funct="I_SUBSCRIBER.CONTRACT.CREDIT_BALANCE.LIST_FIND"/>
  <redirect originalname="SERVICE_DETAIL" funct="I_SUBSCRIBER.CONTRACT.SERVICE.DETAIL"/>
  ...
</functionalstep>
```

## Using a Functional Step in a Menu

You declare the functional step using the `func` attribute of the menu element in the JSPF configuration file.

For example, to specify the functional step called by the `CONTRACT_VIEW` menu item, enter the following `func` attribute:

```
<menu name="CONTRACT.VIEW" func="I_SUBSCRIBER.CONTRACT.VIEW" orgtypes="" roles="">
```

When the user clicks the `CONTRACT_VIEW` menu item, the `func` attribute calls the functional step `I_SUBSCRIBER.CONTRACT.VIEW`.

## Overriding the String Tags

When a display page is used in several workflows, you can change the title of the page for the specific context. You use the `string` element to change the title within the functional step.

For example, you use a display page to display the available rate plans for an existing contract and to display rate plans when signing up for a new contract. The declarations in the JSPF configuration file:

```
// The default title in the Display JSP
<jsp name="rateplan_change.jsp" authenticate="true" roles="GUEST;SUBSCRIBER">
    <string name="title">
        <value name="en">Change rate plan</value>
        <value name="fr">Changer le forfait du contrat</value>
    </string>
</jsp>

...

// Overriding the default name when calling the JSP from a functional step
<functionalstep name="GUEST.CUSTOMER.NEW.RATEPLAN_SELECT" url="rateplan_change.jsp"
form="getRateplanTypeFromContractType">
    <string name="title">
        <value name="en">New Customer Account - Rate plan</value>
        <value name="fr">Nouveau compte - Forfait</value>
    </string>
</functionalstep>
```

## Using Display Pages

A display page only displays information or forms to get user input.

Display pages can generate HTML pages (MyWeb), WML pages (MyWap), VoiceXML pages (Mylvr), and so on.

You can also use a display page as a template to display the same object in different contexts.

For example, you can have a display page listing members. It can be used in the following pages:

- List members of a level
- Search members in an organization
- here the common part is the

## Declaring the Display Page

You have to declare the display page in the JSPF configuration file.

You declare:

- Name of the JSP, Authentication, and access rules
- Forms and input constraints
- String localization

For example:

```
<jsp name="login.jsp" authenticate="false" roles="">
    <form name="login">
        <input name="login" regexp=".*"/>
        <input name="password" regexp=".*"/>
    </form>
    <string name="title">
        <value name="en">Log in to Your Account</value>
        <value name="fr">Accédez à votre compte</value>
    </string>
    <string name="login_disabled">
        <value name="en">Your login has been disabled</value>
        <value name="fr">Votre login a été inactivé</value>
    </string>
</jsp>
```

## Writing a Display Page

You create the JSP file in the directory of your channel. For example, `<home_dir>/channels/MyWeb` for the MyWeb channel.

Your JSP must include the following:

- Declare JSPF variables
- Include the JSPF files
- Include the form handlers that manage the processing of data

Example of a display page:

```
<%-- declare JSP display page name --%>
<%! String strJSP = "JSPdisplaypagefilename";%>
<%-- include the framework jsp pages--%>
<%@ include file="../common/fw/framework_start.jsp" %>
<%@ include file="../common/fw/framework_head.jsp" %>

<%-- include file logic_helper.jsp if a form handler needs to use helper methods %>
<%@ include file="../common/logic_helper.jsp" %>
<%-- include all form handler pages needed %>
<%@ include file="../common/form_handler/getInfos.jsp" %>

<%-- write the HTML and java code to display information--%>

<%@ include file="../common/fw/framework_tail.jsp" %>
<%@ include file="../common/fw/framework_end.jsp" %>
```

## Defining Display Page Access Rules

You can require to be authenticated to access a display page. You use the following to restrict the access to a display page:

- `authenticate` attribute to determine if the user needs to be authenticated.
- `roles` attribute to set the authorized roles.

For example:

```
<jsp name="invoice_view.jsp" authenticate="true" roles="SUBSCRIBER;CUSTADMIN"/>
```

## Setting the Strings

The strings used by a display page are in the JSPF configuration file.

In the , the framework manages the title of your pages. The title is handled by `framework_head.jsp`. You do not need to generate the HTML code for the title in each display page.

When defining a display page, you define all of the labels and their localization the page uses to display information:

For example,

```
<jsp name="rateplan_change.jsp" authenticate="true" roles="GUEST;SUBSCRIBER">
    <string name="title">
        <value name="en">Change rate plan</value>
        <value name="fr">Changer le forfait du contrat</value>
    </string>
    ...
</jsp>
```

---

For more information about localizing and displaying strings in your application, refer to *Localizing Your Application in Developing Telco Service Manager*.

---

## Specifying Input Constraints

You can define a rule to specify the accepted values of each form field the user submits. You use a regular expression to validate the input.

For example:

