

Precision Marketing in WebSphere Commerce

Enable your precision marketing objectives

Use IBM Management Center marketing tools

Explore scenarios and customizations



Rufus Credle
Priyajith Chembakassery
Julie Iammarino
Vikas C Roy



International Technical Support Organization

Precision Marketing in WebSphere Commerce

May 2010

Archived

Note: Before using this information and the product it supports, read the information in “Notices” on page ix.

First Edition (May 2010)

This edition applies to WebSphere Commerce Version 7.

© Copyright International Business Machines Corporation 2010. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contact an IBM Software Services Sales Specialist



Start SMALL, Start BIG, ... **JUST START**
architectural knowledge, skills, research and development . . .
that's IBM Software Services for WebSphere.

Our highly skilled consultants make it easy for you to design, build, test and deploy solutions, helping you build a smarter and more efficient business. **Our worldwide network of services specialists wants you to have it all!** Implementation, migration, architecture and design services: IBM Software Services has the right fit for you. We also deliver just-in-time, customized workshops and education tailored for your business needs. You have the knowledge, now reach out to the experts who can help you extend and realize the value.

For a WebSphere services solution that fits your needs, contact an IBM Software Services Sales Specialist:
ibm.com/developerworks/websphere/services/contacts.html

Contents

Contact an IBM Software Services Sales Specialist	iii
Notices	ix
Trademarks	x
Preface	xi
The team that wrote this book	xi
Now you can become a published author, too!	xiii
Comments welcome	xiv
Stay connected to IBM Redbooks	xiv
Part 1. Precision marketing with WebSphere Commerce	1
Chapter 1. Precision marketing	3
1.1 Overview and objectives	4
1.2 Benefits and capabilities of WebSphere Commerce	5
1.3 Marketing concepts and terminology	6
1.3.1 Web activities	6
1.3.2 Dialog activities	7
1.3.3 Activity templates	8
1.3.4 Customer segment	10
1.3.5 Triggers	12
1.3.6 Branches and Experiments	14
1.3.7 Targets	15
1.3.8 Action	17
1.4 Working from statistics	19
1.5 Customized precision marketing for your business	22
1.5.1 Creating a new target wish list	22
1.5.2 Extending existing customer segment	22
1.5.3 Creating a new trigger to monitor customer's interaction	23
1.5.4 Recommending products based on their attributes	23
Chapter 2. Web activities	25
2.1 Overview	26
2.2 Scenarios	26
2.2.1 Cross promoting products	26
2.2.2 Advertising new products	33
Chapter 3. Dialog activities	37

3.1 Overview	38
3.2 Scenarios	38
3.2.1 Repeat customers and big spenders	38
3.2.2 Abandoned shopping carts	47
Part 2. Customization scenario	51
Chapter 4. Create a new target	53
4.1 Overview	54
4.2 Design	56
4.2.1 Target template definitions	56
4.2.2 Task commands	59
4.2.3 Object definitions	60
4.2.4 Properties view	61
4.2.5 Summary definition	61
4.2.6 Serialization JSPs for the wish list target	61
4.3 Implementation	62
4.3.1 Creating the target template definition	62
4.3.2 Creating the target task commands	64
4.3.3 Creating validation messages	74
4.3.4 Defining the icons to represent the new target	81
4.3.5 Defining a resource bundle and properties file for UI text	82
4.3.6 Creating the object definition for the target	84
4.3.7 Creating the properties view for the target	87
4.3.8 Creating the summary for the target	90
4.3.9 Registering the new classes in the marketing extensions library	91
4.3.10 Adding the target to the element palette in the Activity Builder	92
4.3.11 Creating and registering a serialization JSP for the new target	93
4.4 Test your code	96
4.4.1 Checking the UI customization for new wish list target	96
4.4.2 Creating a new wish list target Web activity	97
4.4.3 Testing your task command with Web stores	98
Chapter 5. Extend the customer segment	99
5.1 Overview	100
5.2 Design	102
5.2.1 Task commands	103
5.2.2 Object definition	103
5.2.3 Properties view	105
5.2.4 Serialization JSPs for the customer segment attribute	105
5.3 Implementation	105
5.3.1 Creating the new customer segment task command	105
5.3.2 Defining a resource bundle and properties file for UI text	113
5.3.3 Creating the object definition	115

5.3.4	Creating the properties view for the customer segment attribute . .	118
5.3.5	Registering the new classes in the marketing extensions library . .	119
5.3.6	Creating and registering a serialization JSP for the new attribute	120
5.4	Test your code	122
5.4.1	Checking the UI customization for the new customer segment definition	122
5.4.2	Creating a dialog activity with the new customer segment	123
5.4.3	Testing your task command with Web stores	123
Chapter 6.	Create a new trigger	125
6.1	Overview	126
6.2	Design	128
6.2.1	Triggering a template definition	128
6.2.2	Task command	132
6.2.3	External event notification	133
6.2.4	Object definition	135
6.2.5	Properties view	136
6.2.6	Summary definition	136
6.3	Implementation	137
6.3.1	Creating the trigger template definition	137
6.3.2	Creating the CustomerProvidesPhoneNumberTriggerTaskCmd interface	138
6.3.3	Setting up external system to inform the marketing services of the external event	141
6.3.4	Defining a resource bundle and properties file for UI text	142
6.3.5	Creating the object definition for the trigger	143
6.3.6	Creating the properties view for the trigger	146
6.3.7	Creating the summary for the trigger	147
6.3.8	Registering the new classes in the marketing extensions library . .	151
6.3.9	Adding the trigger to the element palette in the Activity Builder . .	152
6.3.10	Registering the serialization JSP for the trigger	152
6.4	Test your code	153
Chapter 7.	Customize an existing action	155
7.1	Overview	156
7.2	Design	160
7.2.1	Task command	160
7.2.2	Template definition	161
7.2.3	Object definition	161
7.2.4	Properties view	161
7.2.5	Serialization JSP for the new properties	161
7.3	Implementation	162

7.3.1	Creating the new display product attribute task command	162
7.3.2	Creating the template definition for the new template.	168
7.3.3	Defining a resource bundle and properties file for UI text.	169
7.3.4	Updating the object definition	170
7.3.5	Updating the properties view.	172
7.3.6	Creating the summary definition	174
7.3.7	Registering the new classes in the marketing extensions library . .	175
7.3.8	Creating and registering the serialization JSP for the new template	176
7.4	Test your code.	178
7.4.1	Checking the UI customization for recommend catalog entries action	178
7.4.2	Creating a Web activity to recommend catalog entries with attributes.	178
7.4.3	Testing your task command with Web stores	178
Part 3.	Common Practices	179
	Chapter 8. Common practices	181
8.1	Precision marketing common practices.	182
8.2	Customization common practices	184
Part 4.	Appendices	187
	Appendix A. Triggers, targets, actions, and branching	189
	Appendix B. Additional material	193
	Locating the Web material	193
	Using the Web material	194
	How to use the Web material	194
	Related publications	195
	IBM Redbooks	195
	Online resources	195
	How to get Redbooks.	200
	Help from IBM	200

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:


This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

BladeCenter®
DB2®
IBM®
Rational®

Redbooks®
Redpaper™
Redpapers™
Redbooks (logo) ®

System x®
WebSphere®

The following terms are trademarks of other companies:

Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

This IBM® Redbooks® publication positions WebSphere® Commerce Version 7 in today's marketplace and discusses its enhanced features enabling business users to achieve their vision for precision marketing.

This book helps you to tailor and configure the Marketing tool in IBM Management Center for WebSphere Commerce to create and manage various marketing campaigns as needed by your business.

This book provides several business scenarios which can be implemented through simple customizations. Each scenario addresses a unique requirement which can be mapped with similar business scenarios.

This book has been developed for an experienced WebSphere Commerce design and developer audience.

The team who wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.



Figure 1 From left to right, Rufus Credle, Julie Iammarino, Vikas C Roy, Priyajith Chembakassery

Rufus Credle is a Certified Consulting IT Specialist at the ITSO, Raleigh Center. In his role as Project Leader, he conducts residencies and develops IBM Redbooks and Redpapers™ about network operating systems, enterprise resource planning (ERP) solutions, voice technology, high availability, clustering solutions, Web application servers, pervasive computing, IBM and OEM e-business applications, WebSphere Commerce, IBM industry technology, System x®, and IBM BladeCenter®. Rufus' various positions during his IBM career include assignments in administration and asset management, systems engineering, sales and marketing, and IT services. He has a BS degree in Business Management from Saint Augustine's College. Rufus has been employed at IBM for 29 years.

Priyajith Chembakassery is a Staff Software Engineer in IBM India Software Labs. He has six years experience in the software industry and three years of experience with WebSphere Commerce. He works for the WebSphere Commerce analytics team in Bangalore. He holds an M.sc. degree in Information Technology.

Julie Iammarino is a Consultant with Ascendant Technology in Cleveland, Ohio. She is an IBM WebSphere Commerce 6.0 Certified Developer. Her areas of expertise include the integration of WebSphere Commerce, WebSphere Portal, LDAP, and DB2®. She designs and develops using Java™, EJB, JSTL, JSP, CSS, and JavaScript.

Vikas C Roy is a Staff Software Engineer in IBM India Software Lab. He joined IBM in 2005. Prior to that he had two years of industry experience in product development. His areas of expertise include Core Java, Eclipse RCP and Plug-in development, data structures and design patterns. Vikas is currently working with the WebSphere Commerce product development team and he has written extensively on its various components. He holds an MS degree in Software Systems from BITS Pilani, India.

Thanks to the following people for their contributions to this project:

Tamikia Barrow, Margaret Ticknor, Craig Schmidt
International Technical Support Organization, Raleigh Center

Howard Borenstein, Marketing and Promotions Architect, WebSphere
Commerce Development
IBM Toronto

John McLean, Development Manager, WebSphere Commerce Promotions,
Marketing and Analytics
IBM Toronto

James Fong, Manager, WebSphere Commerce Technology Practice, IBM
Software Services for WebSphere
IBM Toronto

Dick Hrabik, IBM AIM Customer Programs, Program Manager WebSphere
Commerce
IBM Austin

Baron Lam, IBM Software Group, WebSphere Commerce Development
IBM Toronto

Janet Browne, IBM Software Group, Application and Integration Middleware
Software
IBM Toronto

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author - all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:
ibm.com/redbooks
- ▶ Send your comments in an e-mail to:
redbooks@us.ibm.com
- ▶ Mail your comments to:
IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- ▶ Find us on Facebook:
<http://www.facebook.com/IBMRedbooks>
- ▶ Follow us on twitter:
<http://twitter.com/ibmredbooks>
- ▶ Look for us on LinkedIn:
<http://www.linkedin.com/groups?home=&gid=2130806>
- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:
<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>
- ▶ Stay current on recent Redbooks publications with RSS Feeds:
<http://www.redbooks.ibm.com/rss.html>



Part 1

Precision marketing with WebSphere Commerce

Customers differ greatly in terms of their demographics, lifestyles, needs, perceptions, preferences, and behaviors. A business entity needs to understand the current profitability of each individual customer or customers in a segment, as well as their potential lifetime profitability. With precision marketing, delivering the right message at the right time through the right channel to the right people, blended with modern technology, it is possible to treat customers individually. In this section, we discuss the enhanced features of WebSphere Commerce Version 7 to enable business users to achieve their vision for precision marketing.

Precision marketing

This chapter provides an overview of precision marketing and the features available in WebSphere Commerce Version 7 to enable marketing managers to target more precisely to their customers.

This chapter contains the following sections:

- ▶ Overview and objectives
- ▶ Benefits and capabilities of WebSphere Commerce
- ▶ Marketing concepts and terminology
- ▶ Working from statistics
- ▶ Customized precision marketing for your business

In today's competitive business environment, retaining existing and loyal customers is as important as acquiring new customers. These customers are like individual brand ambassadors for your business who spread awareness about your products by word of mouth. With precision marketing, by using the appropriate tools and technology with statistical data analysis capabilities, and by taking the specialized approaches to marketing campaign design and its execution, companies can discover a new source of power. This can result in delivering accurate and exact marketing promotional messages to customer segments.

Be precise. A lack of precision is dangerous when the margin of error is small.

- Donald Rumsfeld

Precision is the quality of being accurate and exact. Precision tools are used to measure things accurately. Similar to that, *Precision Marketing* refers to targeting marketing content based on a user's online behavior. This includes browsing and searching behavior, along with contextual information, such as referral sites and cookies. In other words, precision marketing gives companies the ability to attract people with specially configured marketing messages. Marketing content can be targeted towards both registered and guest users. To do that, IBM Management Center for WebSphere Commerce (also referred to as Management Center in this IBM Redbooks publication) provides a rich and interactive user interface (UI) to business users, enabling precision marketing.

1.1 Overview and objectives

In today's e-commerce business environment, sending promotional e-mails and gift coupons are a common practice followed by on-line retailers. Most of these e-mails are being sent to all their customers, assuming that it might be of interest to someone. Consider a scenario where an e-mail is sent promoting fitness and sports equipment. At first glance, this sounds okay. But these promotional e-mails do not make any sense for customers whose sole interest is music and art. This is just an example of imprecise marketing without knowing your customer base.

With precision marketing, the goal is to improve the efficiency and effectiveness by which a business can attract, retain, and make use of their most profitable customers. The primary objective of precision marketing is to improve the browser-to-buyer conversion ratio. (This means the ratio of customers who browse through the catalogs who also buy the products.) Precision marketing is about turning non-customers into customers and turning existing customers into better customers. This is achieved by getting the right marketing promotions at the right time to get the shopper to buy the products.

Precision marketing seeks to reduce customer acquisition and support costs and improve the overall productivity of marketing spending. In particular, this means minimizing the money that goes to waste by marketing to the wrong people. The wrong people are those who, for whatever reason, are unfit candidates for buying the product or service offered by the company. Therefore, the precise and personalized marketing activities are targeted to the appropriate individual shoppers or the customer segments to increase the conversion rates on orders. These activities can be carried out continuously with the customers to keep interactions on going.

1.2 Benefits and capabilities of WebSphere Commerce

Precision marketing enables retailers to recognize and respond to events across multiple channels and touch points. It allows retailers to follow and build a dialog with the shopper by delivering contextual targeted marketing information to keep the shopper within the brand as they shop across multiple channels. WebSphere Commerce Version 7 has the ability to create precise and timely marketing promotions to target individual customers and the customer segments. A marketing manager can use the interactive UI of IBM Management Center for WebSphere Commerce to define these marketing activities. With powerful capabilities and business user tools, WebSphere Commerce is a proven, flexible solution that scales to meet the various business models, whatever the size of the business. Easy-to-use tools are available for precision marketing, promotions, catalog management, and merchandising to help business users create, manage, and deploy personalized marketing and promotion campaigns to help revolutionize customer shopping experiences across all sales channels.

The following list details precision marketing capabilities in WebSphere Commerce Version 7:

- ▶ Target customers with relevant marketing information based on browsing and shopping behavior.
- ▶ Wait for customers to perform a specific action, such as register with the store or abandon a cart, and respond with a relevant and timely marketing message. This approach uses both e-mail and SMS messages to reach customers.
- ▶ Target customers who participate in social commerce activities that the store offers. Build loyalty by encouraging and rewarding participation.
- ▶ Support for capturing various events from multiple channels, such as shopper behavior on a Web site or any other channel. Process trigger events based on predefined business rules and logic.
- ▶ Respond to trigger events by delivering actions to any channel (for example, an action can be an e-mail message or mobile SMS offering a promotion).
- ▶ Dialog Activity Builder for defining and managing triggers and actions.
- ▶ Assign and reassign shoppers to and from segments, dynamically.
- ▶ Support advanced promotions, including promotions based on product attributes, and caps on discount amount.
- ▶ Build precision marketing customer dialogs quickly with best-in-class business user tooling

With these capabilities, WebSphere Commerce precision marketing helps businesses to:

- ▶ Improve order conversion rates
- ▶ Increase customer satisfaction and loyalty
- ▶ Improve customer experience with personalization
- ▶ Improve brand-stickiness as shoppers switch channels

1.3 Marketing concepts and terminology

WebSphere Commerce has defined and used terminologies and configurations to work with marketing activities. These terminologies are being used in this book to explain the precision marketing concept. You can find them in the IBM Management Center for WebSphere Commerce when creating those activities.

1.3.1 Web activities

A Web activity is an existing feature in WebSphere Commerce that is based on a Web store e-marketing spot. The Web activities allow marketing managers to provide personalized contents on their Web store that is targeted to individual customers and customer segments. This helps in communicating the most relevant and precise marketing contents to the shopper, which increases the sales orders. When a customer views a store page, the Web activity determines what to show the customer based on current and past actions. By using e-marketing spots, Web activities can display catalog entries, categories, contents, merchandising associations, and promotion recommendations from the store. You can target sets of customers or segments within Web activities, which allows you to personalize what customers see in Web store e-marketing spots.

You can create Web activities in a graphical editor called the *Activity Builder*. See Chapter 2, “Web activities” on page 25 for more details. In Figure 1-1, the marketing manager is targeting the registered customers in the store to show them cross-sell merchandising associations based on the customer’s purchase history.

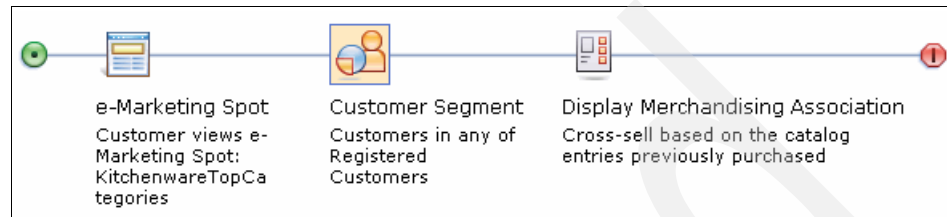


Figure 1-1 Web activity to display the marketing information of the cross sell merchandising associations to registered customer segment

Web activities always start with an e-marketing spot and can have targets, actions, and branches depending on the business scenario required by the marketing manager. By default, the e-marketing spot trigger is included as the first element in all standard Web activity templates. The Web activity is triggered when a customer views the page containing the e-marketing spot. Only the e-marketing spot trigger is applicable to Web activities. Although you cannot include more than one trigger in a Web activity, the results from a Web activity are not restricted to being called from a commerce Web page e-marketing spot. A Web service can also be called to get the results to display (for example, from Portal or from a Java command). See Chapter 2, “Web activities” on page 25 for more details.

1.3.2 Dialog activities

Dialog activities are newly introduced in WebSphere Commerce Version 7. It allows multi-step interactions initiated by specific customer actions. It enables the marketing manager to deliver a continuous, multi-step interaction with their customers. It extends beyond e-marketing spots as it allows you to track customer behavior over a period of time and take actions accordingly. Dialog activities can be triggered by customer events such as placing an order, calling the call center agent, or signing up for a loyalty program. Using customer events to build dialog activities enables the marketing managers to craft the interactions they have with their customers and not leave it to chance. Similar to how many sales organizations have a multi-step sales process to take customers from a prospect to a purchasing customer, dialog activities map out a similar process based on events and interactions.

You can create dialog activities using the Activity Builder. See Chapter 3, “Dialog activities” on page 37 for more details. In Figure 1-2, the marketing manager is targeting all customers who register with the store and sends them a welcome greeting note with promotional details.

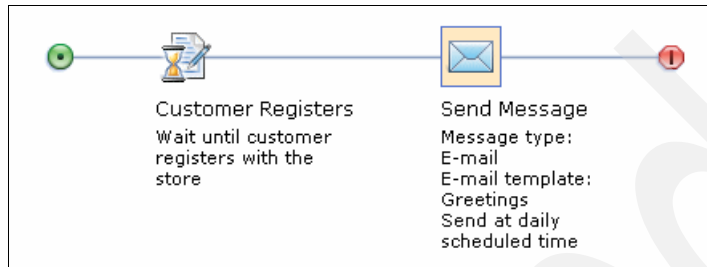


Figure 1-2 Welcome greeting note with promotional details

A dialog activity has an additional feature, called a *trigger*, to execute the actions automatically, based on the conditions specified in the trigger. Dialog activities can start with any trigger in the palette. In most cases, triggers are customer events (for example, a customer registers or places an order). Additionally, a trigger can be an elapsed period of time (for example, a week). When the trigger occurs, the dialog activity proceeds. You can include more than one trigger in a dialog activity. Apart from the triggers, the dialog activities can have targets, actions, and branches depending on the business scenario required by the marketing manager. See Chapter 3, “Dialog activities” on page 37 for more details.

1.3.3 Activity templates

An activity template is a predefined starting point for an activity that a business user uses to create new Web and dialog activities. Choose from one of the available standard activity templates bundled with the IBM Management Center for WebSphere Commerce, or define your own custom activity templates.

IBM Management Center for WebSphere Commerce provides the standard Web and dialog activity templates (shown in Table 1-1 on page 9), which can be used when creating the activities.

Table 1-1 Web and dialog activity templates

Standard Activity Template	Web Activity	Dialog Activity
Blank Web/Dialog Activity	Yes	Yes
Catalog Entry Recommendation	Yes	No
Category Recommendation	Yes	No
Content Recommendation	Yes	No
Merchandising Association Recommendation	Yes	No
Promotion Recommendation	Yes	No

When you define a Web activity in IBM Management Center for WebSphere Commerce, the standard templates shown in Figure 1-3 are displayed.

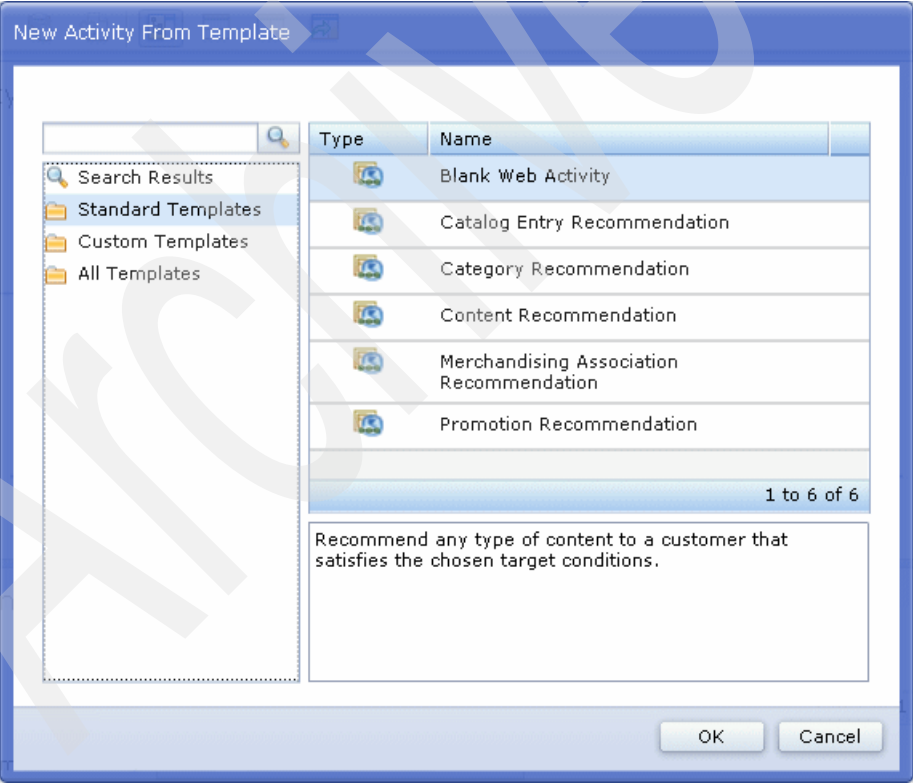


Figure 1-3 New Activity Template window for Web Activity

When using Web activities templates, you must select the e-marketing spot in which to display the activity, and the appropriate asset to display to customers. If the standard activity templates do not meet your needs, you can create custom activity templates for both Web and dialog activities. There are two ways to create a custom activity template:

- ▶ Build an activity to use as a template from scratch.
- ▶ Create a template based on an existing activity.

The management of templates allows the marketing manager to create, change, and delete custom templates as well as create new activities based on the templates. Activity templates are blueprints for a Web or dialog activity, but share the same space with those activities. This means you cannot create a Web or dialog activity with the name of an existing activity template. Similarly, you cannot create an activity template with the name of existing Web or dialog activities.

Activity templates are activity-type specific. This means Web activities can be created using Web activity templates, and dialog activities can be created using dialog activity templates, but there is no support for converting one type of template to another. Also, the activity templates are store-path enabled to support extended sites. The global activity templates are available for use with all stores, but they cannot be modified. Business users cannot create global activity templates, but can create activities from them or make a copy of them into a local store for further customization.

1.3.4 Customer segment

A *customer segment* incorporates registration information, demographics, address information, customer culture, purchase history, and other attributes that define a dynamic group of customers or accounts. Customer segments serve as targets for advertising, promotions, and suggestive selling.

IBM Management Center for WebSphere Commerce supports creating dynamic customer segments using the **Add To or Remove From Customer Segment** action in Web and dialog activities. Using this feature, you can define a dynamic customer segment that gets updated automatically based on customers behavior. Segments are considered dynamic because customers belong to a segment based on personal data and purchase history, both of which might change. For example, you might create segments based on a customer's registration status. If you create a segment that requires customers to be registered to qualify, an unregistered customer is excluded. If that same customer registers with the store later, they become a member of that target segment, and continue to be a member until the segment is deleted. The new current age attribute is a function of the Customer Segment tool which is new in Management Center. With IBM Management Center for WebSphere Commerce,

you can use the new current age according to the supplied birth date to build segments based on a customer's exact age. The customer's age remains accurate over time because the server calculates the age from the supplied birth date.

Customer segments also support static criteria. You can explicitly include or exclude particular customers or accounts, which overrides any defined dynamic criteria. In this way for example, you can explicitly include customers in a segment where they otherwise do not match, or exclude them from a segment even though it was a match. See Figure 1-4.

High Profile Guest Customers

Save and Close Close

General Properties Customer or Segment Registration Demographics Address Purchase Details Miscellaneous

Customer segments ⓘ Ignore customer segments

Included customers ⓘ

* Logon ID	First Name	Last Name
skarl	Karl	Stevenson

1 to 1 of 1

Excluded customers ⓘ

* Logon ID	First Name	Last Name
------------	------------	-----------

0 to 0 of 0

Figure 1-4 Customer segment definition window

IBM Management Center for WebSphere Commerce Customer Segment Tooling provides a full set of functions for managing customer segments created in Management Center. It also allows business users to view, change, and delete the existing customer segments created in WebSphere Commerce Accelerator through an indirect migration approach. For instance, when a business user opens up an existing customer segment created in WebSphere Commerce Accelerator, the data migration happens immediately. After the customer segment is migrated to Management Center, it can be maintained and managed in Management Center Customer Segment Tooling.

Using WebSphere Commerce Accelerator to manage the customer segments being migrated into Management Center or the customer segments created in Management Center is not supported. However, business users can still view those customer segments from WebSphere Commerce Accelerator limited to the

segment attributes supported in WebSphere Commerce Accelerator. In other words, for those new segment attributes supported in Management Center, such as “Use marketing activities to add or remove customers” and “Age range,” it is not shown in WebSphere Commerce Accelerator when a customer segment contains such attribute conditions.

1.3.5 Triggers

A trigger is used to wait for a customer to do some action, or to wait for a certain period of time to elapse. It defines the event that causes the activity to start or continue. When a trigger occurs, the activity flow can continue from where the trigger was defined. IBM Management Center for WebSphere Commerce has two sets of triggers defined for Web and Dialog activities, as shown in Table 1-2.

Table 1-2 Web and dialog activities

Web activities	Dialog activities
Web activities always start with the e-marketing spot trigger. By default, the e-marketing spot trigger is included as the first element in all standard Web activity templates. The Web activity is triggered when a customer views the page containing the e-marketing spot.	Dialog activities can start with any trigger that is available in the palette. In most cases, triggers are customer events, for example, a customer registers or places an order. Additionally, a trigger can be an elapsed period of time, for example, a week. When the trigger you specify occurs, the Dialog activity proceeds.
You cannot include more than one trigger in a Web activity.	You can include more than one trigger in a Dialog activity.

In the Activity Builder palette, triggers are available only for the dialog activities. The following sections discuss the dialog-specific triggers available in IBM Management Center for WebSphere Commerce. See Appendix A, “Triggers, targets, actions, and branching” on page 189 for the icons in palette.

Wait

When a Wait trigger element is reached in a dialog activity, the customer temporarily stops moving along that activity path. This is used to specify a waiting period in the activity. When the waiting period ends, the activity continues and the customer proceeds to the next element in the activity flow. This is an effective way to automate follow-up marketing actions (for example, sending a promotional e-mail to a customer who has not placed an order within a certain period of time).

Customer Places Order

The Customer Places Order trigger specifies waiting to perform an action until a customer completes the order checkout process and submits the order in the Web store. You can use this trigger in a dialog activity to begin or continue the activity as soon as a customer places an order on your site. This trigger provides a means to reach out to customers when they place an order on the site, which is usually an indication that they have an interest in your site. Targeting these customers with marketing materials or promotional offers can encourage them to return to your site and continue shopping.

Customer Abandons Shopping Cart

The Customer Abandons Shopping Cart trigger allows a business user to follow up with their customers if they have abandoned their shopping cart. The business user can specify the number of days of inactivity that defines that the cart has been abandoned. After that, this trigger automatically checks for customers who have abandoned their cart for the number of days specified. You can automate follow-up marketing actions for these customers (for example, offering them incentives to come back to the store and complete their purchase by sending an e-mail or text message, or issuing a coupon).

Customer Registers

The Customer Registers trigger specifies to wait to perform an action until a customer registers in the store. This trigger provides a means to reach out to customers when they first register with the store, which is usually an indication that they have an interest in your site. Targeting these customers with marketing materials or promotional offers can encourage them to make a purchase.

Customer Participates In Social Commerce

This trigger relies on the Web stores or the social commerce events to inform the marketing runtime about the social commerce interactions. You can use this trigger in a dialog activity to begin or continue an activity as soon as a customer participates in social commerce on your site a specified number of times.

Customer Is In Segment

This trigger checks the specified customer segments to determine which customers are in the segments. Those customers then proceed through the dialog activity. You can automate follow-up marketing actions for these customers (such as sending an e-mail or text message, or issuing a coupon). You can set the trigger to run every day or one time only. For example, the Customer Is In Segment trigger can be used to send an e-mail to all registered customers in the store. Then, a Wait trigger can be used to follow up with a customer if they have not placed an order in the store within a specific amount of time.

Customer Celebrates Birthday

The Customer Celebrates Birthday trigger allows a business user to send birthday wishes to a customer or entitle them to a promotion. The business user can specify the number of days before the customer's birthday to participate in the dialog activity. This trigger checks for customers who are celebrating a birthday in the number of days you specify. You can then automate follow-up marketing actions for these customers, such as sending a birthday wish in an e-mail or text message, or issuing a special birthday coupon, or entitling the customer to a birthday promotion. Marketing to customers when they are celebrating a birthday can build loyalty and provide customers with a great excuse to treat themselves.

The customer needs to be a registered customer with the store to invoke the dialog activities by this trigger.

1.3.6 Branches and Experiments

A *branch* allows you to define alternate, or more than one, course of action in an activity flow. The decision nodes in an activity are where shoppers might see or qualify for content based on various conditions. In the IBM Management Center for WebSphere Commerce UI, branches look like and behave similar to experiments, but with notable differences:

- ▶ There are no limits to the number of branches in an activity. A user can create multiple branches in an activity. There is a restriction to one experiment per activity.
- ▶ Branches can be nested. In addition to having multiple branches in an activity, branches can exist within other branches as well.
- ▶ In a Web activity, branching is not allowed on the view e-marketing spot trigger.
- ▶ Branches do not track detailed order statistics.
- ▶ Paths within a branch cannot be marked as a winner, and therefore, a new activity cannot be created from a winner.

You can add a branch element to both Web and dialog activities. The following sections discuss new branch types that have been introduced in IBM WebSphere Commerce V7.

First path for which the customer qualifies

Continue along the first successfully evaluated path. When the paths begin with targets, this means the path in which all the targets evaluate to true (before encountering a trigger or action) is the first successful path. Each customer sees one product depending on the value of their shopping cart. Use this branch type if you want to target customer groups with personalized marketing actions within the same activity. When the activity is triggered, each customer experiences the first path for which he or she qualifies, and no other paths. The server evaluates the customer against the targets at the beginning of each path, starting with the first path and working down to the last path. As soon as the customer qualifies for the targets in a path, she proceeds down the path and experiences the action.

All paths for which the customer qualifies

Evaluate each one of the paths in the branch. Each path is independent of the others. Use this branch type if you want customers experiencing the activity to be evaluated for each path independently of the other paths. Potentially, a customer can experience all the paths in the activity as long as she meets the requirements of the targets at the beginning of each path.

Random path

Select a path randomly based on percentages associated with the paths. Each customer sees one of the two paths. Use this branch type if you want the server to assign which activity path a customer follows randomly, based on a percentage you assign to each path. When the activity is triggered, each customer experiences one of the paths.

An *experiment* is a statistical tool that enables business users to run alternative paths within existing Web activities to determine whether small changes might improve the effectiveness of a Web activity. You can add an experiment to a Web activity to deliver an alternative marketing message. An experiment can include an experimental version of any marketing element, such as a target customer segment, an e-marketing spot used to deliver the marketing message, or the marketing content. In the IBM Management Center for WebSphere Commerce, experiments are created in the Activity Builder, and are represented as optional paths through the Web activity. After the experiment is complete, marketing managers can compare results between the experiment path and the control path using the Statistics tab in the properties view for the experiment.

1.3.7 Targets

A *target* defines which customers experience your marketing activities. When a customer reaches a target in the activity flow, the customer is evaluated against the target criteria. For example, the criteria for a purchase history target might be

that the customer has placed five orders. Targets are typically based on a customer's behavior and segmentation. If you do not include targets in an activity, then the activity applies to all customers.

The following sections discuss targets available with IBM WebSphere Commerce V7 to create Web and dialog activities.

Catalog browsing behavior

This target is used to target customers who have browsed specific parts of your store catalog when shopping on your site over time. For example, if a customer has repeatedly browsed certain categories, that customer is a prime target for advertisements or promotions related to those categories.

External site referral (not for dialog activities)

This target is used to target customers who entered your site from a link on a specific external site.

Social commerce participation

This target is used to target customers who have participated in social commerce on your site a specified number of times.

Customers participate in social commerce when they do any of the following actions on your site:

- ▶ Post a product review or comment, or rate a product
- ▶ Post a blog entry or comment, or rate a blog entry
- ▶ Upload a photo or video

Current page (not for dialog activities)

This target is used to target customers who are currently viewing a specific page of your store. For example, this can be a display page for a specific part of your catalog, or a search results page following a specific keyword search.

Online behavior

This target is used to target customers whose recorded behavior when shopping on your site over time meets certain criteria. The recorded behavior of your customers can provide you with important clues about their interests. You can use these clues to personalize marketing messages. For example:

- ▶ Customers who have searched the site using the keyword “television” in the last week.
- ▶ Customers who have visited store pages with page URLs containing certain data.

Cookie contents (not for dialog activities)

This target is used to target customers whose computers have a cookie from your site that contains certain data (for example, a ZIP code that indicates that the customer lives in a certain geographical area). This is useful for targeting users who have not logged in but have browsed the site, providing useful information.

Day and time

This target is used to specify the days of the week and times of day that an activity is active. If an activity uses the branch element and has more than one path, use this target to set a schedule for each path.

1.3.8 Action

An *action* defines what to do, based on the previous sequence of triggers and (optionally) targets in the activity flow. In a Web activity, an action typically displays something in an e-marketing spot. In a dialog activity, an action can send the customer an e-mail or text message, or add the customer to a customer segment.

The following sections discuss actions that are available with IBM WebSphere Commerce V7 to use in Web and dialog activities.