```
<jsp name="logic_login.jsp" authenticate="false" roles="">
    <form name="login">
        <input name="login" regexp=".*"/>
        <input name="password" regexp=".*"/>
    </form>
```

---

You must also call

`jspHelper.getJSPConfiguration().checkForm(login, request)` in your form or logic handler to validate the input. For more information about this method, refer to the *JSPF Form and Logic Handler Reference*.

For more information on regular expressions, refer to the Jakarta-ORO Java class reference found at <http://jakarta.apache.org/>.

---

## Calling Functional Steps in a JSP

You declare the next functional step in your display page. You can call:

- A logic handler method
- A display page with a form handler method
- A display page

Use several methods to generate an URL the JSPF can use. You use the functional step's `ALIASNAME` to tell the JSPF which JSP is the next step of the workflow.

- To only generate an URL, you use:

```
jspHelper.getUrl("ALIASNAME")
```

For example, in a `<form>` tag in a display page.

- To generate an URL + the system parameters, you use:

```
jspHelper.encodeURLFunct("ALIASNAME", null, false)
```

- To generate an URL + the system parameters + new parameters, you use:

```
Hashtable hashTableOfParameters; // fill the hashtable with  
your parameters
```

```
jspHelper.encodeURLFunct("ALIASNAME", hashTableOfParameters, false)
```

You then test if a functional step is allowed for the current functional step:

```
jspHelper.isFunctionalStepValid ("ALIASNAME")
```

## Using Form Handlers

A JSP form handler is a JSP where is defined at least one form handler method. Others methods could be present within the JSP form handler for local treatment needed by the form handler method. You must use a form handler method to retrieve data to prepare objects for JSP display page. A form handler method is always called by the framework just before your code displaying the data.

A form handler must always be located in a JSP form handler and must be included in the display page that uses it.

No security access definition on JSP form handlers and all the channel form handlers are shared by all the channels.

The form handlers are located in

```
<install_dir>/channels/common/form_handler.
```

## Declaring a Form Handler

A form handler is a JSP that contains one or more form handler methods.

## Writing a Form Handler

To write your own form handler, you:

- Write the new JSP Form handler
- Import the required packages or classes

For example:

```
<%-- import all the needed java classes --%>

<%@ page import =
"com.netonomy.jfn.*,java.lang.reflect.*,java.util.*,java.net.*,com.netonomy.util.ComplexId,com.netonomy..."%>

<%-- declare each form handler method after this comment line--%>
```

---

No security access definitions are in JSP form handlers. Do not forget to include the JSP form handler in the display page in order to call the form handler method.

---

## Writing Form Handler Methods

You can write your own methods inside an existing JSP form handler or a new JSP form handler. When writing your own form handler method, you must:

User the JSPF naming convention `formHandler_<yourmethodname>`

Use the standard parameters:

- `HttpSession`
- `HttpServletRequest`
- `HttpServletResponse`
- `JFNJSPHelper`
- `Hashtable` for results of the method
- `Hashtable` for any errors that occur while processing the method

For example:

```
public void formHandler_Formhandlername (
    HttpSession session,
    HttpServletRequest request,
    HttpServletResponse response,
    JFNJSPHelper jspHelper,
    Hashtable results,
    Hashtable errors) throws Throwable
{
    //your processing code here
}
```

To prepare data for a JSP display page, you can:

- Fill the request attributes
- Set session parameters

## Calling a Form Handler Method

You can call a form handler and a form handler method in a functional step. You use the `url` attribute to specify the form handler and the `form` attribute to call the method.

For example:

```
<functionalstep name="GUEST.CUSTOMER.NEW.RATEPLAN_SELECT" url="rateplan_change.jsp"
form="getRateplanTypeFromContractType">

    <redirect originalname="CHANGE_RATEPLAN" funct="GUEST.CUSTOMER.NEW.CREATE_MEMBER"/>

    <redirect originalname="NEXT_STEP" funct="GUEST.CUSTOMER.NEW.CREATE_MEMBER"/>

    <string name="title">
        <value name="en">New Customer Account - Rate plan</value>
        <value name="fr">Nouveau compte - Forfait</value>
    </string>

</functionalstep>
```



When calling form handler methods using functional steps, you need to make sure the method is located in the form handler. In the example, the form handler method `getRateplanTypeFromContractType` is declared in the `rateplan_change.jsp` display page.