Recommend Promotions

This action associates marketing contents with promotions. The IBM Management Center for WebSphere Commerce UI provides widgets to enable business users to enter promotion elements and content elements in activity definitions. You can use this action in a Web activity to show the promotions in an e-marketing spot. The advantage of the Recommend Promotions action is that it automatically associates the promotion to the content's click action to take the customer to the appropriate promotion page without manually specifying the click action URLs.

Add to or remove from customer segment

This action adds and removes the customers in a segment based on their online behavior. This action categorizes customers for behavioral marketing initiatives and targets them with more precise and updated promotions. To use this action, check the option when creating a customer segment as shown in Figure 1-5 on page 18. Selecting this option allows the marketing runtime to perform additions to or removals from segment operations for the customer dynamically.

Senior Citizen Female Customer Segment

Save and Close Close

General Properties Customer or Segment Registration Demographics Address Purchase Details Miscellaneous

*Name Senior Citizen Female Customer Segen

Description All female customers with age more then 60 years.
Check the date of birth of female customers periodically and add them to this segment if condition satisfies.

Use marketing activities to add or remove customers ☒ Select this checkbox to dynamically add or remove customers from segments.

Figure 1-5 Customer segment definition window

Display recently viewed

This action captures the set of previously viewed products by a customer in a Web store. The marketing manager can configure the total number of products to record for each customer. These recorded products can be further displayed to the customers so that they can add them to the shopping cart.

Issue coupon

This action issues a coupon to the customers. When the customer checks out, she can redeem this coupon if applicable.

Send message

When using this action, you can send a marketing e-mail or text message to a customer or customers in a segment based on the specified activities. You can use predefined message templates when using this action or you can create a new message template before using this action. There are two types of messages that can be used in activities:

- E-mail message
- Text message (SMS) to a mobile device

Before using this action, configure the e-mail and SMS service in WebSphere Commerce server.

Recommend catalog entry

This action can display one or more catalog entries in an e-marketing spot on a store page, provided you have specified one or more catalog entries to display in the spot.

Recommend content

Use this action to display one or more pieces of content that you want to show to customers on store pages (for example, store advertisements or messages, any graphical or textual marketing message).

Recommend category

This action displays one or more categories in an e-marketing spot on a store page when using a Web activity.

Display Merchandising Association

Use this action in a web activity to display one or more predefined merchandising associations in an e-marketing spot on a store page.

For more details on actions in marketing activities, see the WebSphere Commerce Information Center at the following Web page:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center.doc/concepts/csbactions.htm>

1.4 Working from statistics

Success in a precision marketing campaign depends on the function of both the quantity and quality of the data from which the new promotional insights are formed. There are many reasons why companies have not successfully embraced and implemented technical infrastructures, business processes, and organizational controls necessary to exploit their growing data source. One way to get relief from this situation is to capture and integrate the relevant customers' data, a prerequisite to build a substantial marketing solution that can increase business value. With robust information, a customer database can be the organization's most valuable asset and the data can be consistently updated, cleaned, and maintained similar to any other critical and highly valuable information assets.

To call in the statistician after the experiment is done may be no more than asking him to perform a post-mortem examination; he may be able to say you what the experiment died of.

- R.A. Fisher

In IBM Management Center for WebSphere Commerce, all campaigns, activities, experiments, and statistics are gathered and stored automatically in designated database schema. Business users can view these statistics by opening the

activity or e-marketing spot. The following sections describe the four sets of statistics that are collected and available to the business user.

Activity element counters

Activity element counters provide a simple value indicating the number of customers who have reached the particular trigger, target, or action element. Branches and paths do not have counters associated to them.

e-Marketing spot: Activity statistics

This provides the views, clicks, and click ratio statistics for each e-marketing spot on which a Web activity is running. The Statistics tab becomes visible when the activity has e-marketing spots with associated statistics data. If none of the e-marketing spots have any data (for example, because the activity has not been activated yet), the Statistics tab is not displayed.

Experiment statistics

Statistics gathered for an experiment help the marketing manager decide which experimental path was more successful for a particular test. View and click data are provided in the Statistics tab of the experiment properties view.

Experiment elements have an associated customer count. However, the value is different from the other element counts. The customer count for an experiment indicates the number of unique customers who reached the experiment. If the marketing manager has specified a maximum number of customers for the experiment definition, this count value indicates when the experiment ends.

E-mail activity statistics

In IBM Management Center for WebSphere Commerce, the statistics for e-mail activities show up in the Statistics tab of the properties view, when one creates a new e-mail activity and saves it. Definitions of the statistics that are collected and shown for an e-mail activity are discussed in the following list:

- ▶ **Activity send status**

An e-mail activity has three send status values:

- Not sent
- Scheduled but not sent
- Sent

- ▶ **Number of attempted recipients**

This number indicates that the customer belongs to the defined customer segment, to which the e-mail is to be sent.

► Number of e-mails bounced

This number indicates the number of customers to which the e-mail sent has resulted in a failure of delivery. This might be due to a number of reasons (hard bounce back, soft bounce back).

► Number of e-mails delivered

This indicates the total number of assumed recipients of the e-mail.

► Number of e-mails opened

The shows the total numbers of times the e-mail is opened by the customers associated with this activity.

Note: If the same customer opens the same e-mail twice, this number does not account for a re-open and does not get updated.

► Number of e-mails clicked

This indicates the number of e-mails in which a URL is clicked by customers.

Note: If the e-mail contained two URL links and a customer opened them both, the number gets updated by one. Also, clicking the same link repeatedly does not update this number.

► Percentage of e-mails bounced

This number indicates the ratio of the e-mails bounced back to the total number of attempted recipients.

► Percentage of e-mails delivered

This number indicates the ratio of the e-mails delivered to the total number of attempted recipients.

► Percentage of e-mails opened

This number indicates the ratio of the e-mails opened from the total number of delivered e-mails.

► Percentage of e-mails clicked to delivered

This number indicates the ratio of the delivered e-mails in which the URL is clicked.

► Percentage of e-mails clicked to opened

This number indicates the e-mails clicked to opened ratio.

1.5 Customized precision marketing for your business

Precision marketing works using the personalization ID that uniquely identifies a user and allows WebSphere Commerce to present them with personalized content when the user interacts with the business, using any available channel. To create such personalized promotional content, IBM Management Center for WebSphere Commerce provides a rich set of UI elements and activities templates to the marketing manager to map business needs with the tool. Most of the requirements can be achieved by using standard features, but if a business scenario cannot be met by provided tools, the precision marketing business tool can be always extended and customized to suite requirements. In this book, we have selected a few of the product's requirements and explained them individually in part two to explain the various customization paths possible with IBM Management Center for WebSphere Commerce. These scenarios are generic so you can get an idea on how to proceed with similar customization when required.

1.5.1 Creating a new target wish list

In this scenario, a new target element is created to target those customers who have items in their wish list. The items can be any product, a specific product, or it can belong to a category. This target can be used to send coupons and recommendations to a customer and increase the conversion rate of wish list to an order. See Chapter 4, "Create a new target" on page 53 for more details.

1.5.2 Extending existing customer segment

You can define multiple customers segments with many combinations to deal with your customers individually. IBM Management Center for WebSphere Commerce provides the Customer Segment definition wizard to create such segments. You can always add new conditions or modify the existing conditions to create specific customer segments for your business. In this scenario, a new segment is created to target customers who place an order online, but pick up the shipment at a physical location. These customers can later be targeted for in-store promotions. See Chapter 5, "Extend the customer segment" on page 99 for more details.

1.5.3 Creating a new trigger to monitor customer's interaction

In this scenario, a new trigger is created to monitor a customer's interaction with available channels and trigger the chosen action in an activity flow. This trigger can be integrated with external systems to let WebSphere Commerce runtime know when a customer provides their contact number at the point of sale (at a store, an online order checkout, or calls the customer care executive for a predefined number of times). See the Chapter 6, "Create a new trigger" on page 125 for more details.

1.5.4 Recommending products based on their attributes

You can create Web activities to recommend specific products to the customers based on their shopping cart. We have considered a scenario where a marketing manager wants to recommend products based on defined attributes. For example, she wants to recommend random products to the customer's shopping cart with the attribute defined as Season-Clearance-Sell. With this customization, the marketing manager can recommend more generic products by modifying their attributes. See Chapter 7, "Customize an existing action" on page 155 for more details.

Web activities

This chapter provides an overview of the Web activities and its capabilities provided in the WebSphere Commerce Version 7. We provide some scenarios to showcase the usage of the various targets, actions, and branching available for Web activities.

This chapter contains the following sections:

- ▶ Overview
- ▶ Scenarios

2.1 Overview

Web activities control the marketing information that customers see in e-marketing spots on store pages. A Web activity can communicate relevant information to customers that can increase sales and shopping cart sizes for a retailer. Personalized contents for a Web activity are displayed in the predefined place holders called e-marketing spots on the store pages. A Web activity can also be associated with a campaign program in which each Web activity plays a specific part in the campaign.

Web activities are created by the marketing managers using the Marketing tool in the IBM Management Center for WebSphere Commerce. A Web activity starts with the e-marketing spot trigger and must contain an action, such as Recommend Content or Display Recently Viewed. Optionally, a Web activity can contain targets and branches. The trigger for a Web activity is always an e-marketing spot view event.

For more information about the triggers, targets, actions, and branching available for a Web activity, see Appendix A, “Triggers, targets, actions, and branching” on page 189.

2.2 Scenarios

The following sections discuss two sample scenarios where you can apply the targets, actions, and branching capabilities of the Web Activity Builder to provide personalized information to the customer. The scenarios are explained with detailed steps on how to build Web activities using the IBM Management Center for WebSphere Commerce, including a diagram of the Activity Builder.

2.2.1 Cross promoting products

Karl Stevenson is the marketing manager of the Madisons online store. Karl is happy with the performance of his products online. To increase sales further, Karl decides to give a discount on the products for which he can do a cross-sell. He chooses the products in the Desk Lamp category to be cross-promoted with the products in the Desk category.

Karl has provided the requirements to perform the cross-promotion, which are as follows:

Show a 10% off for all the lamps in the Desk Lamps category if:

- *The customer has already purchased furniture that falls under the Desk category or any of its sub-categories.*
- *The customer has added any furniture into the shopping cart that falls under the Desk category or any of its sub-categories.*

Karl understands that he needs to design the activity on paper before getting started with the Marketing tool. First, he determines the prerequisites for such an activity and comes up with the following list.

- ▶ An e-marketing spot on the home page where the promotional ad needs to be displayed.
- ▶ Ad content that states the promotion details, which need to be designed by his user interface (UI) team.
- ▶ A promotion for the products in the Desk Lamp category.

Karl decides that he wants to show the promotional ad on his category display pages for the Desk Lamps category. Because Karl is confident in what he needs to do, he launches the IBM Management Center for WebSphere Commerce and performs the following steps to create the activity:

1. Create a new e-Marketing Spot using the Marketing tool in IBM Management Center for WebSphere Commerce and name it PromotionDisplaySpot.

For more information about creating e-marketing spots, refer to the Information Center at the following Web page:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/tasks/tsbctespot.htm>

2. Create a new category promotion and name it DeskLamp Promotion using the Promotion tool in the IBM Management Center for WebSphere Commerce.

For more information about creating promotions, refer to the Information Center at the following Web page:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/tasks/tprcreat.htm>

3. Create a new content using the Marketing tool and name it DeskLamp Promotion Ad.

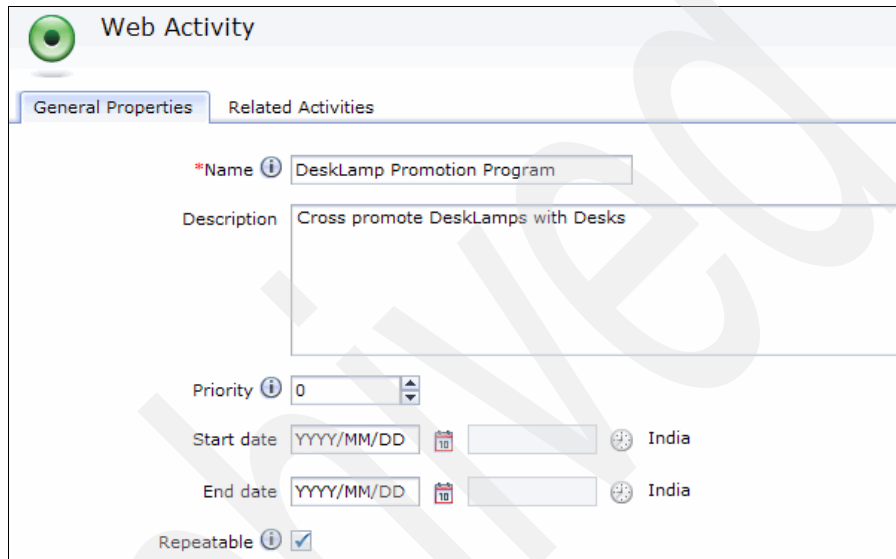
For more information about creating marketing content, refer to the Information Center at the following Web page:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/tasks/tsbmccreate.htm>

4. Create a new Web activity by selecting the **Blank Web activity** option in the Standard templates section. Provide a name and description for the Web activity. See Figure 2-1.

For more information about creating Web activities using the Marketing tool, refer to the Information Center at the following Web page:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/tasks/tsbctwebact.htm>



The screenshot shows the 'Web Activity' dialog box with the 'General Properties' tab selected. The 'Name' field is 'DeskLamp Promotion Program' and the 'Description' is 'Cross promote DeskLamps with Desks'. The 'Priority' is set to 0. The 'Start date' and 'End date' are both set to 'YYYY/MM/DD' with a calendar icon and a time zone dropdown set to 'India'. The 'Repeatable' checkbox is checked.

Figure 2-1 Web activity properties

5. Select the e-marketing spot created for this cross promotion program. The Activity Builder resembles Figure 2-2.

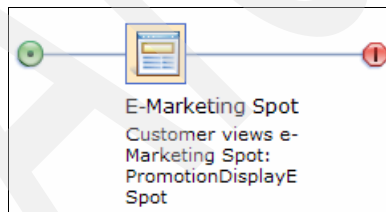


Figure 2-2 The Web activity with the e-marketing spot selected

6. Add the Current Page target that identifies the customers who are (or who are not) on a particular page. See Figure 2-3 for the target properties.

Current Page

*Customer behavior ⓘ Customer is viewing a category ▼

Target customers ⓘ ☒ Who are viewing any of the following categories
☐ Who are not viewing any of the following categories

Desk Lamps Find and Add

*Categories

* Type	* Name	Description
	Desk Lamps	Lamps to illuminate your working environment

1 to 1 of 1

Include subcategories ☐

Figure 2-3 The current page target properties

The Activity Builder resembles Figure 2-4.

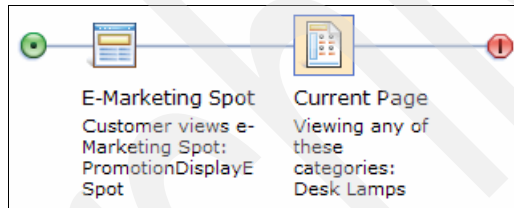


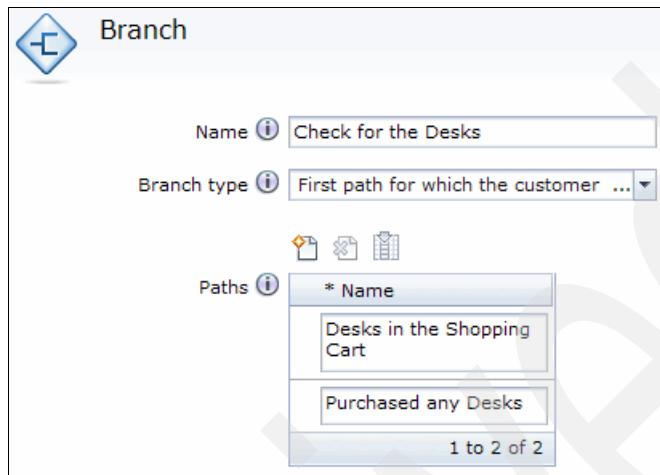
Figure 2-4 The Web activity with the current page target added

Note: Karl requires two paths to account for his two conditions:

If the customer has bought the product or if the customer has added the product in to the shopping cart.

He continues with the steps.

7. Add a branching element from the Branching tools and name the paths. Select **Branch type** as the first path for which the customer qualifies. Save the activity. See Figure 2-5 for the branching properties.



The screenshot shows the 'Branch' configuration window. At the top is a blue diamond icon with a white 'C' and the title 'Branch'. Below this, there are two fields: 'Name' with the value 'Check for the Desks' and 'Branch type' with a dropdown menu showing 'First path for which the customer ...'. Underneath these is a section for 'Paths' with a list box containing two items: 'Desks in the Shopping Cart' and 'Purchased any Desks'. The list box has a header '* Name' and a footer '1 to 2 of 2'. There are also icons for adding, deleting, and moving items.

Figure 2-5 The branch properties

The Activity Builder resembles Figure 2-6.

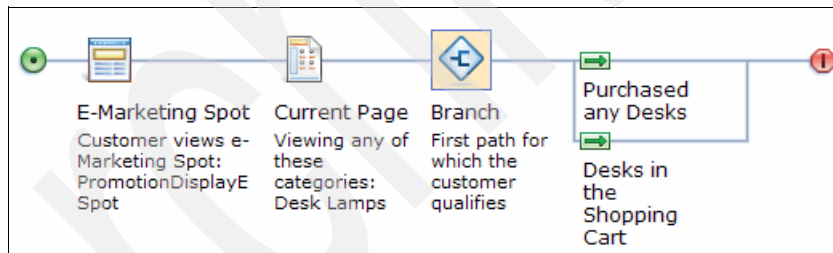


Figure 2-6 The Web activity with the branch added

8. Add the Purchase History target to the first branch (named Purchased any Desks), to filter customers who have not purchased products in the Desks category.

Enter values for the Purchase History target properties. See Figure 2-7 for the target properties.

Purchase History

Purchase history focus: Purchases of catalog entries from ...

Target customers

- ☒ Who have catalog entries from any of the following categories and satisfy the following conditions
- ☐ Who have catalog entries from all of the following categories and satisfy the following conditions
- ☐ Who do not have catalog entries from any of the following categories or who do not satisfy the following conditions

Desks Find and Add

* Categories	* Type	* Name	Description
	Desks	Desks	For work or home use

1 to 1 of 1

Number of catalog entries: At least the following number

*Number: 1

Value of catalog entries: Any value

Time frame: At any time

Figure 2-7 The Purchase history target properties

The Activity Builder resembles Figure 2-8

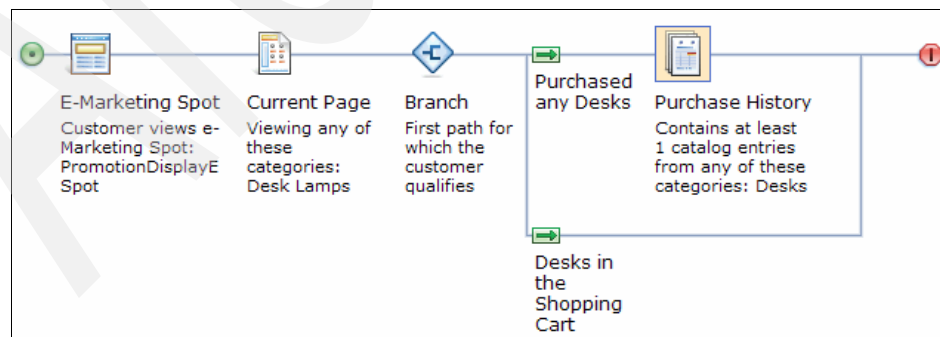


Figure 2-8 The Web activity with the Purchase history target added

9. Add the Shopping Cart target to the second branch (named Desks) in the Shopping Cart, to filter customers who have not added products in the Desks category to the shopping cart. See Figure 2-9 for the target properties.

Shopping Cart

Shopping cart contents: Catalog entries from specific cate ...

Target customers:

- ☒ Who have catalog entries from any of the following categories and satisfy the following conditions
- ☐ Who have catalog entries from all of the following categories and satisfy the following conditions
- ☐ Who do not have catalog entries from any of the following categories or who do not satisfy the following conditions

Find and Add

* Type	* Name	Description
	Desks	For work or home use

1 to 1 of 1

Number of catalog entries: At least the following number

*Number: 1

Value of catalog entries: Any value

Time frame: At any time

Figure 2-9 The Shopping Cart target properties

The Activity Builder resembles Figure 2-10.

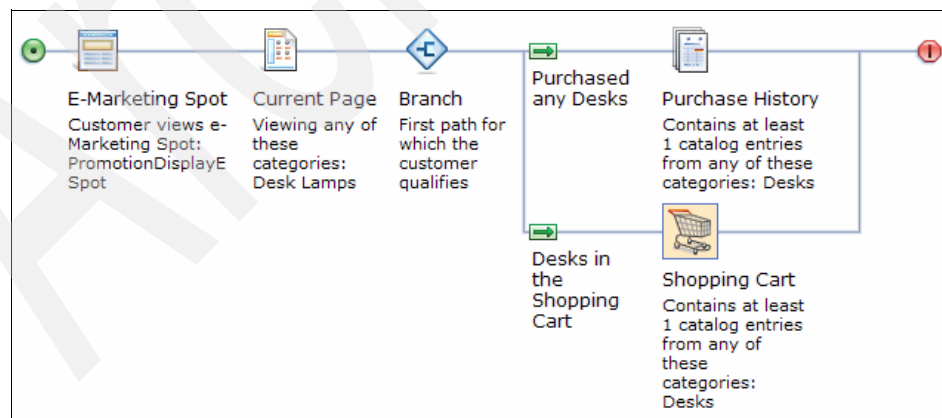


Figure 2-10 The Web activity with the Shopping Cart target added

10. Add the Recommend Promotion action to the Web activity and select the promotion and the content created for this activity. The Activity Builder resembles Figure 2-11.

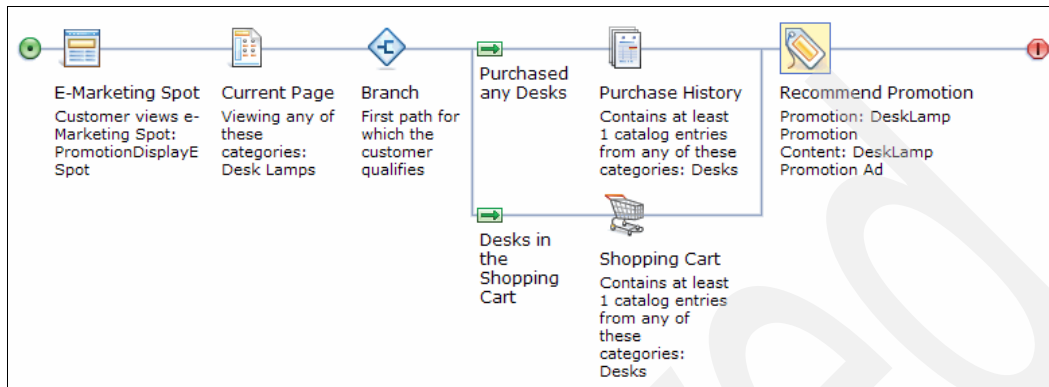


Figure 2-11 The Web activity with the Recommend Promotion action added

11. Save and activate the activity.

Karl contacts his store developer to update the store JSPs for the category display page. The category display includes a snippet to invoke his newly created e-marketing spot. Optionally, Karl can create generic ad content for the customers who do not meet the target criteria set by the Web activity.

2.2.2 Advertising new products

Karl's store has launched a high-end home theater system. Because the system is costly, Karl wants first to target his high income customers. He also thinks it is a good idea to target the customers who have previously shown an interest in home theaters. Karl comes up with the following requirements:

Show the ad for the home theater system on the right side of the store home page for all customers who:

- *Belong to the High-income earners customer segment*
- *Have either visited the category pages for the Home Theater category at least five times in the past or have searched for "Home Theater" in the online store at least once*

Karl finds the required prerequisites to create such an activity and he comes up with the following list:

- ▶ An e-marketing spot on the store home page
- ▶ A customer segment to group his high-income earners
- ▶ The ad content for the home theater system

Karl has asked his UI team to create the ad content. He identifies that the e-marketing spot and the customer segment already exist for his online store. To create this Web activity, Karl opens up the Marketing tool in the IBM Management Center for WebSphere Commerce and performs the following steps:

1. Create a new Web activity by selecting the Blank Web Activity template. Name the activity HomeTheaterAd.
2. Select the e-marketing spot for the activity.
3. Add a branching element to account for the conditions. Three paths need to be created in the branch for using the following targets:
 - Customer Segment target
To find the customers who belong to the High-Income earners group.
 - Catalog Browsing Behavior target
To find the customers who have visited the Home Theater category at least five times.
 - Online Behavior target
To find the customers who have searched for Home Theater at least once.To add a new path in a branch, right-click the branch element and click **Add Path**.
4. Add the Customer Segment target to the first path. Enter the following values:
 - Target customers: **Who are in any of the following customer segments.**
 - Customer segment: **High-Income Earners.**
5. Add the Catalog Browsing Behavior target to the second path. Enter the following values:
 - Target customers: **Who viewed any of the following categories and satisfy the following conditions.**
 - Customer behavior: **Customer viewed a category.**
 - Categories: **Home Theater category.**
 - Frequency: **At least the following number of times**
 - Times: **5**

6. Add the Online Behavior target to the third path. Enter the following values:
 - Online behavior: **Searched keywords.**
 - Target customers: **Who searched for any of the following keywords and satisfy the following conditions**
 - Keyword matching rule: **Search keyword contains one of the following values.**
 - Search keywords: **Home Theater**
7. Add the Recommend Content action after the branch and select the content to be displayed in the e-marketing spot.
8. Save and activate the activity.

The Activity Builder shows the activity. See Figure 2-12.

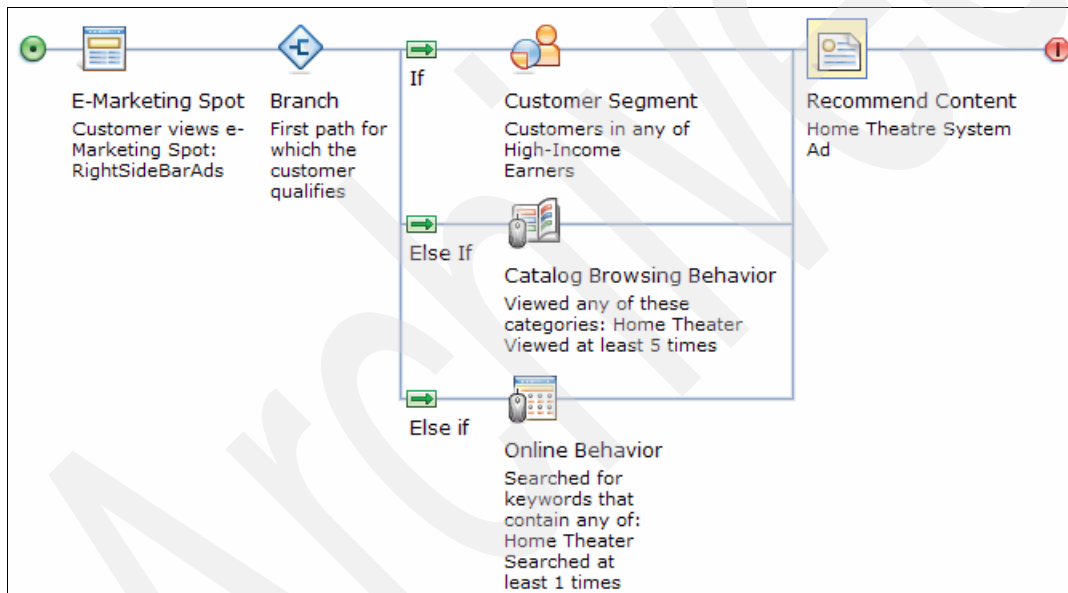


Figure 2-12 The Web activity to recommend the Home-theatre ad for selected customers

After the Web activity is created, you can test the activity following the instructions at the following Web page:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/tasks/tsbtestweb.htm>

Dialog activities

This chapter provides an overview of the dialog activities and its capabilities provided in the WebSphere Commerce Version 7. We provide scenarios to showcase the usage of the various triggers, targets, actions and branching available for dialog activities.

This chapter contains the following sections:

- ▶ Overview
- ▶ Scenarios

3.1 Overview

Dialog activities automate marketing actions based on the specific behavior of your customers. A dialog activity is an interactive dialog with a customer: you wait for the customer to do something, then you respond with a planned marketing action. This dialog can be ongoing, as the customer's behavior on your site changes over time.

Dialog activities are triggered by a specific user-initiated event, such as placing an order or abandoning a shopping cart. You can respond by reaching out to that customer with a relevant and timely marketing action. For example, you can send promotions through e-mail or text message (SMS), issue the customer a coupon, or add the customer to a customer segment.

A dialog activity is composed of triggers, targets, actions, and branching. See Appendix A, "Triggers, targets, actions, and branching" on page 189 for more information about these elements.

3.2 Scenarios

The following scenarios include examples of how to take advantage of the features of dialog activities. Each scenario describes steps on building the dialog activity in IBM Management Center for WebSphere Commerce, including a diagram of what the Activity Builder looks like.

3.2.1 Repeat customers and big spenders

As the marketing manager of the Madisons online store, Karl Stevenson noticed a trend in which after making an initial online purchase, customers were not likely to return for more purchases. He wants to reward registered customers who return to make a second purchase to encourage them to return for another purchase. In addition, he wants to reward registered customers who purchase \$2,000 or more worth of items.

The requirements are as follows:

- ▶ For those customers who complete their second purchase, a 10% coupon is issued for use on the next order.
- ▶ Those customers who spend \$2,000 or more receives a message containing a promotion for free shipping on the next order.
- ▶ If customers meet both criteria, both promotions are received and are eligible to be used together.

Karl lists the prerequisites to create the activity as follows:

- ▶ Create a free shipping promotion to use for customers who spend at least \$2,000 on an order. This promotion is rewarded to customers based on the total order amount.
- ▶ Create a new promotional ad content for the free shipping promotion called Free shipping 2k.
- ▶ Create a new e-Marketing Spot called Free Shipping 2k E-Spot which displays the Free shipping 2k content.
- ▶ Create a new e-mail template called Free Shipping 2k E-mail which inserts content from Free Shipping 2k E-Spot.
- ▶ Create a 10% off coupon for customers who ordered for the second time called 10% Off Next Order.

Note: For more information about how to work with promotions, e-marketing spots, and content in IBM Management Center for WebSphere Commerce, see the Information Center:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center.doc/tasks/tsbemssupert.htm>.

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center.doc/concepts/cprpromotools.htm>.

Karl has created the promotions, the ad content, the e-marketing spot, and the e-mail template using the Marketing tool in the IBM Management Center for WebSphere Commerce. To create the activity he completes the following instructions:

1. Create a new dialog activity selecting the **Blank Dialog Activity** option in the standard template section.
2. Enter the name and description for the newly created dialog activity. See Figure 3-1.

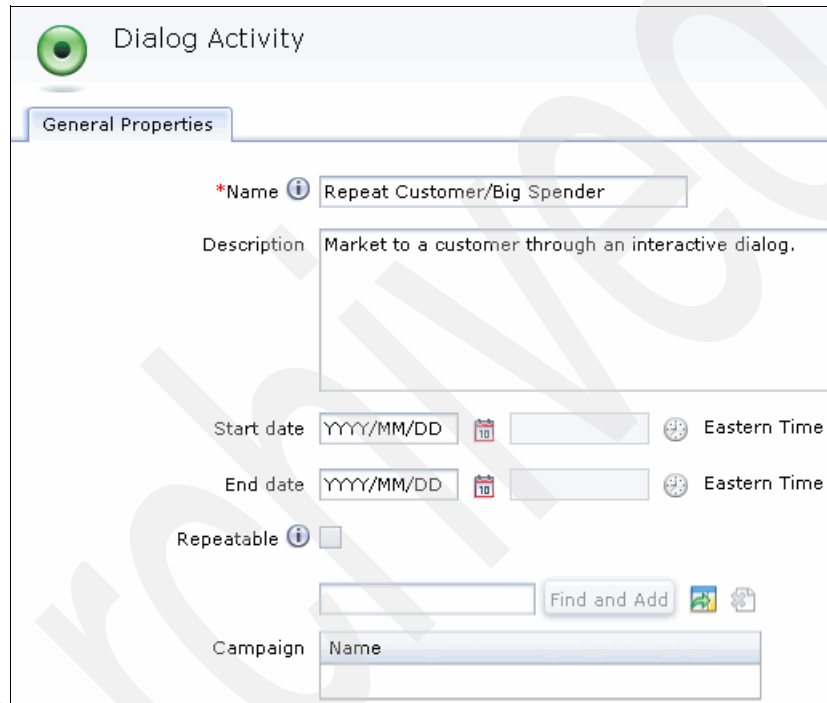


Figure 3-1 New Dialog Activity, Repeat Customer/Big Spender

3. Add a Customer Places Order trigger, as shown in Figure 3-2.

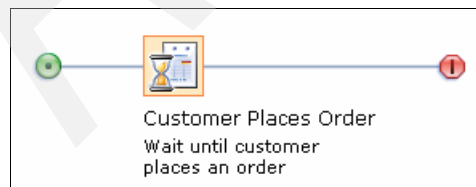



Figure 3-2 Dialog activity with new Customer Places Order trigger


4. Add a Customer Segment target to consider only registered users for this activity. Enter Registered Customers as the value for the Customer segments property, as shown in Figure 3-3.

Customer Segment

Target customers ☒ Target customers in any of the following customer segments
☐ Target customers in all of the following customer segments
☐ Target customers who are not in any of the following customer segments

Find and Add   

*Customer segments

* Type	* Name	Description
	Registered Customers	Customers who have registered with the store

1 to 1 of 1

Figure 3-3 Customer Segment target for Registered Customers

See Figure 3-4 to see what the activity looks like after the customer segment is added.

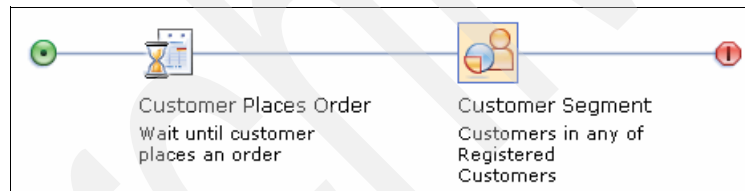
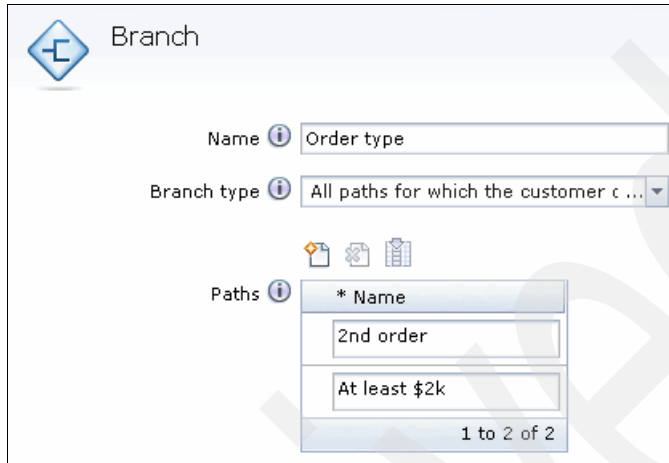


Figure 3-4 Dialog activity with new Customer Segment target

5. Add a branch element to account for the two flows in the scenario. Select the All paths for which the customer qualifies type for the newly added branch. Name the first branch 2nd order and the second branch At least \$2k, as shown in Figure 3-5.



Branch

Name ⓘ Order type

Branch type ⓘ All paths for which the customer c ...