```
<%@ include file="../../../common/form_handler/getRateplan.jsp" %>
```

## Using Logic Handlers

You must use a logic handler method when you need to carry out processing, redirect the user depending on the result of processing or both.

When working with logic handlers, keep in mind:

- A logic handler method is similar to a servlet
- A logic handler method does not generate GUI elements
- A logic handler method is located in a logic handler JSP

## Declaring a Logic Handler

A logic handler is a JSP with one or more logic handler methods.

You can require to be authenticated to access a display page or logic handler. You use the following to restrict the access to a display page:

- `authenticate` attribute to determine if the user needs to be authenticated.
- `roles` attribute to set the authorized roles.

For example:

```
<jsp name="logic_dispatcher.jsp" authenticate="true" roles="SUBSCRIBER;CUSTADMIN"/>
```

## Writing a Logic Handler

To write your own logic handler, you:

- Write the new JSP Logic handler
- Import the required packages and classes
- Include the required JSPF files
  - `framework_start.jsp`
  - `framework_end.jsp`

For example:

```
<%-- import all the needed java classes --%>

<%@ page import =
"com.netonomy.jfn.*,java.lang.reflect.*,java.util.*,java.net.*,com.netonomy.util.ComplexId,com..." %>

<%-- declare JSP logic handler name --%>

<%! String strJSP = "JSPlogichandlerfilename";%>

<%-- include file logic_helper.jsp if you need to use helper method when writing a logic handler --%>
<%@ include file="../common/logic_helper.jsp" %>

<%-- declare each logic handler method after this comment line--%>

<%-- include these 2 files in order to use the JSP exception mechanism in case of java exception --%>
<%@ include file="../common/fwkw/framework_start.jsp" %>
<%@ include file="../common/fwkw/framework_end.jsp" %>
```

From a logic handler you can call:

- A logic handler method
- A JSP Display page with a form handler method
- A JSP Display page

## Writing Logic Handler Methods

You can write your own methods inside an existing logic handler or a new logic handler. When writing your own logic handler method, you must:

- Use the JSPF naming convention `logicHandler_<yourmethodname>`
- Use the standard parameters:
  - `HttpSession`
  - `HttpServletRequest`
  - `HttpServletResponse`
  - `JFNJSPHelper`
  - `Hashtable` for any errors that occur while processing the method

For example:

```
public void logic_Logichandlername (
    HttpSession session,
    HttpServletRequest request,
    HttpServletResponse response,
    JFNJspHelper jspHelper,
    Hashtable errors) throws Throwable
{
}
}
```

## Calling a Logic Handler Method

You can call a logic handler and a logic handler method in a functional step. You use the `url` attribute to specify the logic handler and the `logic` attribute to call the method.

For example:

```
<functionalstep name="GUEST.CUSTOMER.NEW.DO_LEGAL_CONTACT" url="logic_contact.jsp" logic="setLegalContact">  
    <redirect originalname="ON_OK" funct="GUEST.CUSTOMER.NEW.BILLING_CONTACT_CHANGE"/>  
</functionalstep>
```

When calling logic handler methods using functional steps, you need to make sure the method is located in the logic handler. In the example, the logic handler method `setLegalContact` is declared in the JSP logic handler `logic_contact.jsp`.

## Calling the Next Functional Step

To call the next functional step you want to go use the corresponding alias:

```
request.getRequestDispatcher(jspHelper.getUrlAndMoveToPage  
("ALIASNAME")).forward(request, response);
```

Test if a functional step is allowed for the current functional step:

```
jspHelper.isFunctionalStepValid ("ALIASNAME")
```

For example:

```
public void logic_setRoles ( HttpSession      session,  
                           HttpServletRequest request,  
                           HttpServletResponse response,  
                           JFNJspHelper      jspHelper,  
                           Hashtable          errors  
                           ) throws Throwable  
{  
    UserF member = jspHelper.getUser ();  
  
    jspHelper.doNotSend ("roles");  
    /* Gets all the roles selected  
    ObjectId[] ids   = ObjectId.instantiate (request.getParameterValues("roles"));  
  
    if (ids==null)  
    {  
        Hashtable FW_urlParameters = new Hashtable ();  
  
        FW_urlParameters.put ("title", jspHelper.localize("form_entry_error_title","default"));  
        FW_urlParameters.put ("section", jspHelper.localize("form_entry_error_section","default"));  
        FW_urlParameters.put ("message", jspHelper.localize("select_a_role","default"));  
        response.sendRedirect (jspHelper.encodeURLFunct ("GLOBAL.MESSAGE", FW_urlParameters, true));  
        jspHelper.exitJSP();  
    }  
    RoleIF[] roles = new RoleIF[ids.length];  
  
    for (int i=0;i<ids.length;i++)  
    {  
        roles[i]=ObjectRefMgr.getRole (ids[i]);  
    }  
  
    member.doModifyLogin (null, roles);  
    request.getRequestDispatcher(jspHelper.getUrlAndMoveToPage ("ON_OK")).forward(request, response);  
}
```

After calling the next functional step, your logic handler should not execute any other code.

---

You can redirect a specific functional step inside your function by using `request.getRequestDispatcher`. If you do, make sure to include a return (`return;`).

---

## Passing Data From One Page to Another

### Preparing Data in a Form or Logic Handler

You can use the `request.setAttribute ("NAME", VALUE)` function to pass data between pages in the same client request.

The data passed using this method exist only during the life span of the current request and is destroyed afterwards.

Use `request.getAttribute ("NAME")` to retrieve the data in the next page.

For example, in the logic/form handler, you enter:

```
ContractF []contracts = ContractF.findContract(Filter);  
Request.setAttribute ("contracts_to_display",contracts);
```

In the next page:

```
displayContracts = (ContractF[]) request.getAttribute  
("contracts_to_display");
```

## Preventing Propagation of an HTTP Parameter

The JSPF automatically propagates HTTP parameters between pages.

If you do not want to automatically propagate an HTTP parameter, use the method:

```
jspHelper.doNotSend (String paramName)
```

An example:

```
...

    /* Create a new BLM session
    blmSession = new SessionF ();

    /* Add the BLM session to the session manager
    blmSession.add();

    /* Authenticate the user
    blnAuthenticate = blmSession.authenticate(request.getParameter("login"),
                                              request.getParameter("password"));

    jspHelper.doNotSend ("login");
    jspHelper.doNotSend ("password");
    ...
```

## Propagate HTTP Parameter from a Display Page

To propagate HTTP parameters to the next functional step, you can use the:

- FORM (HTTP method POST) element when using HTML forms
- A (HTTP method GET) element when using HTML links

Example of using HTML forms:

```
<form action="<%=jspHelper.getUrl("MODIFY_CONTACT") %>" method="post">
    <%=jspHelper.generateAllParametersAsHiddenFields () %>
    <input type="submit" value="<%=jspHelper.localize
("contact_change_confirm_button","default_text") %>">
    ...
</form>
```

In this example, `generateAllParametersAsHiddenFields()` generates `<input type="hidden" ...>` for each parameter including system parameters.

### Example of using HTML links

```
<a href="<%=jspHelper.encodeURLFunc("MODIFY_CONTACT ", hashTableOfParameters, false) %>">
<%=jspHelper.localize (" MODIFY_CONTACT ")%>
</a>
```

In this example, `encodeURLFunc("MODIFY_CONTACT", hashTableOfParameters, false)` generates the URL followed by the system parameters and the added parameters `hashTableOfParameters`.

## Propagate a New Parameter

In a workflow, if you want a logic handler to make parameters available for `generateAllNonSystemHiddenFields()` or `generateAllParametersAsHiddenFields()` methods, use:

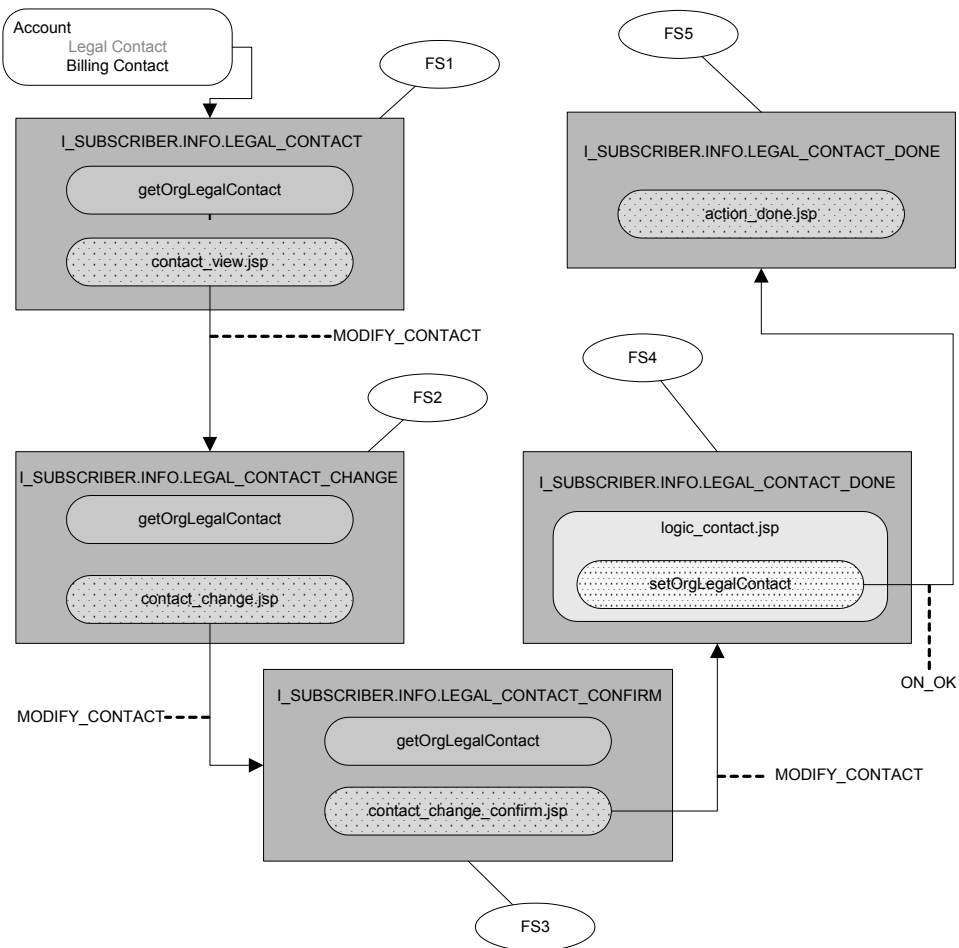
```
jspHelper.doSend (String paramName, String value)
```

## Sample Documented Workflow

Here is the workflow to allow a residential subscriber to view and modify the legal contact:

- 1 Call the page flow from a menu item
- 2 Display the actual values
- 3 Prompt the user to modify the actual values and enter new values
- 4 Ask to confirm the value changed by the user
- 5 Display the submission report

## Diagram of the Documented Workflow



## Define Functional Step for LEGAL\_CONTACT Menu Item

To specify which functional step is called by the `LEGAL_CONTACT` menu item, set the `funct` attribute to `I_SUBSCRIBER.INFO.LEGAL_CONTACT`:

```
<menu name="LEGAL_CONTACT"
funct="I_SUBSCRIBER.INFO.LEGAL_CONTACT" orgtypes="" roles="">
```

When the user clicks on the menu item, the framework calls the `I_SUBSCRIBER.INFO.LEGAL_CONTACT` functional step.



## Functional Step FS1

Define the `I_SUBSCRIBER.INFO.LEGAL_CONTACT` functional step:

```
<functionalstep name="I_SUBSCRIBER.INFO.LEGAL_CONTACT"
  url="contact_view.jsp" form="getOrgLegalContact">
  <redirect originalname="MODIFY_CONTACT" funct="I_SUBSCRIBER.INFO.LEGAL_CONTACT_CHANGE"/>
  <string name="title">
    <value name="en">Legal contact</value>
    <value name="fr">Contact légal</value>
  </string>
</functionalstep>
```

In the `contact_view.jsp` page, call the `MODIFY_CONTACT` alias only if the corresponding functional step `I_SUBSCRIBER.INFO.LEGAL_CONTACT_CHANGE` is allowed for the user role:

```
if (jspHelper.isFunctionalStepValid ("MODIFY_CONTACT") ) {
    %>
    <a href="<%=jspHelper.encodeURLFuncnt ("MODIFY_CONTACT",null,false)%>">
    <%=jspHelper.localize ("contact_view_modify_button","default_text")%>
    </a>
    <%
}
```

## Functional Step FS2

Define the `I_SUBSCRIBER.INFO.LEGAL_CONTACT_CHANGE` functional step:

```
<functionalstep name="I_SUBSCRIBER.INFO.LEGAL_CONTACT_CHANGE" url="contact_change.jsp"
  form="getOrgLegalContact">
  <redirect originalname="MODIFY_CONTACT" funct="I_SUBSCRIBER.INFO.LEGAL_CONTACT_CHANGE_CONFIRM"/>
  <string name="title">
    <value name="en">Change legal contact</value>
    <value name="fr">Changement de contact légal</value>
  </string>
</functionalstep>
```

In the `contact_change.jsp` page, set the action attribute of the form to `MODIFY_CONTACT`:

```
<form name="Contact" action="<%=jspHelper.encodeURLFuncnt ("MODIFY_CONTACT",null,false)%>" method="post">
```

## Functional Step FS3

Define the `I_SUBSCRIBER.INFO.LEGAL_CONTACT_CHANGE_CONFIRM` functional step:

```
<functionalstep name="I_SUBSCRIBER.INFO.LEGAL_CONTACT_CHANGE_CONFIRM" url="contact_change_confirm.jsp"
form="getOrgLegalContact">

    <redirect originalname="MODIFY_CONTACT" funct="I_SUBSCRIBER.INFO.LOGIC_LEGAL_CONTACT_CHANGE_DONE"/>

    <string name="title">

        <value name="en">Please confirm service address information change</value>

        <value name="fr">Confirmez le changement de contact légal</value>

    </string>

</functionalstep>
```

In the `contact_change_confirm.jsp` page, set the action attribute of the form to `MODIFY_CONTACT` and keep all the parameters of the previous page hidden in the form:

```
<form action="<%=jspHelper.getUrl("MODIFY_CONTACT")%>" method="post">

    <%=jspHelper.generateAllParametersAsHiddenFields ()%>

    <input type="submit" value="<%=jspHelper.localize
("contact_change_confirm_button","default_text")%>">

    ...