Paths ⓘ

* Name
2nd order
At least \$2k

1 to 2 of 2

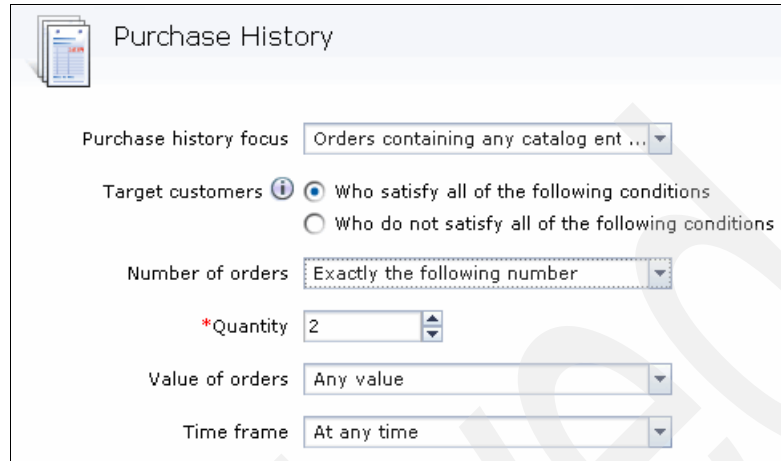
Figure 3-5 Branching for the customer's second order and orders over \$2,000

Figure 3-6 shows what the activity looks like at this point.



Figure 3-6 Dialog activity with new Branch

6. In the first branch, add the Purchase History target to check for customers with exactly two orders, as shown in Figure 3-7.



The image shows a configuration window titled "Purchase History". It contains several settings:

- Purchase history focus:** A dropdown menu set to "Orders containing any catalog ent ...".
- Target customers:** Two radio buttons. The first, "Who satisfy all of the following conditions", is selected. The second is "Who do not satisfy all of the following conditions".
- Number of orders:** A dropdown menu set to "Exactly the following number".
- *Quantity:** A text input field containing the number "2".
- Value of orders:** A dropdown menu set to "Any value".
- Time frame:** A dropdown menu set to "At any time".

Figure 3-7 Purchase History targeting customers with two completed orders

Look at Figure 3-8 to see the progress of the dialog activity.

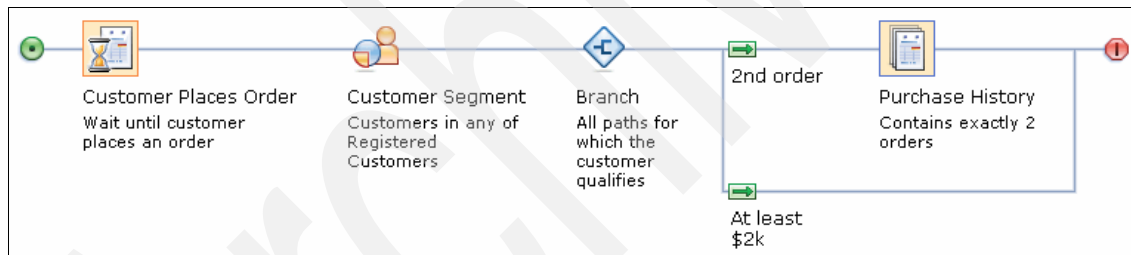


Figure 3-8 Dialog activity with new Purchase History target

7. Add an Issue Coupon action to reward the customer after placing the order. Look at Figure 3-9 on page 44 to see how to configure the coupon.

Issue Coupon

Find and Add

*Coupon promotion ⓘ

* Type	* Administrative Name
10% Off Next Order	

1 to 1 of 1

Figure 3-9 Issue 20% off coupon for customers with two orders

Figure 3-10 shows the dialog activity after adding the Issue Coupon action.

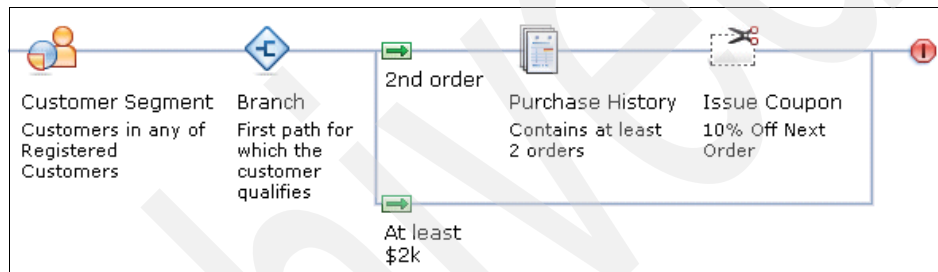


Figure 3-10 Dialog activity with new Issue Coupon action

8. In the second branch, add another Purchase History target to check for orders with a total value of at least \$2,000. Figure 3-11 on page 45 shows how to configure the target.

Purchase History

Purchase history focus: Orders containing any catalog ent ...

Target customers: ☒ Who satisfy all of the following conditions
☐ Who do not satisfy all of the following conditions

Number of orders: Any number

Value of orders: Total value is at least the followin ...

*Amount (USD): 2000

Currency: US Dollar

Time frame: At any time

Figure 3-11 New Purchase History target for orders of at least \$2,000

See Figure 3-12 to see what the dialog activity looks like at this point.

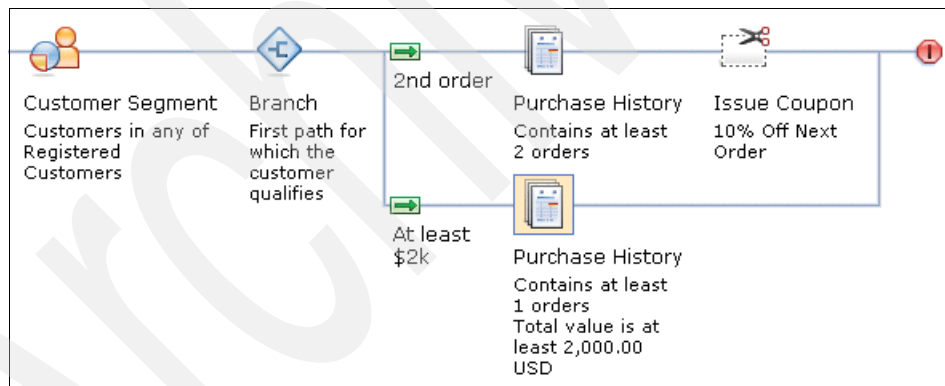
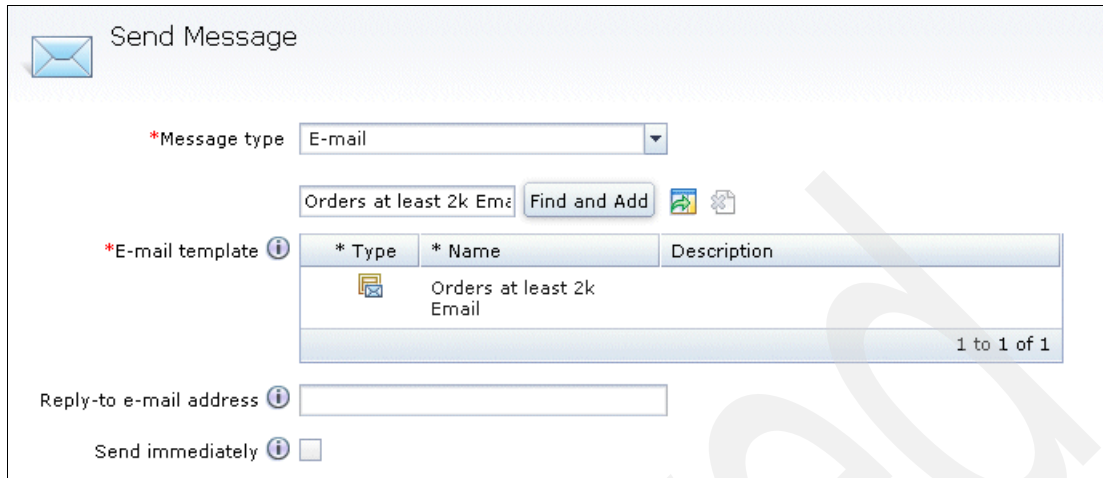


Figure 3-12 Dialog activity with new Purchase History target

9. Add a Send Message action to this branch to send a message with a promotion to the customer. The Send Message action sends the Free Shipping 2k E-mail. The new action is shown in Figure 3-13 on page 46.



Send Message

*Message type: E-mail

Orders at least 2k Email Find and Add

*E-mail template

* Type	* Name	Description
	Orders at least 2k Email	

1 to 1 of 1

Reply-to e-mail address

Send immediately ☐

Figure 3-13 New Send Message activity which e-mail a promotion to the customer

Figure 3-14 shows the dialog activity after adding the Send Message action.

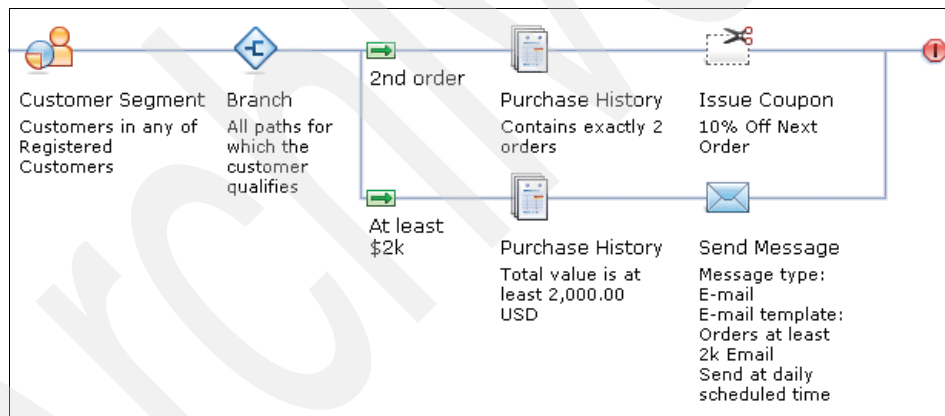


Figure 3-14 Dialog activity with Send Message action

After completing the previous steps, the dialog activity diagram resembles Figure 3-15 on page 47. After the dialog activity is built, it needs to be activated.

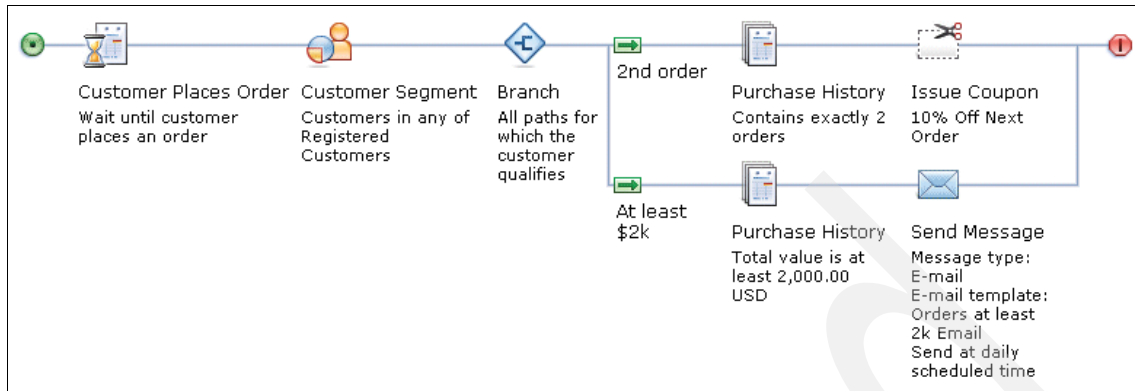


Figure 3-15 Completed dialog activity for Repeat Customers/Big Spenders

3.2.2 Abandoned shopping carts

It has been determined that a high percentage of users browse the store, add items to the shopping cart, but ultimately abandon it. To decrease the amount of stale carts that do not lead to a sale, Karl wants to remind users to return to the store and complete their purchases. Also, Karl wants to issue coupons to customers with high value shopping carts. Customers with carts containing at least \$1,000 worth of merchandise receives coupons, as well as a reminder in the form of a text message (SMS). Customers with carts valued at less than \$1,000 only receive the text message reminder. After a shopping cart has been abandoned for five days, the messages and coupons are sent. Karl determined that the requirements are as follows:

If a cart is abandoned for at least five days containing

- ▶ At least \$2,000 worth of merchandise, a text message reminder and 20% off coupon is sent to the customer
- ▶ At least \$1,000 worth of merchandise, a text message reminder and 10% off coupon is sent to the customer
- ▶ Or less than \$1,000 worth of merchandise, a text message reminder is sent to the customer

Karl lists the prerequisites to create the activity:

- ▶ Create content for the text message reminder for customers with at least \$2,000 worth of items in the abandoned cart, called Abandoned Cart Reminder >=\$2k.
- ▶ Create content for the text message reminder for customers with at least \$1,000 worth of items in the cart, called Abandoned Cart Reminder >=\$1k.

- ▶ Create content for the text message reminder for customers with less than \$1,000 worth of merchandise in the abandoned cart, called Abandoned Cart Reminder.
- ▶ Create a 20% off coupon for customers who have at least \$2,000 worth of items in the abandoned cart, called 20% off Abandoned Cart Coupon.
- ▶ Create a 10% off coupon for customers who have at least \$1,000 worth of items in the abandoned cart, called 10% off Abandoned Cart Coupon.

To build out this scenario, Karl follows the steps listed:

1. Create a new dialog activity and select the Blank Dialog Activity option. Name the dialog activity Abandoned Shopping Carts.
2. Add the Customer Abandons Shopping Cart trigger. Set the minimum number of days the cart has been abandoned to 5 days, and the maximum number of days to 20 days. The maximum number of days property ensures that carts that have been abandoned prior to the activation of this activity are still considered.
3. Add a branch with three branches to cover the three possible scenarios. Set the branch type to First path for which the customer qualifies. Name the first, second and third branches Cart value >= \$2k, Cart value >= \$1k, and Cart value < \$1k respectively.
4. Configure the first branch by completing the following steps.
 - a. Add a Shopping Cart target. Enter the following values:
 - Value of catalog entries: **Total value is at least the following amount.**
 - Amount: **2,000**
 Keep the rest of the defaults.
 - b. Add an Issue Coupon action. Select **20% off Abandoned Cart Coupon** for the coupon promotion.
 - c. Add a Send Message action. Enter the following values:
 - Message type: **Text message (SMS)**
 - Content: *Abandoned Cart Reminder >=\$2k*
5. Configure the second branch by completing the following steps.
 - a. Add a *Shopping Cart* target. Enter the following values:
 - Value of catalog entries: **Total value is at least the following amount.**
 - Amount: **1,000**
 Keep the rest of the defaults.
 - b. Add an Issue Coupon action and select **10% off Abandoned Cart Coupon** for the coupon promotion.

- c. Add a Send Message action. Enter the following values:
 - Message type: **Text message (SMS)**
 - Content: **Abandoned Cart Reminder >= \$1k**
6. Configure the third branch by adding a Send Message action. Enter the following values:
 - Message type **Text Message (SMS)**
 - Content: **Abandoned Cart Reminder**

The dialog activity is now complete. Figure 3-16 is a diagram of the finished activity. Activate and test the new dialog activity to ensure that it functions correctly.

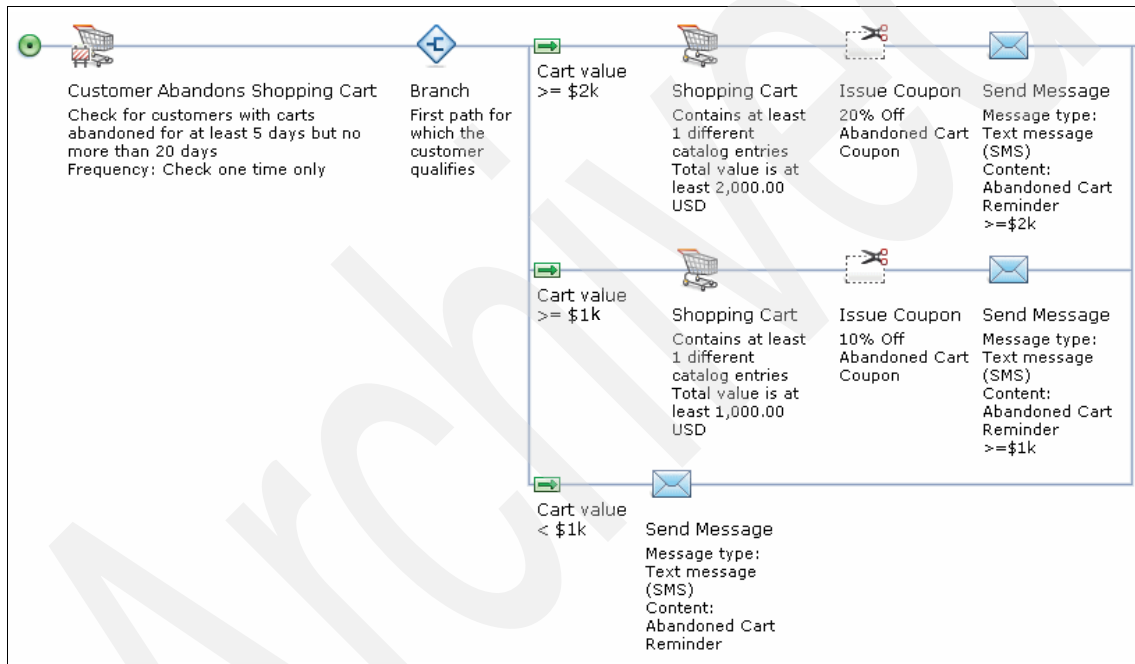


Figure 3-16 Completed dialog activity for the Abandoned Shopping Carts scenario

After the dialog activity is created, test the activity with the instructions at the Information Center:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/tasks/tsbtestdia.htm>



Part 2

Customization scenario

The Marketing tool in IBM Management Center for WebSphere Commerce provides a rich user interface for marketing managers to create and manage various campaigns as needed by the business. If they need to create activities for which there are no widgets provided by IBM Management Center for WebSphere Commerce, they can customize. In this section, we consider some business scenarios that can be implemented through simple customizations. Each scenario addresses a unique requirement that can be mapped with a similar business scenario. We explain the concepts and capabilities of Marketing tool in the following chapters.

Create a new target

Many elements are provided in WebSphere Commerce Version 7 for use in the construction of dialog or Web activities. However, sometimes it is necessary to create new elements if the existing ones are not sufficient for your needs. This chapter describes how to create a new target. The scenario is described in detail, including instructions on how to build out the customization.

This chapter includes the following sections:

- ▶ Overview
- ▶ Design
- ▶ Implementation
- ▶ Test your code

4.1 Overview

Karl conducted a test to determine the percentage of users who eventually purchase catalog entries from their wish list. The percentage was not as high as he wanted it to be, so he decided that targeting users with wish lists can be beneficial to the company's online sales. If users with wish lists can be targeted, coupons or recommendations can be sent out to encourage those users to make purchases. This can increase the number of online orders.

However, currently there is no target for the wish list. If this functionality is to exist on the site, Karl needs to have a new target created. He decided to contact the IT department to create a wish list target.

The requirements are as follows:

Create a new target element to target users with:

- ▶ *Any catalog entry in the wish list*
- ▶ *Any, all, or no specific catalog entries in the wish list*
- ▶ *Any, all, or no catalog entries in specific categories in the wish list*

When completed, the new wish list target looks like Figure 4-1, Figure 4-2 on page 55, and Figure 4-3 on page 55.

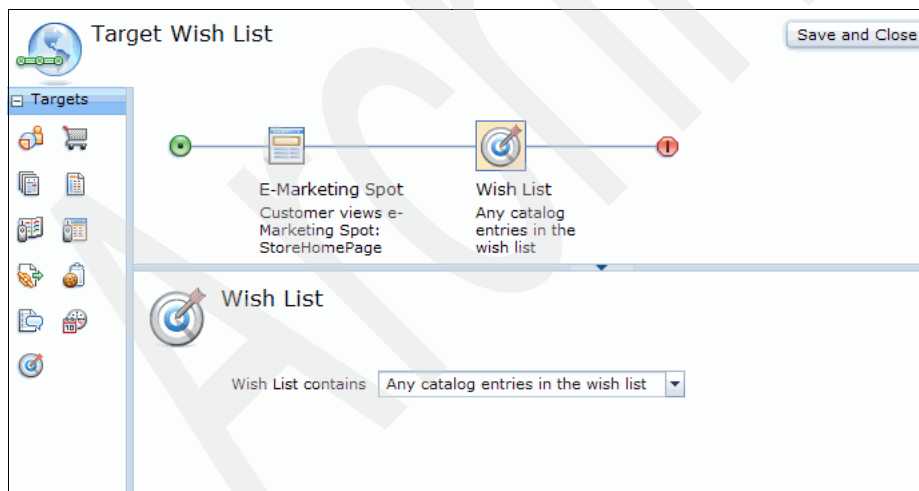


Figure 4-1 New Wish List target for wish lists containing at least one catalog entry

Target Wish List [Save and Close]

Targets

- E-Marketing Spot: Customer views e-Marketing Spot: StoreHomePage
- Wish List: Specific catalog entries in the wish list

Wish List

Wish List contains: Specific catalog entries in the wisl ...

Target customers:

- ☐ Who have any of the following catalog entries in the wish list
- ☐ Who have all of the following catalog entries in the wish list
- ☐ Who do not have any of the following catalog entries in the wish list

[Find and Add] [Add] [Remove] [Reset]

*Catalog entries

* Type	* Code	Name
0 to 0 of 0		

Figure 4-2 New Wish List target for wish lists containing specific catalog entries

Target Wish List [Save and Close] [Close]

Targets

- E-Marketing Spot: Customer views e-Marketing Spot: StoreHomePage
- Wish List: Catalog entries from specific categories in the wish list

Wish List

Wish List contains: Catalog entries from specific cate ...

Target customers:

- ☐ Who have catalog entries from any of the following categories in the wish list
- ☐ Who have catalog entries from all of the following categories in the wish list
- ☐ Who do not have catalog entries from any of the following categories in the wish list

[Find and Add] [Add] [Remove] [Reset]

*Categories

* Type	* Name	Description
0 to 0 of 0		

Figure 4-3 New Wish List target for wish lists containing catalog entries in specific categories

4.2 Design

In this section, we discuss the design of the new target and the steps involved to create it.

The following steps are needed to create a new target:

1. Steps for the server side logic of the target:
 - a. Create the target template definition.
 - b. Create the target task command.
2. Steps for the User Interface (UI):
 - a. Define icons to represent the target.
 - b. Define a resource bundle for the UI text elements.
 - c. Create the object definition for the target.
 - d. Create the properties views for the target.
 - e. Create the summary definition for the target.
 - f. Register the new libraries with the marketing UI.
 - g. Add the target to the element palette in the Activity Builder.
 - h. Create and register serialization JSP files for the new target.

Note: See the Information Center documentation on the IBM Management Center for WebSphere Commerce framework to get a good understanding on marketing customization.

http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center_customization.doc/concepts/csboverview.htm.

For more information about LOB customizations, see IBM Redbooks publication *WebSphere Commerce Line-Of-Business Tooling Customization*, SG24-7619.

4.2.1 Target template definitions

Campaign element template definitions define the campaign element and how it works. Template definitions specify the task command used in the campaign element as well as the element name, arguments needed, how to determine if the event has occurred, and how to evaluate the event. The template definitions are stored in the DMELETEEMPLATE database table as XML fragments. A template definition can be composed of up to three types of XML fragments:

- ▶ Implementation definition
- ▶ Behavior rule
- ▶ Related rule

All campaign elements must have an implementation definition XML fragment. Targets can only be composed of implementation definitions and behavior rules. For this scenario, only the implementation definition is needed.

A campaign element can be composed of one or more template definitions. In our case, we are creating three separate template definitions for each of Karl's desired scenarios. Using three template definitions simplifies the logic each task command needs to perform. The template name is stored in the NAME field of the DMELETEEMPLATE table and maps to the elemTemplateName in the object definition described in 4.2.3, "Object definitions" on page 60. Table 4-1 shows the template names and descriptions used in this scenario.

Table 4-1 Wish list campaign element templates

Template Name	Description
wishListAny	Target users with at least one catalog entry in the wish list
withListProducts	Target users with specific catalog entries in the wish list
wishListCategories	Target users with catalog entries in specific categories in the wish list

The campaign element template definition also defines which parameters are required to be passed to the task command for use in the evaluation. The parameters are name-value pairs that map to the parameters that are configured in the IBM Management Center for WebSphere Commerce. The parameters are defined in the implementation definition XML fragment using the Argument element.

Implementation definition XML fragment for wishListAny

The implementation definition XML for wishListAny only needs to contain the name of the task command used to evaluate the user's wish list.

Implementation definition XML fragment for wishListProducts and wishListCategories

The implementation definition XML fragments for the wishListProducts and wishListCategories templates define the task commands used to perform the evaluations. They must both define parameters to determine which catalog entries or categories need to be checked against the user's wish list in addition to parameters to determine if the wish list needs to contain any, all, or none of the catalog entries or categories listed. The parameters are as shown in Table 4-2.

Table 4-2 Wish list target parameters

Parameter	Value	
anyOrAll	any, all	Used in conjunction with the containsOperator to determine if the wish list must contain any, all, or none of the catalog entries or categories. Used for both wishListProducts and wishListCategories.
containsOperator	=, !=	Used in conjunction with the containsOperator to determine if the wish list must contain any, all, or none of the catalog entries or categories. Used for both wishListProducts and wishListCategories.
catalogEntryIdList	comma delimited string of catalog entry IDs	Only for use in wishListProducts
categoryIdList	comma delimited string of category IDs	Only for use in wishListCategories

For more information regarding campaign element template definitions, see the WebSphere Commerce Version 7.0 Information Center:

http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center_customization.doc/concepts/csbcusteletemp.htm

4.2.2 Task commands

The task commands must correspond with the template definitions defined for the target element. Thus, three task commands are needed for this scenario. The task commands must use the parameters specified in the template definition, if present. The parameters are used in the evaluation to determine if the criteria has been met. Based on the combination of the `anyOrAll` and `containsOperator` parameters shown in Table 4-3, the task command evaluates whether the wish list contains any, all, or none of the catalog entries or categories specified in the `catalogEntryIdList` or `categoryIdList` respectively.

Table 4-3 *anyOrAll and containsOperator values*

<i>anyOrAll</i>	<i>containsOperator</i>	<i>result</i>
all	=	all
any	=	any
any	!=	none

CustomerFilterWishListTargetTaskCmd

This task command corresponds to the `wishListAny` template and returns true if the user has at least one catalog entry in the wish list. No parameters are used because we are checking to verify if the user has catalog entries in the wish list or not.

CustomerFilterWishListProductTargetTaskCmd

This task command corresponds with the `wishListProducts` template. Based on the parameters, it checks if the user's wish list has met the criteria and contains all, any or none of the proper products.

CustomerFilterWishListCategoriesTargetTaskCmd

This task command corresponds with the `wishListCategories` template and checks if the user's wish list has met the criteria based on the parameters entered in the UI and contains catalog entries in all, any, or none of the specified categories.

4.2.3 Object definitions

An object definition is required for the campaign element that describes its characteristics in the Marketing tool within the IBM Management Center for WebSphere Commerce. Every campaign element to be used in a marketing activity extends the `mktFlowElementObjectDefinition` class. More details on the class can be found in the Information Center:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.openlaszlo.api.doc/lzx/commerce/marketing/restricted/activityBuilder/FlowElementObjectDefinition.lzx/mktFlowElementObjectDefinition.html>

Any subclass of the `mktFlowElementObjectDefinition` must contain a dataset with the name template containing the data in XML format. Each XML node has the name of the variable needed for a particular element at runtime defined in the `IMPLXML` column of the `DMELETEMPLATE` table. So, for all variables prefixed with `MARKETING_xyz` in the `IMPLXML` column of the `DMELETEMPLATE` table for a particular flow element, a corresponding `<xyz/>` node is added to the flow element object definition for the use in UI. The default values for the variables can be provided with the template dataset.

Every campaign element definition must have an attribute `elemTemplateName` in the template dataset that maps to the `NAME` column of the `DMELETEMPLATE` table. The wish list target, when dragged to the work area for the first time resembles Figure 4-1 on page 54. The target is initialized to have Any catalog entries in the wish list value for its criteria property. To do this, the `elemTemplateName` can be initialized to the `wishListAny` template in the object definition.

The `wishListAny` template does not have any data to be stored in the `DMELEMENTNVP` because the task command which evaluates the `wishListAny` template looks for a user's wishlist and checks for any catalog entries. The `wishListProducts` template needs to send a list of catalog entry IDs that the user has selected using the UI, the `anyOrAll` attribute, and the `containsOperator` attribute to the marketing engine. The `wishListCategories` template needs to send the list of category IDs that the user has selected using the UI, the `anyOrAll` attribute, and the `containsOperator` attribute.

4.2.4 Properties view

A properties view is required for any variables that the business user needs to provide that are defined in the implementation XML for the campaign element. Every campaign element must extend the `mktFlowElementProperties` class to define the property views. See the Information Center for more info:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center.openlaszlo.api.doc/lzx/commerce/marketing/restricted/activityBuilder/FlowElementProperties.lzx/mktFlowElementProperties.html>

The wish list target prompts the user for the type of the template to be used. Based on the template type selected by the user, the UI changes to capture the appropriate data. For example, if the user chooses the **Specific catalog entries in the wish list** option, the UI changes to allow the user to select a list of catalog entries.

4.2.5 Summary definition

The summary definition is required to display the summary text beneath the campaign element in the Activity Builder work area. The summary text helps the business user to quickly understand how the target is used in the Web or Dialog activity. A summary definition class needs to extend the `mktFlowElementSummary` class. Visit the Information Center for more information:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center.openlaszlo.api.doc/lzx/commerce/marketing/restricted/activityBuilder/FlowElementSummary.lzx/mktFlowElementSummary.html>

For the purpose of this book, we have a simple summary definition that shows the template type the user has currently chosen. The summary definition class needs to implement the `updateSummary` method, which prepares the summary text and set the `summaryText` attribute.

4.2.6 Serialization JSPs for the wish list target

The serialization JSPs are used to retrieve the activity elements from the database tables and display them in the IBM Management Center for WebSphere Commerce UI. If the element object definition does not include any `wcfChildObjectDefinitions` or `wcfReferenceObjectDefinitions`, use the JSP `SerializeActivitygenericElement.jsp` to retrieve the campaign element attributes.

The wish list target has three behaviors. The default behavior is to consider any catalog entries in the wish list. This is mapped to the template name wishListAny. Because this template does not have any child or reference objects, we can use the SerializeActivitygenericElement.jsp. The other templates, wishListProducts and wishListCategories, create references to catalog entries and categories respectively. To retrieve these references we need to create two serialization JSPs.

4.3 Implementation

This section provides the details for implementing the creation of the new target element.

4.3.1 Creating the target template definition

The first step is to create the campaign element template definitions for the new target. As designed, there are three template definitions for the wish list target:

- ▶ wishListAny
- ▶ wishListProducts
- ▶ wishListCategories

Each template definition is added into the DMELETEEMPLATE table in XML format. Example 4-1 shows the SQL statement needed to add the target template definition for users with at least one catalog entry in the wish list.

Example 4-1 Target template definition for wishListAny

```
insert into dmeletemplate
(dmeletemplate_id, dmelementtype_id, name, implxml)
values (10000, 2, 'wishListAny',
'<FlowElementImplementation type="Wish List Contents">
<Implementation invocationType="TaskCommand">
<Class
name="com.madisons.commerce.marketing.commands.elements.CustomerFilterW
ishListTargetTaskCmd">
</Class>
</Implementation>
</FlowElementImplementation>');
```

Example 4-2 shows the SQL statement needed to add the target template definition to target users with specific catalog entries in the wish list.

Example 4-2 Target template definition for wishListProducts

```
insert into dmeletemplate
(dmeletemplate_id, dmelementtype_id, name, implxml)
values (10001, 2, 'wishListProducts',
'<FlowElementImplementation type="Wish List Product Contents">
<Implementation invocationType="TaskCommand">
<Class
name="com.madisons.commerce.marketing.commands.elements.CustomerFilterW
ishListProductTargetTaskCmd">
<Argument name="anyOrAll" value="MARKETING_anyOrAll" />
<Argument name="containsOperator" value="MARKETING_containsOperator" />
<Argument name="catalogEntryIdList"
value="MARKETING_catalogEntryIdList" />
</Class>
</Implementation>
</FlowElementImplementation>');
```

Example 4-3 shows the SQL statement needed to add the target template definition to target users with catalog entries in specific categories in the wish list. Both the wishListProducts and wishListCategories templates define arguments.

Example 4-3 Target template definition for wishListCategories

```
insert into dmeletemplate
(dmeletemplate_id, dmelementtype_id, name, implxml)
values (10002, 2, 'wishListCategories',
'<FlowElementImplementation type="Wish List Category Contents">
<Implementation invocationType="TaskCommand">
<Class
name="com.madisons.commerce.marketing.commands.elements.CustomerFilterW
ishListCategoryTargetTaskCmd">
<Argument name="anyOrAll" value="MARKETING_anyOrAll" />
<Argument name="containsOperator" value="MARKETING_containsOperator" />
<Argument name="categoryIdList" value="MARKETING_categoryIdList" />
</Class>
</Implementation>
</FlowElementImplementation>');
```

4.3.2 Creating the target task commands

After the template definitions have been inserted into the database, the task commands need to be created.

Create the task command targeting users with any catalog entry in the wish list that corresponds to the wishListAny template:

Creating the CustomerFilterWishListTargetTaskCmd interface

Perform the following steps to create the CustomerFilterWishListTargetTaskCmd interface:

1. In Rational® Application Developer for WebSphere Software, click **File** → **New** → **Other**.
2. In the next window, select the Interface wizard and click **Next**.
3. Ensure WebSphereCommerceServerExtensionsLogic/src is entered in the Source folder field.
4. In the Package field, type:
`com.[your company name].commerce.marketing.commands.elements.`
5. Enter CustomerFilterWishListTargetTaskCmd in the name field.
6. Add MarketingCampaignElementTaskCmd as the Extended interface and click **Finish**.
7. In the Java file, set the defaultCommandClassName constant to the name of the CustomerFilterWishListTargetTaskCmdImpl class that is created next.

Example 4-4 shows the CustomerFilterWishListTargetTaskCmd interface.

Example 4-4 CustomerFilterWishListTargetTaskCmd interface

```
package com.madisons.commerce.marketing.commands.elements;

import com.ibm.commerce.marketing.commands.elements
    .MarketingCampaignElementTaskCmd;

public interface CustomerFilterWishListTargetTaskCmd
    extends MarketingCampaignElementTaskCmd {

    public static final String defaultCommandClassName =
        CustomerFilterWishListTargetTaskCmdImpl.class.getName();
}
```

8. Save and close the Java file.

Creating the CustomerFilterWishListTargetTaskCmdImpl class

Perform the following steps to create the CustomerFilterWishListTargetTaskCmdImpl class.

1. Click **File** → **New** → **Other**.
 2. In the next window, select the Class wizard and click **Next**.
 3. Ensure WebSphereCommerceServerExtensionsLogic/src is entered in the Source folder field.
 4. In the Package field, type:
`com.[your company name].commerce.marketing.commands.elements.`
 5. Enter CustomerFilterWishListTargetTaskCmdImpl in the name field.
 6. Browse and enter MarketCampaignElementTaskCmdImpl. Select Superclass and click **OK**.
 7. Add CustomerFilterWishListTargetTaskCmd as the Interface and click **Finish**.
- Example 4-5 shows the class declaration.

Example 4-5 CustomerFilterWishListTargetTaskCmdImpl class declaration

```
public class CustomerFilterWishListTargetTaskCmdImpl
    extends MarketingCampaignElementTaskCmdImpl
    implements CustomerFilterWishListTargetTaskCmd {
    ...
}
```

The task command implements the performExecute method, which performs the actual evaluation for the target. The method evaluates whether or not a user has any catalog entries in the wish list and sets a boolean field with the result. The field is set to true if there are catalog entries in the wish list, and false if there are none. Example 4-6 shows the performExecute method needed to evaluate the user's wish list.

Example 4-6 performExecute method in CustomerFilterWishListTargetTaskCmdImpl

```
public void performExecute() {
    final String METHOD_NAME = "performExecute";

    if (LoggingHelper.isEntryExitTraceEnabled(LOGGER)) {
        LOGGER.entering(CLASSNAME, METHOD_NAME);
    }

    boolean meetsCriteria = false;

    try {
        InterestItemAccessBean iiAB = new InterestItemAccessBean();
    }
}
```

```

        ArrayList interestItemList = new ArrayList();

        // build a list of wish list catalog entries
        Enumeration interestItems = iiAB
            .findAllByUserReferenceNumber(
                getRegisteredMemberIdForPersonalizationId());
        while (interestItems.hasMoreElements()) {
            interestItemList.add(interestItems.nextElement());
        }

        // if at least one item is present in the wish list,
        // the criteria is met
        if (interestItemList != null &&
            interestItemList.size() > 0) {
            meetsCriteria = true;
        }
    } catch (RemoteException e) {
        // TODO: handle exception
    } catch (FinderException e) {
        // TODO: handle exception
    } catch (NamingException e) {
        // TODO: handle exception
    } catch (Exception e) {
        // TODO: handle exception
    }

    // return true or false based on the criteria above
    setReturnValue(meetsCriteria);

    if (LoggingHelper.isEntryExitTraceEnabled(LOGGER)) {
        LOGGER.exiting(CLASSNAME, METHOD_NAME);
    }
}

```

8. Save and close the Java file.

The next step is to create the task command targeting users with specific catalog entries in the wish list. This corresponds to the wishListProducts template.

Create the CustomerFilterWishListProductTargetTaskCmd interface

1. In Rational Application Developer for WebSphere Software, click **File** → **New** → **Other**.
2. In the next window, select the Interface wizard and click **Next**.
3. Ensure WebSphereCommerceServerExtensionsLogic/src is entered in the Source folder field.
4. In the Package field, type:
`com.[your company name].commerce.marketing.commands.elements.`
5. Enter CustomerFilterWishListProductTargetTaskCmd in the name field.
6. Add MarketingCampaignElementTaskCmd as the Extended interface and click **Finish**.
7. In the Java file, set the defaultCommandClassName constant to the name of the CustomerFilterWishListProductTargetTaskCmdImpl class that is created next.

Example 4-7 shows the new CustomerFilterWishListProductTargetTaskCmd interface.

Example 4-7 CustomerFilterWishListProductTargetTaskCmd interface

```
package com.madisons.commerce.marketing.commands.elements;

import com.ibm.commerce.marketing.commands.elements.
    MarketingCampaignElementTaskCmd;

public interface CustomerFilterWishListProductTargetTaskCmd
    extends MarketingCampaignElementTaskCmd {

    public static final String defaultCommandClassName =
        CustomerFilterWishListProductTargetTaskCmdImpl
            .class.getName();
}
```

8. Save and close the file.

Creating the CustomerFilterWishListProductTargetTaskCmdImpl class

Perform the following steps to create the CustomerFilterWishListProductTargetTaskCmdImpl class.

1. Click **File** → **New** → **Other**.
2. In the next window, select the Class wizard and click **Next**.
3. Ensure WebSphereCommerceServerExtensionsLogic/src is entered in the Source folder field.
4. In the Package field, type:
`com.[your company name].commerce.marketing.commands.elements.`
5. Enter CustomerFilterWishListProductTargetTaskCmdImpl in the name field.
6. Browse and enter MarketCampaignElementTaskCmdImpl. Select Superclass and click **OK**.
7. Add CustomerFilterWishListProductTargetTaskCmd as the Interface and click **Finish**.

Example 4-8 shows the class declaration.

Example 4-8 CustomerFilterWishListProductTargetTaskCmdImpl class declaration

```
public class CustomerFilterWishListProductTargetTaskCmdImpl
    extends MarketingCampaignElementTaskCmdImpl
    implements CustomerFilterWishListProductTargetTaskCmd {
    ...
}
```

Example 4-9 shows the constants needed for the task command.

Example 4-9 Constants defined for the CustomerFilterWishListProductTargetTaskCmdImpl class

```
private static final Logger LOGGER = LoggingHelper
    .getLogger(CustomerFilterWishListTargetTaskCmdImpl.class);
private static final String CLASSNAME =
    CustomerFilterWishListProductTargetTaskCmdImpl
        .class.getName();

private static final String ANY_OR_ALL_MISSING =
    "_ERR_ANY_OR_ALL_MISSING";
private static final String CONTAINS_OPERATOR_MISSING =
    "_ERR_CONTAINS_OPERATOR_MISSING";
private static final String CATENTRY_ID_LIST_MISSING =
    "_ERR_CATENTRY_ID_LIST_MISSING";
```

```
// anyOrAll parameter: any, all
private static final String PARAM_ANY_OR_ALL = "anyOrAll";
private static final String ANY = "any";
private static final String ALL = "all";

// containsOperator parameter: =, !=
private static final String PARAM_CONTAINS_OPERATOR =
    "containsOperator";
private static final String EQUALTO = "=";
private static final String NOT_EQUALTO = "!=";

// list of catalog entries
private static final String PARAM_CATENTRY_ID_LIST =
    "catalogEntryIdList";
```

8. The task command must implement the `validateParameters` method, which validates whether the proper parameters have been set in the `DMELEMENTNVP` table. The method checks for the existence of the `catalogEntryIdList`, `anyOrAll`, and `containsOperator` parameters. If any of these parameters are empty or null, an error is logged. Example 4-10 shows the implementation of the `validateParameters` method.

Example 4-10 validateParameters method in the CustomerFilterWishListProductTargetTaskCmdImpl class

```
public List validateParameters(Map elementParameters) {
    final String METHOD_NAME = "validateParameters";

    if (LoggingHelper.isEntryExitTraceEnabled(LOGGER)) {
        LOGGER.entering(CLASSNAME, METHOD_NAME, elementParameters);
    }

    List validationErrors = new ArrayList();

    // log an error if the parameter catalogEntryIdList
    // is not present
    Object catEntries = elementParameters
        .get(PARAM_CATENTRY_ID_LIST);
    List catalogEntryIdList = MarketingUtil
        .parseList(catEntries.toString());
    if (catalogEntryIdList == null
        || catalogEntryIdList.toString().length() == 0) {
        ApplicationError validateError = new ApplicationError(
            ApplicationError.TYPE_GENERIC_ERROR,
```

```

        CATENTRY_ID_LIST_MISSING, null,
        LOGGER.getResourceBundleName());
validationErrors.add(validateError);
    }
    else {
        // validate the business objects exist
        List dataVector =
MarketingUtil.parseList(catalogEntryIdList.toString());
        for (int i = 0; i < dataVector.size(); i++) {
            validateProduct(dataVector.get(i).toString(),
validationErrors);
        }
    }
    // log an error if the parameter anyOrAll is not present
Object anyOrAll = elementParameters.get(PARAM_ANY_OR_ALL);
if (anyOrAll == null || anyOrAll.toString().length() == 0) {
    ApplicationError validateError = new ApplicationError(
        ApplicationError.TYPE_GENERIC_ERROR,
        ANY_OR_ALL_MISSING,
        null, LOGGER.getResourceBundleName());
    validationErrors.add(validateError);
}

// log an error if the parameter containsOperator
// is not present
Object containsOperator = elementParameters.
    get(PARAM_CONTAINS_OPERATOR);
if (containsOperator == null
    || containsOperator.toString().length() == 0) {
    ApplicationError validateError = new ApplicationError(
        ApplicationError.TYPE_GENERIC_ERROR,
        CONTAINS_OPERATOR_MISSING, null, LOGGER
            .getResourceBundleName());
    validationErrors.add(validateError);
}

if (LoggingHelper.isEntryExitTraceEnabled(LOGGER)) {
    LOGGER.exiting(CLASSNAME, METHOD_NAME, elementParameters);
}

return validationErrors;
}

```

9. The task command implements the performExecute method, which performs the actual evaluation for the target. The method evaluates if the user's wish list contains any, all, or none of the specified list of catalog entries.

Example 4-11 shows the performExecute method needed to evaluate the user's wish list.

Example 4-11 performExecute method in CustomerFilterWishListProductTargetTaskCmdImpl

```
public void performExecute() {
    final String METHOD_NAME = "performExecute";

    if (LoggingHelper.isEntryExitTraceEnabled(LOGGER)) {
        LOGGER.entering(CLASSNAME, METHOD_NAME);
    }

    /* Start Section 0 */
    boolean meetsCriteria = false;

    // get the parameters from the DMELEMENTNVP table
    Map parameters = getElementParameters();

    String anyOrAll = (String) parameters.get(PARAM_ANY_OR_ALL);
    String containsOperator = (String) parameters
        .get(PARAM_CONTAINS_OPERATOR);
    String catEntries = (String) parameters
        .get(PARAM_CATENTRY_ID_LIST);
    /* End Section 0 */

    try {
        /* Start Section 1 */
        InterestItemAccessBean iiAB = new InterestItemAccessBean();
        List interestItemList = new ArrayList();

        // retrieve an Enumeration of items in the user's wish list
        Enumeration interestItems = iiAB
            .findAllByUserReferenceNumber(
                getRegisteredMemberIdForPersonalizationId());

        // build a list of wish list catalog entries
        while (interestItems.hasMoreElements()) {
            InterestItemAccessBean interestItem =
                (InterestItemAccessBean) interestItems
                    .nextElement();
            String interestItemId = (String) interestItem
```

```

        .getCatalogEntryReferenceNumber();
        interestItemList.add(interestItemId);
    }
    /* End Section 1 */

    /* Start Section 2 */
    List targetChildList = new ArrayList();

    // parse the comma delimited string of catalog entries
    // into a List
    List catEntriesList = MarketingUtil.parseList(catEntries);

    // build a list of catalog entries
    for (int i = 0; i < catEntriesList.size(); i++) {
        String targetCatEntry = (String) catEntriesList.get(i);

        // retrieve a list of child catalog entries
        // for each catalog entries in the target list
        List temp = ConditionUtil.getChildCatentries(getStoreId(),
            (String) catEntriesList.get(i));

        // add each catalog entry id to the list
        for (int j = 0; j < temp.size(); j++) {
            targetChildList.add(temp.get(j));
        }
    }
    /* End Section 2*/

    /* Start Section 3*/
    // loop through each catalog entry in the target list
    // and compare to the items in the wish list
    checkCatEntriesLabel:
    for (int i = 0; i < targetChildList.size(); i++) {
        String targetCatEntry = (String) targetChildList.get(i);

        if (anyOrAll.equals(ANY) &&
            containsOperator.equals(EQUALTO)) {
            // if at least one target catalog entry is
            // present in the wish list, the criteria is met
            if (interestItemList.contains(targetCatEntry)) {
                meetsCriteria = true;
                break checkCatEntriesLabel;
            }
        } else if (anyOrAll.equals(ALL) &&
            containsOperator.equals(EQUALTO)) {

```



```

        // all target catalog entries must be presented in
        // the wish list for the criteria to be met
        if (!interestItemList.contains(targetCatEntry)) {
            meetsCriteria = false;
            break checkCatEntriesLabel;
        } else {
            meetsCriteria = true;
        }
    } else {
        // no target catalog entries should be presented in the
        // wish list for the criteria to be met
        if (interestItemList.contains(targetCatEntry)) {
            meetsCriteria = false;
            break checkCatEntriesLabel;
        } else {
            meetsCriteria = true;
        }
    }
}

} catch (RemoteException e) {
    // TODO: handle exception
} catch (FinderException e) {
    // TODO: handle exception
} catch (NamingException e) {
    // TODO: handle exception
} catch (CreateException e) {
    // TODO: handle exception
} catch (Exception e) {
    // TODO: handle exception
}

/* End Section 3*/

// return true or false based on the criteria above
setReturnValue(meetsCriteria);

if (LoggingHelper.isEntryExitTraceEnabled(LOGGER)) {
    LOGGER.exiting(CLASSNAME, METHOD_NAME);
}
}
}

```

In Section 0, the parameters that are needed to evaluate the criteria are initialized.

In Section 1, the list of catalog entries from the user's wish list are retrieved and a list of catalog entry IDs is created and used to compare against the catalog entry ID list.

In Section 2, the comma delimited catalog entry parameter string is parsed into a list. The list of IDs is then looped through to get the child catalog entry that the ID is associated with. This is in case the list contains a product and not an item (SKU), because only items are in the wish list.

Section 3 is where most of the logic takes place. The child catalog entry ID list is looped through and each ID is compared to the IDs of the catalog entries in the wish list. According to Table 4-3 on page 59, the values of the anyOrAll and containsOperator parameters determine if the wish list needs to contain any, all or none of the targeted catalog entries.

If the result is all, as soon as a catalog entry ID does not match, the criteria is not met and the task command returns false.

If the result is any, as soon as there is a match, the criteria is met and it returns true.

If the result is none, as soon as there is a match, the criteria is not met and the task command returns false.

4.3.3 Creating validation messages

After the task commands have been created, server side validation messages need to be setup.

Creating a new messagekey class

Perform the following steps to create a new messagekey class:

1. In Rational Application Developer for WebSphere Software, click **File** → **New** → **Other**.
2. In the next window, select the Class wizard and click **Next**.
3. Ensure WebSphereCommerceServerExtensionsLogic/src is entered in the Source folder field.
4. In the Package field, type:
`com.[your company name].commerce.marketing.logging.`
5. Enter WishListMessageKeys in the name field and click **Finish**.

Example 4-12 on page 75 shows the newly created class and the constants used as the message keys.

Example 4-12 *WishListMessageKeys class*

```
public class WishListMessageKeys {
    public final static String
_APP_ELEMENT_TARGET_CUST_NOT_FOUND="_APP_ELEMENT_TARGET_CUST_NOT_FOUND "
public final static String
_APP_ELEMENT_WISHLIST_CATENTRY_NOT_FOUND="_APP_ELEMENT_WISHLIST_CATENTRY_NOT_FO
UND"
public final static String
_APP_ELEMENT_WISHLIST_CATEGORY_NOT_FOUND="_APP_ELEMENT_WISHLIST_CATEGORY_NOT_FO
UND"
}
```

6. Save and close the file.

Creating a properties file

Perform the following steps to create a properties file:

1. Click **File** → **New** → **Other**.
2. In the next window, select the File wizard and click **Next**.
3. In the Enter or select the parent folder field window, type:
WebSphereCommerceServerExtensionsLogic/src/com/[your company name]/commerce/marketing/logging/properties/
4. In the File name field, enter WcMarketingMessages.properties and click **Finish**.
5. Click **File** → **New** → **Other**.
6. In the next window, select the File wizard and click **Next**.
7. In the Enter or select the parent folder field, type
WebSphereCommerceServerExtensionsLogic/src/com/[your company name]/commerce/marketing/logging/properties/.
8. In the File name field, enter WcMarketingMessages_en_US.properties, and click **Finish**.
9. In each of the files, enter the text from Example 4-13.

Example 4-13 *WcMarketingMessages.properties and WcMarketingMessages_en_US.properties*

```
_APP_ELEMENT_TARGET_CUST_NOT_FOUND = Specify the target customers.
_APP_ELEMENT_WISHLIST_CATENTRY_NOT_FOUND = Specify a catalog entry in the
Catalog entries table.
_APP_ELEMENT_WISHLIST_CATEGORY_NOT_FOUND = Specify at least one category in the
Category table.
```

10. Save and close the file.

The next step is to create the task command targeting users with catalog entries in specific categories in the wish list. This corresponds to the *wishListCategories* template.

Creating the CustomerFilterWishListCategoryTargetTaskCmd interface

Perform the following steps to create the CustomerFilterWishListCategoryTargetTaskCmd interface.

1. In Rational Application Developer for WebSphere Software, click **File** → **New** → **Other**.
2. In the next window, select the Interface wizard and click **Next**.
3. Ensure WebSphereCommerceServerExtensionsLogic/src is entered in the Source folder field.
4. In the Package field, type:
`com.[your company name].commerce.marketing.commands.elements`
5. Enter CustomerFilterWishListCategoryTargetTaskCmd in the name field.
6. Add MarketingCampaignElementTaskCmd as the Extended interface and click **Finish**.
7. In the Java file, set the defaultCommandClassName constant to the name of the CustomerFilterWishListCategoryTargetTaskCmdImpl class that is created next.

Example 4-14 shows the new CustomerFilterWishListCategoryTargetTaskCmd interface.

Example 4-14 CustomerFilterWishListCategoryTargetTaskCmd interface

```
package com.madisons.commerce.marketing.commands.elements;

import com.ibm.commerce.marketing.commands.elements
    .MarketingCampaignElementTaskCmd;

public interface CustomerFilterWishListCategoryTargetTaskCmd
    extends MarketingCampaignElementTaskCmd {

    public static final String defaultCommandClassName =
        CustomerFilterWishListCategoryTargetTaskCmdImpl
            .class.getName();

}
```

8. Save and close the file.

Creating CustomerFilterWishListCategoryTargetTaskCmdImpl class

Perform the following steps to create the CustomerFilterWishListCategoryTargetTaskCmdImpl class.

1. Click **File** → **New** → **Other**.
2. In the next window, select the Class wizard and click **Next**.
3. Ensure WebSphereCommerceServerExtensionsLogic/src is entered in the Source folder field.
4. In the Package field, type:
`com.[your company name].commerce.marketing.commands.elements`
5. Enter CustomerFilterWishListCategoryTargetTaskCmdImpl in the name field.
6. Browse and enter MarketCampaignElementTaskCmdImpl. Select Superclass and click **OK**.
7. Add CustomerFilterWishListCategoryTargetTaskCmd as the interface and click **Finish**.

Example 4-15 shows the class declaration.

Example 4-15 CustomerFilterWishListCategoryTargetTaskCmdImpl class declaration

```
public class CustomerFilterWishListCategoryTargetTaskCmdImpl
    extends MarketingCampaignElementTaskCmdImpl
    implements CustomerFilterWishListCategoryTargetTaskCmd {
    ...
}
```

8. Define the required constants for the task command. This is similar to the snippet shown in Example 4-9 on page 68, except that the constant referencing catalogEntryIdList must be changed to see the categoryIdList.

The task command implements the validateParameters method, which validates whether the categoryIdList, anyOrAll, and containsOperator parameters have been set. This is similar to the snippet shown in Example 4-10 on page 69.

The task command implements the performExecute method, which performs the actual evaluation for the target. The method evaluates if the user's wish list contains catalog entries in any, all, or none of the specified list of categories. Example 4-16 on page 78 shows the performExecute method needed to evaluate the user's wish list.

Example 4-16 *performExecute* method in
CustomerFilterWishListCategoryTargetTaskCmdImpl

```
public void performExecute() {
    final String METHOD_NAME = "performExecute";

    if (LoggingHelper.isEntryExitTraceEnabled(LOGGER)) {
        LOGGER.entering(CLASSNAME, METHOD_NAME);
    }

    /* Start Section 0 */
    boolean meetsCriteria = false;

    // get the parameters from the DMELEMENTNVP table
    Map parameters = getElementParameters();

    String anyOrAll = (String) parameters.get(PARAM_ANY_OR_ALL);
    String containsOperator = (String) parameters
        .get(PARAM_CONTAINS_OPERATOR);
    String categories = (String)
        parameters.get(PARAM_CATEGORY_ID_LIST);

    // parse the comma delimited string of categories into a List
    List categoriesList = MarketingUtil.parseList(categories);
    /* End Section 0 */

    /* Start Section 1 */
    try {
        InterestItemAccessBean iiAB = new InterestItemAccessBean();
        List interestItemCatList = new ArrayList();

        // retrieve an Enumeration of items in the user's wish list
        Enumeration interestItems = iiAB
            .findAllByUserReferenceNumber(
                getRegisteredMemberIdForPersonalizationId());

        // build a list of wish list item categories
        while (interestItems.hasMoreElements()) {
            InterestItemAccessBean interestItem =
                (InterestItemAccessBean) interestItems
                    .nextElement();
            String interestItemId = interestItem
                .getCatalogEntryReferenceNumber();

            // retrieve a list of parent categories for the
```

```

// current wish list item
List tempList = MarketingUtilCatalog.getParentCategories(
    Long.valueOf(interestItemId), categoriesList);

// add each category id to the list
for (int i = 0; i < tempList.size(); i++) {
    interestItemCatList.add((String) tempList.get(i));
}
}
/* End Section 1 */

/* Start Section 2 */
// loop through each category in the target list and
// compare to the items in the wish list
checkCategoriesLabel:
for (int i = 0; i < categoriesList.size(); i++) {
    String targetCategory = (String) categoriesList.get(i);
    if (anyOrAll.equals(ANY) &&
        containsOperator.equals(EQUALTO)) {
        // if at least one target category is represented in the
        // wish list, the criteria is met
        if (interestItemCatList.contains(targetCategory)) {
            meetsCriteria = true;
            break checkCategoriesLabel;
        }
    } else if (anyOrAll.equals(ALL)
        && containsOperator.equals(EQUALTO)) {
        // all target categories must be represented in the wish
        // list for the criteria to be met
        if (!interestItemCatList.contains(targetCategory)) {
            meetsCriteria = false;
            break checkCategoriesLabel;
        } else {
            meetsCriteria = true;
        }
    } else {
        // no target categories should be represented in the
        // wish list for the criteria to be met
        if (interestItemCatList.contains(targetCategory)) {
            meetsCriteria = false;
            break checkCategoriesLabel;
        } else {
            meetsCriteria = true;
        }
    }
}
}

```

```

    }
} catch (RemoteException e) {
    // TODO: handle exception
} catch (FinderException e) {
    // TODO: handle exception
} catch (NamingException e) {
    // TODO: handle exception
} catch (Exception e) {
    // TODO: handle exception
}
}
/* End Section 2 */

// return true or false based on the criteria above
setReturnValue(meetsCriteria);

if (LoggingHelper.isEntryExitTraceEnabled(LOGGER)) {
    LOGGER.exiting(CLASSNAME, METHOD_NAME);
}
}
}

```

In Section 0, the parameters are initialized.

In Section 1, the catalog entries from the user's wish list are evaluated to determine the parent category to which it belongs. These categories are added to the variable, `interestItemCatList`, to compare against the list of target category IDs.

In Section 2, the `categoriesList` is looped through and each element is compared to the categories in the variable, `interestItemCatList`. Table 4-3 on page 59 determines how to evaluate the criteria based on the combination of parameters.

9. Save and close the Java file.

Note: We have only presented one approach. There are other design options for this customization, however. Being that the scenario is similar to the Shopping Cart or purchase history targets, those targets can be extended to make use of inherent functionality. We chose to create new task commands because this scenario was not complex.

This completes all the steps to build the server side logic to implement a new target.

4.3.4 Defining the icons to represent the new target

After we have the server side logic ready for the new target, the next step is to implement the user interface changes in the IBM Management Center for WebSphere Commerce tooling.

The first step is to define the icons for the palette, work area, and properties view of the Activity Builder for the new wish list target. Generic icons for triggers, targets and actions are shipped with WebSphere Commerce to be used in customizations. Three sets of the icons are provided to be displayed in the palette, work area and the properties views. They can be found under the paths listed in Table 4-4.

Table 4-4 The location to the generic icons

Path	Used in	Resource name
LOBTools/WebContent/WEB-INF/src/lzx/commerce/marketing/resources/pal/trigger.png	Palette	genericTriggerPalettelcon
LOBTools/WebContent/WEB-INF/src/lzx/commerce/marketing/resources/pal/target.png	Palette	genericTargetPalettelcon
LOBTools/WebContent/WEB-INF/src/lzx/commerce/marketing/resources/pal/action.png	Palette	genericActionPalettelcon
LOBTools/WebContent/WEB-INF/src/lzx/commerce/marketing/resources/dgm/trigger.png	Work area	genericTriggerIcon
LOBTools/WebContent/WEB-INF/src/lzx/commerce/marketing/resources/dgm/target.png	Work area	genericTargetIcon
LOBTools/WebContent/WEB-INF/src/lzx/commerce/marketing/resources/dgm/action.png	Work area	genericActionIcon
LOBTools/WebContent/WEB-INF/src/lzx/commerce/marketing/resources/hdr/trigger.png	Properties views	genericTriggerHeaderIcon
LOBTools/WebContent/WEB-INF/src/lzx/commerce/marketing/resources/hdr/target.png	Properties views	genericTargetHeaderIcon
LOBTools/WebContent/WEB-INF/src/lzx/commerce/marketing/resources/hdr/action.png	Properties views	genericActionHeaderIcon

Karl's IT developer uses the generic icons for target to test his new wish list target implementation. The generic icons are already defined in the LOBTools/WebContent/WEB-INF/src/lzx/commerce/marketing/restricted/MarketingManagementResources.lzx file.

4.3.5 Defining a resource bundle and properties file for UI text

Perform the following steps to define a resource bundle for the text to be displayed in the UI:

1. Create a new Java package inside the LOBTools/src named:
`com.<your_company_name>.marketing.client.lobtools.properties`

Right-click **Java Resources: src** inside the LOBTools project. Click **New** → **Package**. Figure 4-4 shows the new package structure.

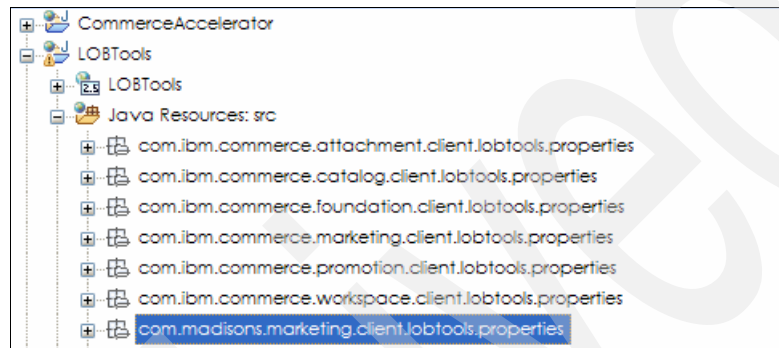


Figure 4-4 The new package to hold the wish list target resource bundle

2. Create a new properties files to hold the text strings. Right-click the new package and click **New** → **File** and enter the name `MarketingLOB.properties`.
3. Enter the snippet in Example 4-17 into the properties file.

Example 4-17 The properties file contents

```
wishListName=Wish List
wishListAny=Any catalog entries in the wish list
wishListProducts=Specific catalog entries in the wish list
wishListCategories=Catalog entries from specific categories in the wish list
wishListCriteria=Wish List contains
anyProducts=Who have any of the following catalog entries in the wish list
allProducts=Who have all of the following catalog entries in the wish list
notAnyProducts=Who do not have any of the following catalog entries in the wish list
anyCategories=Who have catalog entries from any of the following categories in the wish list
```

allCategories=Who have catalog entries from all of the following categories in the wish list
notAnyCategories=Who do not have catalog entries from any of the following categories in the wish list

4. Save the file.
5. Navigate to the following directory:
LOBTools/WebContent/WEB-INF/src/lzx/<your_company_name>/marketing
Create an OpenLaszlo file named MarketingResourceBundle.lzx.
6. Enter the snippet in Example 4-18 in the newly created OpenLaszlo file.

Example 4-18 The resource bundle key definitions

```
<library>
  <class name="extMarketingResourceBundle"
    extends="wcfResourceBundle"
    baseName="com.madisons.marketing.client.lobtools
      .properties.MarketingLOB">

    <wcfResourceBundleKey name="wishListName" />
    <wcfResourceBundleKey name="wishListAny" />
    <wcfResourceBundleKey name="wishListProducts" />
    <wcfResourceBundleKey name="wishListCategories" />
    <wcfResourceBundleKey name="wishListCriteria" />

    <wcfResourceBundleKey name="anyProducts" />
    <wcfResourceBundleKey name="allProducts" />
    <wcfResourceBundleKey name="notAnyProducts" />
    <wcfResourceBundleKey name="anyCategories" />
    <wcfResourceBundleKey name="allCategories" />
    <wcfResourceBundleKey name="notAnyCategories" />

  </class>

  <extMarketingResourceBundle id="extMarketingResources" />
</library>
```

7. Save the file.

The resource bundle id `extMarketingResources` can be used in the object definition, properties views, and the summary class to display the text strings needed for the wish list target.

4.3.6 Creating the object definition for the target

To create the object definition for the wish list target, perform the following steps:

1. Create an OpenLaszlo file named `WishListFlowElementObjectDefinition.lzx` in the `LOBTools/WebContent/WEB-INF/src/lzx/<your_company_name>/marketing/objectDefinitions/activityBuilder/` directory. Use the snippet in Example 4-19.

Example 4-19 The object definition for the Wish list target

```
<library>
<class name="extWishListElementObject"
  extends="mktFlowElementObjectDefinition"
  objectType="wishList"
  displayName="${extMarketingResources.wishListName.string}"
  headerIcon="genericTargetHeaderIcon"
  flowIcon="genericTargetIcon"
  paletteIcon="genericTargetPaletteIcon"
  propertiesClass="extWishListProperties"
  summaryClass="extWishListSummary"
  elemTemplateType="Target">

  <mktFlowElementCreateService>
    <wcfServiceParam
      name="PROPERTY_categoryIdList"
      objectPath="ChildCatalogGroup/CatalogGroupAlias"
      propertyName="catgroupId"
      optional="true"/>
    <wcfServiceParam
      name="PROPERTY_catalogEntryIdList"
      objectPath="ChildCatentry/CatalogEntry"
      propertyName="catentryId"
      optional="true"/>
  </mktFlowElementCreateService>

  <mktFlowElementUpdateService>
    <wcfServiceParam
      name="PROPERTY_categoryIdList"
      objectPath="ChildCatalogGroup/CatalogGroupAlias"
      propertyName="catgroupId"
      optional="true"/>
    <wcfServiceParam
      name="PROPERTY_catalogEntryIdList"
      objectPath="ChildCatentry/CatalogEntry"
```

```

        propertyName="catentryId"
        optional="true"/>
</mktFlowElementUpdateService>

<dataset name="template">
    <elemTemplateName>wishListAny</elemTemplateName>
</dataset>

<wcfReferenceObjectDefinition
    objectGroups="ChildCatalogGroup"
    objectType="ChildCatalogGroup"
    referencedTypes="CatalogGroup,SalesCatalogGroup"
    idProperty="childCatalogGroupId"
    allowDuplicates="false"
    derivedId="true"
    allowedParentTypes="WebActivity,
        WebActivityTemplate,
        DialogActivity,
        DialogActivityTemplate"/>

<wcfReferenceObjectDefinition
    objectGroups="ChildCatalogGroup"
    objectType="ChildInheritedCatalogGroup"
    referencedTypes="InheritedCatalogGroup,
        InheritedSalesCatalogGroup"
    idProperty="childCatalogGroupId"
    allowDuplicates="false"
    derivedId="true"/>

<wcfReferenceObjectDefinition
    objectType="ChildCatentry"
    objectGroups="ChildCatentry"
    referencedTypes="Product,Bundle,
        CatalogGroupSKU,ProductSKU,Kit"
    idProperty="childCatentryId"
    allowDuplicates="false"
    derivedId="true"
    allowedParentTypes="WebActivity,
        WebActivityTemplate,
        DialogActivity,
        DialogActivityTemplate"/>

<wcfReferenceObjectDefinition
    objectType="ChildInheritedCatentry"
    objectGroups="ChildCatentry"

```

```

        referencedTypes="InheritedProduct,
            InheritedBundle,
            InheritedCatalogGroupSKU,
            InheritedProductSKU,
            InheritedKit"
        idProperty="childCatentryId"
        allowDuplicates="false"
        derivedId="true"/>

<wcfPropertyDefinition propertyName="elemTemplateName">
    <wcfPropertyValue
        displayName="{extMarketingResources
            .wishlistAny.string}"
        value="wishlistAny"/>
    <wcfPropertyValue
        displayName="{extMarketingResources
            .wishlistProducts.string}"
        value="wishlistProducts"/>
    <wcfPropertyValue
        displayName="{extMarketingResources
            .wishlistCategories.string}"
        value="wishlistCategories"/>
    </wcfPropertyDefinition>
</class>
</library>

```

The `propertiesClass` and `summaryClass` mentioned in the object definition is created later.

2. Save the file.
3. Register the wish list target object definition in its parent definition. To do this:
 - a. Open the `LOBTools/WebContent/WEB-INF/src/lzx/commerce/marketing/objectDefinitions/activityBuilder/FlowPathElementObjectDefinition.lzx` file.
 - b. Look for the `mktBaseFlowPathElementObject` class definition, and add the `extWishlistElementObject` object definition.

See the snippet in Example 4-20 on page 87.

Example 4-20 Registering the wish list target object in the base flow element object definition

```
.....
<class name="mktBaseFlowPathElementObject"
      extends="wcfChildObjectDefinition"
      displayName="{mktMarketingResources.path.string}"
      isBaseDefinition="true">
  .....
  <extWishlistElementObject />
  .....
</class>
.....
```

4. Save the file.

4.3.7 Creating the properties view for the target

To create the properties view for the wish list target, perform the following steps:

1. Create an OpenLaszlo file named `WishListPropertiesView.lzx` inside the `LOBTools/WebContent/WEB-INF/src/lzx/<your_company_name>/marketing/propertiesViews/activityBuilder` directory. The snippet in Example 4-21 shows the properties definition for the wish list target.

Example 4-21 The wish list properties view definition

```
<library>
  <class name="extWishListProperties"
        extends="mktFlowElementProperties">
    <wcfPropertyGroup name="containerGroup" collapsable="false">
      <wcfPropertyCombobox
        propertyName="elemTemplateName"
        promptText="{extMarketingResources
          .wishListCriteria.string}"/>

      <wcfPropertyGroup name="productsGroup" collapsable="false">
        <wcfEnablementCondition
          propertyName="elemTemplateName"
          enablementValue="wishListProducts"
          conditionId="productsElement"/>

        <wcfMultiValueSelector name="anyOrAll1"
          useRadioButtons="true"
          promptText="{mktMarketingResources.targetCondition.string}">
          <wcfMultiValueSelection name="selectAnyP"
```

```

        displayName="${extMarketingResources
        .anyProducts.string}">
        <dataset name="values">
            <anyOrAll>any</anyOrAll>
            <containsOperator>=</containsOperator>
        </dataset>
    </wcfMultiValueSelection>
    <wcfMultiValueSelection name="selectAllP"
        displayName="${extMarketingResources
        .allProducts.string}">
        <dataset name="values">
            <anyOrAll>all</anyOrAll>
            <containsOperator>=</containsOperator>
        </dataset>
    </wcfMultiValueSelection>
    <wcfMultiValueSelection name="selectNoP"
        displayName="${extMarketingResources
        .notAnyProducts.string}">
        <dataset name="values">
            <anyOrAll>any</anyOrAll>
            <containsOperator>!=</containsOperator>
        </dataset>
    </wcfMultiValueSelection>
</wcfMultiValueSelector>

    <wcfPropertyChildListEditor name="prodEditor"
        objectTypes="ChildCatentry,
        ChildInheritedCatentry"
        listClass="mktProductGrid"
        promptText="${mktMarketingResources.products.string}"
        required="true"/>
</wcfPropertyGroup>

    <wcfPropertyGroup name="categoriesGroup"
        collapsable="false">
        <wcfEnablementCondition propertyName="elemTemplateName"
            enablementValue="wishListCategories"
            conditionId="categoriesElement"/>

        <wcfMultiValueSelector name="anyOrAll2"
            useRadioButtons="true"
            promptText="${mktMarketingResources.targetCondition.string}">
            <wcfMultiValueSelection name="selectAnyC"
                displayName="${extMarketingResources
                .anyCategories.string}">

```



```

        <dataset name="values">
            <anyOrAll>any</anyOrAll>
            <containsOperator>=</containsOperator>
        </dataset>
    </wcfMultiValueSelection>
    <wcfMultiValueSelection name="selectAllC"
        displayName="{extMarketingResources
            .allCategories.string}">
        <dataset name="values">
            <anyOrAll>all</anyOrAll>
            <containsOperator>=</containsOperator>
        </dataset>
    </wcfMultiValueSelection>
    <wcfMultiValueSelection name="selectNoC"
        displayName="{extMarketingResources
            .notAnyCategories.string}">
        <dataset name="values">
            <anyOrAll>any</anyOrAll>
            <containsOperator>!=</containsOperator>
        </dataset>
    </wcfMultiValueSelection>
</wcfMultiValueSelector>

    <wcfPropertyChildListEditor name="catEditor"
        objectTypes="ChildCatalogGroup,
            ChildInheritedCatalogGroup"
        listClass="mktCategoryGrid"
        promptText="{mktMarketingResources.categories.string}"
        required="true"/>

</wcfPropertyGroup>
</wcfPropertyGroup>
</class>
</library>

```

2. Save the file.

4.3.8 Creating the summary for the target

To create the summary definition for the wish list target, perform the following steps:

1. Create an OpenLaszlo file named *WishListSummary.lzx* inside the `<your_company_name>/marketing/propertiesViews/activityBuilder` directory. The snippet in Example 4-22 shows the summary definition for the wish list target.