</form>
```

## Functional Step FS4

Define the `I_SUBSCRIBER.INFO.LOGIC_LEGAL_CONTACT_CHANGE_DONE` functional step:

```
<functionalstep name="I_SUBSCRIBER.INFO.LOGIC_LEGAL_CONTACT_CHANGE_DONE" url="logic_contact.jsp"
logic="setOrgLegalContact">

    <redirect originalname="ON_OK" funct="I_SUBSCRIBER.INFO.LEGAL_CONTACT_CHANGE_DONE"/>

</functionalstep>
```

In the `setOrgLegalContact` logic handler method of the `logic_contact.jsplogic` handler, do not automatically send the parameters to next functional step. So on the next call to `jspHelper.generateAllParametersAsHiddenFields()`, the parameters will not be generated as hidden.

```
jspHelper.doNotSend ("company_name"); jspHelper.doNotSend ("first_name"); jspHelper.doNotSend ("last_name");
jspHelper.doNotSend ("sex"); jspHelper.doNotSend ("title");
jspHelper.doNotSend ("home_number"); jspHelper.doNotSend ("home_street");
jspHelper.doNotSend ("home_zip"); jspHelper.doNotSend ("home_city");
jspHelper.doNotSend ("home_state"); jspHelper.doNotSend ("home_country");
jspHelper.doNotSend ("home_phone"); jspHelper.doNotSend ("fax"); jspHelper.doNotSend ("email");
//Call the next functional step using its alias ON_OK
request.getRequestDispatcher(jspHelper.getUrlAndMoveToPage ("ON_OK")).forward(request, response);
```

## Functional Step FS5

Define the functional step:

```
<functionalstep name="I_SUBSCRIBER.INFO.LEGAL_CONTACT_CHANGE_DONE" url="action_done.jsp">
    <string name="title">
        <value name="en">Submission of change legal address</value>
        <value name="fr">Soumission du changement d'adresse légale</value>
    </string>
    <string name="action_done_heading">
        <value name="en">Your legal address </value>
        <value name="fr">Votre adresse légale </value>
    </string>
</functionalstep>
```

There is no next page in the workflow, even if the done page allows the user to go to another page as referenced in the configuration file. The display page displays a button which redirects the user to the home page.

# Working with New JSPF JSPs

## Writing a JSP

Every JSP file you want to use in the Personalization Manager channels must include certain elements and follow certain rules.

The rules of using the JSPF in your JSPs:

- 1    Declare    variables
- 2    Include the    files
- 3    Include the form handlers that manage the processing of form data
- 4    Use JSPF implicit objects
- 5    Manage JSP workflow and menus

## Declaring JSPF variables

When you code a JSP to use the JSPF, you must declare the following variables:

- `String strJSP = "pagename.jsp"`
- Menu variables:
  - `String selectedTopbar = null`
  - `String selectedMenu = null`
  - `String selectedSubmenu = ""`

---

If you do not use the functional step feature, set these menu variables to the menu names.

---

## Including the JSPF JSPs

Include the following framework files:

- In the JSP Header:

```
<%@ include file="../../../common/fw/framework_start.jsp" %>
<%@ include file="../../../common/fw/framework_head.jsp" %>
```
- In the JSP Footer:

```
<%@ include file="../../../common/fw/framework_tail.jsp" %>
<%@ include file="../../../common/fw/framework_end.jsp" %>
```

## Managing form internal JSPF objects

Managing form internal JSPF objects

Every form managed by a JSP can provide a form handler that is automatically called by the framework engine.

In the JSP Header, include the form handlers:

- `<%@ include file="../../common/form_handler/setContact.jsp" %>`

## Managing internal JSPF objects

Managing internal JSPF objects

Include the formhandlers you want to use to manage the data and context of the page along with the following objects:

- `session` - the HTTP session
- `blmSession` - the BLM session
- `currentJsp` - the current JSP
- `jspHelper` - class that contains JSPF methods
- `errors` - the errors repository
- `results` - the results repository

## Modifying the JSPs

According to your needs, you may need to customize the JSPs. Customizing the JSPs involves:

- Adding a JSP
- Removing a JSP

## Adding a JSP to a channel application

Adding a JSP to a TSM application involves:

- Writing a JSP for the JSPF
- Adding the JSP to the JSPF configuration file
- Adding the JSP to the `form_handlers`
- Adding the JSP to menus
- Adding the JSP to the functional step

## Removing a JSP

Removing a JSP involves:

- Removing the JSP for the JSPF
- Removing the JSP from the JSPF configuration file
- Removing the JSP from the form\_handlers
- Removing the JSP from menus
- Removing the JSP from the functional step

## Examples of a JSPF JSP

The following JSPs are examples of JSPs that use the JSPF for form handling, variable declaration, authentication, etc.

This JSP is a commented template of a JSP, and contains the basic JSP code for forms.

TEMPLATE FOR FRAMEWORK.JSP	
Declaration of variables and inclusion of framework files.	<pre> &lt;%! // JSPF Page Constants String strJSP = "contact_change.jsp"; // JSPF Menu Constants // To use the FunctionalStep feature, set these constants to null String selectedTopbar = null; String selectedMenu = null; String selectedSubmenu = ""; %&gt;  &lt;!-- JSPF Includes     These JSPs contain the HTML used to build the JSPF --&gt; &lt;%% include file="fwk/framework_start.jsp" %&gt; &lt;%% include file="fwk/framework_head.jsp" %&gt; </pre>
Form Handler includes	<pre> &lt;!-- JSPF Form Handler Includes     These JSPs contain the form handlers and Java code that interacts with the BLM - --&gt; &lt;%% include file="form_handler/getContact.jsp" %&gt; &lt;%% include file="form_handler/createCustomer.jsp" %&gt; &lt;%% include file="form_handler/setContact.jsp" %&gt; &lt;%% include file="form_handler/addMember.jsp" %&gt; </pre>
Beginning of HTML code	<pre> &lt;!-- Application Workspace Code START --&gt; &lt;html&gt; &lt;head&gt;     &lt;title&gt;&lt;/title&gt;     &lt;link rel="stylesheet" href="&lt;%= PATH_WWW %&gt;styles.css" type="text/css"&gt; &lt;/head&gt; </pre>

TEMPLATE FOR FRAMEWORK.JSP	
Body of HTML code	<pre> &lt;body&gt;  Your HTML code goes here. You can look into the objErrors and objResults for information  reported by a form handler. Starting from here, you should avoid any processing and just read variables. Do not forget to localize your texts by using objXMLjsp.localize ().  For example, if the configuration for this JSP contains:  &lt;string name="Z"&gt;      &lt;value name="en"&gt;Cake&lt;/value&gt;      &lt;value name="fr"&gt;Gâteau&lt;/value&gt;  &lt;/string&gt;  You would use:  &lt;%= objHelper.localize ("Z", "Default String for Z") %&gt; </pre>
Declaring forms	<pre> Your forms must be declared as follows:  &lt;form action="&lt;%=objHelper.encodeURLFuncnt("MODIFY_CONTACT",null,false)%&gt;" method="post"&gt;  &lt;div class="listAText"&gt;&lt;%=localize(request,"contact_change_message","default_t ext")%&gt;&lt;/div&gt;  You may call a form from an hyperlink. Use a java script function to avoid the GET method. The java script  function may set the value of some of the input elements before submitting it :  &lt;a href="javascript:Y (value0, value1)"&gt;link to a form&lt;/a&gt;  &lt;/body&gt;  &lt;/html&gt; </pre>
Setting the value of form inputs	<pre> &lt;%  &lt;td class="mandatoryItemText"&gt;&lt;%=localize(request,"company_name_field","defau lt_text")%&gt;:&amp;nbsp;&lt;/td&gt;      &lt;td class="listAText"&gt;&lt;input type="text" name="company_name" value="&lt;%= objHelper.nonNullString(objContact.getCompanyName()) %&gt;"&gt;&lt;/td&gt;  %&gt; </pre>
End of HTML code	<pre> &lt;!-- Application Workspace Code END --&gt; </pre>
Inclusion of framework_end.jsp	<pre> &lt;%@ include file="fwk/framework_tail.jsp" %&gt;  &lt;%@ include file="fwk/framework_end.jsp" %&gt;  &lt;!-- end-of-page --&gt; </pre>

## Reloading JSPF Configuration Information

When the application server starts, the JSPF configuration file is loaded into memory. Once this file is loaded, changes made to this file are not taken into account. This presents a problem because when you develop a TSM application, you often have to modify the contents of this file.

In order to keep you from having to stop and restart your application server every time you modify this file, you can use a special JSP. The `framework_appreset.jsp` JSP reloads the JSPF configuration file. This file is located in `<home_dir>/channels/common/fwk.`

---

This JSP is for development purposes only and should not be used in production environments.

---

### To reload the JSPF configuration file

Run the `framework_appreset.jsp`. When called, this JSP reloads the JSPF configuration file.

---

The full path of the JSPF configuration file must be declared in the `jfnApplication.properties`. If you use `res:/` for paths, this JSP does not work.

---



# Index

## A

Authentication  
in JSP declaration • 33

## C

Channels  
about • 15  
adding • 66  
contents • 63, 64, 65  
customizing • 58, 68  
IVR Channel • 21, 22  
JSPs • 63  
menus • 64, 65  
WAP channel • 20, 21  
Web channel • 15, 16, 18  
workflows • 65  
CID (Customer Interaction Datastore)  
adding sample data • 61  
removing sample data • 61  
cidAdminTool Administration Tool  
create\_demo\_cid\_test command • 61  
drop\_demo\_cid\_structure command • 61

## D

Display Pages  
about • 52, 83  
and the JSPF • 50, 52, 83  
declaring • 83  
defining access to • 84  
Display Pages - using • 83, 84  
functional steps • 86  
strings • 82, 85

## F

Form Handler JSPs  
about • 54, 87  
calling methods • 88  
declaring • 87  
location • 87  
passing data • 93, 94, 95  
writing • 87, 88  
framework\_head.jsp Framework JSP

about • 50  
location • 50  
using • 51  
framework\_menuend.jsp Framework JSP  
about • 50  
location • 50  
framework\_menustart.jsp Framework JSP  
about • 50  
location • 50  
framework\_start.jsp Framework JSP  
about • 50  
location • 50  
using • 51  
framework\_tail.jsp Framework JSP  
about • 50  
location • 50  
Functional Steps  
about • 41, 80  
declaring in the JSPF • 41, 42, 43  
example • 43, 95, 96, 97, 98, 99  
in menus • 82  
redirects • 42, 81, 86, 92  
strings • 43, 82  
using • 81

## H

Help  
technical support • ix

## J

JSPF

- about • 26
- channels • 14, 15
- components • 27
- display pages • 52, 83
- features • 26
- form handler JSPs • 54, 87, 88
- Framework JSPs • 49, 50, 51, 52
- JSP programming • 33, 34, 35, 101
- location • 27
- logic handler JSPs • 56, 89, 90, 91, 92
- mandatory includes • 100
- reloading configuration file • 104
- removing JSPs • 102
- sample JSP • 102
- variables • 100
- JSPF Configuration File
  - about • 30
  - contents • 30, 33, 36, 41
  - form settings • 34
  - functional steps • 41, 42, 43
  - JSP page declarations • 33, 34, 35, 83, 87, 89
  - menus • 36, 37, 38, 39
  - reloading • 104
  - string settings • 34, 37, 43
- jfnApplication.properties Configuration File
  - about • 27, 28
  - and channels • 16, 21, 22, 30, 63, 66
  - location • 28
  - using • 28, 66
- JSPF Classes
  - about • 48
  - jar file • 48
- JSP
  - adding to JSPF • 100, 101
  - and the JSPF • 33, 34, 35
  - channels • 63
  - declaring in JSPF configuration file • 33, 34, 35, 100
  - example • 102
  - form settings • 34
  - framework JSPs • 50, 51, 102
  - JSPF variables • 100
  - JSP - mandatory includes • 100, 101
  - removing from JSPF • 102

## L

- Languages
  - localizing • 34
- Logic Handler JSPs

- about • 56, 89
- calling functional steps from • 92
- declaring • 89
- location • 89
- passing data • 93, 94, 95
- writing • 89, 90, 91

## M

- Media
  - adding • 66
  - and channels • 66
- Menus
  - about • 36, 64
  - contents • 36, 64
  - customizing • 76, 77, 78, 79
  - declaring • 37, 39
  - IVR channel • 65
  - strings • 37
  - structure • 38, 76, 77, 78
  - WAP channel • 65
  - Web channel • 64
- MyIVR Channel
  - about • 21
  - contents • 21
  - customizing • 58
  - features • 22
  - location • 22
  - menus • 65
- MyWAP Channel
  - about • 20
  - contents • 20
  - customizing • 58
  - features • 21
  - location • 21
  - menus • 65
- MyWeb Channel
  - about • 15
  - contents • 15
  - customizing • 58
  - display options • 18, 19, 20
  - features • 16
  - location • 16
  - menus • 64
- MyWeb.xml Configuration File
  - about • 30
  - location • 30
  - using • 30, 33, 36, 41

## P

- Parameters

- about • 68
- adding • 69
- examples • 72
- groups • 71
- in the CID • 68
- localizing • 70
- values • 69, 70

#### Personalization Data

- adding to channel • 68, 69
- association between values and parameter • 70
- default values • 69, 70
- example • 72, 73
- in the CID • 69, 70
- localizing • 70
- parameter groups • 71

#### Personalization Manager

- about • 14
- channels • 15
- components • 14
- sample data • 59, 61

## R

#### Roles

- in JSP declaration • 33

## S

#### Sample Data

- about • 59, 61
- contents • 59, 60
- inserting into CID • 61
- reference • 59
- removing from CID • 61
- Sample Data - services • 59
- users • 60

## W

#### Workflows

- about • 41
- and Functional Steps • 41
- declaring in the JSPF • 41, 42, 43
- example • 43, 95, 96, 97, 98, 99
- redirects • 42, 81, 86, 92
- strings • 43, 82