Example 4-22 The wish list summary definition

```
<library>
  <class name="extWishListSummary" extends="mktFlowElementSummary">
    <mktFlowSummaryParam name="elemTemplateName"
      propertyName="elemTemplateName"/>
    <method name="updateSummary" args="e">
      <![CDATA[
        var summary = "";
        if(this.resolvedParams["elemTemplateName"] ==
          "wishListAny") {
          summary = extMarketingResources.wishListAny.string;
        } else if(this.resolvedParams["elemTemplateName"] ==
          "wishListProducts") {
          summary = extMarketingResources.wishListProducts
            .string;
        } else if(this.resolvedParams["elemTemplateName"] ==
          "wishListCategories") {
          summary = extMarketingResources.wishListCategories
            .string;
        }
        this.setSummaryText(summary);
      ]]>
    </method>
  </class>
</library>
```

2. Save the file.

4.3.9 Registering the new classes in the marketing extensions library

The object definition, the properties view, the summary definition, the resource files, and the resource bundle need to be defined in the marketing extension library located at

LOBTools/WebContent/WEB-INF/src/lzx/commerce/marketing/MarketingExtensionsLibrary.lzx. This makes the customized assets available for the marketing UI.

The MarketingExtensionsLibrary.lzx resembles Example 4-23.

Example 4-23 Registering the new assets in the Marketing extension library

```
<library>
.....

<!-- The resource bundle for the new target -->
<include href="../../madisons/marketing
/MarketingResourceBundle.lzx"/>

<!-- The resource file which contains references to the images used
in the icons-->
<include href="../../madisons/marketing/resources
/extMarketingManagementResources.lzx"/>

<!-- The wish list target object definition class -->
<include href="../../madisons/marketing/objectDefinitions
/activityBuilder/WishListFlowElementObjectDefinition.lzx"/>

<!-- The wish list target properties view class -->
<include href="../../madisons/marketing/propertiesViews
/activityBuilder/WishListPropertiesView.lzx"/>

<!-- The wish list target summary class -->
<include href="../../madisons/marketing/propertiesViews
/activityBuilder/WishListSummary.lzx"/>
.....
</library>
```

4.3.10 Adding the target to the element palette in the Activity Builder

The new wish list target needs to be added in the element palette of the Activity Builder so that business users can pick the target from the palette and use it to create activities. The elements in the palette are defined as a dataset in the Activity Builder. They are grouped based on the type of campaign element (for example, triggers, targets, actions or branching).

There are four versions of the Activity Builder. Table 4-5 shows the Activity Builder type and the OpenLaszlo file that defines a particular Activity Builder type.

Table 4-5 The Activity Builder types and the path to the corresponding OpenLaszlo files

Type	Path
Web Activity Builder	LOBTools/WebContent/WEB-INF/src/lzx/commerce/marketing/propertiesViews/WebActivityBuilder.lzx
Web activity template builder	LOBTools/WebContent/WEB-INF/src/lzx/commerce/marketing/propertiesViews/WebActivityTemplateBuilder.lzx
Dialog Activity Builder	LOBTools/WebContent/WEB-INF/src/lzx/commerce/marketing/propertiesViews/DialogActivityBuilder.lzx
Dialog activity template builder	LOBTools/WebContent/WEB-INF/src/lzx/commerce/marketing/propertiesViews/DialogActivityTemplateBuilder.lzx

To add the wish list target in the Activity Builder, the wish list target definition must be placed under the Targets group in the Activity Builder properties view. Choose the Activity Builder type in which you want to place the new target. If you want to add the trigger in all four Activity Builder types, add the new object reference in all the above files. Example 4-24 shows the snippet that includes the target definition in the Web Activity Builder.

Example 4-24 The new wish list target added in the Web Activity Builder palette

```
.....
<class name="mktWebActivityBuilder"
  extends="mktActivityBuilder"
  generalPropertiesClass="mktGeneralWebActivityProperties"
  flowConnectorClass="mktWebActivityFlowConnector"
  ..... >
  <dataset name="palette">
    <Group resourceBundle="mktMarketingResources"
      name="targets"
```

```

        helpText="help_palette_targets"
        helpLink="concepts/csbtargets.htm">
        .....
        <Element objectType="wishList"/>
        .....
    </Group>
    .....

```

4.3.11 Creating and registering a serialization JSP for the new target

Serialization JSPs need to be created to retrieve the attributes saved for the wish list target. Because the wishListAny template does not have any reference objects, we do not have to create a new JSP. For the wishListProducts and the wishListCategories templates, we need to create two JSPs. Execute the following steps to create and registry the serialization JSPs.

1. Create a JSP file `SerializeActivitywishListCategories.jsp` inside the `LOBTools/WebContent/jsp/<your_company_name>/marketing` directory. The snippet in Example 4-25 shows the contents of the JSP.

Example 4-25 The JSP snippet for the wishListCategories template

```

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<object objectType="wishList">
    <parent>
        <object objectId="${element.parentElementIdentifier.name}"/>
    </parent>
    <elemTemplateName>
        <![CDATA[${element.campaignElementTemplateIdentifier.externalIdentifier
.name}]]>
    </elemTemplateName>
    <elementName>
        ${element.campaignElementIdentifier.name}
    </elementName>
    <sequence>${element.elementSequence}</sequence>
    <customerCount readonly="true">${element.count}</customerCount>
    <c:forEach var="elementVariable"
        items="${element.campaignElementVariable}">
        <c:choose>
            <c:when test="${elementVariable.name == 'categoryIdList'}">
                <c:choose>
                    <c:when test="${categoryUniqueIDs != ''}">
                        <c:set var="categoryUniqueIDs"
value="${categoryUniqueIDs}${','}${elementVariable.value}" />

```

```

        </c:when>
        <c:otherwise>
            <c:set var="categoryUniqueIDs"
                value="\${elementVariable.value}" />
        </c:otherwise>
    </c:choose>
</c:when>
<c:otherwise>
    <\${elementVariable.name}>
        <![CDATA[\${elementVariable.value}]]>
    </\${elementVariable.name}>
</c:otherwise>
</c:choose>
</c:forEach>
<c:if test="\${categoryUniqueIDs != ''}">
    <c:set var="uniqueIDs" value="\${categoryUniqueIDs}" />
    <jsp:directive.include
file="../../commerce/marketing/restricted/GetCategoriesById.jsp" />
</c:if>
<c:forEach var="userDataField"
    items="\${element.userData.userDataField}">
    <x_\${userDataField.typedKey}>
        <![CDATA[\${userDataField.typedValue}]]>
    </x_\${userDataField.typedKey}>
</c:forEach>
</object>

```

2. Create a JSP file `SerializeActivitywishListProducts.jsp` inside the `LOBTools/WebContent/jsp/<your_company_name>/marketing` directory. The snippet in Example 4-26 shows the JSP contents.

Example 4-26 The JSP snippet for the wishListProducts template

```

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<object objectType="wishList">
    <parent>
        <object objectId="\${element.parentElementIdentifier.name}" />
    </parent>
    <elemTemplateName>
        <![CDATA[\${element.campaignElementTemplateIdentifier.externalIdentifier.name}]]>
    </elemTemplateName>
    <elementName>
        \${element.campaignElementIdentifier.name}
    </elementName>

```

```

</elementName>
<sequence>${element.elementSequence}</sequence>
<customerCount readonly="true">${element.count}</customerCount>
<c:forEach var="elementVariable"
  items="${element.campaignElementVariable}">
  <c:choose>
    <c:when test="${elementVariable.name ==
      'catalogEntryIdList'}">
      <c:choose>
        <c:when test="${productUniqueIDs != ''}">
          <c:set var="productUniqueIDs"
            value="${productUniqueIDs}${','}${elementVariable.value}" />
        </c:when>
        <c:otherwise>
          <c:set var="productUniqueIDs"
            value="${elementVariable.value}" />
        </c:otherwise>
      </c:choose>
    </c:when>
    <c:otherwise>
      <${elementVariable.name}>
        <![CDATA[${elementVariable.value}]]>
      </${elementVariable.name}>
    </c:otherwise>
  </c:choose>
</c:forEach>
<c:if test="${productUniqueIDs != ''}">
  <c:set var="uniqueIDs" value="${productUniqueIDs}" />
  <jsp:directive.include
    file="../../commerce/marketing/restricted/GetProductsById.jsp" />
</c:if>
<c:forEach var="userDataField"
  items="${element.userData.userDataField}">
  <x_${userDataField.typedKey}>
    <![CDATA[${userDataField.typedValue}]]>
  </x_${userDataField.typedKey}>
</c:forEach>
</object>

```

3. Create the struts action mapping for the new serialization JSPs. Open the LOBTools/WebContent/WEB-INF/struts-extension.xml file. Add the snippet shown in Example 4-27 inside the <action-mappings> section of the struts configuration.

Example 4-27 The struts action mapping for the serialization JSPs

```
<action path="/SerializeActivityElement-wishListAny"
include="/jsp/commerce/marketing/restricted/SerializeActivitygenericElement.jsp
" />
<action path="/SerializeActivityElement-wishListProducts"
include="/jsp/madisons/marketing/SerializeActivitywishListProducts.jsp" />
<action path="/SerializeActivityElement-wishListCategories"
include="/jsp/madisons/marketing/SerializeActivitywishListCategories.jsp" />
```

After the command and UI customization are completed, build the project for Java changes and build the openlazlo project for the UI changes. For more details on building the OpenLaszlo project, see the WebSphere Commerce Information Center:

http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center_customization.doc/tasks/tdeolbuild.htm

Restart the WebSphere Commerce server to test your code.

4.4 Test your code

Perform the following procedures to test the new wish list target.

4.4.1 Checking the UI customization for new wish list target

You can see the newly added wish list target in the Target palette when creating a Web activity. Use this target to specify various combinations of the Web activities and save the definition. You can retrieve them and make adjustments. After you save the Web activities definitions with wish list targets, you can test the scenario.

4.4.2 Creating a new wish list target Web activity

Karl received a list of products that are aging and going to be discontinued because the newer versions of same products are in plan to launch soon. He decided to give an instant discount on those products and sell them immediately. He figured the most precise way to get this sale done was by targeting customers who have any of the specified catalog entries in their wish list. Karl created the following Web activity to offer a 10% discount to any customers who have specified catalog entries in the wish list. Make sure to activate this Web activity too. See Figure 4-5.

E-Marketing Spot
Customer views e-Marketing Spot: HomePageRightSideBarAds

Wish List
Specific catalog entries in the wish list

Recommend Content
AmountOffOrder

Wish List

Wish List contains: Specific catalog entries in the wish list

Target customers:

- ☒ Who have any of the following catalog entries in the wish list
- ☐ Who have all of the following catalog entries in the wish list
- ☐ Who do not have any of the following catalog entries in the wish list

Find and Add

*Catalog entries

* Type	* Code	Name
	TATA-0201	"Hawthorne" Table Glasses.
	TATA-0101	"Villagois" Table Glasses
	TASI-0101	5-Piece Silverware Set
	KIES-0101	Coffee and Espresso Bar

1 to 4 of 4

Figure 4-5 Discount for customers with items that are in the Web activity definition

4.4.3 Testing your task command with Web stores

Log on to the Web store and create a wish list with any of the catalog entries mentioned in the activity definition. If you browse the store home page, you can see the promotional content in e-marketing spot on the right.

See the WebSphere Commerce Information Center to know more about testing the targets in Web and Dialog activities:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center.doc/refs/rsbtesttar.htm>

Extend the customer segment

This chapter provides a sample customization scenario to demonstrate how to extend the customer segment based on the needs of the user. With this customization, a marketing manager can define a *Pick up in Store* customer segment to target customers that enter the physical store to pick up their order.

This chapter includes the following sections:

- ▶ Overview
- ▶ Design
- ▶ Implementation
- ▶ Test your code

5.1 Overview

IBM Management Center for WebSphere Commerce has enhanced customer segment tooling that allows the business user to define more precise customer segments and target them for a marketing campaign. The IBM Management Center for WebSphere Commerce user interface (UI) for the customer segment definition has many properties to choose from and combine to get the desired customer segment. Sometimes, however, the business need is specific and cannot be achieved by the inherent features. In this case, customization is required. The base customer segment feature can be extended and a new behavior enabled. In this section, we discuss a scenario where the marketing manager wants to create a custom segment to contain registered users who have used the Pick up in Store option when placing orders. To accomplish this, a new property is added in the customer segment definition window. This property can be used independently or in conjunction with other properties to define a customer segment.

WebSphere Commerce allows customers to browse products in the online store, add them in the shopping cart, place the order, but choose the option to pick up the items from a nearby physical store. The customer can save on shipping costs by picking up their products at a store. This minimizes the customer's waiting time, as they do not have to wait for the product to be delivered to their location. Customers can retrieve their purchase at a time of their convenience after it is made available for pickup at their preferred or nearby physical store location.

When using the Pick up in Store purchase mode, some additional sales for the retailer are also made possible, as the customer is present in the store when picking up their products. Karl observed from his analytic reports that the majority of the working professionals place their orders in the online store because of the great price, but pick up the items at the physical stores. The reason can be that they are at work during business hours and cannot receive the shipment, or they like to visit the physical stores to see more products. Whatever the reason, Karl finds these customers suitable to target for in-store promotions. To do that, Karl wants to define new customer segments, such as a Pick up in Store segment with some combinations such as Sporadic Pick up in Store customers and Frequent Pick up in Store customers. For example, customers who have less than five Pick up in Store orders are placed in the Sporadic segment and customers with more than five Pick up in Store orders are placed in the Frequent segment.

There are currently no inherent properties in the customer segment definition window to create such segments. However, the IBM Management Center for WebSphere Commerce Marketing tool allows you to customize the base behavior. Karl needs to have new properties created for the Pick up in Store

customer segment definition. He contacted the IT department to add new properties to define those customer segments. The requirements for the IT department are as follows:

- ▶ Add a new drop-down properties list on the Purchase Details tab of the customer segment definition wizard. This allows the business user to select the Pick up in Store properties when defining the customer segment. This can be used with other properties to narrow down the buyers in the segment depending on the business scenario.
- ▶ Select the criteria to determine the number of Pick up in Store orders the customer needs to have to be in the customer segment. Karl needs these attributes to define separate segments for Sporadic and Frequent Pick up in Store customers.
- ▶ When selecting one of the options, a new text box to accept numbers is displayed beneath the list. This number helps parameterize the segment condition based on the analytic report.

The new customer segment UI looks like Figure 5-1 and Figure 5-2 on page 102.

The screenshot shows a web application window titled "Frequent pick up in store customers". It has a tabbed interface with tabs: "General Properties", "Customer or Segment", "Registration", "Demographics", "Address", "Purchase Details" (selected), and "Miscellaneous". There are "Save and Close" and "Close" buttons in the top right. The main content area has four rows, each with a label and a dropdown menu:

- Row 1: "Amount spent since registration" with a dropdown showing "Ignore amount spent".
- Row 2: "Orders" with a dropdown showing "Ignore number of orders".
- Row 3: "Pick up in store orders" with a dropdown showing "Ignore number of Pick up in store ...". A yellow tooltip is visible over this dropdown, showing "Ignore number of Pick up in store Orders".
- Row 4: "Last purchase date" with a dropdown showing "Ignore last purch".

Figure 5-1 New drop down properties list

Figure 5-2 Capturing the required number of Pick up in Store orders to qualify for the condition

After this customization is completed, Karl can create new customer segments with these properties, and update the existing customer segments to add these properties. He can retrieve these customer segment definitions at any time to make adjustments. When the IT department reviews these requirements, they learn that they cannot use a WebSphere Commerce command to filter the customers based on the Pick up in Store properties. They decide to extend the existing task commands to complete this task. The next two sections describe the overall design and implementation for this requirement.

5.2 Design

In this section, we discuss the design of the new customer segment and the high level steps involved in creating it.

The following steps are needed to create a new customer segment:

1. Steps for the server side logic of the customer segment:
 - a. Create the new customer segment task command.
 - b. Register the task command.
2. Steps for the IBM Management Center for WebSphere Commerce User Interface (UI)
 - a. Define the resource bundle for the UI text elements.
 - b. Create the object definition for the customer segment attribute.

- c. Create the properties view for the customer segment attribute.
- d. Register the new libraries for the customer segment attributes with the marketing UI.
- e. Create and register the serialization JSP for the new customer segment attribute.

5.2.1 Task commands

WebSphere Commerce provides a **CheckUserInMemberGroupCmd** command to filter out customers based on customer segment definitions. This class has a method to evaluate each condition to determine if the customer meets the definition of the condition. A customer belongs to the customer segment if they meet all the condition criteria. You need to extend this command and add new conditions to filter the Pick up in Store customers for your segment. For more information about API details and the methods to override, see the WebSphere Commerce Information Center:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.api.doc/com/ibm/commerce/membergroup/commands/CheckUserInMemberGroupCmd.html>

5.2.2 Object definition

An object definition is required for the new customer segment attribute to collect the input data from the IBM Management Center for WebSphere Commerce. The new attribute definition object extends from the `wcfChildObjectDefinition` class. For more information about the `wcfChildObjectDefinition` class visit the Information Center:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.openlaszlo.api.doc/lzx/commerce/foundation/restricted/ObjectDefinition.lzx/wcfChildObjectDefinition.html>

The `idProperty` attribute of the top object has the value `elementName`.

The IBM Management Center for WebSphere Commerce framework provides two classes:

- `mktCSElementAndOrListConditionObjectDefinition`
- `mktCSElementSimpleConditionObjectDefinition`

These classes define whether the segment attribute has conditions that are logically ANDed or ORed, or if the customer segment has a simple condition. For example, if the segment attribute condition is to check whether a certain value is greater than or equal to (\geq) some number, you can define the child object

definition for the condition to extend the `mktCSElementAndOrListConditionObjectDefinition` class.

If you have a more complex condition for a customer segment that does a logical AND or OR of two simple conditions (for example, if the segment condition is to check if a value is less than x and the value is greater than y [where x and y are numbers]), you have to define a new child object definition for the complex condition that extends the `mktCSElementAndOrListConditionObjectDefinition` class. For more information about these classes, see the Information Center:

- ▶ <http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.openlaszlo.api.doc/lzx/commerce/marketing/restricted/customerSegment/CustomerSegmentElementObjectDefinition.lzx/mktCSElementSimpleConditionObjectDefinition.html>
- ▶ <http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.openlaszlo.api.doc/lzx/commerce/marketing/restricted/customerSegment/CustomerSegmentElementObjectDefinition.lzx/mktCSElementAndOrListConditionObjectDefinition.html>

Because the scenario we are dealing with is to check for customers whose number of Pick up in Store orders is equal to or greater than or equal to some number, we can use a simple condition in our object definition for the attribute.

We define three templates:

- ▶ ignore
- ▶ equal
- ▶ greaterOrEqual

Table 5-1 lists the behavior of the templates.

Table 5-1 *Templates and their behavior*

Template	Behavior
ignore	Does not consider the Pick up in Store orders of a customer when evaluating the customer segment.
equal	When the business users chooses the <i>equal</i> template, the marketing UI prompts the user to enter the number of Pick up in Store orders that need to be checked against the customers orders when evaluating the segment. The customer who has the exact number of Pick up in Store orders is qualified to be part of the segment.
greaterOrEqual	When the business user chooses the <i>greaterOrEqual</i> template, the marketing UI prompts the user to enter the number of Pick up in Store orders that need to be checked against the customers orders when evaluating the segment. The customer who has the Pick up in Store orders greater than or equal to the value entered in the UI is qualified to be part of the segment.

5.2.3 Properties view

The properties view is required to capture the values of the properties that the business users enter through the UI. The new segment attribute allows the business user to select the template defined in the Table 5-1. Based on the template, the UI changes to prompt the user to enter the number of the Pick up in Store orders. After the properties view is created, it also needs to be added to the Purchase Details tab in the existing customer segment properties view.

5.2.4 Serialization JSPs for the customer segment attribute

The serialization JSPs retrieve the segment data stored in the database and displays them in the customer segment UI.

5.3 Implementation

This section provides the details for implementing the new customer segment.

5.3.1 Creating the new customer segment task command

To create the new task command, extend the `CheckUserInMemberGroupCmdImpl` class and override the `evaluate(String, String, String, Qualifier)` method. For the new behavior, this task command evaluates whether a customer also belongs to the Pick up in Store segment specified in Web or dialog activities.

Creating an interface to define the global constants

1. In Rational Application Developer for WebSphere Software, click **File** → **New** → **Other**.
2. In the next window, select the interface wizard and click **Next**.
3. Ensure `WebSphereCommerceServerExtensionsLogic/src` is entered in the Source folder field.
4. In the Package field, type:
`com.[your company name].commerce.marketing.commands.`
5. Enter `ExtConstants` in the name field and click **Finish**. Example 5-1 shows the new `ExtConstants` interface with constants.

Example 5-1 The ExtConstants interface to define the constants

```
package com.madisons.commerce.marketing.commands;
```

```
public interface ExtConstants {

    String VARIABLE_PICKIN_STORE_ORDER = "bopisOrders";

    int DEFAULT_MINIMUM_PICKIN_STORE_ORDERS = 1;

}
```

Create the task command for Pick up in Store customer segment

1. In Rational Application Developer for WebSphere Software, click **File** → **New** → **Other**.
2. In the next window, select the class wizard and click **Next**.
3. Ensure WebSphereCommerceServerExtensionsLogic/src is entered in the Source folder field.
4. In the Package field, type:
`com.[your company name].commerce.marketing.commands.`
5. Enter `ExtCheckUserInMemberGroupCmdImpl` in the name field.
6. Add `CheckUserInMemberGroupCmdImpl` as the extended Super class and add `ExtConstants` in the extended Interface and click **Finish**. Example 5-2 shows the new `ExtCheckUserInMemberGroupCmdImpl` class.

Example 5-2 The ExtCheckUserInMembergroupCmdImpl task command to filter Pick up in Store customer segment

```
package com.madisons.commerce.marketing.command;

import com.ibm.commerce.membergroup.commands.CheckUserInMemberGroupCmd;

public class ExtCheckUserInMemberGroupCmdImpl
    extends CheckUserInMemberGroupCmdImpl
    implements ExtConstants {

}
```

This command overrides the `evaluate (String, String, String, Qualifier)` method which evaluates the segment for Pick up in Store customers. If the variable is Pick up in Store, the command gets the orders for the customer and after examining the orders, it returns true or false based on the results. Example 5-3 shows the `evaluate` method implementation for this command.

Example 5-3 The `evaluate (String, String, String, Qualifier)` method to check the Pick up in Store customers in segment

```

/**
 * This method evaluates the values for variable
 * <code>VARIABLE_PICKIN_STORE_ORDER</code> to determine the customer
 * segment for <code>Pick up in Store</code> orders. Any customer,
who
 * will have such orders will be added in this segment.
 *
 * @param variable the variable name for checking the
 * <code>VARIABLE_PICKIN_STORE_ORDER</code> property.
 * @param operator the operator to evaluate the condition.
 * @param value the value to check for the condition.
 * @param qualifiers other qualifiers.
 * @return <code>true</code>, if the condition matches for this
 * variable, <code>false</code> otherwise.
 * @see CheckUserInMemberGroupCmd#evaluate(String,
 * com.ibm.commerce.condition.OpenCondition.Parameter[])
 */
public boolean evaluate(
    String variable,
    String operator,
    String value,
    Qualifier[] qualifiers) {

    final String METHODNAME = "evaluate(String, String, String,
        Qualifier[])";
    if (LoggingHelper.isEntryExitTraceEnabled(LOGGER)) {
        ECTrace.entry(ECTraceIdentifiers.COMPONENT_MARKETING,
            getClass().getName(), METHODNAME,
            new Object[]{variable, operator,
                value, qualifiers});
    }
    boolean result = false;

    /*
     * Check the variable for VARIABLE_PICKIN_STORE_ORDER to
     * handle the logic, else delegate it to the super class.
     */

```

```

        if(null != variable &&
            VARIABLE_PICKIN_STORE_ORDER.compareTo(variable) == 0){
            result = evaluatePickInStoreOrdersCondition(operator, value,
                qualifiers);
        } else {
            result = super.evaluate(variable, operator, value,
                qualifiers);
        }
        if (LoggingHelper.isEntryExitTraceEnabled(LOGGER)) {
            ECTrace.exit(ECTraceIdentifiers.COMPONENT_MARKETING,
                getClass().getName(), METHODNAME, result);
        }
        return result;
    }
}

```

The `evaluatePickInStoreOrdersCondition` (Example 5-4) is a custom method to examine the customer's orders, only when the variable is `VARIABLE_PICKIN_STORE_ORDER`.

Example 5-4 The helper method `evaluatePickInStoreOrdersCondition(String, String, Qualifier)` to figure out the configured number of orders of type In-Store Pickup

```

/**
 * A helper method to check the orders of a customer for
 * <code>BOPIS</code> property.
 *
 * An additional logic can be provided in this method to check for
 * the minimum/configured number of <code>BOPIS</code> orders
 * to qualify this customer in the 'Pick up in Stores Customer'
 * segment.
 *
 * @param operator the operator to evaluate the condition.
 * @param value the value to check for the condition.
 * @param qualifiers other qualifiers.
 * @return <code>true</code>, if the condition matches for this
 *         variable, <code>false</code> otherwise.
 * @see #getConfiguredNumberOfInStorePickupOrders()
 */
protected boolean evaluatePickInStoreOrdersCondition(
    String operator,
    String value,
    Qualifier[] qualifiers){

    final String METHODNAME =
        "evaluatePickInStoreOrdersCondition(String, String, Qualifier[])";
    boolean result = false;

```

```

int numInStorePickupOrders = 0;

List orders = getOrders();
Iterator orderIds = orders.iterator();
while(orderIds.hasNext()){
    String orderId = (String)orderIds.next();
    try {
        result = checkInStorePickupOrders(orderId);
        /*
         * If total number of BOPIS orders for this customer match
         * the configured number, then break this loop as there is
         * no need to check further. This will also improve the
         * overall performance of this command when the Order
         * history list is huge for this customer.
         */
        if(result && OPERATOR_EQUAL_TO.equals(operator)){
            if(++numInStorePickupOrders ==
                getConfiguredNumberOfInStorePickupOrders(value))
            { break; }
        }

        if(result &&
            OPERATOR_GREATER_THAN_OR_EQUAL_TO.equals(operator)){
            if(++numInStorePickupOrders >=
                getConfiguredNumberOfInStorePickupOrders(value))
            { break; }
        }
        // Reset the result flag
        result = false;
    } catch (EException e) {
        ECTrace.trace(ECTraceIdentifiers.COMPONENT_MARKETING,
            getClass().getName(), METHODNAME, "Order Id: " + orderId
                + "\n" + e.getMessage());
    }
}
return result;
}

```

The `getConfiguredNumberOfInStorePickupOrders` method gets the values for the required number of Pick up in Store orders attribute defined in the UI. The `getOrders` method gets the list of Order IDs for the customer to check the eligibility for the segment. Example 5-5 shows the implementation of these two methods.

Example 5-5 The `getConfiguredNumberOfInStorePickupOrders` method

```
/**
 * This method returns the configured number for BOPIS order to let
 * customers qualify for 'Pick up in Stores Customer' segment.
 *
 * @param value
 * @return the configured number for BOPIS order.
 * @see ExtConstants#DEFAULT_MINIMUM_PICKIN_STORE_ORDERS
 */
protected int getConfiguredNumberOfInStorePickupOrders(String
    value){
    int minumumPickinStoreOrders =
        DEFAULT_MINIMUM_PICKIN_STORE_ORDERS;
    try{
        minumumPickinStoreOrders = Integer.parseInt(value);
    } catch(NumberFormatException exception){
        minumumPickinStoreOrders =
            DEFAULT_MINIMUM_PICKIN_STORE_ORDERS;
    }

    return minumumPickinStoreOrders;
}

/**
 * This method gets the list of orders (only Id) for the member.
 * @return list of order id.
 * @see #getUser()
 */
protected List getOrders(){
    final String METHODNAME = "getOrders()";
    List orderIds = new ArrayList();
    try {
        String memberId = getUser().getMemberId();
        OrderSearchDataBean bean = new OrderSearchDataBean();
        bean.setCommandContext(getCommandContext());
        bean.setPageSize(100);
        WhereClauseCondition condition = new WhereClauseCondition();
        Set setFromTable = new HashSet();
```

```

        setFromTable.add("ORDERS");
        condition.appendANDCondition(new WhereClauseFreeFormCondition(
            "ORDERS.MEMBER_ID = " + memberId, setFromTable, null));
        Vector orderList = bean.findOrders(condition, "ORDERS",
            "ORDERS_ID", "1", "100");
        for(int i = 0; i < orderList.size(); ++i){
            orderIds.addAll((Vector)orderList.get(i));
        }
    } catch (ECSYSTEMException e) {
        ECTrace.trace(ECTraceIdentifiers.COMPONENT_MARKETING,
            getClass().getName(), METHODNAME, e.getMessage());
    } catch (RemoteException e) {
        ECTrace.trace(ECTraceIdentifiers.COMPONENT_MARKETING,
            getClass().getName(), METHODNAME, e.getMessage());
    } catch (CreateException e) {
        ECTrace.trace(ECTraceIdentifiers.COMPONENT_MARKETING,
            getClass().getName(), METHODNAME, e.getMessage());
    } catch (FinderException e) {
        ECTrace.trace(ECTraceIdentifiers.COMPONENT_MARKETING,
            getClass().getName(), METHODNAME, e.getMessage());
    } catch (NamingException e) {
        ECTrace.trace(ECTraceIdentifiers.COMPONENT_MARKETING,
            getClass().getName(), METHODNAME, e.getMessage());
    }
    return orderIds;
}

```

-
7. After the list of orders is created, the `checkInStorePickupOrders` method checks the order items one by one in a loop for the `InStorePickup` attribute to determine the segment. After the `InStorePickup` attribute is detected, the loop is broken out of as there is no need to keep checking the attributes after it is found. Example 5-6 shows the method.

Example 5-6 The `checkInStorePickupOrders` method

```

/**
 * This method checks the individual order items for BOPIS
 * properties.
 *
 * @param orderId the order Id to fetch the order items.
 * @return <code>true</code>, if the condition matches for BOPIS,
 * <code>true</code> otherwise.
 * @throws ECException
 * @see CheckInStorePickupCmd
 */

```

```

protected boolean checkInStorePickupOrders(
    String orderId) throws ECEException {

    final String METHODNAME = "checkInStorePickupOrders(String)";
    boolean result = false;
    OrderDataBean orderDataBean = new OrderDataBean();
    orderDataBean.setOrderId(orderId);
    DataBeanManager.activate(orderDataBean);
    OrderItemAccessBean[] accessBean = orderDataBean.getOrderItems();
    for(int i = 0; i < accessBean.length; ++i){
        CheckInStorePickupCmd checkInStorePickupCmd =
            (CheckInStorePickupCmd)CommandFactory
                .createCommand(CheckInStorePickupCmd.NAME,
                    getCommandContext().getStoreId());
        checkInStorePickupCmd.setCommandContext(getCommandContext());
        checkInStorePickupCmd.setOrderItem(accessBean[i]);
        checkInStorePickupCmd.execute();
        result = checkInStorePickupCmd.isInStorePickupOrderItem();
        // If any of the order items is kind of 'In Store Pickup',
        // then no need to check the entire order items and break
        // the loop.
        if(result){ break; }
    }
    return result;
}

```

8. Save and close the Java file. This completes all the steps required to build the server side logic to extend the customer segment.

Register the task command with WebSphere Commerce server

After the task command is created, register the command with the WebSphere Commerce server. You might need to add an entry in the CMDREG table to override the base **CheckUserInMemberGroupCmd** implementation with your custom command. This ensures that the runtime executes the custom command whenever the **CheckUserInMemberGroupCmd** command is called. Example 5-7 shows the sample SQL to register this command.

Example 5-7 CMDREG insert sql statement

```
INSERT INTO CMDREG VALUES(  
    0,  
    'com.ibm.commerce.membergroup.commands.CheckUserInMemberGroupCmd',  
    null,  
    'com.madisons.commerce.marketing.commands.ExtCheckUserInMemberGroupCmdI  
mpl',  
    null,  
    null,  
    'Local',  
    null);
```

5.3.2 Defining a resource bundle and properties file for UI text

See 4.3.5, “Defining a resource bundle and properties file for UI text” on page 82 to create a new resource bundle.

Perform the following steps to define a resource bundle for the text to be displayed in the UI

1. Enter the snippet in Example 5-8 in to the properties file and save the file.

Example 5-8 The UI strings to be added in the properties file for the new customer segment attribute

```
customerSegment_BOPISOrders_Ignore=Ignore number of Pick up in store  
Orders  
customerSegment_BOPISOrders_Equal=Number of Pick up in store orders  
must be equal to the following number  
customerSegment_BOPISOrders_Greater=Number of Pick up in store orders  
must be greater than equal to the following number  
customerSegment_NumberOfBOPISOrders=Number of Pick up in store orders  
customerSegment_BOPISOrders=Pick up in store orders
```

2. Update the
LOBTools/WebContent/WEB-INF/src/lzx/<your_company_name>/marketing/MarketingResourceBundle.lzx file and enter the resource bundle keys in the file. See the snippet in Example 5-9.

Example 5-9 Defining the resource bundle keys in the resource bundle file

```
<class name="extMarketingResourceBundle"
    extends="wcfResourceBundle"
    baseName="com.madisons.marketing.client.lobtools.properties.MarketingLOB">

    .....

    <wcfResourceBundleKey
        name="customerSegment_BOPISOrders_Ignore" />
    <wcfResourceBundleKey
        name="customerSegment_BOPISOrders_Equal" />
    <wcfResourceBundleKey
        name="customerSegment_BOPISOrders_Greater" />
    <wcfResourceBundleKey
        name="customerSegment_BOPISOrders" />
    <wcfResourceBundleKey
        name="customerSegment_NumberOfBOPISOrders" />

    .....

</class>
```

5.3.3 Creating the object definition

To create the object definition for the customer segment attribute, execute the following steps:

1. Create an OpenLaszlo file `BOPISOrdersObjectDefinition.lzx` under the directory `LOBTools/WebContent/WEB-INF/src/lzx/<your_company_name>/marketing/objectDefinitions/customerSegment` and use the snippet in Example 5-10.

Example 5-10 The object definition for the customer segment attribute

```
<library>
  <class name="extBOPISOrdersListObjectDefinition"
    extends="wcfChildObjectDefinition"
    objectType="BOPISOrdersList"
    baseType="mktCSElementAndOrListConditionObjectDefinition">
    <wcfChildObjectDefinition objectType="BOPISOrdersCondition"
      baseType="mktCSElementSimpleConditionObjectDefinition">
      <dataset name="template">
        <conditionVariable>bopisOrders</conditionVariable>
      </dataset>

      <wcfPropertyDefinition propertyName="conditionValue"
        required="true" type="integer" minValue="0"
        forceParentValidation="true"
        displayName="{extMarketingResources.customerSegment_NumberOfBOPISOrders.string}" />
      </wcfChildObjectDefinition>
    </class>

    <class name="extBOPISOrdersObjectDefinition"
      extends="wcfChildObjectDefinition"
      objectType="BOPISOrders" singleInstance="true"
      templateTypeProperty="template"
      idProperty="elementName">
      <dataset name="template">
        <elementName>BOPISOrders</elementName>
        <template>ignore</template>
      </dataset>

      <extBOPISOrdersListObjectDefinition />

      <wcfObjectTemplate templateType="ignore"
        displayName="{extMarketingResources.customerSegment_BOPISOrders_Ignore.string}">
```

```

        <dataset name="template" />
    </wcfObjectTemplate>

    <wcfObjectTemplate templateType="equal"
displayName="${extMarketingResources.customerSegment_BOPISOrders_Equal.
string}">

        <dataset name="template">
            <object objectType="BOPISOrdersList">
                <elementName>bopisOrders</elementName>
                <conditionUsage>orListCondition</conditionUsage>
                <object objectType="BOPISOrdersCondition">
                    <conditionOperator>=</conditionOperator>
                </object>
            </object>
        </dataset>

    </wcfObjectTemplate>

    <wcfObjectTemplate templateType="greaterOrEqual"
displayName="${extMarketingResources.customerSegment_BOPISOrders_Greate
r.string}">

        <dataset name="template">
            <object objectType="BOPISOrdersList">
                <elementName>bopisOrders</elementName>
                <conditionUsage>orListCondition</conditionUsage>
                <object objectType="BOPISOrdersCondition">
                    <conditionOperator>>=</conditionOperator>
                </object>
            </object>
        </dataset>

    </wcfObjectTemplate>
</class>
</library>

```

2. Register the new object in the `TopAndListObjectDefinition.lzx` file under the `LOBTools/WebContent/WEB-INF/src/lzx/commerce/marketing/objectDefinitions/customerSegment` directory. The `mktTopAndListObjectDefinition` class is the top level condition class which does the logical AND of all the customer segment attribute conditions for a particular customer segment. The new segment attribute object needs to be defined as a child of the `mktTopAndListObjectDefinition` class. For more information about the `mktTopAndListObjectDefinition` class, refer to the Information Center:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.openlaszlo.api.doc/lzx/commerce/marketing/objectDefinitions/customerSegment/TopAndListObjectDefinition.lzx/mktTopAndListObjectDefinition.html>

The snippet highlighted in Example 5-11 shows adding the new segment attribute object as a child of the top object.

Example 5-11 Registering the segment attribute object as a child of the customer segment top object

```
<class name="mktTopAndListObjectDefinition"
  extends="wcfChildObjectDefinition" objectType="TopAndList"
  idProperty="elementName">

  .....

  <extBOPISOrdersObjectDefinition/>

  .....

</class>
```

3. Save the files.

5.3.4 Creating the properties view for the customer segment attribute

To create the properties view for the customer segment attribute, perform the following steps:

1. Create an LOBTools/WebContent/WEB-INF/src/lzx/madisons/marketing/propertiesViews/customerSegment/BOPISOrdersPropertiesView.lzx OpenLazlo file. The snippet in Example 5-12 shows the properties view for the new segment attribute.

Example 5-12 Properties view for the customer segment attribute

```
<library>
  <class name="extBOPISOrdersProperties" extends="wcfPropertyGroup"
    displayGrouping="true" collapsable="false">
    <wcfPropertyCombobox propertyName="template"
      objectPath="TopAndList/BOPISOrders"
      promptText="{extMarketingResources.customerSegment_BOPISOrders.string}"
    />

    <wcfPropertyGroup name="bopisOrdersGroup"
      collapsable="false">
      <wcfEnablementOrCondition conditionId="singleInput">
        <wcfEnablementCondition conditionId="equal"
          objectPath="TopAndList/BOPISOrders"
          propertyName="template"
          enablementValue="equal" />
        <wcfEnablementCondition conditionId="greaterOrEqual"
          objectPath="TopAndList/BOPISOrders"
          propertyName="template"
          enablementValue="greaterOrEqual" />
      </wcfEnablementOrCondition>
      <wcfPropertyStepper

        objectPath="TopAndList/BOPISOrders/BOPISOrdersList/BOPISOrdersCondition"
        "
          propertyName="conditionValue" required="true"

        promptText="{extMarketingResources.customerSegment_NumberOfBOPISOrders
          .string}"
          minimumValue="0" />
      </wcfPropertyGroup>
    </class>
  </library>
```

2. After the property view is defined, place the new attribute in the appropriate tab of the customer segment UI. We are placing the new attribute in the Purchase Details tab. To do this, open the LOBTools/WebContent/WEB-INF/src/lzx/commerce/marketing/propertiesViews/CustomSegmentPropertiesView.lzx file. The CustomSegmentPropertiesView.lzx defines the customer segment UI properties, the various tabs, and the widgets in each of the tab. Locate the tab pane with the name purchaseDetailsTab and insert the property definition of the new attribute within that section. After inserting the new property definition, the section resembles Example 5-13. The new attribute property is highlighted.

Example 5-13 Adding the new attribute property within the Purchase details tab of the customer segment UI

```
<wcfPropertyTabPane name="purchaseDetailsTab"
text="{mktMarketingResources.csPurchaseDetailsTab.string}">
  <wcfPropertyPane>
    <wcfPropertyGroup name="purchaseDetailsGroup"
      collapsable="false" displayGrouping="true">
      <mktAmountSpentProperties/>
      <mktOrderFulfilledProperties/>
      <extBOPISOrdersProperties />
      <mktLastPurchaseDateProperties/>
    </wcfPropertyGroup>
  </wcfPropertyPane>
</wcfPropertyTabPane>
```

3. Save the files.

5.3.5 Registering the new classes in the marketing extensions library

The object definition class, the properties view class, and the resource bundle need to be defined in the marketing extensions library, which is located at LOBTools/WebContent/WEB-INF/src/lzx/commerce/marketing/MarketingExtensionsLibrary.lzx. The snippet in Example 5-14 on page 120 shows registering the new classes in the marketing extension library file.

Example 5-14 Registering the new classes and the resource bundle in the marketing extensions library file

```
<library>

.....

<include
href="../../madisons/marketing/MarketingResourceBundle.lzx"/>
  <include
href="../../madisons/marketing/objectDefinitions/customerSegment/BOPISOrdersObjectDefinition.lzx"/>
    <include
href="../../madisons/marketing/propertiesViews/customerSegment/BOPISOrdersPropertiesView.lzx"/>
      .....
    </include>
  </include>
</include>

</library>
```

5.3.6 Creating and registering a serialization JSP for the new attribute

Serialization JSP is required to retrieve the new attribute data stored in the data. Perform the following steps to create and register the serialization JSPs.

1. Create a `SerializeCustomerSegmentBOPISOrders.jsp` JSP file inside the `LOBTools/WebContent/jsp/<your_company_name>/marketing` directory. The snippet in the Example 5-15 shows the contents of the JSP.

Example 5-15 SerializeCustomerSegmentBOPISOrders.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<c:set var="objectType" value="BOPISOrdersCondition"/>
<c:set var="parent" value="${element.parentElementIdentifier.name}"/>

<c:if test="${empty element.simpleConditionVariable}">
  <c:set var="objectType" value="BOPISOrdersList"/>
  <c:set var="parent" value="BOPISOrders"/>

  <c:forEach var="searchElement" items="${allElements}">
    <c:if test="${searchElement.parentElementIdentifier.name ==
element.memberGroupConditionElementIdentifier.name}">
      <c:set var="foundChild" value="${searchElement}"/>
    </c:if>
  </c:forEach>
</c:if>
```



```

<c:choose>
  <c:when test="\${foundChild.simpleConditionOperator} == '='">
    <c:set var="template" value="equal"/>
  </c:when>
  <c:when test="\${foundChild.simpleConditionOperator} == '>='">
    <c:set var="template" value="greaterOrEqual"/>
  </c:when>
  <c:otherwise>
    <c:set var="template" value="ignore"/>
  </c:otherwise>
</c:choose>

<object objectType="\${parent}">
  <parent>
    <object objectId="\${element.parentElementIdentifier.name}"/>
  </parent>
  <elementName>\${parent}</elementName>
  <template>\${template}</template>
</object>
</c:if>

<object objectType="\${objectType}">
  <parent>
    <object objectId="\${parent}"/>
  </parent>

  <elementName><![CDATA[\${element.memberGroupConditionElementIdentifier.name}]]></elementName>

  <conditionUniqueId><![CDATA[\${element.memberGroupConditionElementIdentifier.uniqueID}]]></conditionUniqueId>

  <conditionVariable><![CDATA[\${element.simpleConditionVariable}]]></conditionVariable>

  <conditionOperator><![CDATA[\${element.simpleConditionOperator}]]></conditionOperator>

  <conditionValue><![CDATA[\${element.simpleConditionValue}]]></conditionValue>

  <conditionUsage><![CDATA[\${element.conditionUsage}]]></conditionUsage>
  <conditionNegate><![CDATA[\${element.negate}]]></conditionNegate>
</object>

```

Based on the condition operator, we identify the template used in the segment attribute.

2. Create the struts action mapping for the new serialization jsps. Open the LOBTools/WebContent/WEB-INF/struts-extension.xml file and add the snippet shown in Example 5-16.

Example 5-16 The struts action mapping for the serialization JSP

```
<action path="/SerializeCustomerSegment-bopisOrders"
include="/jsp/madisons/marketing/SerializeCustomerSegmentBOPISOrders.js
p" />
```

3. Save the files.

After the task command and UI customization are completed, you can build the project for Java changes and build openlazo project for the UI changes. Restart the WebSphere Commerce server to test your code.

5.4 Test your code

You can follow the steps to recreate the scenario and to debug your code if you run into any problems.

5.4.1 Checking the UI customization for the new customer segment definition

When you use the customized UI to create a new customer segment definition for Pick up in Store customers, you can find them in the MGPCONDELE and MBRGRPCOND tables. There is a separate record for the newly created customer segment.

Verify the newly-created customer segment definition to make sure that the UI has retrieved all the data from the database and that it can be modified again. This lets you know that the UI customization has been done correctly.

5.4.2 Creating a dialog activity with the new customer segment

The new customer segment can be used with either Web or dialog activities. For testing purposes, we create a dialog activity that targets those customers after they have placed an order and if they have any history of Pick up in Store orders. If they meet the criteria, they are sent a special promotional e-mail. The dialog activity definition resembles Figure 5-3.

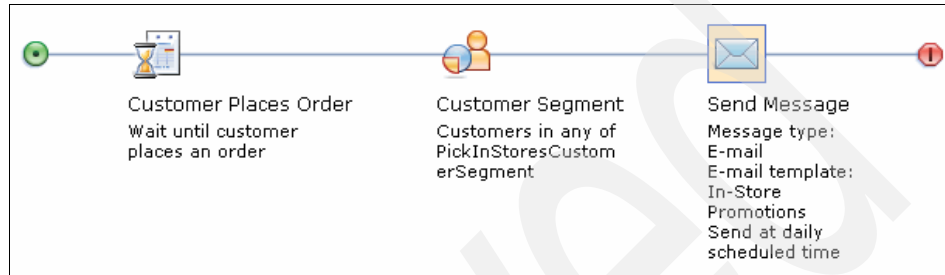


Figure 5-3 A Dialog activity for Pick up in Store customer segment, when executed, it sends in-store promotions to the ordering customers if they fall under this segment

5.4.3 Testing your task command with Web stores

You have verified the UI customization and created a Web activity for this customer segment. You need to test the task command. This task command evaluates each ordering customers after they check out. To invoke your command, make sure that the customer already has a sufficient number of orders (specified in the customer segment definition) that were placed with the Pick up in Stores shipping option.

See the following Web page to test the Place Order trigger in the activity:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center.doc/refs/rsbtestdia.htm>

Create a new trigger

This chapter provides a sample customization scenario to demonstrate how to extend the functionality of the Marketing tool in the IBM Management Center for WebSphere Commerce by creating a new trigger. For the customization, we include an overview, design and implementation to give you a starting point for your own customizations.

The following sections are included in this chapter:

- ▶ Overview
- ▶ Design
- ▶ Implementation
- ▶ Test your code

6.1 Overview

Although WebSphere Commerce Version 7 provides many triggers to use in dialog activities, you might need to create your own based on your business needs. The creation of new triggers is an easy process and is described in detail in the following sections. The steps needed to create a trigger are similar in nature to the method for creating a target, which was described in Chapter 4, “Create a new target” on page 53.

Karl Stevenson of Madisons Online decided that he wants to set up a marketing activity based on a customer providing his or her phone number. When making purchases, customers can either enter a phone number online or give a phone number to the sales person in the store. Customers who provide their phone number by calling the call center are monitored. He wants to use this customer-initiated event to trigger an action, such as issuing a coupon or sending the customer a message. He also wants to set the number of times the customer needs to provide a phone number for the marketing activity to be triggered, as well as the dates on which it occurs. An external application determines if a customer provided a phone number at the point of sale (POS) or called the call center. The application then signals the marketing activity that the event has occurred.

He wants the marketing activity to behave as follows:

- ▶ Wait until the customer make a purchase and provides a phone number three times
- ▶ Wait until the customer calls the call center two times in the last 30 days

Karl realizes that the best way to achieve the results of the scenario is to create a new marketing trigger.

The requirements for the new trigger are as follows:

- ▶ Create a trigger which is triggered by an external application
- ▶ Check if the customer provided his/her phone number
 - At POS in the store
 - Called the call center
- ▶ Set the number of times a customer has to provide the phone number before the activity continues
- ▶ Set the time frame in which the customer needs to provide the phone number

When completed, the new trigger resembles Figure 6-1 on page 127.

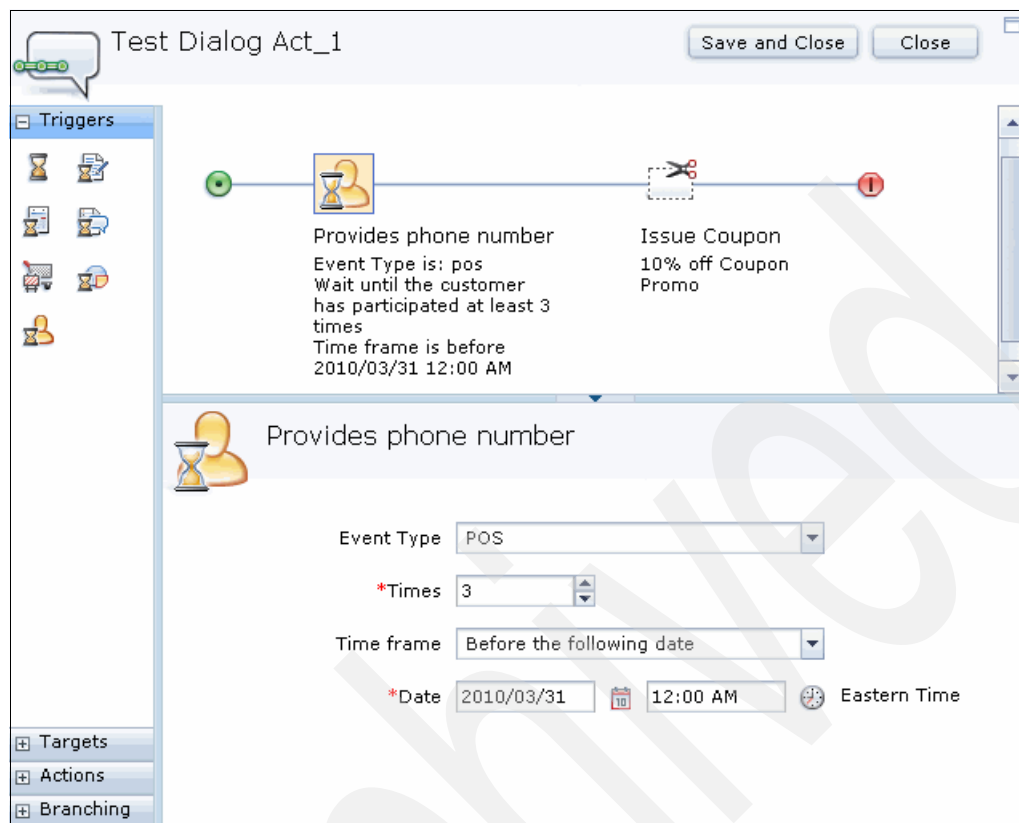


Figure 6-1 Customer provides phone number trigger

6.2 Design

We discuss the design of the trigger in this section. We also list the high level steps needed to create the trigger. Many of the steps are similar to those of creating a new target, as mentioned in 4.2, “Design” on page 56. However, there are some differences, which are noted.

The following steps are used to create a trigger. The steps are customized for our needs in this scenario:

1. Steps to create the server side logic for the trigger
 - a. Create the trigger template definition.
 - b. Create the trigger task command.
 - c. Inform the marketing services of the external event for the trigger.
2. Steps for the User Interface (UI)
 - a. Define a resource bundle for the UI text elements.
 - b. Create the object definition for the trigger.
 - c. Create the properties view for the trigger.
 - d. Create the summary for the trigger.
 - e. Register the new libraries with the marketing UI.
 - f. Add the trigger to the element palette in the Activity Builder.
 - g. Register the serialization JSP file for the new trigger.

6.2.1 Triggering a template definition

Every new campaign element requires one or more template definitions. For this customization, only one template definition is necessary. We use all three XML fragments in the definition:

- ▶ Implementation definition
- ▶ Behavior rule
- ▶ Related rule

The name we are using for the trigger template definition is `providePhoneNumber`, which is stored in the `NAME` field of the `DMELETEEMPLATE` table. See 4.2.1, “Target template definitions” on page 56 or the Information Center for more information about the campaign element template definitions:

http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center_customization.doc/concepts/csbcusteletemp.htm

Implementation definition XML fragment

There are three types of triggers:

- ▶ Daily check
- ▶ Customer event
- ▶ Elapsed time

The trigger type determines when the trigger is processed as well as the function of the task command. A daily check trigger is processed once a day at a scheduled time. The task command of this type of trigger is required to call the `forwardTriggersForProcessing` method, which determines which customers have met the requirements and forwards the trigger for each customer. It can also perform validation of the input parameters. A customer event trigger is processed the next time the `SendMarketingTriggers` job runs after the defined event or customer behavior occurs. The task command for this trigger type is only required if input parameter validation is needed. The elapsed time trigger is processed the next time the `SendMarketingTriggers` job runs after the defined amount of time has passed and requires a task command for user interface validation. For more information about the types of triggers, read the Information Center:

http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center_customization.doc/concepts/tsbtypediaact.htm

The implementation definition defines the task command that is used to validate the input parameters. Because we are concerned with the customer behavior and want the trigger to be processed as soon as it occurs, the new trigger is the customer event type. Thus, the task command is only needed to validate the parameters when the business user activates the activity that is using the trigger. For more information about the implementation definition, see the Information Center:

http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center_customization.doc/concepts/csbimplementtar.htm

Behavior rule XML fragment

The behavior rule defines how to detect if the defined customer behavior has occurred and the command to call from the external application. The command name we are using is **CustomerProvidesPhoneNumber**, which is not an actual Commerce command, but is a name used to identify the invoking of the trigger. This command name is important, as it is used by the external application to notify the trigger that the event has occurred. The behavior rule also defines how the behavior is processed and recorded. This fragment is required because we are focused on the behavior of the customer, such as calling the call center or providing a phone number at the point of sale, and need to detect when the

proper behavior occurs. When the event is detected through an external application, the behavior is recorded. To detect if the event has occurred, the rule defines variable name-value pairs to match in the event.

Table 6-1 lists the variable name-value pairs needed for this trigger:

Table 6-1 Behavior rule variable name-value pairs

name	value	type
eventType	MARKETING_eventType	NVP
MARKETING_beforeAfterOnDate	MARKETING_date1	CURRENTDATE
MARKETING_beforeDate	MARKETING_date2	CURRENTDATE

The variable name-value pairs have three elements:

- ▶ Name
- ▶ Value
- ▶ Type

Depending on the type of event that the trigger needs to match, those elements have different meanings. In our case, we need to match the type of event that has occurred, either a customer calling the call center or providing a phone number at the point of sale. We are using the eventType name-value pair to match this event. The name is the name of the name-value pair to match and the value is the value of the name-value pair to match. For example, if the business user selects the call center event type, the MARKETING_eventType value is callCenter. Thus, eventType needs to equal that value for the trigger to occur.

We are also concerned with the date on which the event occurs. In this case, for example, the MARKETING_beforeAfterOnDate name is the operator that describes how the actual event date relates to the specified date. The value, for example, MARKETING_date1, is the date to match. So if the business user selects before the date as the operator and March 31, 2010 as the date to match, the actual date must occur before March 31, 2010 in order for the target to occur.

The other parameters needed are listed in Table 6-2.

Table 6-2 Behavior rule parameters

Parameter	Value	Description
command	CustomerProvidesPhoneNumber	Name of the command to match when the event has occurred.
action	record	Forwards the occurrence of the event to the marketing services so the behavior can be recorded for the customer.
comparison	recordAll	Matches any value in the actual name-value pair and each value is recorded in the user behavior.
maxSize	MARKETING_numberOfTimes	The maximum number of times the marketing services records a customer's behavior associated with a particular value.
maxTotalSize	MARKETING_numberOfTimes	The maximum number of times the marketing services records a customer's behavior in total.
relativeDays	MARKETING_daysOperator	This parameter specifies how many days relative to the current date the event must occur and is the operator selected by the user.

More information about the behavior rule is provided in the WebSphere Commerce V. 7 Information Center:

http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center_customization.doc/concepts/csbbehavior.htm

Related rule XML fragment

The related rule fragment is required to check if the desired event has occurred the specified number of times. It functions in conjunction with the behavior rule that detects that the event happened and records it, but does not check the number of times. The related rule follows through and checks how many times the event has occurred against the recorded data, and if it matches the rest of criteria, including the dates on which the event needs to occur.

The related rule defines a target task command that is used to perform the checks. We are able to reuse the existing user behavior target task command, **CustomerFilterUserBehaviorTargetTaskCmd**, which performs this task based on the parameters from the user interface for the trigger.

The related rule XML fragment uses the same format as the implementation definition XML fragment for a target as described in 4.2.1, “Target template definitions” on page 56. It defines arguments needed for the task command. The arguments needed for the related rule are listed in Table 6-3.

Table 6-3 Arguments in the related rule XML fragment

Name	Value
anyOrAll	any
containsOperator	=
dataList	*
numberOfTimesOperator	MARKETING_numberOfTimesOperator
numberOfTimes	MARKETING_numberOfTimes
daysOperator	MARKETING_daysOperator
daysValue	MARKETING_days
matchLocations	history
eventType	MARKETING_eventType

For more information about the related rule XML fragment, see the following Information Center article:

http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center_customization.doc/concepts/csbrelated.htm

6.2.2 Task command

The task command for the trigger must correspond to the command defined in the implementation definition. The task command ensures that the proper parameters are available when the dialog activity containing the trigger is activated.

CustomerProvidesPhoneNumberTriggerTaskCmd

This is the only task command used in this customization. It performs a validation on the parameters entered in the IBM Management Center for WebSphere Commerce when the activity is activated. If the parameters are not present or invalid, the activation fails.

6.2.3 External event notification

Triggers sometimes need to detect events and determine if the specific criteria has been met. When an event occurring outside of WebSphere Commerce is used in a trigger, the marketing services must be notified that the event has taken place. A Process MarketingTriggers service call must be made to record the user behavior for evaluation. There are various ways of calling the Process MarketingTriggers service, such as through a URL, using the processMarketingTrigger or the evaluateMarketingTrigger method in the MarketingFacadeClient. We use the URL to notify the services. The URL we are using is similar to the following:

```
http://localhost/webapp/wcs/stores/servlet/MarketingTriggerProcessServiceEvaluate?DM_ReqCmd=CustomerProvidesPhoneNumber&phoneNumber=2165554589&eventType=callCenter&URL=TopCategoriesDisplay?storeId=10051&catalogId=10051
```

The DM_ReqCmd parameter maps to the command name in the behavior rule in the campaign element template definition stored in DMELETEEMPLATE. The phone number parameter is used to identify the customer associated with the external event. However, the marketing services requires the customer's personalization ID to have this event affect any applicable dialog and Web activities for the customer. A custom method needs to be implemented that maps the information the external system has about the customer to the personalization ID assigned to the customer in the WebSphere Commerce database. The **MarketingServicesTaskCmdImpl** command needs to be extended to implement the getPersonalizationId method. This method can retrieve the personalization ID based on another piece of information about the customer (such as their logon ID, e-mail address, or phone number).

The code sample shown in Example 6-1 illustrates how to locate the customer's personalization ID based on the phone number provided in the call to the Process MarketingTrigger service.

Example 6-1 Customer's personalization ID for an external event

```
public class MyCompanyMarketingServicesTaskCmdImpl
extends MarketingServicesTaskCmdImpl
implements MarketingServicesTaskCmd {

    /**
     * IBM copyright notice field.
     */
    public static final String COPYRIGHT =
com.ibm.commerce.copyright.IBMCopyright.SHORT_COPYRIGHT;
    /**
```

```

    * The name of this class.
    */
    public static final String CLASSNAME =
"com.mycompany.commerce.marketing.dialog.util.MyCompanyMarketingServicesTaskCmdImpl";
    private static final Logger LOGGER =
LoggingHelper.getLogger(MyCompanyMarketingServicesTaskCmdImpl.class);

    /**
    * This method will find a customer's personalization ID based on the provided
    phone number.
    * @param triggerParameters: The trigger parameters passed in the service call.
    This should
    * include a parameter named phone. The external system has the customer's phone
    number,
    * and it is assumed that the customer has provided the same number in their
    WebSphere Commerce
    * account. This method maps the customer's phone number to their personalization
    ID.
    * @return: The customer's personalization ID.
    */
    public String getPersonalizationId(StringBuffer triggerParameters) {
        final String METHOD_NAME = "getPersonalizationId";
        if (LoggingHelper.isEntryExitTraceEnabled(LOGGER)) {
            LOGGER.entering(CLASSNAME, METHOD_NAME);
        }
        String pznId = null;
        try {
            String phoneNumber =
MarketingUtil.getDataFromTriggerParametersString(triggerParameters.toString(),
"phone");
            if (LoggingHelper.isTraceEnabled(LOGGER)) {
                LOGGER.log(Level.FINE, CLASSNAME, METHOD_NAME, "phoneNumber:" +
phoneNumber);
            }
            if (phoneNumber != null) {

                List pznIdList = new
ServerJDBCHelperAccessBean().executeParameterizedQuery(
                    "select users.personalizationid from users, address where
users.users_id = address.member_id and address.phone1 = ?",
                    new Object[]{ phoneNumber } );
                if (LoggingHelper.isTraceEnabled(LOGGER)) {
                    LOGGER.log(Level.FINE, CLASSNAME, METHOD_NAME, "pznIdList:" +
pznIdList);
                }
            }
        }
    }

```

```

        if (pznIdList != null && pznIdList.size() == 1) {
            List pznIdEntry = (List)pznIdList.get(0);
            pznId = pznIdEntry.get(0).toString();
        }
        if (LoggingHelper.isTraceEnabled(LOGGER)) {
            LOGGER.log(Level.FINE, CLASSNAME, METHOD_NAME, "pznId:" + pznId);
        }
    }
} catch (Exception e) {
    if (LoggingHelper.isTraceEnabled(LOGGER)) {
        LOGGER.log(Level.FINE, CLASSNAME, METHOD_NAME, "Exception:" + e);
    }
    e.printStackTrace();
}
if (LoggingHelper.isEntryExitTraceEnabled(LOGGER)) {
    LOGGER.exiting(CLASSNAME, METHOD_NAME);
}
return pznId;
}
}

```

Note: To use the personalization feature, the personalization ID must be enabled in the WebSphere Commerce site. See the following Web page for information about enabling personalization:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.admin.doc/tasks/tseenablepersonalizationid.htm>

6.2.4 Object definition

Each campaign element is required to have an object definition which is used to collect input from the user interface in the IBM Management Center for WebSphere Commerce. See 4.2.3, “Object definitions” on page 60 for more information about object definitions. The object definition defines all of the parameters and their possible values that are configurable, including the *eventType*, *numberOfTimes*, and the date parameters. These parameters match those parameters in the behavior rule and related rule XML fragments which begin with *MARKETING_*. The *elemTemplateName* is also an important element of the object definition. This name must match the value in the NAME field in the DMELETEEMPLATE database table, which is *providePhoneNumber*. Default values can be provided in the *dataset* object, so we default the *eventType* to *callCenter*.

For more information about creating an object definition for a campaign element, read the section in Information Center:

http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center_customization.doc/tasks/tsbcustctobdef.htm.

6.2.5 Properties view

A properties view defines the properties that need to be captured. The properties used reference the parameters specified in the object definition. Because we allow the business user to select which event to capture, we need a drop down menu for the eventType property. We also need to add the numberOfTimes property as well as the date properties. The object that displays the date properties, mktBaseDateProperties, can be reused from the mktFlowElementProperties class. The date properties allow the target dates and date operators to be selected. Up to two dates can be chosen. Also, date operators determines how the actual date relates to the specified dates.

6.2.6 Summary definition

The summary definition displays a summary of the trigger as configured by the business user. We need to show the selected eventType, numberOfTimes, and date range of the trigger. As the user edits the trigger, the summary reflects those changes.

6.3 Implementation

This section provides detailed steps for creating the new trigger based on the design.

6.3.1 Creating the trigger template definition

A campaign element template definition must be created for the trigger. Example 6-2 shows the SQL statement used to insert the template definition into the table.

Example 6-2 Trigger element template definition

```
insert into dmeletemplate
(dmeletemplate_id, dmelementtype_id, name, implxml, behaviorxml,
relatedxml)
values (10003, 2, 'providePhoneNumber',
'<Trigger type="CustomerProvidesPhoneNumberTrigger">
<Implementation invocationType="TaskCommand">
<Class name="com.madisons.commerce.marketing.commands.elements
.CustomerProvidesPhoneNumberTriggerTaskCmd">
</Class>
</Implementation>
</Trigger>',
'<BehaviorRule command="CustomerProvidesPhoneNumber" action="record"
comparison="recordAll" maxSize="MARKETING_numberOfTimes"
maxTotalSize="MARKETING_numberOfTimes"
relativeDays="MARKETING_daysOperator">
<Variable name="eventType" value="MARKETING_eventType" type="NVP" />
<Variable name="MARKETING_beforeAfterOnDate" value="MARKETING_date1"
type="CURRENTDATE"/>
<Variable name="MARKETING_beforeDate" value="MARKETING_date2"
type="CURRENTDATE"/>
</BehaviorRule>',
'<FlowElementImplementation type="Customer Provides Phone Number">
<Implementation invocationType="TaskCommand">
<Class name="com.ibm.commerce.marketing.commands.elements
.CustomerFilterUserBehaviorTargetTaskCmd"><Argument name="anyOrAll"
value="any" />
<Argument name="containsOperator" value="=" />
<Argument name="dataList" value="*" />
<Argument name="numberOfTimesOperator"
value="MARKETING_numberOfTimesOperator" />
<Argument name="numberOfTimes" value="MARKETING_numberOfTimes" />
```

```
<Argument name="daysOperator" value="MARKETING_daysOperator" />
<Argument name="daysValue" value="MARKETING_days" />
<Argument name="matchLocations" value="history" />
<Argument name="eventType" value="MARKETING_eventType" />
</Class>
</Implementation>
</FlowElementImplementation>');
```

6.3.2 Creating the CustomerProvidesPhoneNumberTriggerTaskCmd interface

The task command corresponds with the command referenced in the implementation definition:
com.madisons.commerce.marketing.commands.elements.CustomerProvidesPhoneNumberTriggerTaskCmd.

Creating the interface

Perform the following steps to create the interface.

1. In Rational Application Developer for WebSphere Software, click **File** → **New** → **Other**.
2. In the next window, select the Interface wizard and click **Next**.
3. Ensure WebSphereCommerceServerExtensionsLogic/src is entered in the Source folder field.
4. In the Package field, type:
`com.[your_company_name].commerce.marketing.commands.elements`
5. Enter CustomerProvidesPhoneNumberTriggerTaskCmd in the name field.
6. Add MarketingCampaignElementTaskCmd as the Extended interface and click **Finish**.
7. In the Java file, set the defaultCommandClassName constant to the name of the CustomerProvidesPhoneNumberTriggerTaskCmdImpl class that is created next.

Example 6-3 on page 139 shows the new interface.

Example 6-3 CustomerProvidesPhoneNumberTriggerTaskCmdImpl interface

```
package com.madisons.commerce.marketing.commands.elements;

import com.ibm.commerce.marketing.commands.elements
    .MarketingCampaignElementTaskCmd;

public interface CustomerProvidesPhoneNumberTriggerTaskCmd
    extends MarketingCampaignElementTaskCmd {

    public static final String defaultCommandClassName =
        CustomerProvidesPhoneNumberTriggerTaskCmdImpl.class
            .getName();
}
```

8. Save and close the file.

Creating the CustomerProvidesPhoneNumberTriggerTaskCmdImpl class

Perform the following steps to create the CustomerProvidesPhoneNumberTriggerTaskCmdImpl class:

1. Click **File** → **New** → **Other**.
2. In the next window, select the Class wizard and click **Next**.
3. Ensure WebSphereCommerceServerExtensionsLogic/src is entered in the Source folder field.
4. In the Package field, type:
com.[your company name].commerce.marketing.commands.elements
5. Enter CustomerProvidesPhoneNumberTriggerTaskCmdImpl in the name field.
6. Browse and enter MarketCampaignElementTaskCmdImpl. Select Superclass and click **OK**.
7. Add CustomerProvidesPhoneNumberTriggerTaskCmd as the Interface and click **Finish**.

Example 6-4 shows the class declaration.

Example 6-4 CustomerProvidesPhoneNumberTriggerTaskCmdImpl class declaration

```
public class CustomerProvidesPhoneNumberTriggerTaskCmdImpl
    extends CustomerFilterUserBehaviorTargetTaskCmdImpl
    implements CustomerProvidesPhoneNumberTriggerTaskCmd {
    ...
}
```

Example 6-5 shows the constants used in the task command.

Example 6-5 Constant declarations

```
private static final Logger LOGGER = LoggingHelper.getLogger(  
    CustomerProvidesPhoneNumberTriggerTaskCmdImpl.class);  
private static final String CLASSNAME =  
    "CustomerCallsCallCenterTriggerTaskCmdImpl";  
  
private static final String NUMBER_OF_TIMES_MISSING =  
    "_ERR_NUMBER_OF_TIMES_MISSING";  
private static final String PARAM_NUMBER_OF_TIMES = "numberOfTimes";
```

The task command implements the `validateParameters` method, which ensures the existence of the proper parameters that are set in the user interface. Example 6-6 shows the `validateParameters` method.

Example 6-6 validateParameters method in CustomerProvidesPhoneNumberTriggerTaskCmdImpl

```
public List validateParameters(Map elementParameters)  
{  
    final String METHOD_NAME = "validateParameters";  
  
    if(LoggingHelper.isEntryExitTraceEnabled(LOGGER))  
    {  
        LOGGER.entering(CLASSNAME, METHOD_NAME, elementParameters);  
    }  
  
    List validationErrors = new ArrayList();  
  
    // log an error if the parameter numberOfTimes is not present  
    Object numberOfTimes =  
        elementParameters.get(PARAM_NUMBER_OF_TIMES);  
    if(numberOfTimes == null  
        || numberOfTimes.toString().length() == 0)  
    {  
        ApplicationError validateError = new ApplicationError(  
            ApplicationError.TYPE_GENERIC_ERROR,  
            NUMBER_OF_TIMES_MISSING, null,  
            LOGGER.getResourceBundleName());  
        validationErrors.add(validateError);  
    }  
  
    // log an error if the parameters associated with the  
    // dates are not present
```

```

        validateDates(elementParameters, validationErrors);

        if(LoggingHelper.isEntryExitTraceEnabled(LOGGER))
        {
            LOGGER.exiting(CLASSNAME, METHOD_NAME, elementParameters);
        }

        return validationErrors;
    }

```

8. Save and close the file.

6.3.3 Setting up external system to inform the marketing services of the external event

The external system must be configured with the URL that notifies the marketing services that the event has occurred. The specifics of this system is out of scope for this scenario, but it can detect when either a customer has called the call center or provided a phone number at the POS. At this point, the system calls the marketing services through an URL. As mentioned, the URL is similar to the following example:

```

http://hostname:port/webapp/wcs/stores/servlet/MarketingTriggerProcessServiceEvaluate?DM_ReqCmd=CustomerProvidesPhoneNumber&phoneNumber=2165554589&eventType=callCenter&URL=TopCategoriesDisplay?storeId=10051&catalogId=10051

```

The system calls the **CustomerProvidesPhoneNumber** command and provides the user information as well as the eventType. The system does not need to know which eventType is actually being targeted. It only needs to know that one of the events has occurred.

This completes the server side logic of the customization. In the next section, we create the files needed for the UI.

6.3.4 Defining a resource bundle and properties file for UI text

Perform the following steps to define a resource bundle.

1. Create a new Java package inside LOBTools/src. Right-click Java Resources: src inside the LOBTools project. Click **New** → **Package**. Enter `com.<your_company_name>.marketing.client.lobtools.properties` in the name field and click **Finish**.
2. Create a new properties file. Right-click the new package and select **New** → **File** and enter the name `MarketingLOB.properties` and click **Finish**.
3. Enter the name `MarketingLOB.properties` and click **Finish**.
4. Example 6-7 shows the text to enter in the properties file. If a custom properties file was created for another customization, you can use that file instead and add the lines in Example 6-7 at the end of the existing file.

Example 6-7 Custom properties file for the trigger

```
providePhoneNumberName=Provides phone number
eventType = Event Type
callCenter = Call Center
pos = POS
```

5. Save the file.
6. Create a new file in the following directory:

```
LOBTools/WebContent/WEB-INF/src/lzx/<your_company_name>/marketing
called <your_company_name>MarketingResourceBundle.lz
```

Example 6-8 shows the contents of the new file. It is possible to reuse the custom resource bundle file if it was created for another customization. To do so, add the lines in bold into the existing file.

Example 6-8 Custom resource bundle

```
<library>

<class name="extMarketingResourceBundle"
  extends="wcfResourceBundle"
  baseName="com.madisons.marketing.client.lobtools.properties
    .MarketingLOB">

  <wcfResourceBundleKey name="providePhoneNumberName" />
  <wcfResourceBundleKey name="eventType" />
  <wcfResourceBundleKey name="callCenter" />
  <wcfResourceBundleKey name="pos" />
```

```

</class>

<extMarketingResourceBundle id="extMarketingResources" />

</library>

```

7. Save and close the file.
8. To use the properties defined in this file, the resource bundle id, extMarketingResources is used.

6.3.5 Creating the object definition for the trigger

Perform the following instructions to create the object definition for the trigger:

1. Create a new OpenLaszlo file called ProvidePhoneNumberFlowElementObjectDefinition.lzx in the LOBTools/WebContent/WEB-INF/src/lzx/<your_company_name>/marketing/objectDefinitions/activityBuilder/ directory.
2. Copy the content in Example 6-9 into the new file.

Example 6-9 The object definition for the Customer Provides Phone Number Trigger

```

<library>
  <class
    name="extProvidePhoneNumberElementObject"
    extends="mktFlowElementObjectDefinition"
    objectType="providePhoneNumber"
    displayName=
      "${extMarketingResources
        .providePhoneNumberName.string}"
    headerIcon="genericTriggerHeaderIcon"
    flowIcon="genericTriggerIcon"
    paletteIcon="genericTriggerPaletteIcon"
    propertiesClass="extProvidePhoneNumberProperties"
    summaryClass="extProvidePhoneNumberSummary"
    elemTemplateType="Trigger">

    <mktFlowElementCreateService />
    <mktFlowElementUpdateService />

    <dataset name="template">
      <elemTemplateName>providePhoneNumber</elemTemplateName>
      <numberOfTimes>1</numberOfTimes>
      <daysOperator></daysOperator>
      <days></days>
    </dataset>
  </class>
</library>

```

```

        <beforeAfterOnDate></beforeAfterOnDate>
        <date1></date1>
        <beforeDate></beforeDate>
        <date2></date2>
        <eventType>callCenter</eventType>
    </dataset>

    <wcfPropertyDefinition propertyName="eventType"
        displayName="{extMarketingResources.eventType.string}">
        <wcfPropertyValue
            displayName="{extMarketingResources
                .callCenter.string}" value="callCenter" />
        <wcfPropertyValue
            displayName="{extMarketingResources
                .pos.string}" value="pos" />
    </wcfPropertyDefinition>

    <wcfPropertyDefinition propertyName="numberOfTimes"
        required="true" type="number" minValue="1"
        maximumSize="5"
        displayName="{mktMarketingResources.times.string}">
    </wcfPropertyDefinition>

    <wcfPropertyDefinition propertyName="days" required="true"
        type="number" maximumSize="5"
        displayName="{mktMarketingResources.days.string}">
        <wcfEnablementOrCondition conditionId="5">
            <wcfEnablementCondition conditionId="51"
                propertyName="daysOperator" enablementValue="&lt;=" />
            <wcfEnablementCondition conditionId="52"
                propertyName="daysOperator" enablementValue="&gt;" />
        </wcfEnablementOrCondition>
    </wcfPropertyDefinition>

    <wcfPropertyDefinition propertyName="date1" required="true"
        displayName="{mktMarketingResources.date1.string}">
        <wcfEnablementOrCondition conditionId="6">
            <wcfEnablementCondition conditionId="61"
                propertyName="beforeAfterOnDate"
                enablementValue="&gt;=" />
            <wcfEnablementCondition conditionId="62"
                propertyName="beforeAfterOnDate"
                enablementValue="&gt;" />
            <wcfEnablementCondition conditionId="63"
                propertyName="beforeAfterOnDate"

```



```

        enablementValue="="/>
        <wcfEnablementCondition conditionId="64"
            propertyName="beforeAfterOnDate"
            enablementValue="&lt;"/>
    </wcfEnablementOrCondition>
</wcfPropertyDefinition>

<wcfPropertyDefinition propertyName="date2" required="true"
    displayName="{mktMarketingResources.date2.string}">
    <wcfEnablementAndCondition conditionId="7">
        <wcfEnablementCondition conditionId="71"
            propertyName="beforeAfterOnDate"
            enablementValue="&gt;="/>
        <wcfEnablementCondition conditionId="72"
            propertyName="beforeDate" enablementValue="&lt;="/>
    </wcfEnablementAndCondition>
</wcfPropertyDefinition>

<wcfStartDateEndDateValidator validatorId="dateValidator"
    startDatePropertyName="date1" endDatePropertyName="date2"/>
</class>

```

This file defines all of the elements used in the trigger. The propertiesClass and summaryClass mentioned in the object definition is created in the upcoming sections.

Although the icons for the palette, work area, and properties view can be created to represent the trigger, we use the generic trigger icons in this customization. The icons are set in the object definition and are as follows:

- genericTriggerHeaderIcon
- genericTriggerIcon
- genericTriggerPalettIcon

For information about how to define custom icons, read the following section in Information Center:

http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center_customization.doc/tasks/tsbcustpalicons.htm

3. Save and close the file.
4. Register the provide phone number trigger object definition in the parent definition. Open the following file:

LOBTools/WebContent/WEB-INF/src/lzx/commerce/marketing/objectDefinitions/activityBuilder/FlowPathElementObjectDefinition.lzx

5. Search for the `mktBaseFlowPathElementObject` element and add the `extProvidePhoneNumberElementObject` object definition, as shown in Example 6-10.

Example 6-10 Registering the trigger object in the base flow element object definition

```
...
<class name="mktBaseFlowPathElementObject"
      extends="wcfChildObjectDefinition"
      displayName="${mktMarketingResources.path.string}"
      isBaseDefinition="true">
...
<extProvidePhoneNumberElementObject />
...
```

6. Save the file.

6.3.6 Creating the properties view for the trigger

Perform the following steps to create the properties view for the trigger:

1. Create an OpenLaszlo file called `ProvidePhoneNumberPropertiesView.lzx` in the following directory:

LOBTools/WebContent/WEB-INF/src/lzx/<your_company_name>/marketing/pr
opertiesViews/activityBuilder

Example 6-11 shows the content of the newly created file.

Example 6-11 The provides phone number trigger properties view

```
<library>
<!--
  Properties view for the {@link extProvidePhoneNumberElementObject
  Customer Provides Phone Number trigger}.
-->
<class name="extProvidePhoneNumberProperties"
      extends="mktFlowElementProperties">
  <wcfPropertyGroup name="container" collapsable="false">
    <wcfPropertyCombobox
      promptText="${extMarketingResources.eventType.string}"
      propertyName="eventType"/>
    <wcfPropertyStepper minimumValue="1" maximumValue="5"
      propertyName="numberOfTimes"
      promptText="${mktMarketingResources.times.string}" />
    <mktBaseDateProperties/>
  </wcfPropertyGroup>
</class>
</library>
```

2. Save and close the file.

The order of the elements in the properties view dictates where the elements are in the UI. In this case, the eventType combo box appears first, followed by the numberOfTimes element and the date properties.

6.3.7 Creating the summary for the trigger

Perform the following steps to create the summary for the trigger:

1. Create a new OpenLaszlo file called ProvidePhoneNumberSummary.lzx in the following directory:

LOBTools/WebContent/WEB-INF/src/lzx/<your_company_name>/marketing/propertiesViews/activityBuilder/.

The content for the file is shown in Example 6-12.

Example 6-12 The summary definition for the provides phone number trigger

```
<library>
  <class name="extProvidePhoneNumberSummary"
    extends="mktFlowElementSummary">
    <mktFlowSummaryParam name="elemTemplateName"
      propertyName="elemTemplateName"/>
    <mktFlowSummaryParam name="daysOperator"
      propertyName="daysOperator"/>
    <mktFlowSummaryParam name="days" propertyName="days"/>
    <mktFlowSummaryParam name="date1" propertyName="date1"/>
    <mktFlowSummaryParam name="date2" propertyName="date2"/>
    <mktFlowSummaryParam name="beforeDate"
      propertyName="beforeDate"/>
    <mktFlowSummaryParam name="beforeAfterOnDate"
      propertyName="beforeAfterOnDate"/>
    <mktFlowSummaryParam name="numberOfTimes"
      propertyName="numberOfTimes"/>
    <mktFlowSummaryParam name="eventType"
      propertyName="eventType"/>

    <method name="updateSummary" args="e">
      <![CDATA[
        var summaryText = "";
        var dateText = this.getDateSummary();
        var numberText = this.getNumberSummary();
        var eventText = this.getEventType();

        if(eventText != "") {
```

```

        summaryText = eventText;
        if(numberText != "") {
            summaryText += "\n" + numberText;
            if(dateText != "") {
                summaryText += "\n" + dateText;
            }
        }
    }

    this.setSummaryText(summaryText);
    ]]>
</method>

<method name="getEventType">
    <![CDATA[
        var summary = "";

        if ((this.resolvedParams["eventType"] != null) &&
            (this.resolvedParams["eventType"] != "")) {
            summary = "Event Type is: " +
                this.resolvedParams["eventType"];
        }

        return summary;
    ]]>
</method>

<!-- @keywords private -->
<method name="getDateSummary">
    <![CDATA[
        var summary = "";

        if (this.resolvedParams["daysOperator"] == "<=") {
            if(this.resolvedParams["days"] != "") {
                var days =
                    wcfGlobalizationUtil
                        .formatNumberToDisplayFormat(
                            this.resolvedParams["days"]);
                summary = mktMarketingResources.replaceValues(
                    "summaryPerformedWithinLastXDays", [days]);
            }
        }else if (this.resolvedParams["daysOperator"] == ">") {
            if(this.resolvedParams["days"] != "") {
                var days =
                    wcfGlobalizationUtil

```

```

        .formatNumberToDisplayFormat(
            this.resolvedParams["days"]);
        summary = mktMarketingResources.replaceValues(
            "summaryPerformedAtLeastXDaysAgo", [days]);
    }
} else if (this.resolvedParams["beforeAfterOnDate"] ==
"<") {
    if(this.resolvedParams["date1"] != "") {
        var date1 = wcfDateUtil.formatUI(
            wcfDateUtil.parse(
                this.resolvedParams["date1"],
                wcfDateUtil.DATE_TIME_FORMAT),
            wcfDateUtil.UI_DATE_TIME_FORMAT);
        summary = mktMarketingResources.replaceValues(
            "summaryPerformedBeforeXDate", [date1]);
    }
} else if (this.resolvedParams["beforeAfterOnDate"] ==
">") {
    if(this.resolvedParams["date1"] != "") {
        var date1 = wcfDateUtil.formatUI(
            wcfDateUtil.parse(
                this.resolvedParams["date1"],
                wcfDateUtil.DATE_TIME_FORMAT),
            wcfDateUtil.UI_DATE_TIME_FORMAT);
        summary = mktMarketingResources.replaceValues(
            "summaryPerformedAfterXDate", [date1]);
    }
} else if (this.resolvedParams["beforeAfterOnDate"] ==
">=" && this.resolvedParams["beforeDate"] ==
"<=") {
    if(this.resolvedParams["date1"] != "" &&
        this.resolvedParams["date2"] != "") {
        var date1 = wcfDateUtil.formatUI(
            wcfDateUtil.parse(
                this.resolvedParams["date1"],
                wcfDateUtil.DATE_TIME_FORMAT),
            wcfDateUtil.UI_DATE_TIME_FORMAT);
        var date2 = wcfDateUtil.formatUI(
            wcfDateUtil.parse(
                this.resolvedParams["date2"],
                wcfDateUtil.DATE_TIME_FORMAT),
            wcfDateUtil.UI_DATE_TIME_FORMAT);
        summary = mktMarketingResources.replaceValues(
            "summaryPerformedBetweenXandYDates",
            [date1, date2]);
    }
}

```

```

    }
    }else if (this.resolvedParams["beforeAfterOnDate"] ==
        "=") {
        if(this.resolvedParams["date1"] != "") {
            var date1 = wcfDateUtil.formatUI(
                wcfDateUtil.parse(
                    this.resolvedParams["date1"],
                    wcfDateUtil.DATE_TIME_FORMAT),
                    wcfDateUtil.UI_DATE_FORMAT);
            summary = mktMarketingResources.replaceValues(
                "summaryPerformedOnXDate", [date1]);
        }
    }

    return summary;
]]>
</method>

<!-- @keywords private -->
<method name="getNumberSummary">
    <![CDATA[
        var summary = "";

        if(this.resolvedParams["numberOfTimes"] &&
            (this.resolvedParams["numberOfTimes"] != "")) {
            var numberOfTimes = wcfGlobalizationUtil
                .formatNumberToDisplayFormat(
                    this.resolvedParams["numberOfTimes"]);
            summary = mktMarketingResources.replaceValues(
                "summaryPerformedWaitAtLeast", [numberOfTimes]);
        }

        return summary;
    ]]>
</method>
</class>
</library>

```

2. Save and close the file.

This file checks which options have been selected and dynamically determines which text to display based on those selections.

6.3.8 Registering the new classes in the marketing extensions library

After creating the new classes for the trigger, they need to be registered and defined in the marketing extension library. Doing so ensures that the new files are available to the UI.

Open the
LOBTools/WebContent/WEB-INF/src/lzx/commerce/marketing/MarketingExtensionsLibrary.lzx file.

The snippet in Example 6-13 shows the content of the file.

Example 6-13 Registering the new classes in the marketing extensions library

```
<library>
...
<include href="../../madisons/marketing
/MadisonsMarketingResourceBundle.lzx"/>
<include href="../../madisons/marketing/objectDefinitions
/activityBuilder
/ProvidePhoneNumberFlowElementObjectDefinition.lzx"/>
<include href="../../madisons/marketing/propertiesViews
/activityBuilder
/ProvidePhoneNumberPropertiesView.lzx"/>
<include href="../../madisons/marketing/propertiesViews
/activityBuilder/ProvidePhoneNumberSummary.lzx"/>
...
</library>
```

6.3.9 Adding the trigger to the element palette in the Activity Builder

After creating all of the classes, the trigger needs to be added to the element palette in the Activity Builder so it can be used in activities. Table 4-5 on page 92 shows the four types of Activity Builders and the associated files. Triggers can only be used in dialog activities, so only those files related to the Dialog Activity Builder are relevant.

1. Open either the Dialog Activity Builder file or the Dialog Activity Template Builder file.
2. To add the trigger to the element palette, add the following text to the file:

```
<Element objectType="providePhoneNumber"/>
```

The snippet in Example 6-14 shows where to add the snippet in the context of the Activity Builder file.

Example 6-14 Adding the new wish list target to the Dialog Activity Builder

```
<dataset name="palette">
  <Group resourceBundle="mktMarketingResources" name="triggers"
    helpText="help_palette_triggers"
    helpLink="concepts/csbtriggers.htm">
    ...
    <Element objectType="providePhoneNumber"/>
    ...
  </Group>
  ...
</dataset>
```

6.3.10 Registering the serialization JSP for the trigger

Every campaign element requires a serialization JSP to retrieve the element's attributes. Because the provides phone number trigger does not use any reference objects, a generic serialization JSP can be used instead of creating a new one. The JSP needs to be registered by creating a struts action mapping.

1. Open the LOBTools/WebContent/WEB-INF/struts-extension.xml file.
2. Add the content from Example 6-15 in the <action-mappings> section of the file.

Example 6-15 Struts action mapping for the serialization JSP

```
<action
  path="/SerializeActivityElement-providePhoneNumber"
  include="/jsp/commerce/marketing/restricted
    /SerializeActivitygenericElement.jsp" />
```

6.4 Test your code

After all of the steps have been completed, the projects can be built, the code tested, and deployed to production. Perform the instructions within the following paragraphs to test the customizations.

To test that the customizations have been completed successfully, the IBM Management Center for WebSphere Commerce UI changes needs to be verified. Log into the IBM Management Center for WebSphere Commerce and check to see if the new trigger icon is displayed. The icon is the last in the list of icons in the trigger palette.

Next, create a new dialog activity and select the new trigger as the first element in the activity. Configure the trigger to catch customers who have provided their phone number at POS at least 2 times before 1/31/2010. This is the criteria we are looking to match in the test. Add an Issue Coupon action and save and activate the new dialog activity. For more detailed information about dialog activities, refer to Chapter 3, “Dialog activities” on page 37.

After creation, the Dialog activity looks similar to Figure 6-2.

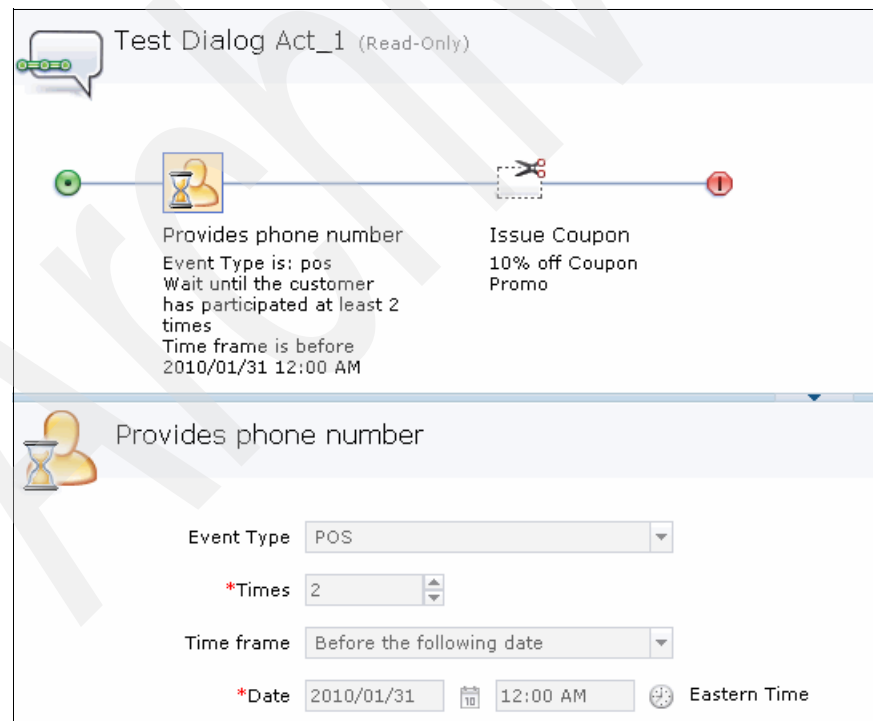


Figure 6-2 New Dialog activity using the provides phone number trigger

The next step is to mimic the occurrence by notifying the marketing services through the URL that was defined in 6.2.3, “External event notification” on page 133. The following URL notifies the marketing services that the desired action has taken place. Please note that the store ID and catalog ID must represent the storeId and catalogId of your store.

http://localhost/webapp/wcs/stores/servlet/MarketingTriggerProcessServiceEvaluate?DM_ReqCmd=CustomerProvidesPhoneNumber&phoneNumber=2165554589&eventType=pos&URL=TopCategoriesDisplay?storeId=10051&catalogId=10051

Table 6-4 lists the parameters needed for the URL for the trigger set up in this example.

Table 6-4 External event notification URL for the provides phone number trigger

Parameter	Value
DM_ReqCmd	CustomerProvidesPhoneNumber
phoneNumber	Customer's phone number
eventType	pos
URL	A valid store URL.

Each time the URL is called, the event is recorded for the customer. After the URL is called twice, the necessary criteria has been met and the customer continues participating in the dialog activity. The coupon is issued to the customer and added to their coupon wallet. The coupon can be accessed from the My Coupon page of the customer's account.

When all steps have been completed successfully, the trigger is ready for production.

For more tips on how to test triggers, see the following section in the Information Center:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center.doc/refs/rsbtestdia.htm>

Customize an existing action

This chapter provides a sample customization scenario to allow marketing managers to recommend catalog entries based on attributes. With this customization, attributes can be chosen to define a catalog entry recommendation and customers are able to see these recommendations on the store pages.

This chapter includes the following sections:

- ▶ Overview
- ▶ Design
- ▶ Implementation
- ▶ Test your code

7.1 Overview

To attract more customers to shop in the online store, the business user can create Web activities to display personalized e-marketing spots on the store pages. When a customer views these pages, the conditions in the Web activities control the logic of what to show the customer at that point of time. A business user can target sets of customers to personalize the content of what they see in the e-marketing spots. There are many ways you can create the recommendations to the customers, such as:

- ▶ Catalog Entry Recommendation
- ▶ Category Recommendation
- ▶ Merchandising Association Recommendation
- ▶ Promotion Recommendation
- ▶ Content Recommendation

In this chapter, we discuss the creation of a new mechanism to recommend catalog entries based on their attributes. This means, irrespective of the catalog entries in various categories or in various groups, we can recommend them to the customer if any of them share similar attributes as defined in the recommendation activity. Consider the following scenario to understand the requirements for this customization.

Karl is preparing the store for the upcoming holiday season. He wants to clear out catalog entries that have reported minimal sales during the year, some of which are nearing their expiry. He tried using the catalog entry recommendations Web activities, but the catalog entries he wanted to recommend for clearance were spread over many categories. He was looking for a feature in Management Center where he can tag all clearance catalog entries with some code name and then use this code name to create the catalog entry recommendation activities. He consults his IT department to implement such a customization to recommend catalog entries based on their attribute values. The requirements are as follows:

- ▶ Change the user interface (UI) for Recommend Catalog Entry action in the IBM Management Center for WebSphere Commerce to provide an option to choose between catalog entries or attribute values. This way, Karl can reuse the same Web activity window to define the catalog entry recommendation with attributes. This requires minimal UI changes for the IT department to enhance the existing recommend catalog entry action.
- ▶ When Karl selects the option to choose catalog entry attributes for the recommendation, the UI accepts the attribute name, their values, and a numeric value to restrict the total number of recommendations to be displayed in the e-marketing spot.

- ▶ Karl can add multiple attribute values for an attribute name. For example, he can define the attribute name as Color and also specify two values for it: Red and Black (all case insensitive). Also, he can specify the total number of records as 5.
- ▶ According to the definition, all catalog entries that have the color attribute with the values of red or black are retrieved. The resulting list of catalog entries is ordered based on the required logic (random, group by price, manufacture, ratings) and five records are selected to display in the e-marketing spot.

The new UI for Recommend Catalog Entry with Attributes resembles Figure 7-1, Figure 7-2 on page 158, and Figure 7-3 on page 159.

The screenshot shows a web application interface for 'Recommend Catalog Entry'. At the top, there is a breadcrumb trail with icons and text: 'E-Marketing Spot' (Customer views e-Marketing Spot: ShoppingCartRight SideBarAds) and 'Recommend Catalog Entry'. Below this, the main heading is 'Recommend Catalog Entry' with a catalog icon. A red box highlights the 'Criteria' section, which contains two radio buttons: 'Recommend catalog entries' (selected) and 'Recommend catalog entries using attributes'. Below the criteria section is a search bar with a 'Find and Add' button and three icons. A table labeled '*Catalog entries' has columns for '* Type', '* Code', and 'Name'. The table is currently empty, and the status at the bottom right indicates '0 to 0 of 0'.

Figure 7-1 A new option in Recommend Catalog Entry action to choose the criteria for recommending catalog entries based on their attributes

E-Marketing Spot
Customer views e-Marketing Spot:
ShoppingCartRight
SideBarAds

Recommend Catalog Entry
Recommend products with
attribute values: for the
attribute:

Recommend Catalog Entry

Criteria ☐ Recommend catalog entries
☒ Recommend catalog entries using attributes

*Attribute name

*Number of catalog entries to recommend

*Attribute values

0 to 0 of 0

Figure 7-2 The properties definition for the new criteria when selected

E-Marketing Spot
Customer views e-Marketing Spot: ShoppingCartRight SideBarAds

Recommend Catalog Entry
Recommend 5 products with attribute values: Red, Black for the attribute: Color

Recommend Catalog Entry

Criteria ☐ Recommend catalog entries
☒ Recommend catalog entries using attributes

*Attribute name

*Number of catalog entries to recommend

*Attribute values

* Attribute values
Red
Black

1 to 2 of 2

Figure 7-3 This window shows all the required values expected by this action

After this customization is completed, Karl can create attribute-based catalog entry recommendations. A custom task command must be created to filter the catalog entries based on the chosen attributes and create the e-marketing spot content. The next two sections describe the overall design and implementation in detail.

7.2 Design

In this section, we discuss the design required to define a catalog entry recommendation with attributes and the high level steps involved for creating it.

These are the steps to implement catalog entry recommendations with attributes:

1. Steps for the server side logic of the customer segment:
 - a. Define a template definition for the catalog entries with attribute display.
 - b. Create the new display catalog entry attribute action task command.
 - c. Register the task command.
 - d. Create a template definition for the new template.
2. Steps for the IBM Management Center for WebSphere Commerce User Interface (UI)
 - a. Define the resource bundle for the UI text elements.
 - b. Update the existing object definition of the catalog entry recommendations action to include the new child objects.
 - c. Updated the properties view of the catalog entry recommendations action to include the new properties.
 - d. Create a new summary definition for the changed element properties.
 - e. Register the new summary definition with the marketing UI.
 - f. Create and register serialization JSP for the new elements.

7.2.1 Task command

For the existing catalog entry recommendation feature, WebSphere Commerce has the **DisplayProductActionTaskCmd** task command to find the catalog entries specified in the activity definition and add them to the e-marketing spot. For our requirements, we can use similar logic, but instead of getting the catalog entries from the activity definition, we search the catalog entries based on the provided attribute name and values. After the list of catalog entries is retrieved, randomly pick N catalog entries to display in the e-Marketing Spot. To do that, create a new task command, **DisplayProductAttributeActionTaskCmdImpl**, which extends from the **MarketingCampaignElementTaskCmdImpl** task command. This is a marketing campaign command that determines the association promotions with Web activities at runtime.

For more information regarding the API details and the methods to override, see the WebSphere Commerce Information Center:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.api.doc/com/ibm/commerce/marketing/commands/elements/MarketingCampaignElementTaskCmd.html>

7.2.2 Template definition

A template definition already exists for the behavior for the recommend catalog entry action (`displayProduct`). To simplify the customization, we create a new template for this requirement (*`displayProductWithAttrib`*). The new template defines how the action behaves when the user selects the option of recommending the catalog entry based on the attribute values. The old template definition remains untouched so that the existing behavior works the way it was working earlier.

7.2.3 Object definition

The existing object definition for the recommend catalog entry action needs to be updated to include the new parameters that the business user wants to enter through Management Center. The existing object definition requires a child object to see the attribute values entered when the user selects the template to recommend catalog entry using attribute values. Also, the attribute value parameters need to be passed along with the create service and update service so that those are available for the marketing runtime.

7.2.4 Properties view

The existing properties view for the recommend catalog entry action needs to be updated to include the new properties. The business user must be able to select the template type that he wants to use, whether to recommend the catalog entry by selecting the SKUs or to recommend the catalog entries by selecting the attribute name and the possible values that need to be matched. If the business user selects the recommending catalog entries based on the attributes template, he is prompted to enter the attribute name, the number of catalog entries that need to be recommended, and the list of attribute values that need to be matched.

7.2.5 Serialization JSP for the new properties

A new serialization JSP must be created to retrieve the new properties entered by the user.

7.3 Implementation

This section provides the details for implementing the new catalog entry recommendation action.

7.3.1 Creating the new display product attribute task command

Extend the `MarketingCampaignElementTaskCmdImpl` task command from the marketing element commands package to implement the new behavior for recommending the catalog entries with attributes. In the new command, override the `performExecute` method to create the custom e-marketing spot content. This is a call back method invoked by the runtime when this action gets executed in the Web activity workflow. An interface must also be created to externalize the command.

Creating the `DisplayProductAttributeActionTaskCmd` interface

Perform the following steps to create the `DisplayProductAttributeActionTaskCmd` interface.

1. In Rational Application Developer for WebSphere Software, click **File** → **New** → **Other**.
2. In the next window, select the Interface wizard and click **Next**.
3. Ensure `WebSphereCommerceServerExtensionsLogic/src` is entered in the Source folder field.
4. In the Package field, type:
`com.[your company name].commerce.marketing.commands`
5. Enter `DisplayProductAttributeActionTaskCmd` in the name field.
6. Enter `MarketingCampaignElementTaskCmd` in the extend interface field and click **Finish**.

Example 7-1 shows the new `DisplayProductAttributeActionTaskCmd` interface with the defined constants.

Example 7-1 The `DisplayProductAttributeActionTaskCmd` interface to externalize the task command

```
package com.madisons.commerce.marketing.commands;

import com.ibm.commerce.marketing.commands.elements
    .MarketingCampaignElementTaskCmd;

/**
 * This command interface extends from Marketing Campaign to
```

```

    * recommend the products based on their attributes.
    * @see MarketingCampaignElementTaskCmd
    */
public interface DisplayProductAttributeActionTaskCmd
    extends MarketingCampaignElementTaskCmd {

    /**
     * Default implementation command class for this interface.
     */
    String defaultCommandClassName =
        DisplayProductAttributeActionTaskCmdImpl.class.getName();

    /**
     * This attribute will be passed by marketing runtime as
     * per the definition for Product's attribute recommendation.
     * Used for attribute name.
     */
    String PARAM_CATENRTY_ATTRIBUTE_NAME = "attribName";

    /**
     * This attribute will be passed by marketing runtime as per the
     * definition for Product's attribute recommendation.
     * Used for attribute values. It can be comma separated values for
     * one or more attributes.
     */
    String PARAM_CATENTRY_ATTRIBUTE_LIST = "attribValueList";

    /**
     * This attribute will be passed by marketing runtime as per the
     * definition for Product's attribute recommendation.
     * Used for total number of random product recommendation required.
     */
    String PARAM_TOTAL_RESULTS_REQUIRED = "numProducts";
}

```

Creating the DisplayProductAttributeActionTaskCmdImpl class

Perform the following steps to create the DisplayProductAttributeActionTaskCmdImpl class.

1. In Rational Application Developer for WebSphere Software, click **File** → **New** → **Other**.
2. In the next window, select the Class wizard and click **Next**.
3. Ensure WebSphereCommerceServerExtensionsLogic/src is entered in the Source folder field.
4. In the Package field, type:
`com.[your company name].commerce.marketing.commands`
5. Enter DisplayProductAttributeActionTaskCmdImpl in the name field.
6. Add MarketingCampaignElementTaskCmdImpl as the extended Super class and add DisplayProductAttributeActionTaskCmd in the extended interface and click **Finish**.

Example 7-2 shows the new DisplayProductAttributeActionTaskCmdImpl class.

Example 7-2 The DisplayProductAttributeActionTaskCmdImpl task command to recommend product with attributes

```
package com.madisons.commerce.marketing.commands;

import com.ibm.commerce.marketing.commands.elements
    .MarketingCampaignElementTaskCmdImpl;

public class DisplayProductAttributeActionTaskCmdImpl
    extends MarketingCampaignElementTaskCmdImpl
    implements DisplayProductAttributeActionTaskCmd {
}
```

This command overrides the performExecute method, which retrieves the catalog entries matching the search criteria for specified attributes and values in the activity definition. With these catalog entries, it randomly picks N records and create the promotional contents for e-marketing spot. Example 7-3 on page 165 shows the performExecute method of this command.

```
/**
 * This method overrides the super class implementation with the
 * new behavior to recommends products with matching attributes and
 * the specified size.
 *
 * @see CatEntrySearchListDataBean
 */
public void performExecute() {
    final String METHODNAME = "performExecute";

    try{
        Map parameters = getElementParameters();
        String attributeName = (String)parameters.get(
            PARAM_CATENRTY_ATTRIBUTE_NAME);
        String attributeValues = (String)parameters.get(
            PARAM_CATENTRY_ATTRIBUTE_LIST);
        String numResults = (String)parameters.get(
            PARAM_TOTAL_RESULTS_REQUIRED);

        List attributeList = MarketingUtil.parseList(attributeValues);
        if(null != attributeValues){
            StringTokenizer st = new StringTokenizer(
                attributeValues, MarketingUtil.COMMA);
            while(st.hasMoreTokens()){
                attributeList.add(st.nextToken());
            }
        }
    }
    /**
     * Get the list of unique catalog Ids for the given
     * attribute and values.
     */
    List catalogIds = getCatalogsIdsForAttributes(
        attributeName, attributeList);

    /**
     * Create the product recommendation with specified number of
     * products in the list.
     */
    recommendProductsWithAttributes(catalogIds, numResults);

    // Command ran successfully
    setReturnValue(true);
} catch(Exception e){
```

```

        e.printStackTrace();
        ECTrace.trace(ECTraceIdentifiers.COMPONENT_MARKETING,
            getClass().getName(), METHODNAME, e.getMessage());
    }
    super.performExecute();
}

```

The performExecute method retrieves the parameters from the runtime and gets the attribute name, values, and total number of recommendations the business manager has configured in the Web activity definition. Using these attributes, this method perform a catalog search to get the matching catalog entries. Example 7-4 shows the getCatalogsIdsForAttributes(String, List) method to get the appropriate catalog entries.

Example 7-4 The getCatalogsIdsForAttributes(String, List) method to search for all qualified catalog entries matching to the attribute and values

```

/**
 * This method retrieves the catalog Ids from the specified
 * attributes and values.
 *
 * @param attributeName the attribute name
 * @param attributeList the list of attribute values
 * @return the list of catalog entry Ids
 *
 * @see CatalogEntrySearchDataBean
 */
protected List getCatalogsIdsForAttributes(String attributeName,
    List attributeList) throws Exception{
    final String METHODNAME = "getCatalogsIdsForAttributes(String,
        List)";

    List catalogIds = new ArrayList();
    CatEntrySearchListDataBean catEntrySearchDB = null;
    Iterator iterator = attributeList.iterator();
    while(iterator.hasNext()){
        /*
         * Initialize the catalog search data bean with attribute
         * parameters.
         * The search in not case sensitive and only for String type
         * attributes.
         */
        String attribute = (String)iterator.next();
        catEntrySearchDB = new CatEntrySearchListDataBean();
        catEntrySearchDB.setAttributeName1(attributeName);
        catEntrySearchDB.setAttributeValue1(attribute);
    }
}

```

```

        catEntrySearchDB.setAttributeValueCaseSensitive("no");
        catEntrySearchDB.setAttributeValueOperator1("EQUAL");
        catEntrySearchDB.setAttributeValueType1("STRING");
        DataBeanManager.activate(catEntrySearchDB);
        CatalogEntryDataBean[] catalogItems =
            catEntrySearchDB.getResultList();
        if(null != catalogItems && catalogItems.length > 0){
            for(int i = 0; i < catalogItems.length; ++i){
                if(!catalogIds.contains(catalogItems[i]
                    .getCatalogEntryID())){
                    catalogIds.add(catalogItems[i].getCatalogEntryID());
                }
            }
        }
    }
    return catalogIds;
}

```

After the list of catalog entries are created, it is passed to an another helper method to shuffle these catalog entries, select any N records, and create the content for the e-marketing spot in the store. Example 7-5 shows the `recommendProductsWithAttributes(List, String)` method to shuffle the records and creating the marketing content.

Example 7-5 The `recommendProductsWithAttributes(List, String)` method to create the dynamic product recommendations for the e-marketing spot

```

/**
 * This is a helper method to recommend the marketing contents for
 * the e-Spot.
 *
 * @param attributeList the list of catalog Ids
 * @param numResults total number of required recommendations
 *
 * @see EMarketingSpotDataBean
 */
protected void recommendProductsWithAttributes(List catalogIds,
        String numResults) throws Exception{

    final String METHODNAME =
        "recommendProductsWithAttributes(String, List, String)";

    int resultSize = Integer.parseInt(numResults);
    /**
     * Randomize the list of catalog Ids to to pick any N records
     * from top.

```

```

    * Change the code to use any other logic like picking products
    * from unique parent categories or products with in-range price
    * value to align with total shopping cart value.
    */
Collections.shuffle(catalogIds);
List recommendProductList = catalogIds.subList(0, resultSize);

if(null != recommendProductList) {
    Iterator catalogListIterator = recommendProductList
        .iterator();
    while (catalogListIterator.hasNext()) {
        String catalogEntryId = (String)catalogListIterator.next();
        EMarketingSpotDataBean emsDataBean = new
            EMarketingSpotDataBean(EMarketingSpotDataBean
                .DISPLAY_TYPE_CATALOG_ENTRY,
                catalogEntryId,
                getActivity(), getElementId(),
                getExperimentTestElements());
        addEMarketingSpotDataBean(emsDataBean);
    }
}
}

```

7. Save and close the Java file. This completes the steps required to create a task command to recommend the catalog entries with attributes.

7.3.2 Creating the template definition for the new template

A campaign template definition needs to be created for the new template in the recommend catalog entry action. Refer to the Information Center for more information about the campaign template definition:

http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center_customization.doc/concepts/csbcusteltemp.htm

Execute the SQL query in the Example 7-6 to create the template definition for the new template to be used in the Recommend catalog entry action.

Example 7-6 SQL query to insert the template definition

```

insert into dmeletemplate
(dmeletemplate_id, dmelementtype_id, name, implxml)
values (15000, 3, 'displayProductWithAttrib',
'<FlowElementImplementation type="ProductAttributes">

```



```
<Implementation invocationType="TaskCommand"><Class
name="com.madisons.commerce.marketing.commands.elements.DisplayProductA
ttributeActionTaskCmd"><Argument name="attribValueList"
value="MARKETING_attribValueList" /><Argument name="attribName"
value="MARKETING_attribName" /><Argument name="numProducts"
value="MARKETING_numProducts"
/></Class></Implementation></FlowElementImplementation>');
```

7.3.3 Defining a resource bundle and properties file for UI text

See 6.3.4, “Defining a resource bundle and properties file for UI text” on page 142 to create a new resource bundle.

1. Enter the snippet in Example 7-7 in the properties file and save it.

Example 7-7 Entries for a custom properties file

```
attributeValues=Attribute values
recommendCatEnt=Recommend catalog entries
recommendCatEntUsingAttrib=Recommend catalog entries using attributes
recommendCriteria=Criteria
attributeName=Attribute name
numProducts=Number of catalog entries to recommend
summaryDisplayProductWithAttributes=Display product with attributes
summaryDisplayProductWithAttributesContains=Recommend {0} products with
attribute values: {1} for the attribute: {2}
```

2. Update the LOBTools/WebContent/WEB-INF/src/lzx/madisons/marketing/MarketingResourceBundle.lzx file and enter the resource bundle keys in the file, as shown in Example 7-8.

Example 7-8 Resource bundle key definition for the new properties

```
<library>
  <class name="extMarketingResourceBundle"
    extends="wcfResourceBundle"
    baseName="com.madisons.marketing.client.lobtools.properties.MarketinLOB">
    .....

    <wcfResourceBundleKey name="attributeValues" />
    <wcfResourceBundleKey name="recommendCatEnt" />
    <wcfResourceBundleKey name="recommendCatEntUsingAttrib" />
    <wcfResourceBundleKey name="recommendCriteria" />
    <wcfResourceBundleKey name="attributeName" />
    <wcfResourceBundleKey name="numProducts" />
```

```

        <wcfResourceBundleKey
            name="summaryDisplayProductWithAttributes" />
        <wcfResourceBundleKey
            name="summaryDisplayProductWithAttributesContains" />

.....
    </class>
    <extMarketingResourceBundle id="extMarketingResources" />
</library>

```

7.3.4 Updating the object definition

The object definition for the existing recommend catalog entries action need to be updated to include the new attributes which capture the attribute details. Perform the following steps to do so.

1. Open the OpenLaszlo file
LOBTools/WebContent/WEB-INF/src/lzx/commerce/marketing/objectDefinitions/activityBuilder/DisplayProductFlowElementObjectDefinition.lzx. Update it to include a new child object to hold the attribute names, add the attribute names parameter list in the create service, and the update service. Remove the child object validator that mandates catalog entry objects as child objects to the top object. Remove this validator because if the user chooses the catalog entry recommendations using attributes, then no catalog entry child objects are part of the top object. The object definition looks like Example 7-9.

Example 7-9 The updated object definition for the recommend product action

```

<class name="mktDisplayProductElementObject"
    extends="mktFlowElementObjectDefinition"
    objectType="displayProduct"
displayName="{mktMarketingResources.displayProductName.string}"
    headerIcon="displayProductHeaderIcon"
    flowIcon="displayProductIcon"
    paletteIcon="displayProductPaletteIcon"
    propertiesClass="mktDisplayProductProperties"
    summaryClass="extDisplayProductSummary"
    elemTemplateType="Action"
    helpLink="concepts/csbactdiscatentry.htm">

    <mktFlowElementCreateService>
        <wcfServiceParam name="PROPERTY_catalogEntryIdList"
            objectPath="ChildCatentry/CatalogEntry"
            propertyName="catentryId" optional="true"/>
    </mktFlowElementCreateService>
</class>

```

```

        <wcfServiceParam name="PROPERTY_attribValueList"
            objectPath="attribValue" propertyName="attribValue"
            optional="true" />
    </mktFlowElementCreateService>

    <mktFlowElementUpdateService>
        <wcfServiceParam name="PROPERTY_catalogEntryIdList"
            objectPath="ChildCatentry/CatalogEntry"
            propertyName="catentryId" optional="true"/>
        <wcfServiceParam name="PROPERTY_attribValueList"
            objectPath="attribValue" propertyName="attribValue"
            optional="true" />
    </mktFlowElementUpdateService>

    <dataset name="template">
        <elemTemplateName>displayProduct</elemTemplateName>
    </dataset>

    <wcfReferenceObjectDefinition objectType="ChildCatentry"
        objectGroups="ChildCatentry"
        referencedTypes="Product,Bundle,CatalogGroupSKU,
            ProductSKU,Kit"
        idProperty="childCatentryId" allowDuplicates="false"
        derivedId="true"
        allowedParentTypes="WebActivity,WebActivityTemplate,
            DialogActivity,DialogActivityTemplate"/>
    <wcfReferenceObjectDefinition objectType="ChildInheritedCatentry"
        objectGroups="ChildCatentry"
        referencedTypes="InheritedProduct,InheritedBundle,
            InheritedCatalogGroupSKU,InheritedProductSKU,InheritedKit"
        idProperty="childCatentryId" allowDuplicates="false"
        derivedId="true"/>

    <wcfChildObjectDefinition objectType="attribValue"
        idProperty="attribValue"
        displayName="{extMarketingResources.attributeName.string}"
        creatable="true">
        <wcfPropertyDefinition propertyName="attribValue"
            required="true" maximumSize="254" trim="true"
            displayName="{extMarketingResources.attributeName.string}"/>
    </wcfChildObjectDefinition>

    <wcfPropertyDefinition propertyName="elemTemplateName">
        <wcfPropertyValue
            displayName="{extMarketingResources.recommendCatEnt.string}"

```

```

        value="displayProduct"/>
    <wcfPropertyValue
displayName="{extMarketingResources.recommendCatEntUsingAttrib.string}
"
        value="displayProductWithAttrib"/>
    </wcfPropertyDefinition>
</class>

```

2. Save the file.

7.3.5 Updating the properties view

The properties view must be updated to include the properties defined in the customization. Because the requirement says that the business user can enter multiple attribute values for an attribute (which matches the attribute values of the catalog entries), we need a grid-based widget where we can capture a list of attribute values. This can be achieved by creating a new child editor object extending the `wcfObjectGrid` class.

1. Open the `LOBTools/WebContent/WEB-INF/src/lzx/commerce/marketing/listViewDefinitions/ChildListEditorListViews.lzx` file and insert the bolded portion of the snippet in Example 7-10.

Example 7-10 Creating a child list editor to add attribute values for an attribute name

```

<library>
.....
    <class name="extAttribValueGrid" extends="wcfObjectGrid"
        preferenceKey="extAttribValueGrid">
        <wcfGridText name="attribValue" propertyName="attribValue"
            editable="true"
            text="{extMarketingResources.attributeValues.string}"
            visible="true" width="150" required="true"/>
        </class>
    .....
</library>

```

2. Save the file.
3. Open the OpenLaszlo file `LOBTools/WebContent/WEB-INF/src/lzx/commerce/marketing/propertiesViews/activityBuilder/DisplayProductPropertiesView.lzx` and update it to include the new properties, as shown in Example 7-11 on page 173.

```
<class name="mktDisplayProductProperties"
  extends="mktFlowElementProperties">
  <wcfPropertyGroup name="group" collapsable="false">
    <wcfPropertyRadioGroup propertyName="elemTemplateName"
      promptText="{extMarketingResources.recommendCriteria.string}"
    />

    <wcfPropertyGroup name="productsGroup" collapsable="false">
      <wcfEnablementCondition propertyName="elemTemplateName"
        enablementValue="displayProduct"
        conditionId="productsElement" />
      <wcfPropertyChildListEditor name="prodEditor"
        objectTypes="ChildCatentry,ChildInheritedCatentry"
        listClass="mktProductGrid"
        promptText="{mktMarketingResources.products.string}"
        required="true" />
    </wcfPropertyGroup>

    <wcfPropertyGroup name="productAttribGroup" collapsable="false">
      <wcfEnablementCondition propertyName="elemTemplateName"
        enablementValue="displayProductWithAttrib"
        conditionId="productAttribElement" />

      <wcfPropertyInputText propertyName="attribName"
        required="true"
        promptText="{extMarketingResources.attributeName.string}" />
      <wcfPropertyStepper
        propertyName="numProducts" required="true"
        promptText="{extMarketingResources.numProducts.string}"
        minimumValue="1" maximumValue="20" />

      <!-- Start Section 1 -->
      <wcfPropertyGroup name="attribValuesGroup"
        collapsable="false">
        <wcfPropertyChildListEditor name="attribValueEditor"
          objectTypes="attribValue" listClass="extAttribValueGrid"
          promptText="{extMarketingResources.attributeValues.string}"
          required="true"/>
      </wcfPropertyGroup>
      <!-- End Section 1 -->

    </wcfPropertyGroup>
  </wcfPropertyGroup>
```

</class>

We define a new property list editor for the attribute values as shown in Section 1 in the snippet using the list editor created in the Example 7-10 on page 172.

4. Save the file.

7.3.6 Creating the summary definition

A summary definition needs to be created for the recommend catalog entry action to include the summary when the user selects the recommend catalog entries using the attributes option. A new file needs to be created because the summary definitions for the existing campaign elements are defined in the LOBTools\WebContent\WEB-INF\src\lzx\commerce\marketing\restricted directory. It is not recommended to modify a restricted file because the fix pack updates to the WebSphere Commerce suite update the files residing in the restricted directory.

1. Create a new OpenLaszlo file ExtDisplayProductSummary.lzx under the LOBTools\WebContent\WEB-INF\src\lzx\<your_company_name>\marketing\propertiesViews\activityBuilder directory and copy the snippet shown in Example 7-12.

Example 7-12 The summary definition for the Recommend catalog entry action

```
<library>
  <class name="extDisplayProductSummary"
    extends="mktFlowElementSummary">
    <mktFlowSummaryParam name="productNames"
      objectPath="ChildCatentry/CatalogEntry/CatalogEntryDescription"
      propertyName="name"/>
    <mktFlowSummaryParam name="elemTemplateName"
      propertyName="elemTemplateName"/>
    <mktFlowSummaryParam name="numProducts"
      propertyName="numProducts"/>
    <mktFlowSummaryParam name="attribName"
      propertyName="attribName"/>
    <mktFlowSummaryParam name="attribValueList"
      objectPath="attribValue" propertyName="attribValue"/>

    <handler name="oninit">
      <![CDATA[
        this.updateSummaryDel.register(
          extMarketingResources.summaryDisplayProductWithAttributes,
          "onstring");
      ]]>
    </handler>
  </class>
```

```

        this.updateSummaryDel.register(
extMarketingResources.summaryDisplayProductWithAttributesContains,
        "onstring");
    ]]>
</handler>

<method name="updateSummary" args="e=null">
<![CDATA[
    if(this.resolvedParams["elemTemplateName"] ==
        "displayProduct") {
        var products = this.getValuesList("productNames");
        if(products != "") {
            products = lz.text.prototype.escapeText(products);
        }
        this.setSummaryText(products);
    } else {
        var summary = "";
        var attribValues =
            this.getValuesList("attribValueList");
        var attribName = this.resolvedParams["attribName"];
        var numProducts = this.resolvedParams["numProducts"];
        summary = extMarketingResources.replaceValues(
            "summaryDisplayProductWithAttributesContains",
            [numProducts, attribValues, attribName]);
        this.setSummaryText(summary);
    }
    ]]>
</method>
</class>
</library>

```

2. Save the file.

7.3.7 Registering the new classes in the marketing extensions library

The resource bundle and the summary definition need to be registered in the marketing extensions library to make the classes available to the marketing UI. The marketing extensions library is located in the LOBTools/WebContent/WEB-INF/src/lzx/commerce/marketing/MarketingExtensionsLibrary.lzx directory.

1. Open the file and update it to include the bolded portion of the snippet in Example 7-13 on page 176.

```
<library>
.....

    <include href="../../../madisons/marketing
        /MarketingResourceBundle.lzx"/>
    <include
href="../../../madisons/marketing/propertiesViews/activityBuilder/ExtDisplayProductSummary.lzx"/>

.....
</library>
```

2. Save the file.

7.3.8 Creating and registering the serialization JSP for the new template

We need to create a serialization JSP for the new template that we have created. The new template is used when the user selects the recommending catalog entries using attributes options. Perform the following steps to create and register the serialization JSP for the new template.

1. Create a `SerializeActivitydisplayProductWithAttrib.jsp` file inside the `LOBTools/WebContent/jsp/<your_company_name>/marketing` directory. The snippet in the Example 7-14 shows the content of the JSP.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<object objectType="displayProduct">
    <parent>
        <object objectId="{element.parentElementIdentifier.name}"/>
    </parent>
    <elemTemplateName>
<![CDATA[{element.campaignElementTemplateIdentifier.externalIdentifier.name}]]>
    </elemTemplateName>
    <elementName>
        {element.campaignElementIdentifier.name}
    </elementName>
    <sequence>{element.elementSequence}</sequence>
    <customerCount readonly="true">{element.count}</customerCount>
    <c:forEach var="elementVariable"
```



```

        items="${element.campaignElementVariable}">
    <c:if test="${elementVariable.name != 'attribValueList'}">
        <${elementVariable.name}>
            <![CDATA[${elementVariable.value}]]>
        </${elementVariable.name}>
    </c:if>
    <c:if test="${elementVariable.name == 'attribValueList'}">
        <object objectType="attribValue">
            <attribValue>
                <![CDATA[${elementVariable.value}]]>
            </attribValue>
        </object>
    </c:if>
</c:forEach>
<c:forEach var="userDataField"
    items="${element.userData.userDataField}">
    <x_${userDataField.typedKey}>
        <![CDATA[${userDataField.typedValue}]]>
    </x_${userDataField.typedKey}>
</c:forEach>
</object>

```

2. Create the struts action mapping for the new serialization JSP. Open the LOBTools/WebContent/WEB-INF/struts-extension.xml file and add the snippet in Example 7-15.

Example 7-15 OBTools/WebContent/WEB-INF/struts-extension.xml

```

<action
    path="/SerializeActivityElement-displayProductWithAttrib"
    include="/jsp/madisons/marketing/SerializeActivitydisplayProductWithAttrib.jsp" />

```

3. Save the files.

After the task command and UI customization are completed, you can build the project for Java changes and build the openlaszlo project for the UI changes. For more details on building the OpenLaszlo project, see the WebSphere Commerce Information Center:

http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center_customization.doc/tasks/tdeolbuild.htm

Restart the WebSphere Commerce server to test your code.

7.4 Test your code

Follow the steps to recreate the scenario and to debug your code if you run into any problems.

7.4.1 Checking the UI customization for recommend catalog entries action

When you use the customized UI to create a catalog entry recommendation activity using the newly created parameters and save it, make sure that UI retrieves all the data properly from the database and it can be modified again. This notifies you that UI customization has been done correctly.

7.4.2 Creating a Web activity to recommend catalog entries with attributes

You can use this action only with Web activities campaign. Enter the values as shown in Figure 7-3 on page 159 to create a catalog entry recommendation definition for a season's clearance sale. Make sure that your store has those attributes and values used in the definition. Specify the valid dates for this activity, save the changes, and activate it.

7.4.3 Testing your task command with Web stores

After you verify the UI customization and create a Web activity for this catalog entry recommendation, you need to test the task command. This task command is called by marketing runtime and passed all the required parameters to the performExecute method. Log in to the store and open the shopping cart. You can see the catalog entry recommendations limited to the number you specified in the definition. Each time you refresh the shopping cart or revisit it, a different set of catalog entry recommendations is displayed.



Part 3

Common Practices

IBM WebSphere Commerce follows standard design patterns and practices so that it is easy to understand and develop customization assets with it. You can use these practices for your development and customizations. In this section we discuss common practices used in implementing precision marketing in WebSphere Commerce as well as customizations of the IBM Management Center for WebSphere Commerce.

Common practices

In this chapter, we discuss the common practices for using the IBM Management Center for WebSphere Commerce for precision marketing and customization.

The following sections are included in this chapter:

- ▶ Precision marketing common practices
- ▶ Customization common practices

8.1 Precision marketing common practices

In this section, we discuss the common practices to be followed when using the IBM Management Center for WebSphere Commerce tooling for precision marketing. The tools are flexible and allow you to create various activities for your business needs. However, there are certain common practices that are to be followed to ensure that the performance is not hindered in any way. If you follow these guidelines, there is minimal impact on your system when the activities are activated.

- Enable and configure command caching.

Command caching can improve the performance of the Marketing tool by storing the results of Web service calls that load the business objects to display to the customer. By doing so, this limits the amount of redundant queries that need to be performed on the database. Instead of querying the database each time, the data can be retrieved from the command cache, which improves response time as well as reduces the system load.

For more information and instructions on how to set up command caching, see the WebSphere Commerce Version 7 Information Center:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.admin.doc/concepts/cdcmarcacommcac.htm>

- Configure business object caching.

Business object caching can improve performance because it stores marketing business objects in addition to user data. This eliminates the need for querying the database for the information regarding the marketing rules for a customer. The MarketingCache stores data relating to marketing business objects such as activity and element business objects, data pertaining to e-marketing spots, and trigger listener business objects. The MarketingUserCache caches customer information. By managing the size of the cache, performance can be improved.

For more information, visit the Information Center:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.admin.doc/concepts/cdcbusobjcache.htm>

- Configure data recording.

Data recording can be used to improve the runtime performance regarding the recording of user behavior. There are two options used in configuring user data recording which can help with performance. By default, the data is stored in the cache, but can also be configured to be persisted to the database if desired. Also, the amount of data in the cache before persisting to the database can be configured. The data is persisted to the database in batch mode when the configured batch size has been reached.

Read the Inforcenter for more details regarding data recording:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.admin.doc/concepts/cdcperfareasconsider.htm>

- View the activities scheduled for an e-marketing spot.

When creating a new Web activity, use the Show Activity List for e-Marketing Spots to see how many activities are scheduled on a particular spot. This information can be used so the e-marketing spots are not overloaded with activities, as this might affect the performance. The content in e-marketing spots are resolved dynamically based on the customer who is viewing the spot. This can be intensive if there are many activities on one spot.

Priority can be set on each activity to determine which is displayed first if more than one is scheduled on the same e-marketing spot and there is a conflict. Setting the priority can help so the system does not need to work as hard to determine what the content is for a customer.

- Create simple activities.

Avoid complex activities that require a large amount of campaign elements. Simple activities are easier to manage and do not require as much processing. Complex activities can be created if necessary.

- Test each new activity in the development environment.

All activities must first be tested in the development environment to see the performance impact. This allows you to see which activities are database intensive and which are not. Some campaign elements have more of an impact than others, such as the shopping cart and purchase history targets. Activities that use these elements need more processing time. Testing the activities first ensures that they work properly before deploying them to the production server.

- Refrain from using the send immediately option.

The send immediately flag can impact performance as e-mails can be sent at any time as soon as a customer reaches the Send Message action of an activity. The performance can be monitored if they are sent at a scheduled time each day.

- Configure the e-mail activity to occur on off peak hours.

The e-mail activity needs to be scheduled to run during off peak hours to lessen the impact on performance during peak hours.

- Schedule the SendTriggers action to run during off peak hours.

The SendTriggers job needs to be scheduled during the off peak hours so performance is not affected during business hours. This job must run before the scheduled time of the e-mail activity because it processes the triggers and determines which customers need to be e-mailed.

8.2 Customization common practices

- Use only the documented OpenLaszlo API classes, not classes marked as private or restricted.

OpenLaszlo files belong to two groups:

- Customizable files
- Restricted files

Customizable files are the recommended files to alter when customizations are needed. Although it is recommended that new files be created when creating new classes, the classes can be put directly into the pre-existing files. These files are not changed if a WebSphere Commerce fix is installed. Restricted files are replaced if a fix is installed that makes them poor locations for customizations to take place. Refrain from changing the restricted classes for this reason.

- Use the predefined extension points when extending the OpenLaszlo API classes.

There are two extension points for customizations. Changes can occur directly in any OpenLaszlo file that is not in a restricted directory. Also, there are some configuration files that can be created in predefined locations. These files are merged with the predefined configurations. The new configuration takes precedence if it is defined in both locations.

The files that can be created are as follows:

- `WC_installdir/xml/config/com.ibm.commerce.component-ext/wc-object-relational-metadata.xml`
- `WC_installdir/xml/config/com.ibm.commerce.component-ext/wc-business-object-mediator.xml`
- `WC_installdir/LOBTools/WebContent/WEB-INF/config/com.ibm.commerce.component-ext/get-data-config.xml`
- `WC_installdir/LOBTools/WebContent/WEB-INF/config/com.ibm.commerce.component-ext/wc-component-clientobjects.xml`

- Use proper naming conventions when extending or creating new functionality.

Following the WebSphere Commerce naming conventions, a new task command has an interface named similar to `NewTriggerTaskCmd` and the implementation class is named similar to `NewTriggerTaskCmdImpl`. If the command is extending an existing command in Commerce, the package mimics the same package structure, but use the company name in place of `ibm`, such as `com.madisons.commerce.marketing.commands.elements`. Using this package structure makes it easier to determine which classes were extended as well as makes it easier for deployment.

When extending any of the OpenLaszlo classes, it is recommended to prefix the class name with *ext* to identify them as extensions as opposed to new files. See the following link in Information Center for more information about the IBM Management Center for WebSphere Commerce file and class naming conventions:

http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center_customization.doc/concepts/ctffilenaming.htm

- ▶ Use WebSphere Commerce Developer when making developing customizations to make customizing easier.
- ▶ Create multiple template definitions for complex campaign elements.

For complex campaign elements, it is recommended that multiple template definitions are created to simplify the creation of the campaign element. Multiple template definitions allow for the separation of the UI into manageable pieces.

For example, in the wish list target created in Chapter 4, “Create a new target” on page 53, the campaign element was split into three pieces:

- wishListAny
- wishListProducts
- wishListCategories

Each of the templates are responsible for a certain task.

- ▶ Split target, trigger, and action task commands into multiple commands.

If the requirements for a new campaign element are complex, it is recommended that multiple task commands are created to handle the logic. This simplifies the logic in each of the task commands and limit the amount of conditionals and checks needed. Also, each task command is mapped to a template definition.

In the wish list target created in Chapter 4, “Create a new target” on page 53, three task commands were used to perform the logic:

- *CustomerFilterWishListTargetTaskCmd*
- *CustomerFilterWishListProductTargetTaskCmd*
- *CustomerFilterWishListCategoryTargetTaskCmd*

Each of the task commands performed a specific action for the campaign element.

Note: Performance varies from system to system. The considerations listed are to be used as guidelines to follow, but do not necessarily improve performance on your system. If you want more information about how to improve performance in WebSphere Commerce, visit the Information Center:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.admin.doc/concepts/cpmperformance.htm>





Appendixes















This section has brief Marketing tool element matrixes to show what campaign elements are available in IBM Management Center for WebSphere Commerce and where they can be used in Web activities and dialog activities. Useful Information Center links about these topics are presented at the end of the appendixes.



Triggers, targets, actions, and branching

This appendix provides a summary of the triggers, targets, actions and branching available for Web activities and dialog activities in WebSphere Commerce Version 7. See Table A-1.

Table A-1 Triggers, targets, actions and branching available for Web activities and Dialog activities

Icon on palette	Element type	Element category	Web activity	Dialog activity
	Wait	Trigger	No	Yes
	Customer Registers	Trigger	No	Yes
	Customer Places Order	Trigger	No	Yes
	Customer Participates in Social Commerce	Trigger	No	Yes

Icon on palette	Element type	Element category	Web activity	Dialog activity
	Customer Abandons Shopping Cart	Trigger	No	Yes
	Customer Celebrates Birthday	Trigger	No	Yes
	Customer Is In Segment	Trigger	No	Yes
	Customer Segment	Target	Yes	Yes
	Shopping Cart	Target	Yes	Yes
	Purchase History	Target	Yes	Yes
	Current Page	Target	Yes	No
	Catalog Browsing Behavior	Target	Yes	Yes
	Online Behavior	Target	Yes	Yes
	External Site Referral	Target	Yes	No
	Cookie Contents	Target	Yes	No
	Social Commerce Participation	Target	Yes	Yes
	Day/Time	Target	Yes	Yes
	Send Message	Action	No	Yes

Icon on palette	Element type	Element category	Web activity	Dialog activity
	Issue Coupon	Action	No	Yes
	Add To or Remove From Customer Segment	Action	Yes	Yes
	Recommend Catalog Entry	Action	Yes	No
	Recommend Content	Action	Yes	No
	Recommend Category	Action	Yes	No
	Display Merchandising Association	Action	Yes	No
	Recommend Promotion	Action	Yes	No
	Display Recently Viewed	Action	Yes	No
	Branch	Branching	Yes	Yes
	Experiment	Branching	Yes	No

See the following Commerce InfoCenter URLs to get more details on campaign elements, and functionality available in IBM Management Center for WebSphere Commerce using precision marketing.

► Targets

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/concepts/csbtargets.htm>

► Triggers

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/concepts/csbtriggers.htm>

- ▶ **Actions**
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/concepts/csbactions.htm>
- ▶ **Customer segments**
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/concepts/csbsegover.htm>
- ▶ **Branching**
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/concepts/csbbranches.htm>
- ▶ **Experiments**
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/concepts/csbexpooverview.htm>
- ▶ **Web activities**
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/concepts/csbwebacts.htm>
- ▶ **Dialog activities**
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center.doc/concepts/csbdiaacts.htm>
- ▶ **Standard Activity Templates**
http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/concepts/csbtemplates_V7.htm
- ▶ **Custom Activity Templates**
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/concepts/csbcustemp.htm>
- ▶ **e-mail Templates**
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/concepts/csbeatover.htm>
- ▶ **e-mail activities**
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/concepts/csbmailacts.htm>
- ▶ **e-marketing spots**
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center.doc/concepts/csbspotover.htm>
- ▶ **Campaigns**
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/concepts/csbcams.htm>
- ▶ **Marketing Content**
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/concepts/csbcontent.htm>

Additional material

This book refers to additional material that can be downloaded from the Internet as described.

Locating the Web material

The Web material associated with this book is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser at:

<ftp://www.redbooks.ibm.com/redbooks/SG247790>

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG247790.

Using the Web material

The additional Web material that accompanies this book includes the following files:

<i>File name</i>	<i>Description</i>
SG247790_Scripts.zip	Zipped Code Samples

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

For information about ordering these publications, see “How to get Redbooks” on page 200. Note that some of the documents referenced here might be available in softcopy only.

- ▶ *WebSphere Commerce Line-Of-Business Tooling Customization*, SG24-7619

Online resources

These Web sites are also relevant as further information sources:

- ▶ Creating an e-marketing spot
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/tasks/tsbctespot.htm>
- ▶ Creating promotions
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/tasks/tprcreat.htm>
- ▶ Creating marketing content to display on store pages
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/tasks/tsbmccreate.htm>
- ▶ Creating a Web activity
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/tasks/tsbctwebact.htm>
- ▶ Testing Web activities
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/tasks/tsbtestweb.htm>

- ▶ Working with e-marketing spots
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center.doc/tasks/tsbemssupert.htm>
- ▶ Promotions tool
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center.doc/concepts/cprpromotools.htm>
- ▶ Testing Dialog activities
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/tasks/tsbtestdia.htm>
- ▶ IBM Management Center for WebSphere Commerce framework
http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center_customization.doc/concepts/ctfcmcframework.htm
- ▶ Campaign element template definitions
http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center_customization.doc/concepts/csbcusteletemp.htm
- ▶ Abstract Class mktFlowElementObjectDefinition
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.openlaszlo.api.doc/lzx/commerce/marketing/restricted/activityBuilder/FlowElementObjectDefinition.lzx/mktFlowElementObjectDefinition.html>
- ▶ Abstract Class mktFlowElementProperties
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center.openlaszlo.api.doc/lzx/commerce/marketing/restricted/activityBuilder/FlowElementProperties.lzx/mktFlowElementProperties.html>
- ▶ Abstract Class mktFlowElementSummary
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center.openlaszlo.api.doc/lzx/commerce/marketing/restricted/activityBuilder/FlowElementSummary.lzx/mktFlowElementSummary.html>
- ▶ Building OpenLaszlo applications
http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center_customization.doc/tasks/tdeolbuild.htm

- ▶ Tips for testing targets in Web and Dialog activities
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center.doc/refs/rsbtesttar.htm>
- ▶ com.ibm.commerce.membergroup.commands Interface
 CheckUserInMemberGroupCmd
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.api.doc/com.ibm/commerce/membergroup/commands/CheckUserInMemberGroupCmd.html>
- ▶ Class wcfChildObjectDefinition
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.openlaszlo.api.doc/lzx/commerce/foundation/restricted/ObjectDefinition.lzx/wcfChildObjectDefinition.html>
- ▶ Abstract Class mktCSElementSimpleConditionObjectDefinition
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.openlaszlo.api.doc/lzx/commerce/marketing/restricted/customerSegment/CustomSegmentElementObjectDefinition.lzx/mktCSElementSimpleConditionObjectDefinition.html>
- ▶ Abstract Class mktCSElementAndOrListConditionObjectDefinition
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.openlaszlo.api.doc/lzx/commerce/marketing/restricted/customerSegment/CustomSegmentElementObjectDefinition.lzx/mktCSElementAndOrListConditionObjectDefinition.html>
- ▶ Class mktTopAndListObjectDefinition
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.openlaszlo.api.doc/lzx/commerce/marketing/objectDefinitions/customerSegment/TopAndListObjectDefinition.lzx/mktTopAndListObjectDefinition.html>
- ▶ Types of Dialog activity triggers and when they are processed
http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center_customization.doc/concepts/tsbtypediaact.htm
- ▶ Implementation definition: for triggers
http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center_customization.doc/concepts/csbimplemmenttar.htm

► Behavior rule definition

http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center_customization.doc/concepts/csbbehavior.htm

► Related rule definition

http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center_customization.doc/concepts/csbrelated.htm

► Ways to inform the marketing services of external events for custom triggers and targets

http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center_customization.doc/concepts/csbtriggerevent.htm

► Enabling personalization ID

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.admin.doc/tasks/tseenablepersonalizationid.htm>

► Creating the object definition for the campaign element

http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center_customization.doc/tasks/tsbcustctobdef.htm

► Defining icons to represent the campaign element

http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center_customization.doc/tasks/tsbcustpalicons.htm

► Tips for testing triggers and actions in Dialog activities

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center.doc/refs/rsbtestdia.htm>

► com.ibm.commerce.marketing.commands.elements - Interface MarketingCampaignElementTaskCmd

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.api.doc/com/ibm/commerce/marketing/commands/elements/MarketingCampaignElementTaskCmd.html>

► Command caching for marketing

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.admin.doc/concepts/cdcmarcaccommcac.htm>

- ▶ Marketing business object cache overview
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.admin.doc/concepts/cdcbusobjcache.htm>
- ▶ Enabling JSP caching
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.admin.doc/tasks/tdcjspcachecont.htm>
- ▶ Data recording; improving runtime performance
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.admin.doc/concepts/cdcperfareasconsider.htm>
- ▶ IBM Management Center for WebSphere Commerce file and class naming conventions
http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center_customization.doc/concepts/ctffilenaming.htm
- ▶ Performance
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.admin.doc/concepts/cpmperformance.htm>
- ▶ Targets in marketing activities
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/concepts/csbtargets.htm>
- ▶ Triggers in marketing activities
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/concepts/csbttriggers.htm>
- ▶ Actions in marketing activities
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/concepts/csbacktions.htm>
- ▶ Customer segments
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/concepts/csbsegover.htm>
- ▶ Using the Branch element in Web and Dialog activities
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/tasks/tsbbranching.htm>
- ▶ Activity templates
http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/concepts/csbtemplates_V7.htm

- ▶ Custom activity templates

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/concepts/csbcustemp.htm>

- ▶ E-mail templates

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/concepts/csbeatover.htm>

- ▶ E-mail activities

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/concepts/csbsmailacts.htm>

- ▶ Working with e-marketing spots

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/tasks/tsbemssupert.htm>

- ▶ Campaigns

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/concepts/csbcamps.htm>

- ▶ Marketing content

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/concepts/csbcontent.htm>

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks publications, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services



Redbooks®

Precision Marketing in WebSphere Commerce

Enable your precision marketing objectives

This IBM Redbooks publication positions WebSphere Commerce Version 7 in today's marketplace and discusses its enhanced features enabling business users to achieve their vision for precision marketing.

Use IBM Management Center marketing tools

This book helps you to tailor and configure the Marketing tool in IBM Management Center for WebSphere Commerce to create and manage various marketing campaigns as needed by your business.

Explore scenarios and customizations

This book provides several business scenarios which can be implemented through simple customizations. Each scenario addresses a unique requirement which can be mapped with similar business scenarios.

This book has been developed for an experienced WebSphere Commerce design and developer audience.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks