Intelligent Offer Management (IOM),
Version 2.3.1

IOM Concepts Guide,

with Tutorials

NUANCE

Notice

Intelligent Offer Management (IOM) Version 2.3.1
IOM Concepts Guide

1198 E. Arques Avenue, Sunnyvale, CA 94085, U.S.A.
Printed in the United States of America.

Last updated 6/9/08.

Contents

List of Tables

List of Figures

Chapter 1 Release Notes for IOM Version 2.3.1

Here is a summary of the major new features in Intelligent Offer Management (IOM) Version 2.3.1.

Service Infrastructure Improvements

Many improvements have been made to the IOM service itself that may not be readily visible to the end-user. These include:

- Database schema enhancements
- Infrastructure redesign: service-oriented architecture

IOM Tool GUI Redesign

One of the key features of Version 2.3.1 is a redesign of the graphical user interface of the IOM Tool. This redesign is based on the following principles:

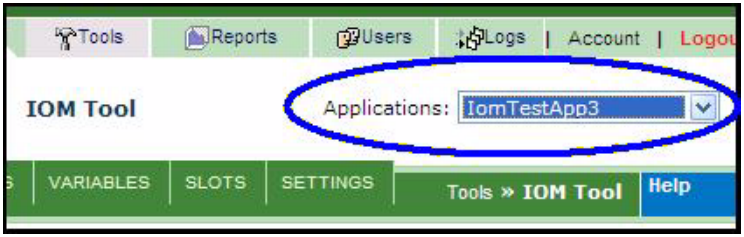
- Consistency from page to page in placement of action and other buttons
- New Campaign "wizard mode"

  A new "wizard" has been added to the IOM Tool to guide you through the process of creating a new campaign and adding offers to it.

The IOM Tool is fully documented in *IOM Tool User Guide*.

Organization by Application name

All your Intelligent Offer Management (IOM) information is categorized by application. You are allowed to access only those specific applications to which you have permission. In the upper right corner, the pulldown menu labeled **Application** lists the applications that you have access to.



New Logical Operators for Upsell Segments: Greater Than and Less Than

In defining upsell segment rules with variables of the Number data type, you can now use the logical operators "Greater Than" and "Less Than."

`Greater Than`	Subscribers for whom the value of the specified variable is greater than Value #1.
`Less Than`	Subscribers for whom the value of the specified variable is less than Value #1.

New Functionality for Interactive Promotions: Accept, Decline, Error, and Help

IOM 2.3 includes new types of prompts and support files you can associate with interactive promotions. In former versions of IOM, the only types of prompts supported were the Initial prompt, a grammar file, and a destination map.

Interactive promotions in IOM 2.3 support the following types of files:

- Initial: played when an offer is made. All interactive promotions must have at least this initial prompt.
- **Accept:** played when the caller accepts the offer.
- **Decline:** played when the caller declines.
- **Error:** played in the event of an error.

- **Help:** played when caller says “Help.”
- Grammar files
- Destination Files

Note: About grammar in IOM 2.3: to make full use of the Accept or Decline prompts, your grammars must be written in terms of “accept” or “decline,” rather than as simple “yes or no.” This depends on the nature of your prompts.

HTTP API

IOM now includes an HTTP application programming interface. See the *IOM Developer and Datafeed Reference*.

Note: The JSP API for IOM Version 1.6.x is considered legacy. You are encouraged to use the HTTP API, instead.

IOM Documentation

Intelligent Offer Management (IOM) is documented in the following books.

Book Title	Audience	Description
IOM Concepts Guide	All readers	■ Overview concepts for IOM: □ Theory of general operation □ Components of the IOM system □ Definitions of key terminology ■ Tutorials of campaigns from start to finish: □ Campaign design □ Implementing campaign objects in the IOM Tool □ Voice application design with a sample application ■ Release notes for latest version of IOM
IOM Tool User Guide	Campaign administrators	Step-by-step use of the IOM Tool for all tasks: ■ Managing campaign objects ■ Importing and exporting applications, campaigns, or offers ■ IOM user set-up in C3

Book Title	Audience	Description
IOM Developer and Datafeed Reference	Voice application developers	All details relating to programming IOM-based voice applications and IOM datafeed: ■ Theory of programming, development methodology ■ XML-over-HTTP API ■ Developing and testing an IOM datafeed, alarms, and service monitors.
IOM Legacy JSP ASP Reference	Developers supporting IOM 1.6 legacy applications	Details related to the pre-IOM-2.3 legacy Java Server Pages Application Programming Interface.

Known Issues in IOM Version 2.3.1

The section details known issues in IOM Version 2.3.1 and their workarounds.

The numbers in the titles of these sections are bug numbers for reference only.

Whitespace in Multi-Valued Strings in Segment Rules (430)

In creating segment rules with multi-valued, comma-delimited strings, IOM Tool does not validate the multi-valued field and does not trim the whitespace: it displays the entered whitespace.

Workaround

None.

Data are stored properly in the database. Whitespace is not valid in multi-valued strings, as documented. In the actual comparison at runtime, whitespace is trimmed and is not considered in the comparison at all.

Renaming with a Name that Already Exists (232)

If you rename a campaign, offer, promotion, segment, variable, or slot to a name that already exists, no error message is displayed.

Workaround

Do not re-use names. Keep track of your object names.

Promotion Fields Not Refreshed (257)

On the **Edit Promotion** page, after you change the promotion name or description, **Update** to save the changes, and then click **Cancel**, the original name or description is displayed.

Workaround

This is page refresh issue. Refresh the page and the changed name or description will be displayed.

Offer Frequency is Reset to Default on Page (398)

After you create an offer with any frequency, the frequency settings are properly stored and acted upon by the IOM Engine. However, if you edit the offer, change the frequency,and click **Cancel**, on the page the frequency is displayed as the default, not the settings that are stored.

Workaround

Navigate away from the page or refresh the page after you click **Cancel**.

Variable Values Associated with Test Phone Numbers Not Imported (422)

The segment variables and rules associated with test telephone numbers (test MINs) are not properly imported. Segments and rules are no created.

Workaround

Recreate the test data associated with the test telephone numbers after you import.

Errors after Session Timeout of 30 Minutes (427)

Sometimes, after an IOM session has been idle for 30 minutes, data errors can occur.

Workaround

Do not allow your web browser page to be idle for more than 30 minutes. Instead, logout and then later login again.

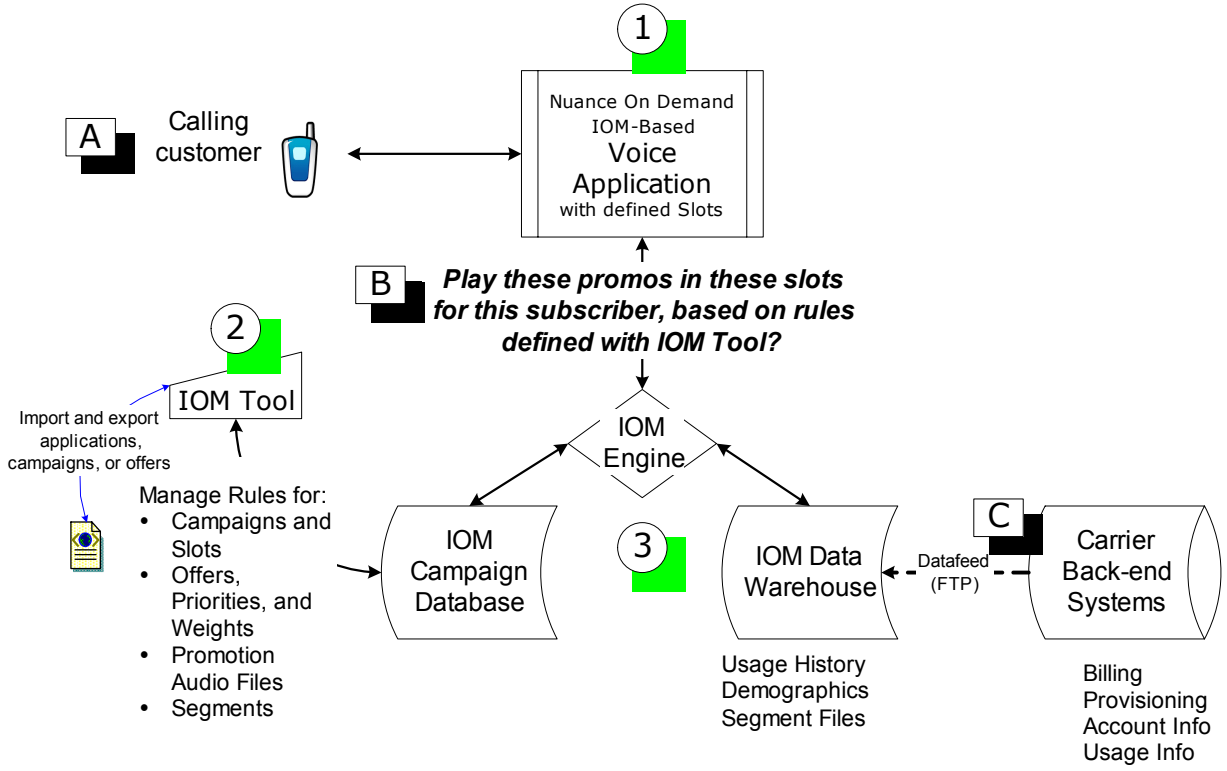
Chapter 2 Overview to Intelligent Offer Management (IOM)

Nuance's Intelligent Offer Management (IOM) is a marketing system and runtime service for carriers to target promotions to their subscribers to increase average revenue per user (ARPU). With IOM, voice application design is simplified, because the marketing logic is maintained in IOM rather than in the application.

The following figure illustrates:

- **A, B, C**: How IOM works in actual practice with a calling subscriber
- **1, 2, 3**: The design of the IOM system itself

Figure 1 How IOM Works



How IOM Works

A: The subscriber calls the voice application.

B: The subscriber is presented with an offer. The voice application exchanges information with the IOM Engine to determine, according to previously defined rules about the subscriber's profile and other information, whether to make an offer: play a promotion of a product or service at certain pre-defined points (voice application "slots").

C: Offline, the carrier feeds information to Nuance about usage and other data that the IOM Engine can use to determine the appropriateness of offers for subscribers.

Components of IOM

The IOM system is composed of three primary parts: a voice application designed for IOM, the IOM Tool, and the IOM Engine and databases.

1: Every IOM marketing campaign is associated with a voice application. The voice application has been designed with specific "slots": locations in the application flow where promotional offers can be made to the subscriber.

2: The IOM Tool has several purposes:

- To create and edit voice application slots.
- To create and edit segment variables.
- To manage the rules used by the IOM Engine that determine how all the campaign's objects interact to present promotional offers to a calling subscriber.
- To manage all the objects of a campaign: offers with priorities and weights, segments, and promotion files.

3: The IOM Engine is the runtime service that, based on the rules defined with IOM Tool, determines whether or not promotional offers should made to a specific calling subscriber.

Note: All of these concepts are defined in "Key Concepts: Campaign Objects" on page 21.

General Approach to Using IOM

Below is an overview of the steps for using the IOM Tool to create campaign objects for an IOM-enabled voice application. Using IOM is usually collaborative work between both a voice application developer and a customer service representative or other person who uses the IOM Tool.

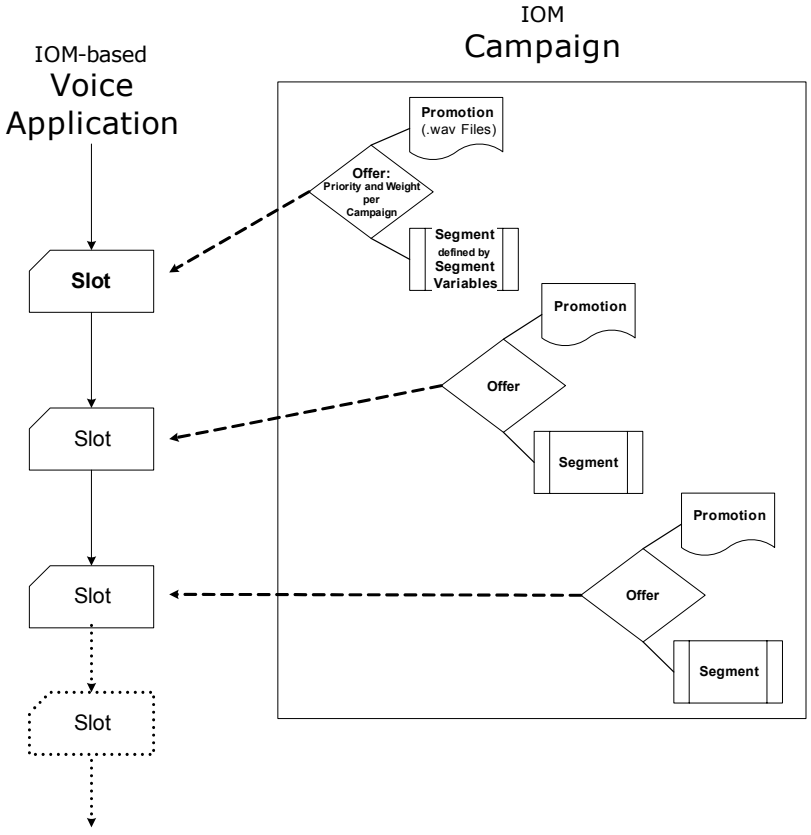
What	How	See...
1. Create a new application or select from existing applications.	With IOM tool	*IOM Tool User Guide*
2. Create voice application slots	With IOM tool	*IOM Tool User Guide*
3. Define segments: ■ Create variables ■ Create segments: □ Define upsell segment rules □ Upload fulfillment segment files	With IOM tool	*IOM Tool User Guide*
4. Create all campaign objects: ■ Campaign ■ Offers with promotions and segments ■ Assign offers with priorities and weights to campaign	With IOM Tool	*IOM Tool User Guide*
5. Code voice application: ■ Map slots in voice application ■ Optionally create destination map	With HTTP API and coding	*IOM Developer and Datafeed Reference*
6. Test and refine voice application: ■ Modify campaign objects ■ Put voice application into production	With IOM Tool and coding	*IOM Tool User Guide*
7. Optionally implement a datafeed	With datafeed framework	*IOM Developer and Datafeed Reference*

The following figure shows the hierarchical relation of the key IOM concepts: campaign objects managed with the IOM Tool.

In a nutshell:

An IOM marketing *campaign* is associated with a *voice application* that has well-defined *slots*, which are points in the application where the IOM Engine will determine whether to plug in an *offer* to play a *promotion* (a set of audio messages) about a product or service targeted at the calling subscriber's market *segment* defined by rules based on *segment variables*.

Figure 2 Campaign Objects: Conceptual Hierarchy



Campaigns and Voice Application Slots

With a defined starting and ending date, a *campaign* is associated with a voice application that has predefined "slots." A *slot* is any point in a voice application where an offer can be made. Any number of slots can be defined for a voice application. Typical slots in a voice application are at the beginning before the main menu and after successful completion of some task.

A campaign can be enabled or disabled at will.

The campaign itself is composed of offers, each with a priority, weight, and associated promotions for particular segments.

Offers

Offers, which can be associated with one or more campaigns, consist of the following:

- Promotions and optional Segment

  Independently of any campaign, an offer always has promotions and can be assigned a segment.
- Priority, Weight, and Slot

  When you associate an offer with a campaign, you must assign it a priority and weight and associate it with a slot in the voice application.
- Offer Frequency (Optional)

  When you associate an offer with a campaign, you can specify the frequency of the offer: how often it is played, such as once a month, every call, and so forth.
- Status and Test-Enabled

  An offer's status can be "on" or "off." A status of "on" means the campaign is actively being presented to subscribers.

  Likewise, a campaign's "test-enabled" status can be "on" or "off." With test-enabled "on," even though the campaign is not active to subscribers, its offers can be tested.

Promotions

Promotions, which can be associated with any offer, are audio and other files representing a product, a service, or a simple message. There are two types of promotions: *interactive* and *informational*.

Interactive Promotion

An *interactive promotion* is played to subscribers who meet the conditions for an offer: the target segment and the priority and weight of the offer. The subscriber is asked to perform some action, such as accept the offer or decline the offer. Based on the subscriber's action, your voice application can do anything it wants, such as to transfer the subscriber to a specific point in the voice application.

An interactive promotion is physically represented by a set of audio files (prompts) and optional support files.

Prompts are categorized into subtypes, based on their content:

- Initial: played when an offer is made. All interactive promotions must have at least this initial prompt.
- Accept: played when the caller accepts the offer.
- Decline: played when the caller declines.
- Error: played in the event of an error.
- Help: played when caller says "Help."

The optional support files a voice application might need that can be associated with interactive promotions can be any of the following:

- Auxiliary prompt audio files, such as re-prompts or "thank you" messages
- Grammar files that (for example) interpret "accept" or "decline" responses.
- Destination files, which define points in the voice application to which to transfer control after a subscriber's response, such as a named `<form>` or `<subdialog>`.

Informational Promotion

An *informational promotion* is an audio file that has no call to action—it is simply to inform the subscriber about a product or service. After the informational promotion is played, the subscriber is returned to the point after the slot in the voice application where they would have arrived regardless of the played promotion.

Informational promotions have only one type of prompt: Initial.

Segments and Variables

A *segment* is a group of logical rules, based on segment *variables,* that defines a specific market segment of subscribers.

Variables are of two types: *demographic* and *usage*.

- **Demographic Variables** describe a market segment's characteristics such as age, location, and so on. Optionally, IOM can evaluate offers against demographic data sent by your voice application.
- **Usage Variables** describe a segment's buying or other patterns, telephone usage, purchase information, purchasing profiles, or other data. IOM evaluates offers against data supplied to it by a datafeed from the carrier's billing or CRM systems.

There are two general types of segments: *upsell* and *fulfillment*.

Upsell Segments

Upsell segments are defined by rules based on segment variables. This is the most common type.

With the IOM Tool, you can create any kind of segment variable. Because billing and CRM data vary from carrier to carrier, determining what variable names to create is a function of the what data is available and the type and depth of segmentation required by the offer. You should create variables based on the carrier's back-end system data and the kind of rules you want to create.

Some examples of segment variables:

- Account Balance, Days Until Due Date, Products On Account, Market, Rate Plan
- TotalMinutes, TotalTimes

After you define segment variables, with the IOM Tool, for each segment you specify rules based on those segment variables that tell the IOM Engine how to determine the appropriateness of an offer. For example, to describe the segment "People whose account balance is due within three days" using a variable called **Days until Due Date**, you might define a rule like this:

Days Until Due Date Equals 3

Or to describe the segment "People whose total minutes over the last four weeks is between 100 and 300," you could use a variable called **Total Minutes** to define a rule like this:

Total Minutes Between 100, 300

Fulfillment Segments

With fulfillment segments, the carrier has done all the marketing in advance and outside the IOM system, with direct mail or billing inserts (for example). The carrier creates a segment file, which is a list of look-up keys (usually, Mobile Identification Numbers, or MINs) that correspond to the recipient's carrier used in the marketing campaign. This segment file is loaded into the IOM database by an IOM datafeed (as described in the *IOM Developer and Datafeed Reference*). When a subscriber calls your voice application, it looks up the key to determine whether or not to play a prompt to fulfill the carrier's promotion.

Priority and Weight

With the IOM Tool, when you associate an offer with a particular campaign, you must assign the offer a priority and a weight.

- When there are competing offers for a slot, the IOM Engine selects the offer based on priority. An offer's *priority* is High, Medium, or Low and is the relative importance of the offer with respect to other offers configured for the same voice application slot. All offers can have the same or different priority.
- The offer's *weight* is a numerical value from 1 to 100 you assign to the offer with the IOM Tool. The weight determines how often an offer is chosen. If offers have the same priority, the IOM Engine chooses based on the offer's weight. In addition to priority, weight gives more granular control over the selection of an offer by the IOM Engine. For example, if two offers in the same slot have the same priority, the IOM Engine makes its selection based on weight.

Some examples:

Priority: If two offers are configured for the same slot, one with high priority and one with medium priority, the offer with high priority is selected first. On the other hand, if the segment of the high priority offer does not match or if the promotion has already been played to the caller, the medium priority offer is evaluated.

Weight: If two offers in the same slot have the same priority, but one has a weight of 75 while the other has a weight of 95, the offer with weight 95 is selected for evaluation.

Random Selection: If two offers in the same slot have the same priority and the same weight, then the offers are chosen at random, such that each will be played half the time.

Chapter 4 Tool Tutorials: Managing IOM Campaigns

In these tutorials, we will design and create three Intelligent Offer Management (IOM) campaigns and all the campaign objects they need. In "Tutorials: Writing the Voice Application" on page 89, we will also look at the design of voice application that uses the campaigns.

Description of the Campaigns

Our company's name is National Wireless. We want to let our English-speaking customers know about two things. We will create three campaigns:

1. A **Global Voice Mail Outage** campaign to play a message to all subscribers whenever our entire voice mail system is under maintenance. We want to be able to turn this message on and off at will.
2. A **Market-based Voice Mail Outage** campaign to play a message to subscribers in the Boston area whenever our system in that area is under maintenance. We want to be able to turn this message on and off at will.
3. A **New Promotions** campaign. We have three types of products to offer:
   - A Loyalty Program offer for subscribers who have been with us more than three months and who have at least $5 in their accounts.
   - A Text Messaging offer for subscribers who send 30 or more messages a month and who do not have our TextMsg200 product
   - An Accessories offer for all subscribers.

Here's what we have to do to create these campaigns with the IOM Tool:

1. We have two locations that we have chosen as strategic spots in the call flow for offers. So we need to create two IOM **slots** where these campaigns and messages can be played.
2. We have four specific kinds of customers, one kind for each possible message. So we need to define specific IOM market **segments** by creating **variables** and **rules** that define these segments. (The Accessories message will be played to all subscribers, so we do not need to define a segment for it.)
3. We have five different promotions that can be played to our subscribers. So we have created a minimum of five audio files and associated grammars and destination files, or IOM **promotions**, one for each message (as shown below), that we will upload to IOM.

Table 1 Promotions Created for the Campaigns

Promotion	Text	Audio File Name
a. Global service outage	"We're sorry but we are upgrading the voicemail system in your area and voice mail voice mail for subscribers in that area will not be available. The estimated time for this outage is three hours."	`GlobalVoiceMail Outage.wav`
b. Boston-specific service outage	"We're sorry but we are upgrading the voicemail system in the Boston area and voice mail voice mail for subscribers in that area will not be available. The estimated time for this outage is three hours."	`BostonVoiceMail Outage.wav`
c. Loyalty Program	"By the way, National Wireless is offering a special bonus to select customers. If you refill your account now, we we'll give you an additional 20% bonus. Would you like to take advantage of this great deal?"	`LoyaltyInital Prompt.wav`
d. Text Messaging	"So I've noticed that you've been sending a number of text messages lately but don't have one of National Wireless's text messaging bundles. If you send a lot of text messages, a text messaging bundle is actually cheaper per message compared to the pay-per-use plan. Would you like to hear more about of text messaging bundles?"	`TextMsg200Upsell InitialPrompt .wav`
e. Accessories	"We are offering many different phone accessories. Would you like to hear more about these great deals?"	`iom_accessories .wav`

4. We also have to upload other auxiliary files associated with some of these promotions, such as supporting prompts, grammars and destination files, that are needed by our voice application.

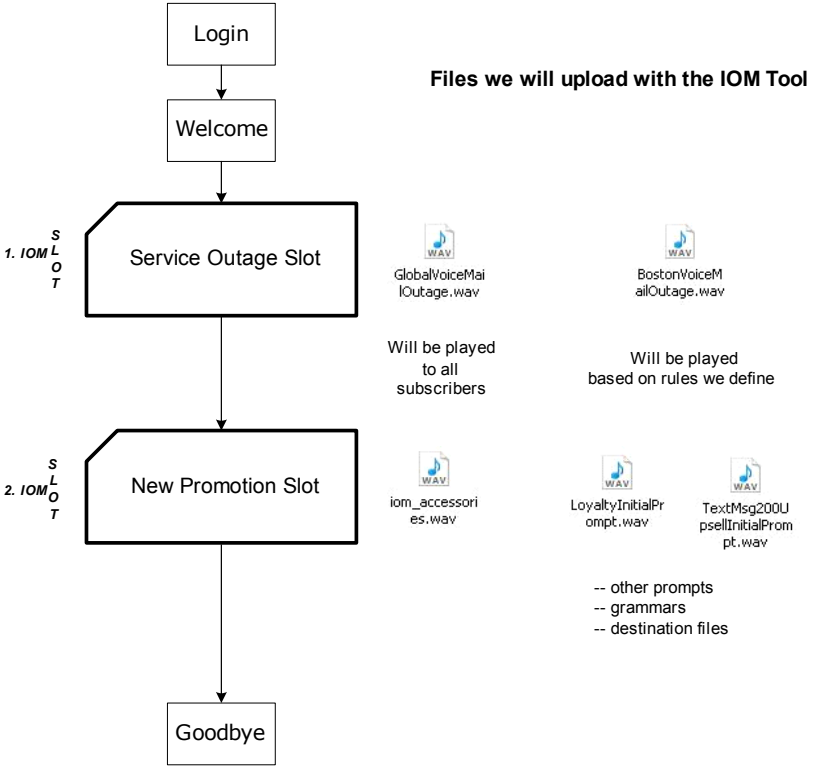
Overview to the Voice Application

We will write a simple voice application that uses all three IOM campaigns. The figure below is the high-level design of the voice application. In "Tutorials: Writing the Voice Application" on page 89, we will look at the voice application in greater detail.

The voice application has been designed with two IOM **slots** we will create with the IOM Tool:

1. Service Outage Slot: At the beginning of the call after the welcome greeting.
2. New Promotion Slot: After the Service Outage slot.

Figure 3 Voice Application Slots



Overview to Building Campaigns with the IOM Tool

Now that we have a good working definition of our campaigns, we will use the IOM Tool to create them.

Note: To guide our work with the IOM tool, we can refer to the steps in "General Approach to Using IOM" on page 19.

Tutorial Objectives: Use the IOM Tool to create three campaigns.

Set-up: A description of our campaigns, with working definitions of the messages (promotions, or audio files), the voice application slots, and the market segments we need to create. We have created five promotions (or audio files) that can be played by our campaigns. We also have auxiliary files needed by our voice application, such as supporting prompts, grammars, and destination files.

Description of Tutorials

We will learn how to:

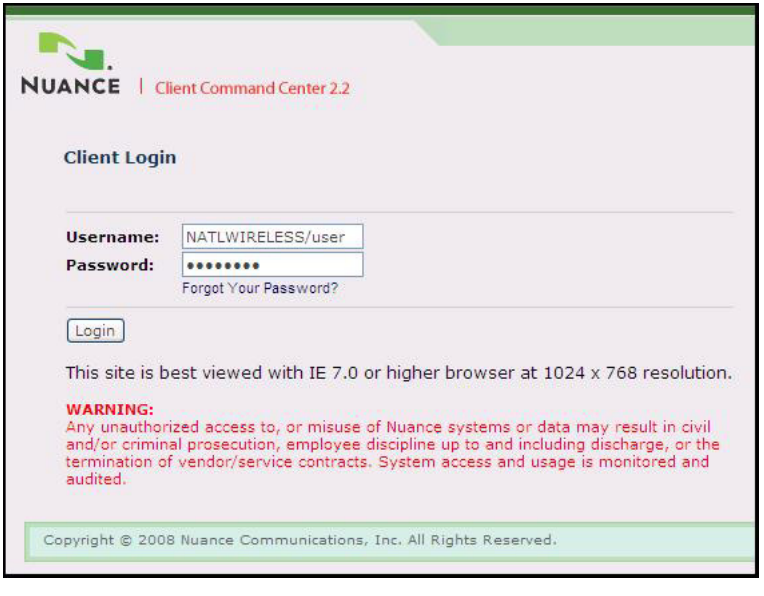
1. Login to the IOM Tool.
2. Select an IOM application.
3. Create slots to be used by our voice application.
4. Create variables to define market segments.
5. Use those variables to build rules that define the market segments.
6. Use the Campaign Wizard to create the Global Voice Mail Outage campaign with a single offer that includes an informational promotion.
7. Create the Market-based Voice Mail Outage campaign with one offer that includes an informational promotion.
8. Enable the Global Voice Mail Outage campaign.
9. Create the New Promotions Campaign with three offers that include interactive promotions (audio files and other associated files).
10. In Chapter 5, write the voice application that exercises our campaigns.

1: Logging In

To login, you need to know the URL to use, your domain and username, and your password. You will be given this information by your Nuance Account Representative.

1. With your web browser, open the appropriate URL.
2. In the **Username** field, enter the *DOMAIN*/*username* you have been told by your Nuance Account Representative.
3. In the **Password** field, enter your password.

If you are an IOM Tool administrator, see Appendix A in the *IOM Tool User Guide* for details about creating domains and users.

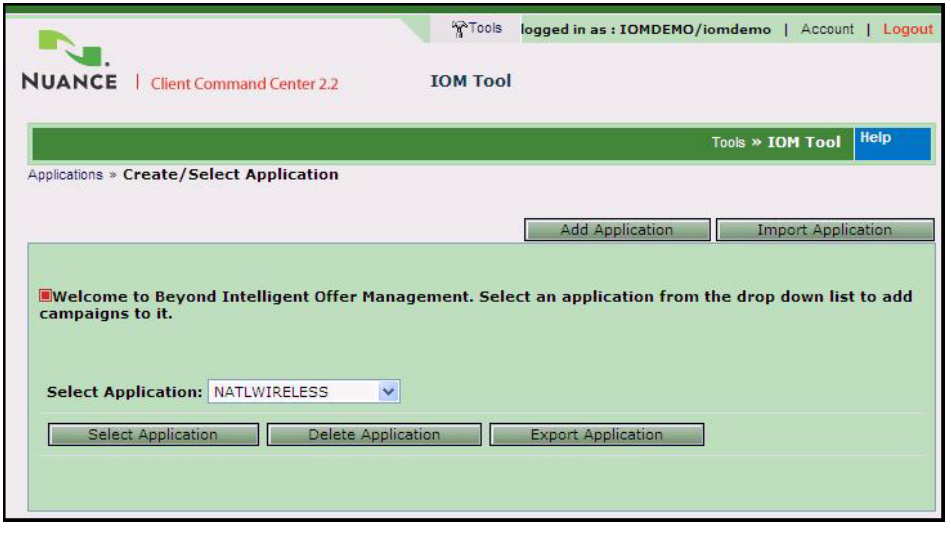


4. Click **Login**.

2: Selecting the Application

After you login, you must select the application you want to work with. (If you don't see the name of your application in the pulldown menu, an application for your company has probably not yet been created).

For information about creating applications, see "Managing Applications" in the *IOM Tool User Guide*.



1. From the pulldown menu, select the name of the desired application.
2. Click **Select Application**.

3: Creating Slots

Now we will create the slots used by our voice application where IOM can possibly play the promotions of our campaign. We need to create two slots: **Service Outage Slot** and **New Promotion Slot**. We usually need to create slots only one time, because they rarely change, and when they do change, we must also change our voice application.

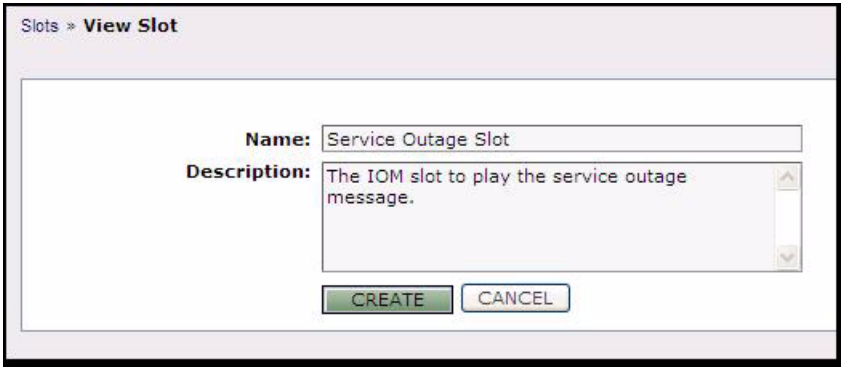
For detailed information about slots, see Chapter 9 in the *IOM Tool User Guide*.

1. On the top menu bar, click **Slots**.

2. On the far right, click **NEW SLOT**. 

3. Enter the name and description of the first slot and click **Create**.

We need to enter the *exact* name of the slot that our voice application will refer to, with spaces (if any) and capitalization.



4. Enter the name and description of the second slot and click **Create**.

We're done with the slots for now, but we'll use them again when we add offers to our campaigns.

4: Creating Variables

Now we'll define the market segments of our subscribers that we want to hear our specific promotions. This is the heart of Intelligent Offer Management (IOM), and it can appear to be very complex, but it's not. It's simple logic. We are defining the marketing logic in IOM itself, rather than in our voice application, so our voice application is easier to design and maintain.

The key idea is that we need to tell IOM how to *distinguish the differences among our subscribers* so our voice application can play messages to them. Because IOM is a program, we need to be very specific about these distinctions. We will create **variables** with which we can make these distinctions, and then in the next section we'll use the variables to build rules that distinguish the segments.

Deciding on Variable Names

Let's look again at the definition of the five messages that we came up with earlier in "Description of the Campaigns". We need to translate these into IOM terms. We'll go one-by-one through these definitions and decide what variables we need. Then, we'll use the IOM Tool to create them.

Table 2 Variables to Define for the Campaigns

Description	Explanation	Variables Needed
A global voicemail outage message for all subscribers	We need to know if the subscriber uses our voicemail product.	Let's call this variable "**Products**."

Description	Explanation	Variables Needed
A voicemail outage message specific to the Boston area	We need variables that state: ■ Whether the subscriber uses our voice mail product ■ The subscriber's geographic area.	■ We'll use our "**Products**" variable. ■ Let's call this variable "**Market**."
A Loyalty Program offer for subscribers who have been with us more than three months and who have $5 in their accounts.	We need two distinguishing variables: ■ How long the caller has been a subscriber. ■ How much money the subscriber has with us.	■ Let's call this variable "**Longevity**." ■ Let's call this variable "**Account Balance**."
A Text Messaging offer for subscribers who send 30 or more messages a month and who do not have our TextMsg200 product	We need two distinguishing variables: ■ Number of messages the subscriber sends per month. ■ Which products the subscriber has.	■ Let's call this variable "**Text Messages**." ■ We'll use our "**Products**" variable.
An Accessories offer for all customers	We do not need to distinguish subscribers, because everybody will hear the message.	none

We've determined that we need five variables. Now we're ready to create them.

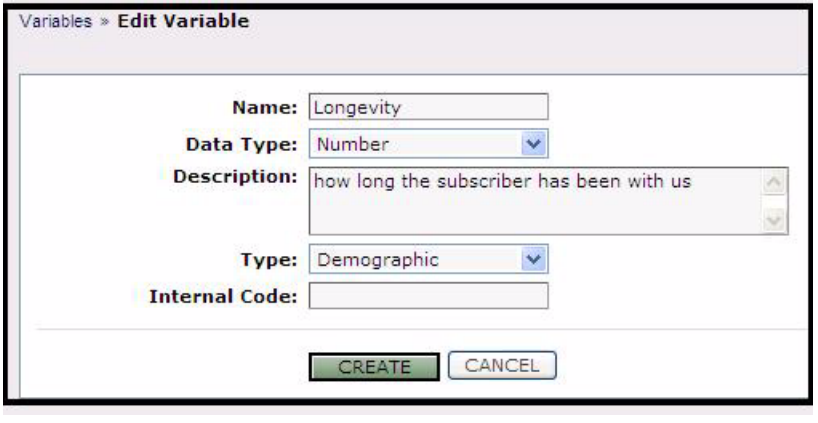
Creating the Variables

1. In the IOM Tool, on the top menu bar, click **Variables**.

2. On the far right, click **NEW VARIABLE**. 

3. Enter the name of the first variable, **Longevity**.

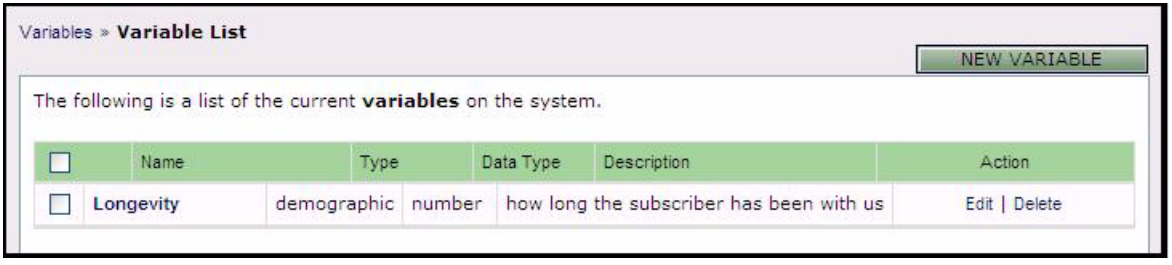
4. Longevity is a number—the number of months a subscriber has been with our company—so with the **Data Type** pulldown menu, set it to **Number**.



5. Fill in the description of the Longevity variable.

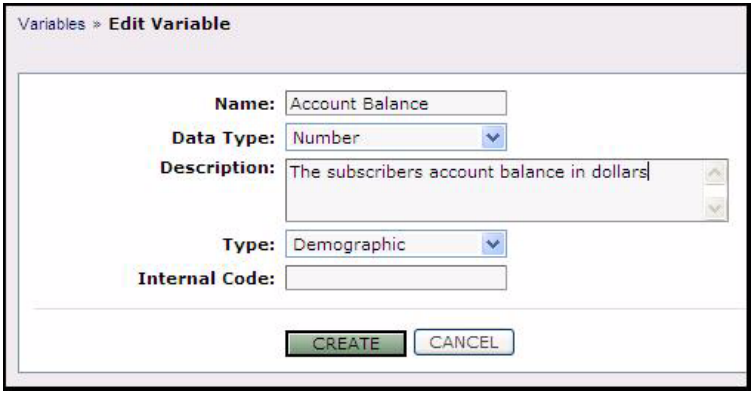
6. Leave its **Type** as the default, **Demographic**.

7. Click **CREATE**. The new variable is listed on the Variables page.

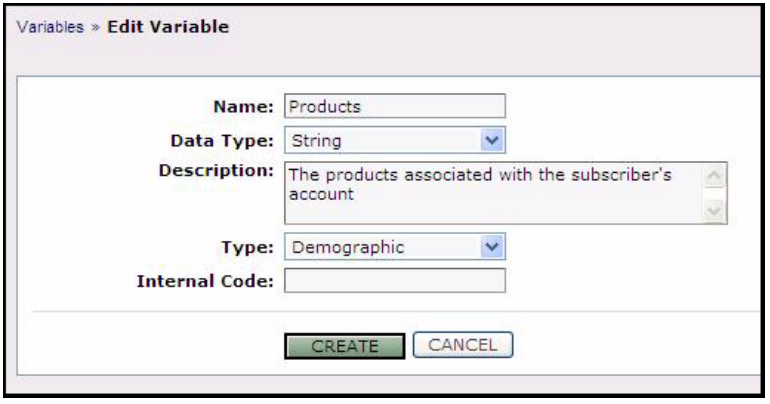


We usually need to create a variable only one time, because they do not change very often. We can reuse them in many different rules to define many different market segments.

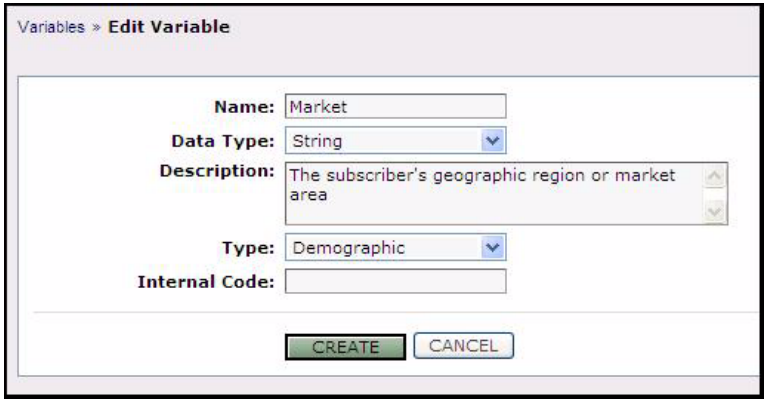
8. Now create the remaining four variables, clicking **NEW VARIABLE**, filling in the details, and clicking **CREATE** for each one.
   - **Account Balance** is data type **Number** and type **Demographic**.



- **Products** is data type **String** and type **Demographic**.



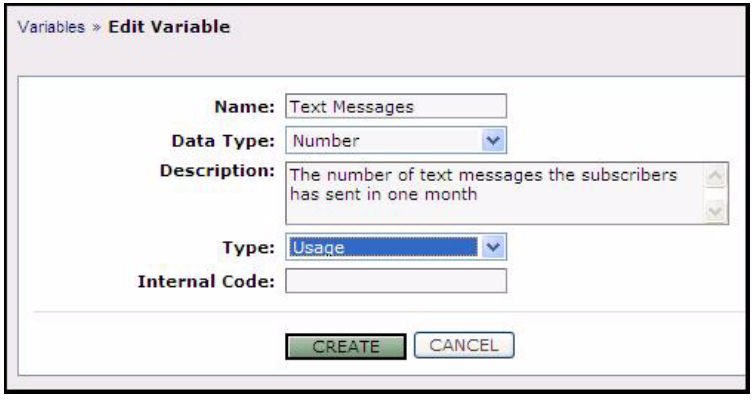
- **Market** is data type **String** and type **Demographic**.



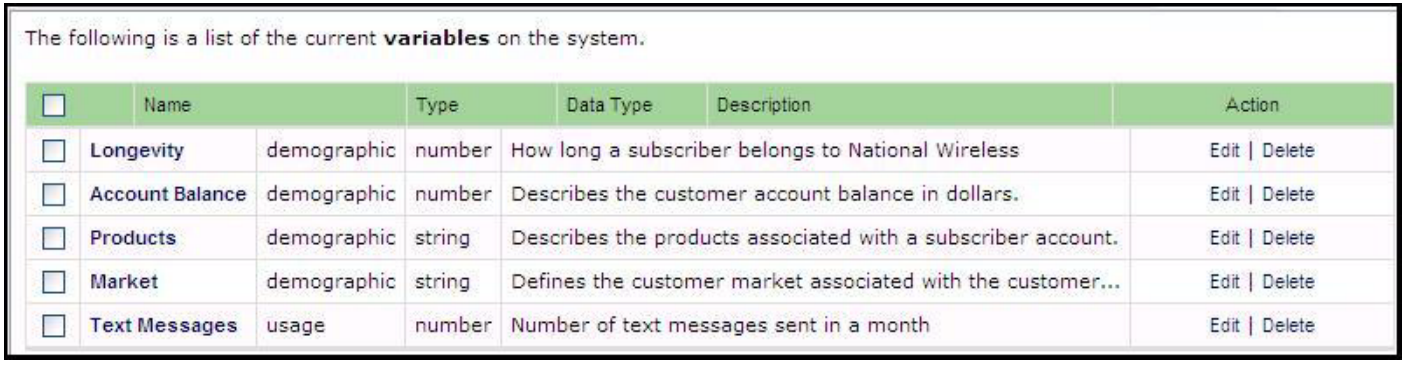
Usage variables represent data coming from the client's back-end systems by way of an IOM data-feed. In this case, the client has recorded the number of text messages sent by the subscriber. The voice application, at runtime, will use the backend data to make a comparison.

For more differences between Demographic and Usage variables, see "Segments and Variables" on page 24.

- **Text Messages** is data type **Number** and type **Usage**.



Our complete list of variables looks like this:



	Name		Type	Data Type	Description	Action
☐	Longevity	demographic	number	How long a subscriber belongs to National Wireless		Edit | Delete
☐	Account Balance	demographic	number	Describes the customer account balance in dollars.		Edit | Delete
☐	Products	demographic	string	Describes the products associated with a subscriber account.		Edit | Delete
☐	Market	demographic	string	Defines the customer market associated with the customer...		Edit | Delete
☐	Text Messages	usage	number	Number of text messages sent in a month		Edit | Delete

Now we're ready to use our variables to build rules that define the market segments.

5: Defining Segments with Variables and Rules

We have to define four different market segments of subscribers. We will use the variables we made in the last section to create rules that define the segments.

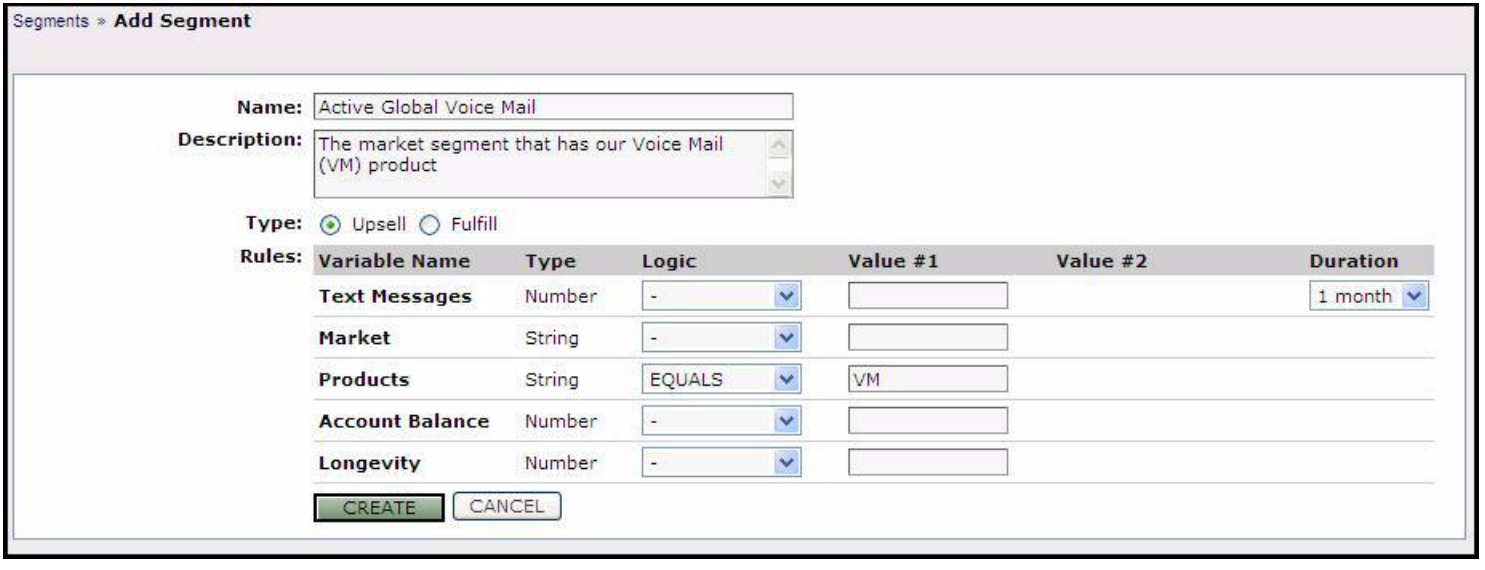
Table 3 Segments to Define with Rules and Variables

Segment Name	Description	Rules Based on Variables
Active Global Voice Mail Users	Subscribers who have our "VM" (voicemail) product	Products
Active Market-based Voice Mail Users	Subscribers in the Boston area geographic market who have our "VM" product	Market Products
Heavy Text Message Users without TextMsg200	Subscribers who send more than 30 text messages per month and who do not have our TextMsg200 product	Text Messages Products
Loyal Customer Segment	Subscribers for more than 3 months who have between $0 and $5 in their accounts	Longevity Account Balance

1. In the IOM Tool, on the top menu bar, click **Segments**.

2. Click **NEW SEGMENT**. 

3. Define the rule that describes the **Active Global Voice Mail Users** segment.



a. Enter the name and description of the segment.

b. This is an upsell segment, so leave the **Type** of the segment to the default, **Upsell**.

c. "Active Global Voice Mail Users" are those subscribers who have our "VM" product. So the rule that defines this segment is the Products variable set like this:
Products EQUALS VM

d. Click **CREATE**.

For the difference between Upsell and Fulfillment segments, see "Segments and Variables" on page 24.

4. Now define the **Active Market-based Voice Mail Users** segment.

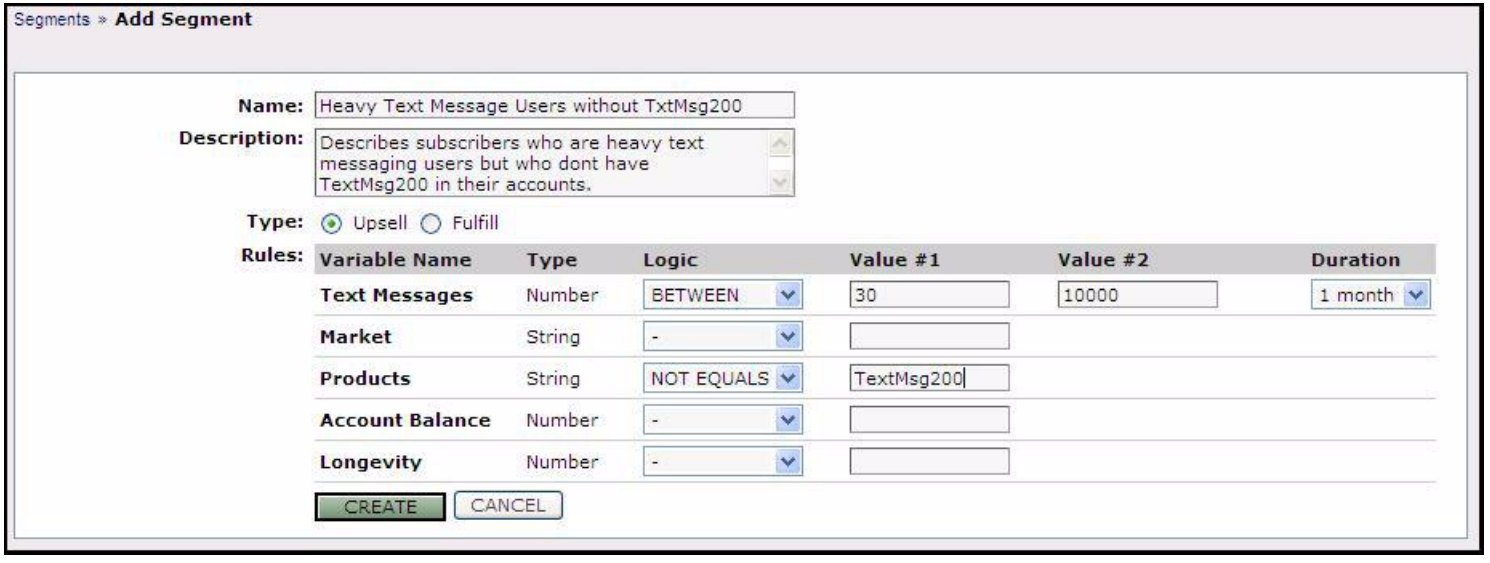
a. Enter the name and description of the segment.

b. This is an upsell segment, so leave the **Type** of the segment to the default, **Upsell**.

c. "Active Market-based Voice Mail Users" are those subscribers who have our "VM" product and who live in the Boston area. So set the Market and Products variables like this:
Market EQUALS Boston
Products EQUALS VM

d. Click **CREATE**.

5. Now, the **Heavy Text Message Users without TextMsg200** segment.



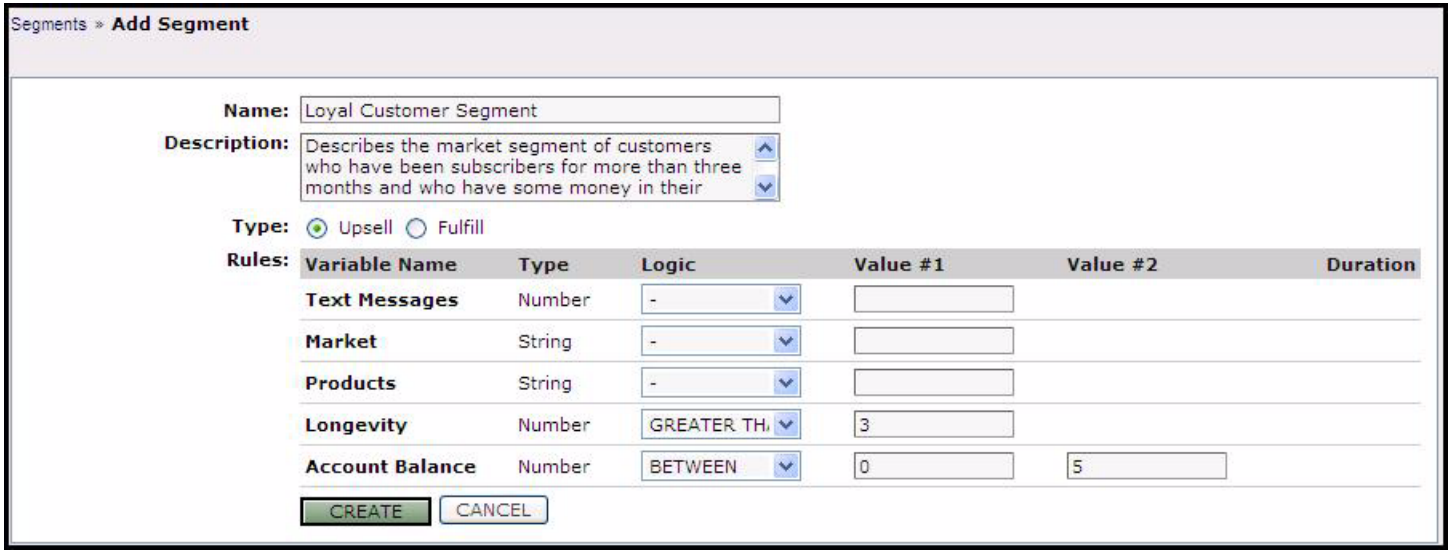
a. Enter the name and description of the segment.

b. This is an upsell segment, so leave the **Type** of the segment to the default, **Upsell**.

c. "Heavy Text Message Users without TextMsg200" send more than 30 message a month but do *not* have our TextMsg200 product. So set the Text Messages and Products variables like this:
Text Message BETWEEN 30 1000 Duration: 1 Month
Products NOT EQUALS TextMsg200

d. Click **CREATE**.

6. Finally, define the **Loyal Customer Segment**.



a. Enter the name and description of the segment.

b. This is an upsell segment, so leave the **Type** of the segment to the default, **Upsell**.

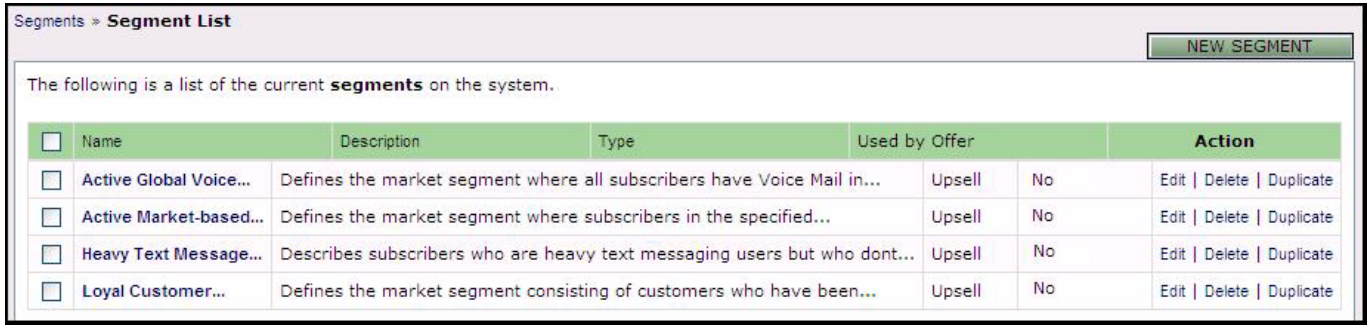
c. The "Loyal Customer Segment" are subscribers who have been customers for more than three months and who have between $0 and $5 in their accounts. So set the Account Balance and Longevity variables like this:

Longevity GREATER THAN 3

Account Balance BETWEEN 0 5

d. Click **CREATE**.

We're done. Our complete list of segments looks like this:



We are now ready to create campaigns to target offers to these segments.

6: Using the Campaign Wizard

Now we'll use the New Campaign Wizard to put together the pieces for the first of the three campaigns, the Global Voice Mail Outage campaign, which we want to use whenever we have a planned outage for maintenance.

How the Wizard Works

The Campaign Wizard works from the top down, starting with the campaign, moving to offers, and ending at promotions. We will be guided through each step.

1

A *Campaign* definition that contains...

2

An *Offer*, targeted at the Active Global Voice Mail Users market segment and associated with the Service Outage Slot, that contains...

3

A *Promotion* with an audio message in the file GlobalVoiceMailOutage.wav that we will upload.

Running the Wizard

1. In the IOM Tool, on the top menu bar, click **Campaigns**.

2. Click **NEW CAMPAIGN**. 

   This starts the Campaign Wizard.

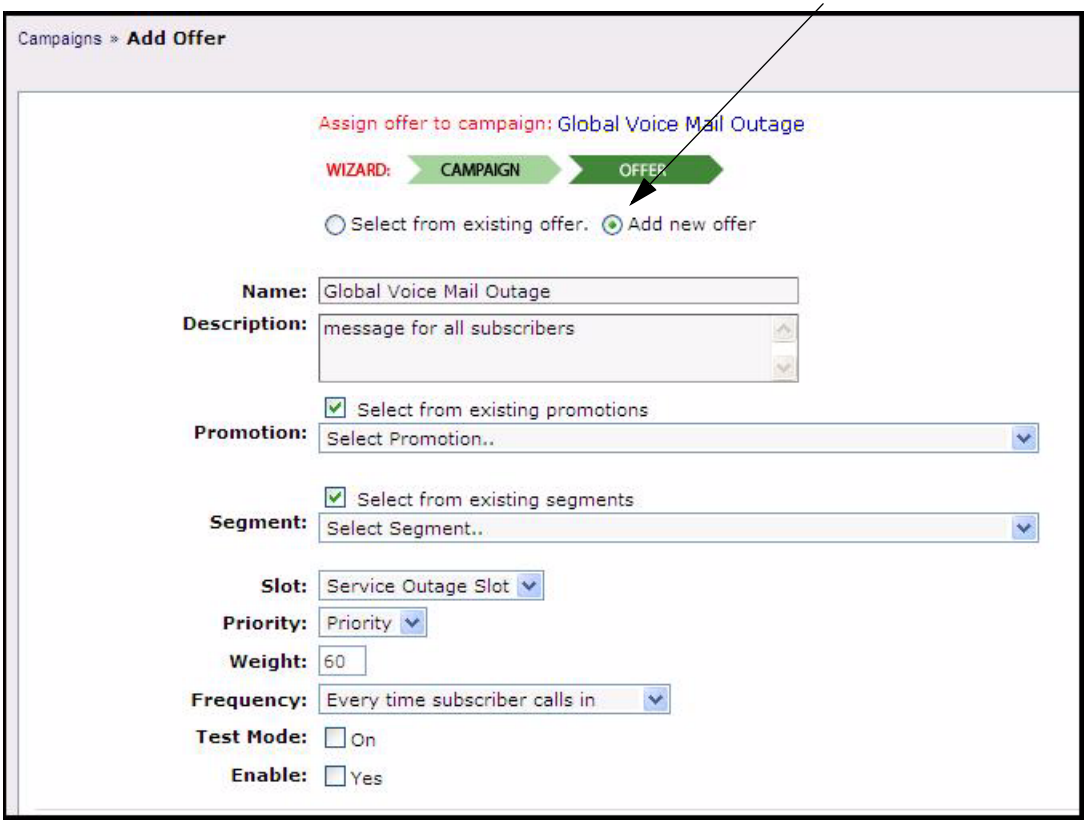
3. Enter the name and description of the first campaign, Global Voice Mail Outage campaign:

4. Because we will use this campaign again and again, it has no ending date. On the far right, click **Forever**.

5. We do not want this campaign enabled right now, because we do not have any planned maintenance at this time. For the **Enabled** field, click **No**.

6. Click **NEXT** to add offers to the campaign.
7. On the **Add Offer** page, we have many choices. Let's examine each one.

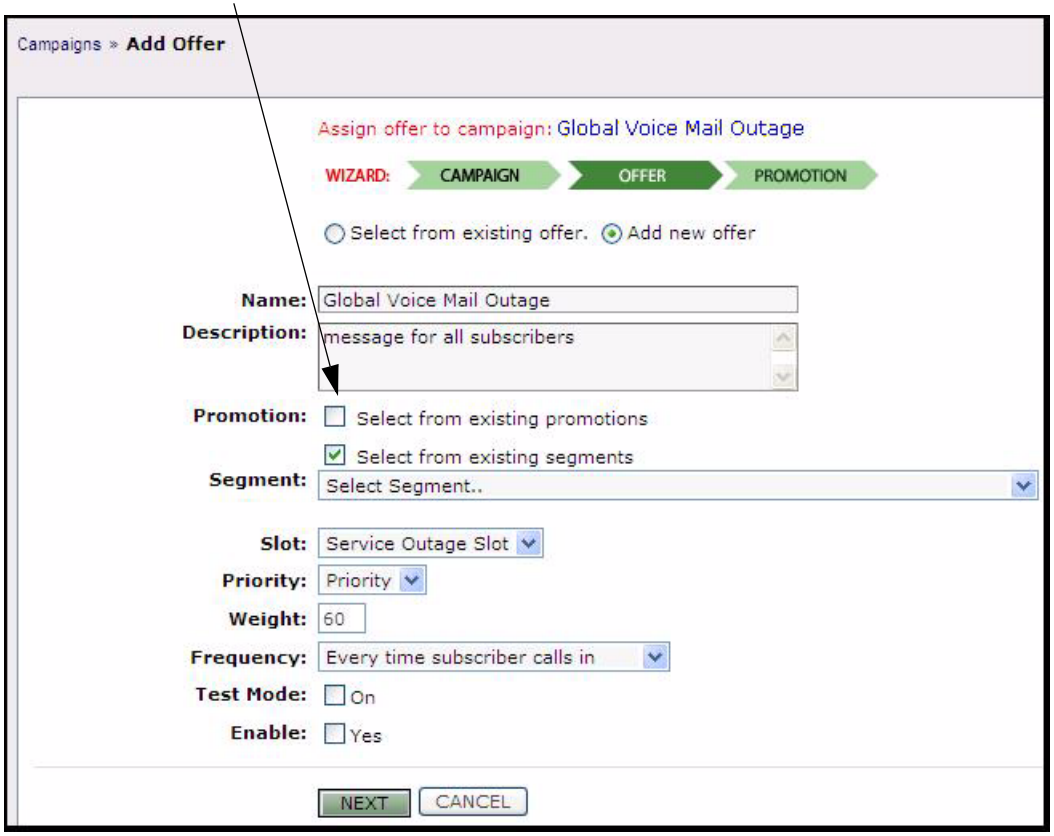
8. We have not yet created offers for this campaign, so click **Add new offer**.



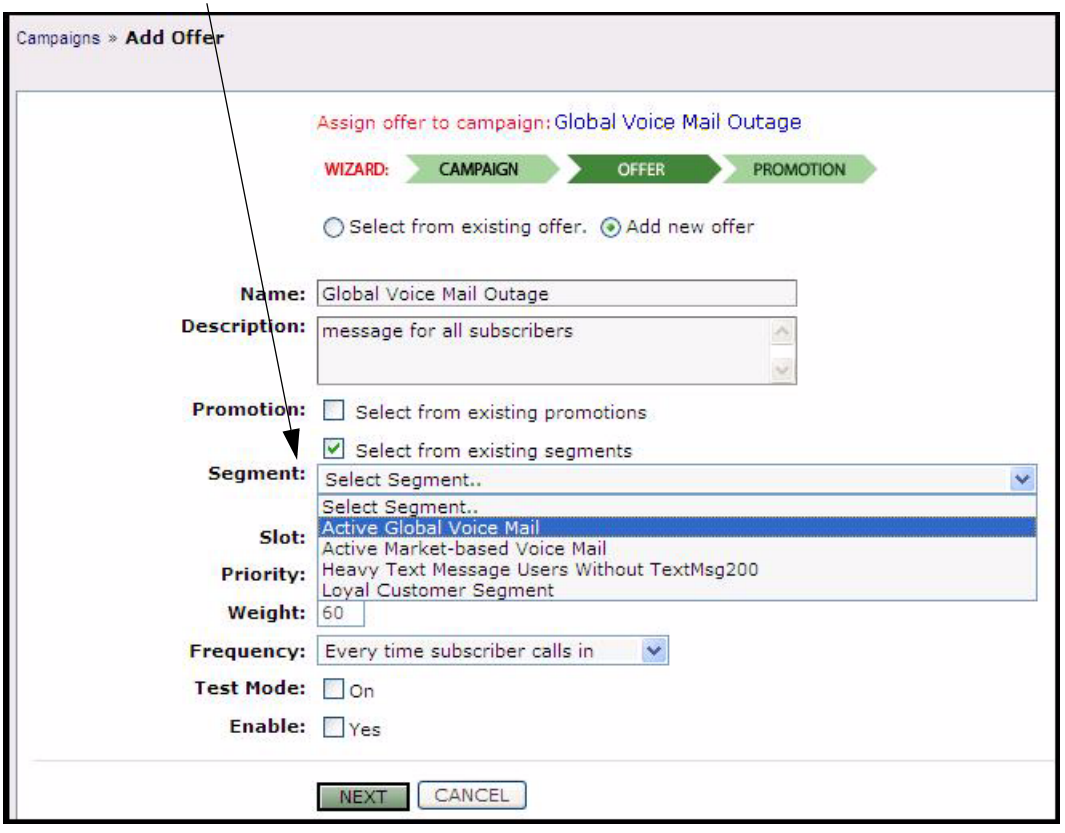
9. Enter the name and description of the offer.

10. We have not yet defined the promotion in the IOM Tool for this campaign, so uncheck **Select from existing promotions**.

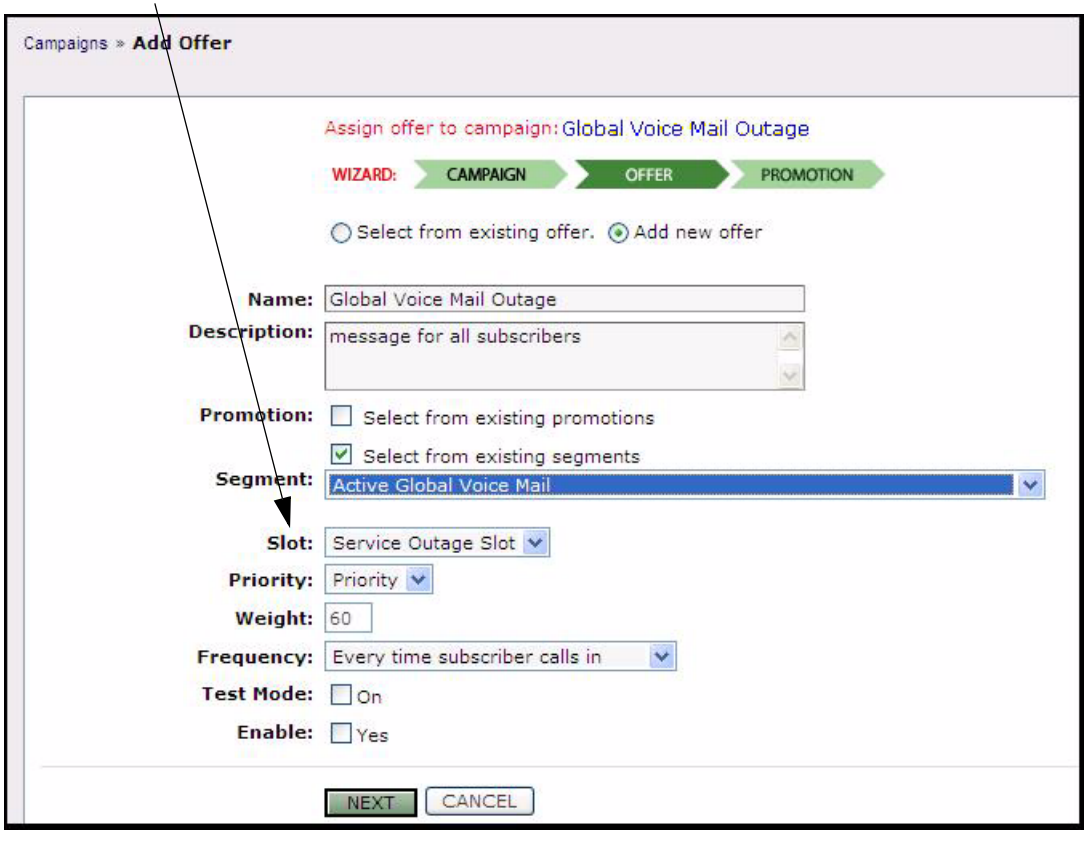
The New Campaign Wizard has added the third step, PROMOTION (in light green at the far right top). When we finish defining the offer and click NEXT, the wizard will take us to the **Add Promotion** page so we can upload our promotion audio file.



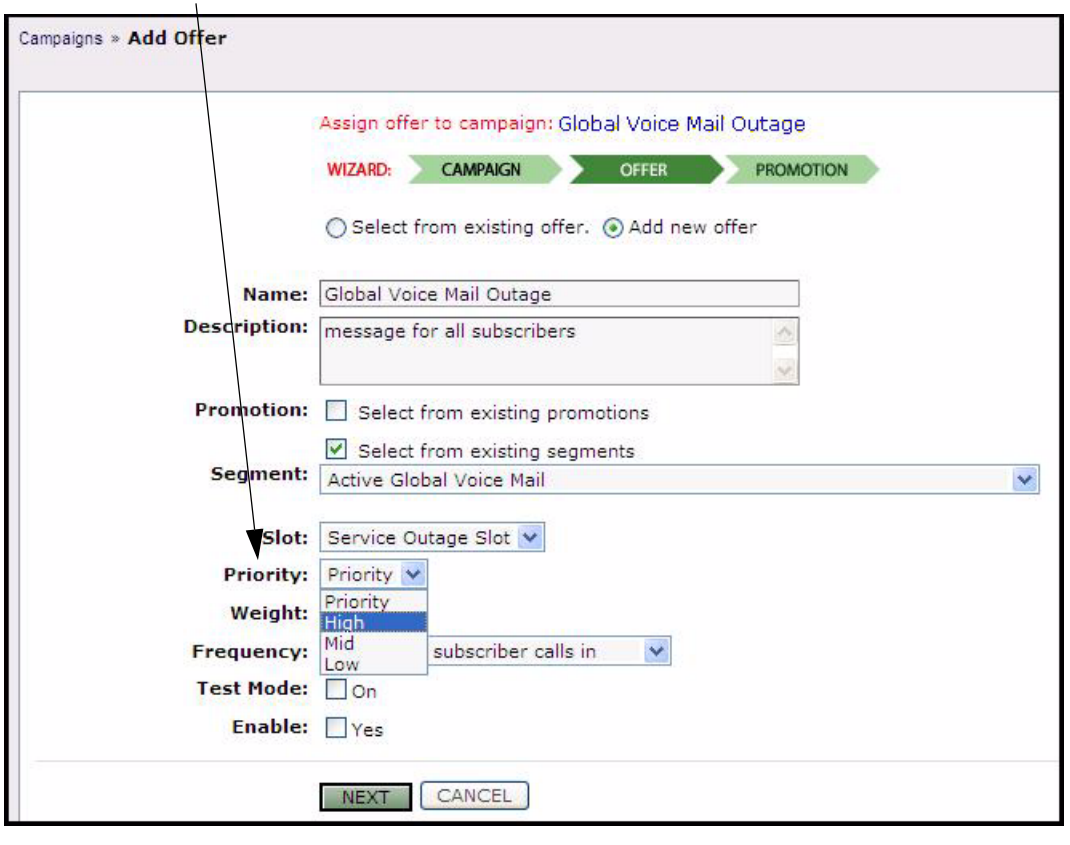
11. We created segments in "5: Defining Segments with Variables and Rules" on page 41. With the pulldown menu under **Select from existing segments**, select our **Active Global Voice Mail Users** segment.



12. We created slots in "3: Creating Slots" on page 33. The slot we want, the **Service Outage Slot**, has already been selected by default.



13. This message is urgent. We want everyone to hear it. From the **Priority** pulldown menu, select **High**.



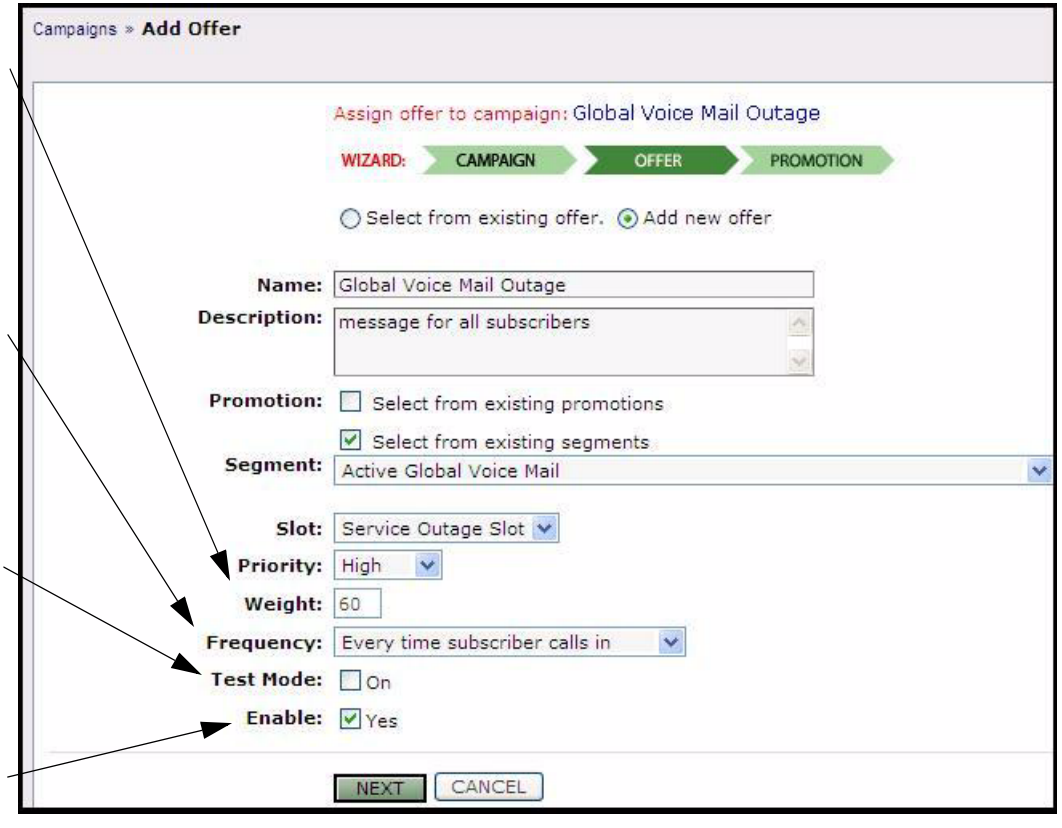
14. Now we'll set the remaining fields for this offer.

For the **Weight** of the offer, the default **60** is acceptable.

For **Frequency**, the default **Every time the subscriber** calls in is exactly what we want.

We are not testing this offer, so leave **Test Mode** unchecked. (For information about test mode, see *IOM Tool User Guide*.)

We want the offer to be playable to our subscribers. For **Enable**, check **Yes**.



15. To create the offer, click **NEXT**. The Wizard moves to the **Add Promotion** page.
16. We created the audio file for this promotion when we defined the campaign (see "Description of the Campaigns"). On the **Add Promotion** page we will upload it to IOM.

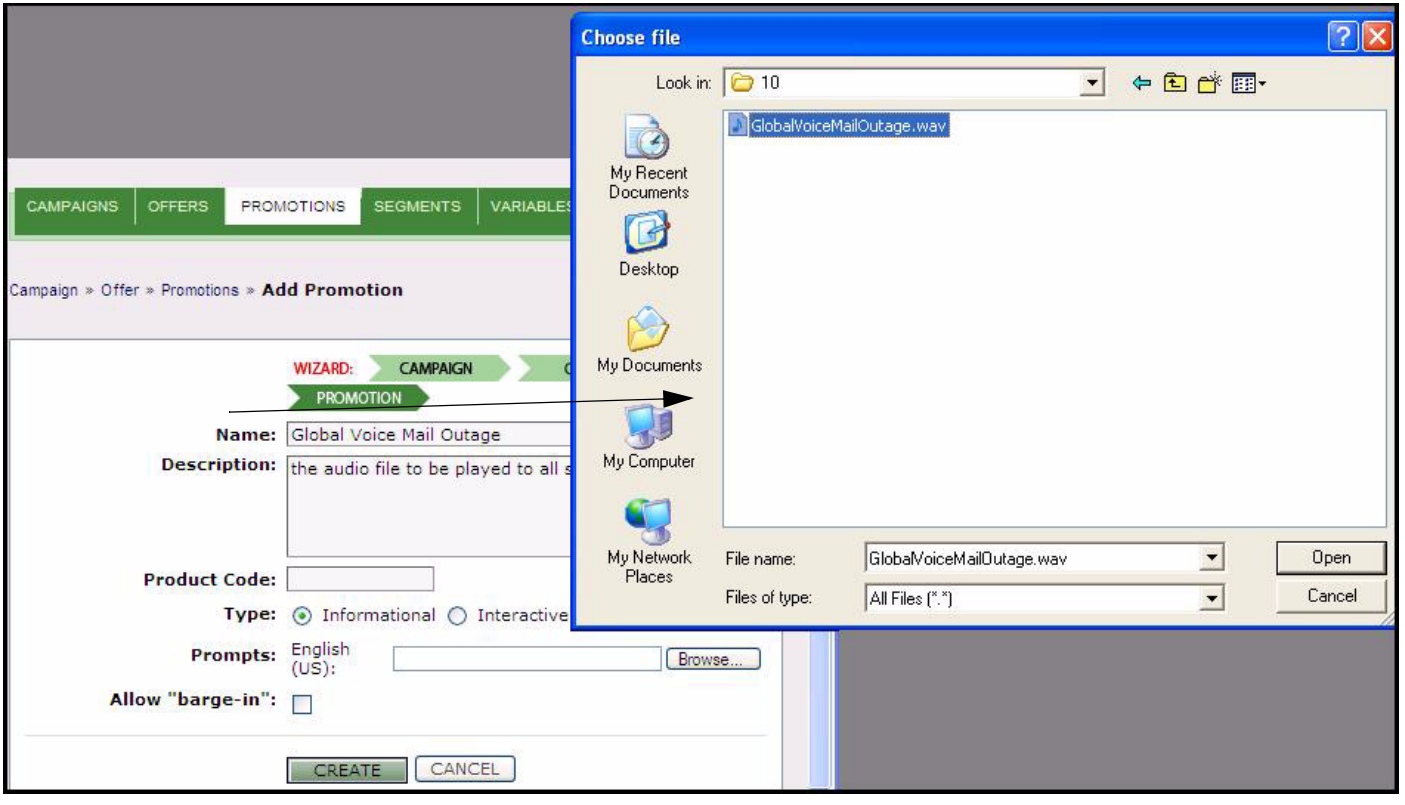
Enter the name and description of the promotion.

We don't want to associate any **Product Code**, so leave this field blank.

This is an *informational promotion*: the subscriber who hears it does not have to do anything in response, merely listen to it. See "Informational Promotion" on page 24 for definitions.

17. We're ready to upload the audio file. Next to the **Prompts** field, click **Browse**.

18. We've got the `GlobalVoiceMailOutage.wav` file on our local computer, so navigate the disk and open the audio file.



19. Click **CREATE**.

The promotion is created.
The audio file is uploaded.

We are returned to the first step of the Campaign Wizard (the Campaign definition page), where we can add more offers and promotions to the campaign, if we want. But we're done. We don't want to add any more offers to the Global Voice Mail Outage campaign.

In the top menu bar, click **Campaigns** to see the entire list of campaigns we've created.

Our first campaign is now created and ready to use.

The status of the campaign is **Disabled**, because we selected **No** for the **Enabled** field when we created the campaign. The tutorial "8: Enabling the Global Voice Mail Outage Campaign" on page 67 shows how to enable the campaign.

7: Creating the Market-based Voice Mail Outage Campaign

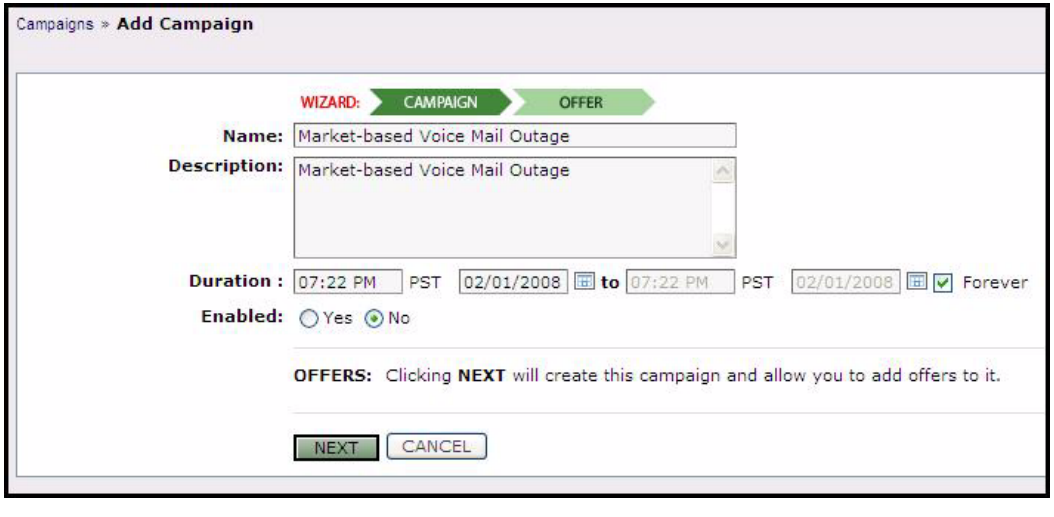
In this tutorial, we'll use the Campaign Wizard again to create the second campaign, the Market-based Voice Mail Outage campaign that will play a message specific to the Boston area.

The only things that differ between the Market-based Voice Mail Outage campaign and the global one are the offer and the targeted segment.

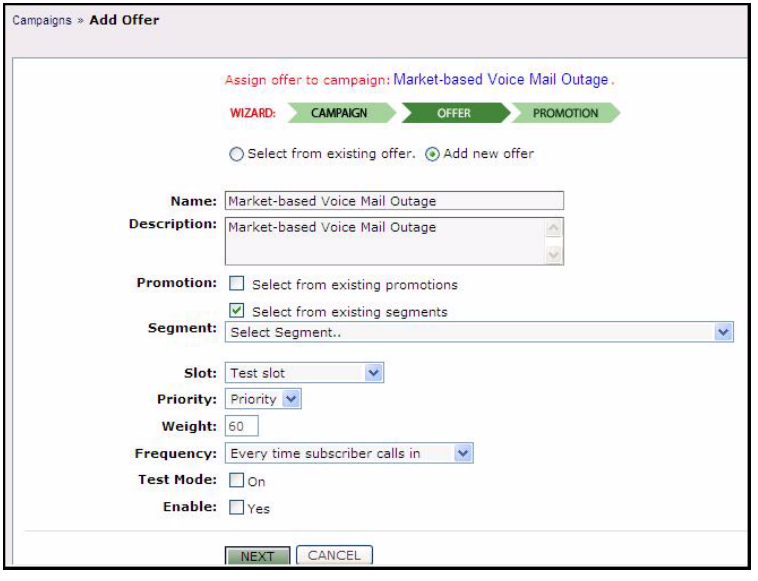
1. In the IOM Tool, on the top menu bar, click **Campaigns**.

2. Click **NEW CAMPAIGN**. 

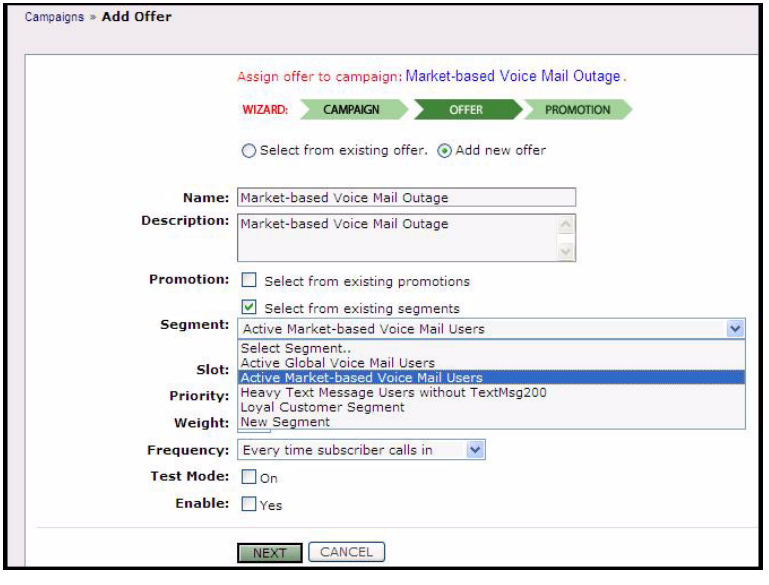
3. Enter the name and description of the campaign: Market-based Voice Mail Outage.



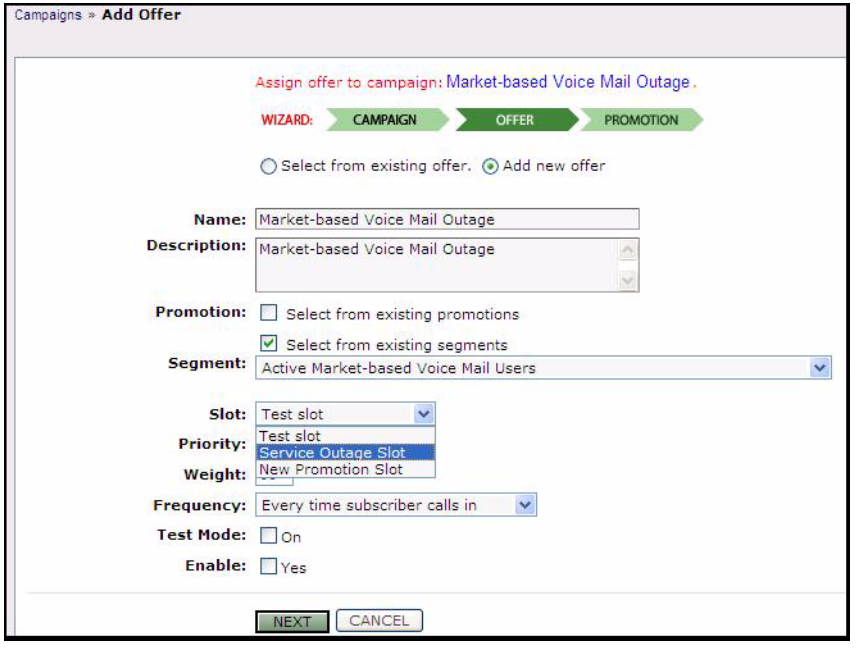
4. Because we will use this campaign again and again, it has no ending date. On the far right, click **Forever**.
5. We do not want this campaign enabled right now, because we do not have any planned maintenance at this time. For the **Enabled** field, click **No**.
6. Click **NEXT** to add offers to the campaign.
7. On the **Add Offer** page:
   a. We have not yet created offers for this campaign. Click **Add new offer**.
   b. Enter the name and description of the offer.
   c. We have not yet uploaded the promotion (audio files) in the IOM Tool for this campaign. Uncheck **Select from existing promotions**.



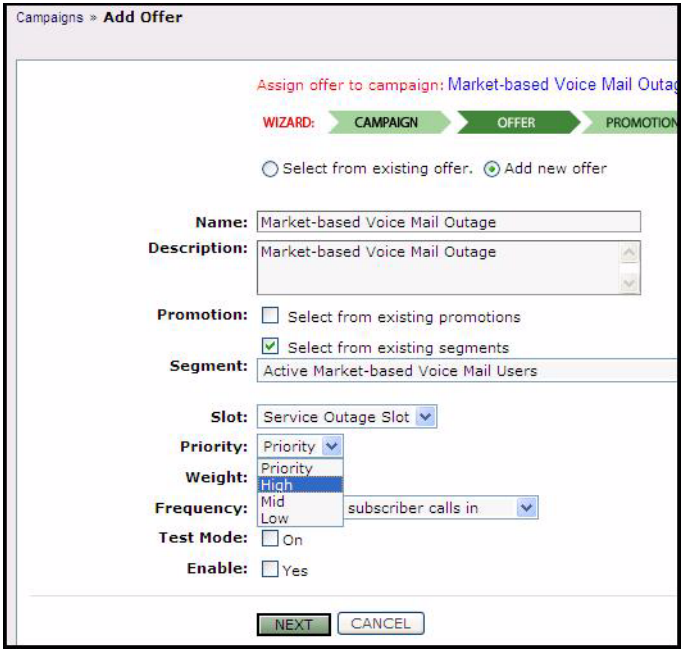
8. We created segments in "5: Defining Segments with Variables and Rules" on page 41. With the pulldown menu under **Select from existing segments**, select the **Active Market Based Voice Mail Users** segment.



9. We created slots in "3: Creating Slots" on page 33. From the **Slot** pulldown menu, select the **Service Outage Slot**.



10. This message is urgent. We want everyone to hear it. From the **Priority** pulldown menu, select **High**.



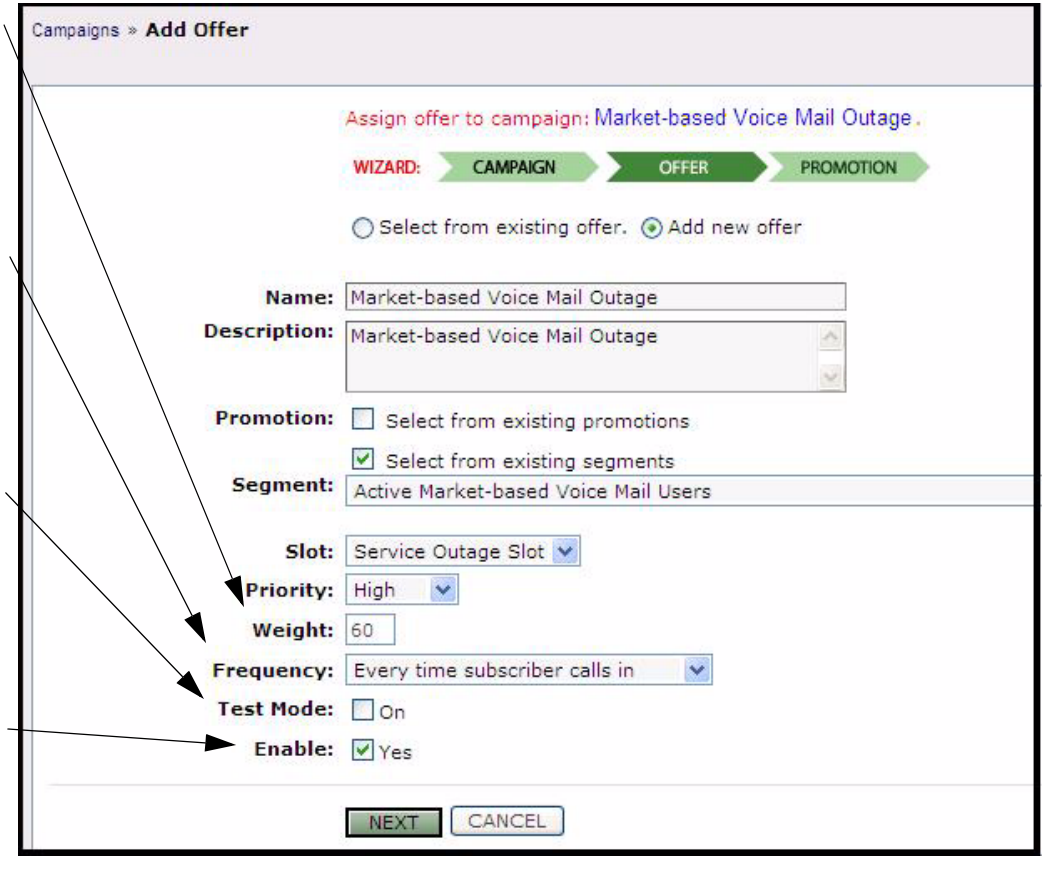
11. Now we'll set the remaining fields for this offer.

For the **Weight** of the offer, the default **60** is acceptable.

For **Frequency**, the default **Every time the subscriber** calls in is exactly what we want.

We are not testing this offer, so leave **Test Mode** unchecked. (For information about test mode, see *IOM Tool User Guide*.)

We want the offer to be playable to our subscribers. For **Enable**, check **Yes**.



12. To create the offer, click **NEXT** to move to the **Add Promotion** page.

13. We created our audio file promotion when we defined the campaign (see "Description of the Campaigns"). On the **Add Promotion** page we will upload it to IOM.

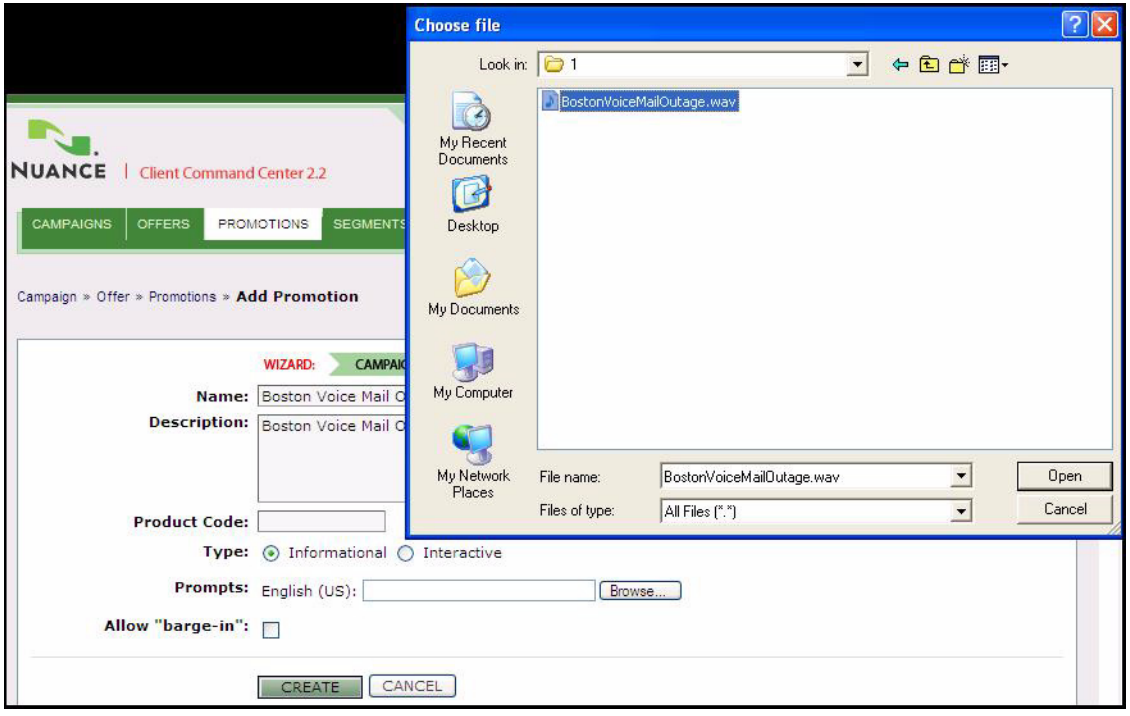
Enter the name and description of the promotion.

We don't want to associate any **Product Code**, so leave this field blank.

This is an *informational promotion*: the subscriber who hears it does not have to do anything in response, merely listen to it. See "Informational Promotion" on page 24 for definitions.

14. We're ready to upload the audio file. Next to the **Prompts** field, click **Browse**.

15. We've got the `MarketBasedVoiceMailOutage.wav` file on our local computer, so navigate the disk and open the audio file.



16. Click **CREATE**.

The promotion is created, and the audio file is uploaded. We are returned to the start of the Campaign Wizard, but we're done defining the Market-based Voice Mail Outage campaign.

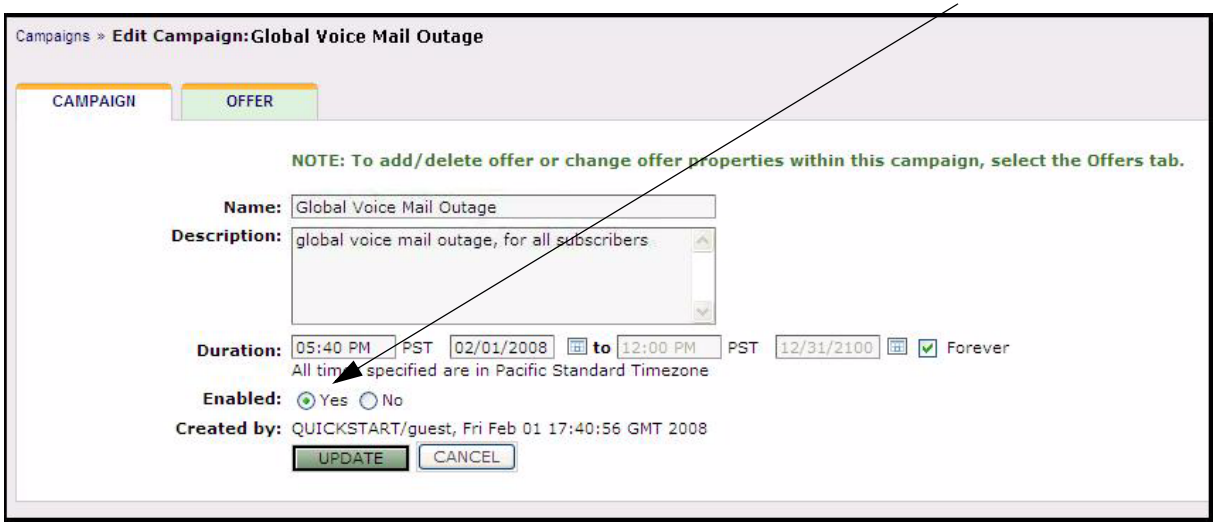
8: Enabling the Global Voice Mail Outage Campaign

Let's suppose that we now have planned maintenance on our voice mail system that will cause a global outage. We are ready to enable the campaign so all subscribers will hear it.

1. In the IOM Tool, on the top menu bar, click **Campaigns**.

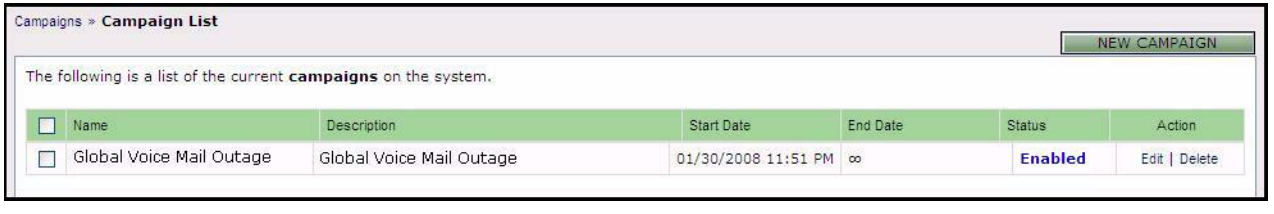
2. On the campaigns list, click **Edit** at the far right for the campaign.

3. On the **Edit Campaign** page, for the **Enabled** field, click the **Yes** radio button.



4. Click **UPDATE** to save the change.

The campaigns list now shows the campaign is **Enabled**.



Note: As an exercise for yourself, use the IOM Tool to enable the other campaign, Market-based Voice Mail Outage campaign.

9: Creating the New Promotions Campaign

Let's create the New Promotions Campaign. Instead of using the Campaign Wizard to work from the top down (as we did in "6: Using the Campaign Wizard" on page 47), we will work from the bottom up:

1. First we'll create the promotions.

2. Then we will add those promotions to offers.

3. Last we will add the offers to the campaign.

Recall that the New Promotions Campaign has three offers, each with its specific promotional message:

- A Loyalty Program offer for subscribers who have been with us more than three months and who have at least $5 in their accounts.
- A Text Messaging offer for subscribers who send 30 or more messages a month and who do not have our TextMsg200 product.
- An Accessories offer for all subscribers.

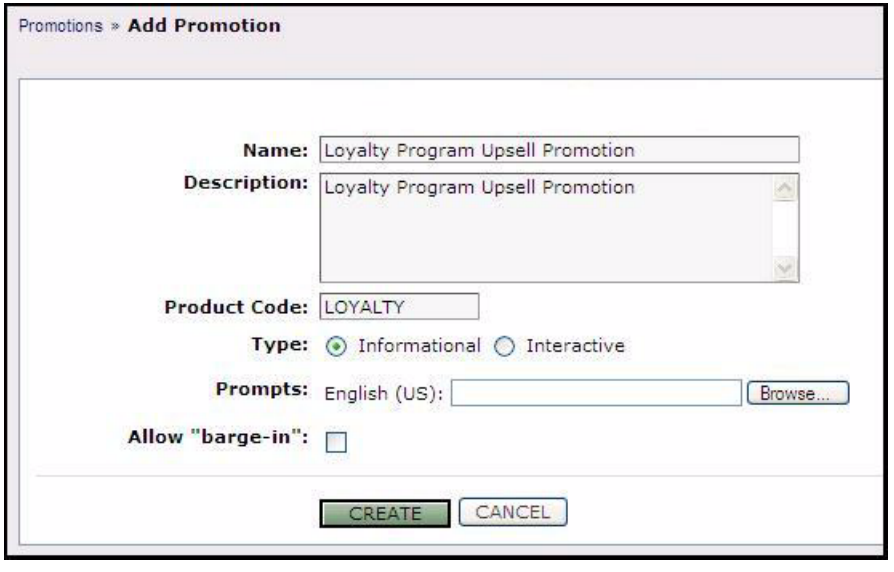
These are *interactive promotions*: the customer must act on them by either accepting or declining the offer, and our voice application has been written to process the customer's responses. Each interactive promotion has a set of associated files needed by the voice application, which we will upload to IOM: supporting audio files, grammar files, and destination files. See "Tutorials: Writing the Voice Application" on page 89 for tutorials about the voice application.

Creating the Promotions

1. In the IOM Tool, on the top menu bar, click **Promotions**.

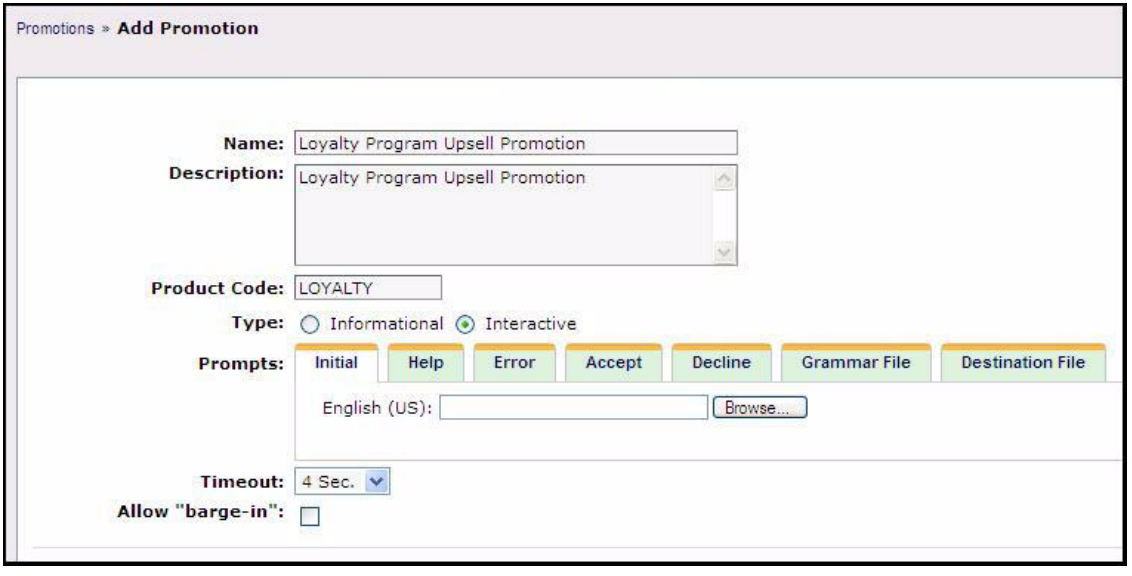
2. On the **Promotions** page, click **NEW PROMOTION**. 

1. Enter the name and description of the promotion.



2. For **Product Code**, enter LOYALTY.

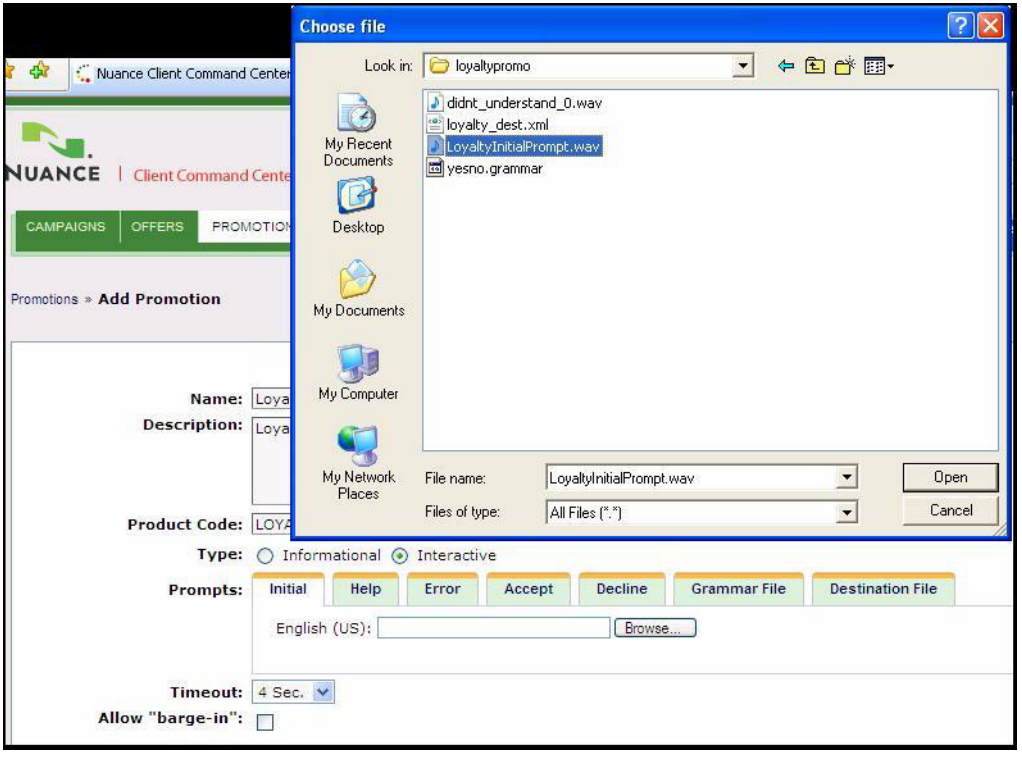
3. This is an interactive promotion, so for **Type**, click **Interactive**. Several new fields appear.



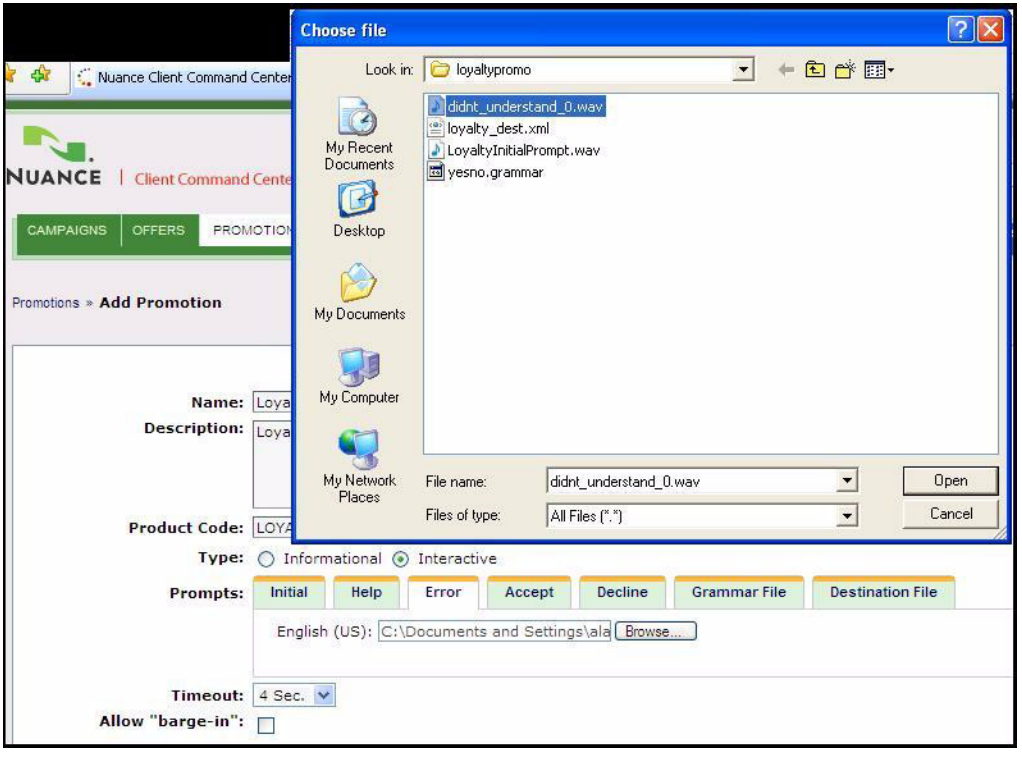
We have worked with our voice application developer in advance to determine the exact audio and other files we need to upload, as shown below. We have all these files on our local computer, and we will upload them all in succession.

Prompt/File Type	File Name
Initial	`LoyaltyInitialPrompt.wav`
Error	`didnt_understand.wav`
Grammar File	`yesno.grammar`
Destination File	`loyalty_dest.xml`

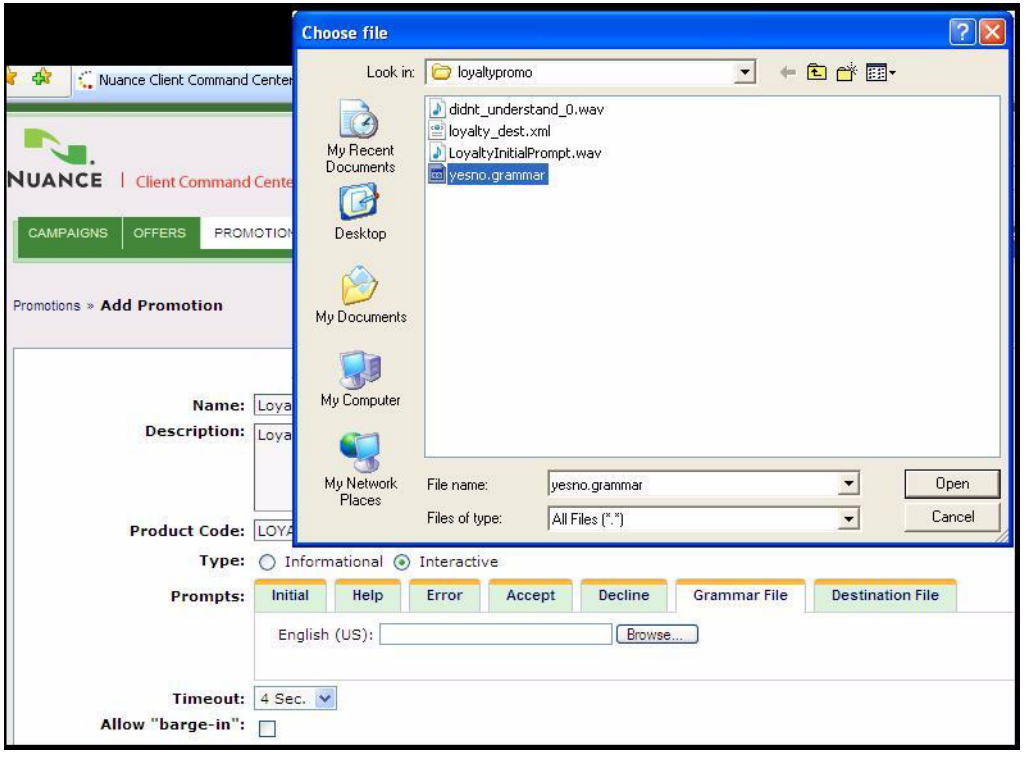
4. Click the first tab, **Initial**, to upload the Initial prompt. Navigate our local disk, and open the `LoyaltyInitialPrompt.wav` audio file.



5. Next, click the **Error** tab to upload the Error prompt, navigate our local disk, and open the `didnt_understand.wav` audio file.

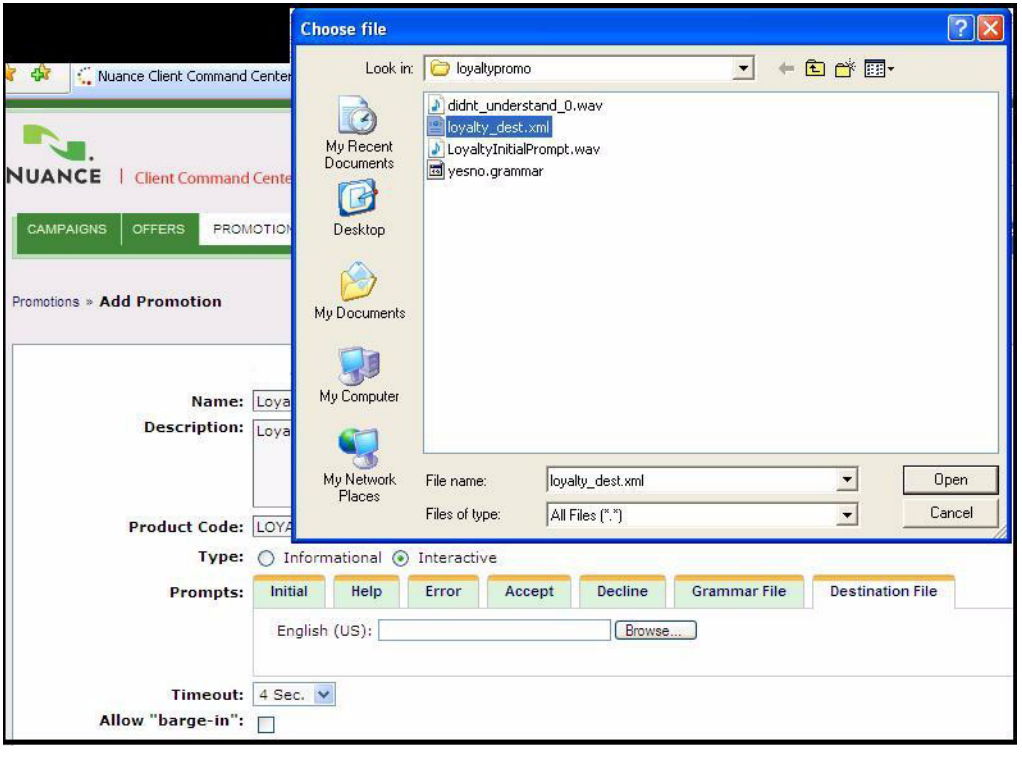


6. Next comes the grammar that will interpret the subscriber's response. Click the **Grammar File** tab to upload the grammar, navigate our local disk, and open the `yesno.grammar` file.



For information about destination files, see "Returning to a Named Destination" on page 102.

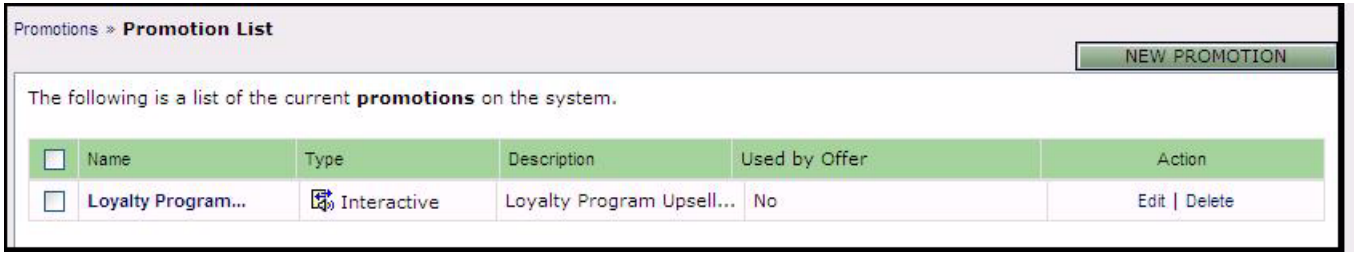
7. Finally comes the destination file, which defines the point in our voice application to which the subscriber will be transferred after accepting the offer. Click the **Destination File** tab to upload the file, navigate our local disk, and open the `loyalty_dest.xml` file.



See the chapter on promotions in *IOM Tool User Guide*.

8. Leave the **Timeout** and **Allow "barge-in"** fields to their defaults.
9. We are ready to create the Loyalty Program Upsell Promotion. Click **CREATE**.

We are returned to the Promotions list, where we see that the Loyalty Program Upsell Promotion has been created.



10. Now create the two other interactive promotions, using the information shown below.

Click **NEW PROMOTION**, enter the details, set the **Type** to **Interactive**, upload the files for the appropriate prompt, and click **CREATE** for each promotion.

Table 4 Prompts, Grammars, and Destination Files for Promotions

Promotion Name	Prompt/File Type	File Name
TextMsg200 Promotion Product Code: TEXTMSG200	Initial	`TextMsg200InitialPrompt.wav`
	Error	`didnt_understand.wav`
	Grammar File	`yesno.grammar`
	Destination File	`textmsg200_dest.xml`
Accessories Promotion Product Code: ACCESSORIES	Initial	`AccessoriesInitialPrompt.wav`
	Error	`didnt_understand.wav`
	Grammar File	`yesno.grammar`
	Destination File	`iom_accessories_dest.xml`

We have now created all three promotions, as shown on the Promotions List.

Now we'll add the promotions to offers.

Adding Existing Promotions to Offers

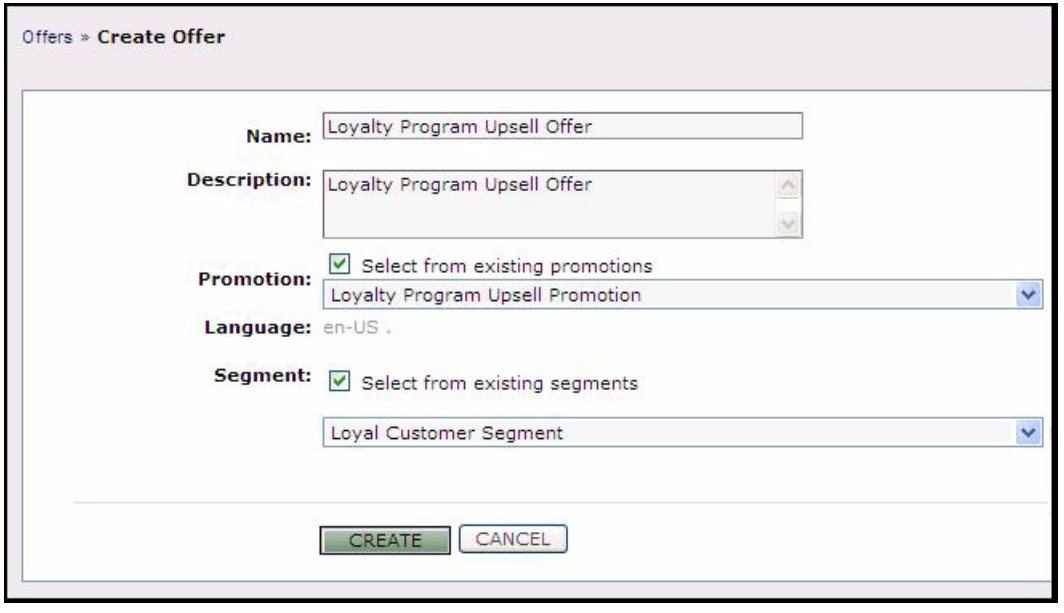
Now we will create three offers, selecting the promotions we created in the previous section, and selecting the appropriate market segment that we created in "5: Defining Segments with Variables and Rules" on page 41. Here are the details of the offers:

Offer Name	Promotion	Segment
Loyalty Program Upsell Offer	Loyalty Program Upsell Promotion	Loyal Customer Segment
TextMsg200 Offer	TextMsg200 Promotion	Heavy Text Message Users without TextMsg200
Accessories Promotion	Accessories Promotion	None. Will be offered to all subscribers.

1. In the IOM Tool, on the top menu bar, click **Offers**.

2. Click **NEW OFFER**. 

3. First, the Loyalty Program Upsell Offer. Enter the name and description.



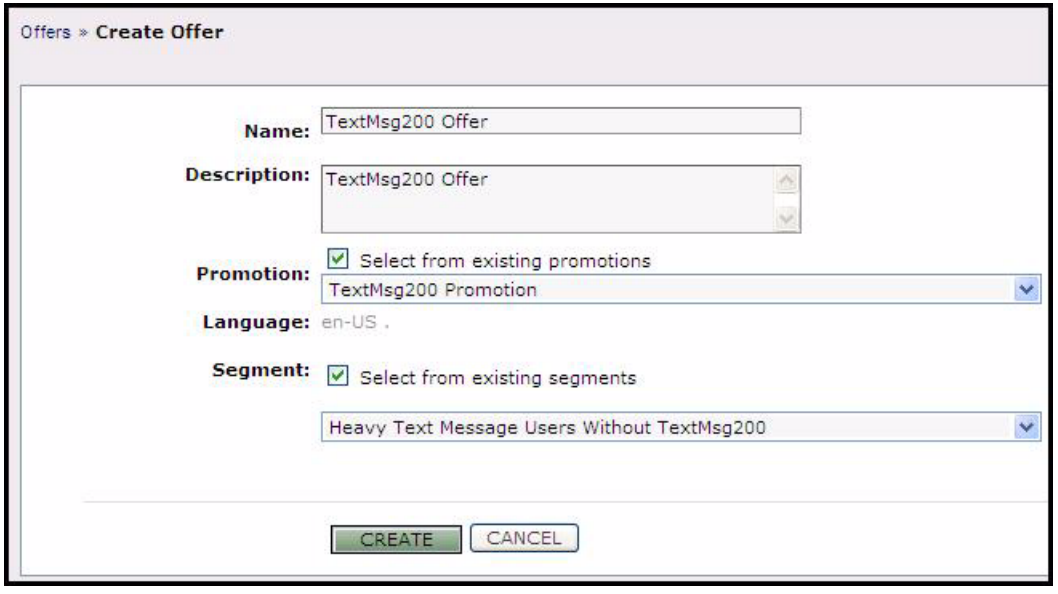
   a. For **Promotion**, select the **Loyalty Program Upsell Promotion**.

   b. For **Segment**, select the **Loyal Customer Segment**.

4. Click **CREATE**.

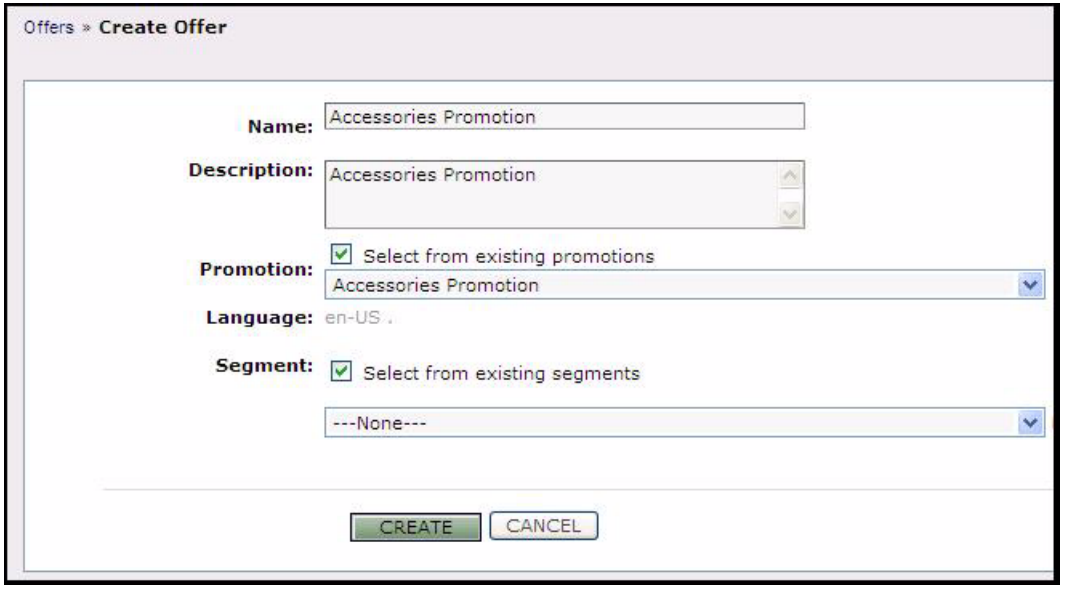
5. Click **NEW OFFER**. 

6. Next, the TextMsg200 Offer. Enter the name and description.



   a. For **Promotion**, select the **TextMsg200 Promotion**.
   b. For **Segment**, select the **Heavy Text Message Users Without TextMsg200**.
7. Click **CREATE**.
8. Click **NEW OFFER**. 

9. Finally, the Accessories Promotion. Enter the name and description.



   a. For **Promotion**, select the **Accessories Promotion**.

   b. For **Segment**, do not select anything. This offer will be made to all subscribers.

10. Click **CREATE**.

All of our offers are now created, as shown in the Offers List. Notice that none of the offers are used by any campaign yet.

Now we'll add them to a new campaign.

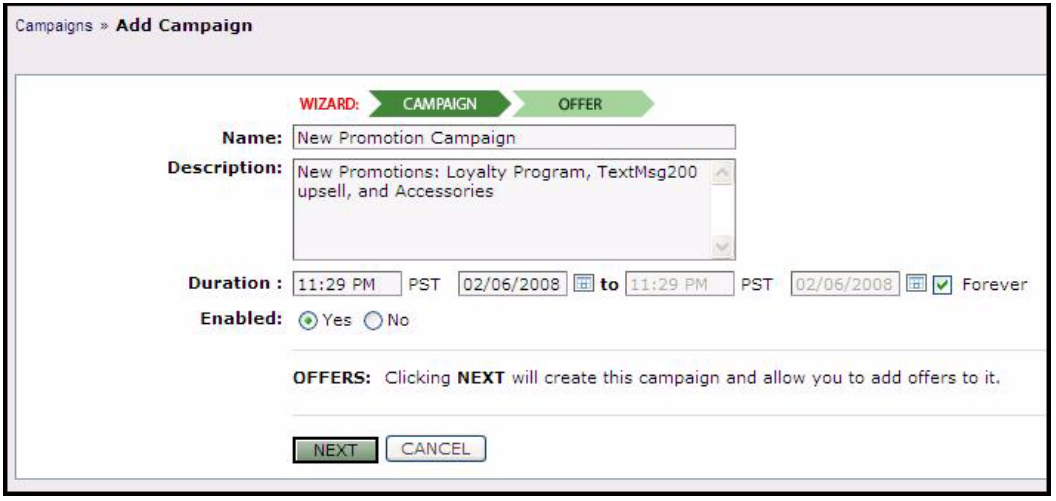
Adding Existing Offers to the Campaign

1. In the IOM Tool, on the top menu bar, click **Campaigns**.

2. Click **NEW CAMPAIGN**. 

   This starts the Campaign Wizard.

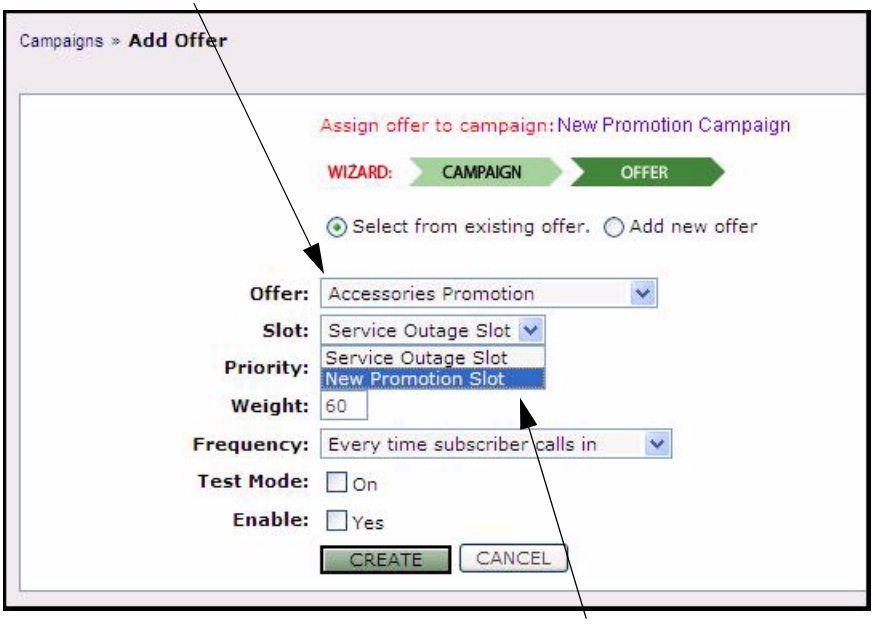
3. Enter the name and description of the campaign: New Promotion Campaign.



4. Because we will use this campaign again and again, it has no ending date. On the far right, click **Forever**.
5. For the **Enabled** field, leave it as the default **Yes**.
6. Click **NEXT** to move to the **Add Offer** page.

7. We will select from existing offers, which we created in the last section.

   The Accessories Promotion offer is already selected.



8. The offer should be associated with the **New Promotion Slot**.

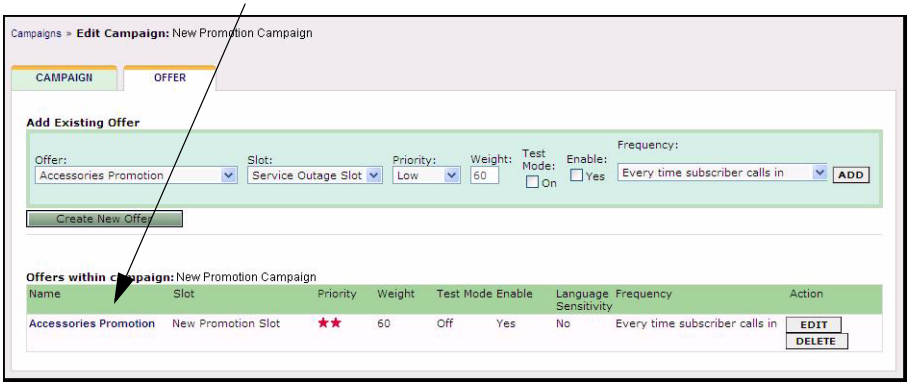
9. Set the **Priority** to **Medium**.

   Use the defaults for **Weight**, **Frequency**, and **Test Mode**.

   For **Enable** click **Yes**.

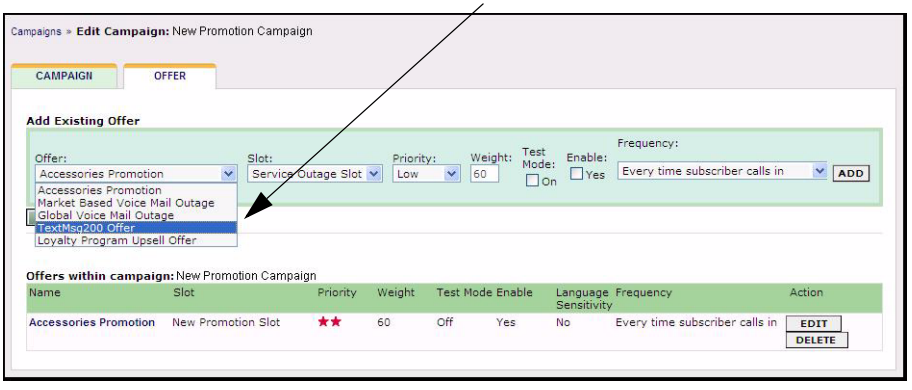
10. Click **CREATE**.

We are taken to the **OFFER** tab of the **Edit Campaign** page. The Accessories Promotion offer has been added to the campaign.

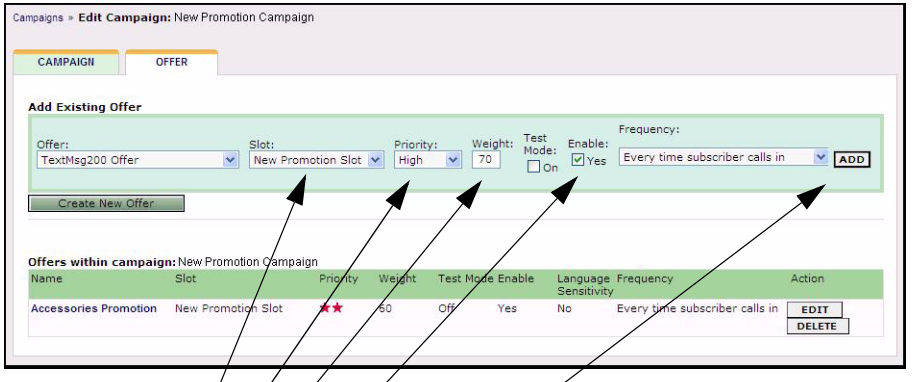


On the **OFFER** tab, we can add the other two offers to the campaign.

11. Under **Add Existing Offer**, from the **Offer** pulldown menu, select the TextMsg200 Offer:



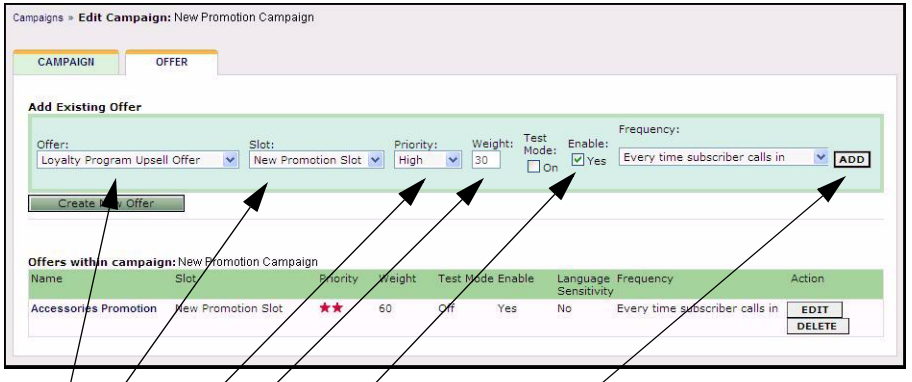
12. Now set the following fields:



a. The **Slot** is **New Promotion Slot**.
b. The **Priority** is **High**.
c. The **Weight** is **70**.
d. The offer is enabled.

13. On the far right, click **ADD**.

14. Now add the remaining offer.



a. The **Offer** is the **Loyalty Program Upsell Offer**.

b. The **Slot** is **New Promotion Slot**.

c. The **Priority** is **High**.

d. The **Weight** is **30**.

e. The offer is enabled.

15. On the far right, click **ADD**.

We're done. We have added all the offers to the campaign, which is now ready for use.

Chapter 5 Tutorials: Writing the Voice Application

These code-walkthrough tutorials highlight programming aspects of an IOM-based sample voice application. The source code and a testing environment for the sample application are available for your use and learning; contact your Nuance Account Representative for details about how to access it.

The purpose of the sample application is to demonstrate how the IOM HTTP Application Programming Interface (API) works with some example campaigns. IOM HTTP API calls and syntax are documented in the *IOM Developer and Datafeed Reference*. You should get a copy of this essential book.

The IOM campaigns that are the basis of the sample application are detailed in "Description of the Campaigns" on page 27. You should familiarize yourself with the purpose and general nature of the campaigns before starting to look at the code.

Note: The sample application is written in Voice XML (VXML). The tutorial code-walkthroughs assume that you have at least a rudimentary working knowledge of VXML. Although you might prefer to write voice applications in some other language, it is VXML that is actually interpreted by the Nuance platform.

Tutorial Objectives: Understand various pieces of code in the IOM sample voice application

Set-up: A description of our campaigns and all the campaign objects created with the IOM Tool and ready to use. See Chapter 4.

Description of Tutorials

We will walk through the code of the sample in these ways:

1: A Look at Main

2: Holding the Results from getPromotion.do

3: Testing the Promotion Type

4: Getting the Prompts with getPromotionData.do

- Playing the Prompt

- Calculating the Call Time

5: Reporting the Results of the Promotion

6: Working with Interactive Promotions

- Playing the Help or Error Prompt
- Exercising a Grammar File
- Playing the Accept or Decline Prompts
- Returning to a Named Destination
- Reporting a Purchase

7: Miscellaneous Useful Code

- Looping Through Destination Keys
- Error Catching

Programs in the Sample Application

The sample application consists of a set of program modules, each of which calls a particular IOM HTTP API for a specific purpose.

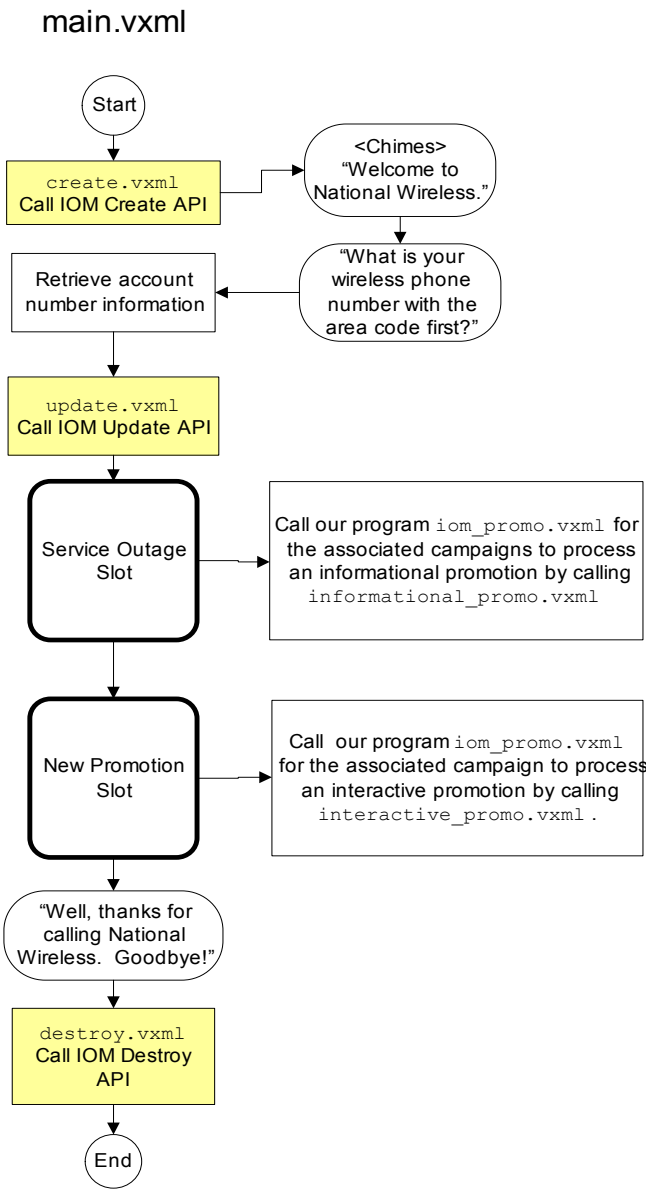
Table 5 Programs in the Sample Application

Program Name	Description and Related IOM HTTP API
`main.vxml`	The main program of the application.
`create.vxml`	Calls IOM `create.do`.
`update.vxml`	Calls IOM `update.do`.
`iom_promo.vxml`	Calls IOM `getPromotion.do`. This subdialog may call one of the following subdialogs:
■ `informational_promo.vxml`	Calls IOM `getPromotionData.do` to play either a bargeinable or non-bargeinable prompt.
■ `interactive_promo.vxml`	Calls IOM `getPromotionData.do` to retrieve the initial and other prompts, grammar, and destination file.
■ `iom_destination.vxml`	Fulfills the promotion.
`report_promotion.vxml`	Calls IOM `reportPromotion.do` to report promotion's acceptance.
`report_purchase.vxml`	Calls IOM `reportPurchase.do` to report a purchase resulting from the promotion.
`destroy.vxml`	Calls IOM `destroy.do`

1: A Look at Main

The `main.vxml` program has a straightforward structure.

Figure 4 A Look at main.vxml



1. An IOM session is initialized with `create.vxml`, which calls IOM's `create.do` API.
2. The caller is prompted for his telephone number, the associated account data is retrieved, and the IOM session is refreshed with `update.vxml`, which calls IOM's `update.do`.
3. `iom_promo.vxml` is called twice, once for each slot, to call IOM's `getPromotion.do` API, which returns the URLs to retrieve the prompts associated with the slots.

   We will if-test against the promotion type: interactive or informational. Depending on the type, `iom_promo.vxml` transfers control to either `informational_promo.vxml` or `interactive_promo.vxml`.
4. A "goodbye" message is played, and the IOM session is terminated with `destroy.vxml`.

Setting IOM_SERVICE_URL and Other Variables

The main program sets variables used in the calls to the IOM HTTP API, rather than having hardcoded values on each call:

```
<!-- Prefix on every call to the IOM API: host, port, and location
of APIs -->
<var name="IOM_SERVICE_URL"
      expr="'http://10.10.10.10:80/services/iomapi/'"/>
<var name="appName" expr="'NATLWIRELESS'"/>
<!-- desired response format from IOM -->
<var name="format" expr="'json'"/>
<var name="locale" expr="'en-US'"/>
```

Running the Slots

The voice application has two IOM slots, which we created with the IOM Tool (see "3: Creating Slots" on page 33).

The form `get_promotion` in `main.vxml` calls `iom_promo.vxml` for each of the slots. Each call to `iom_promo.vxml` is in a subdialog in `main.vxml`:

We must specify the *exact* name of the slot that was created with the IOM Tool, with spaces and capitalization (if any).

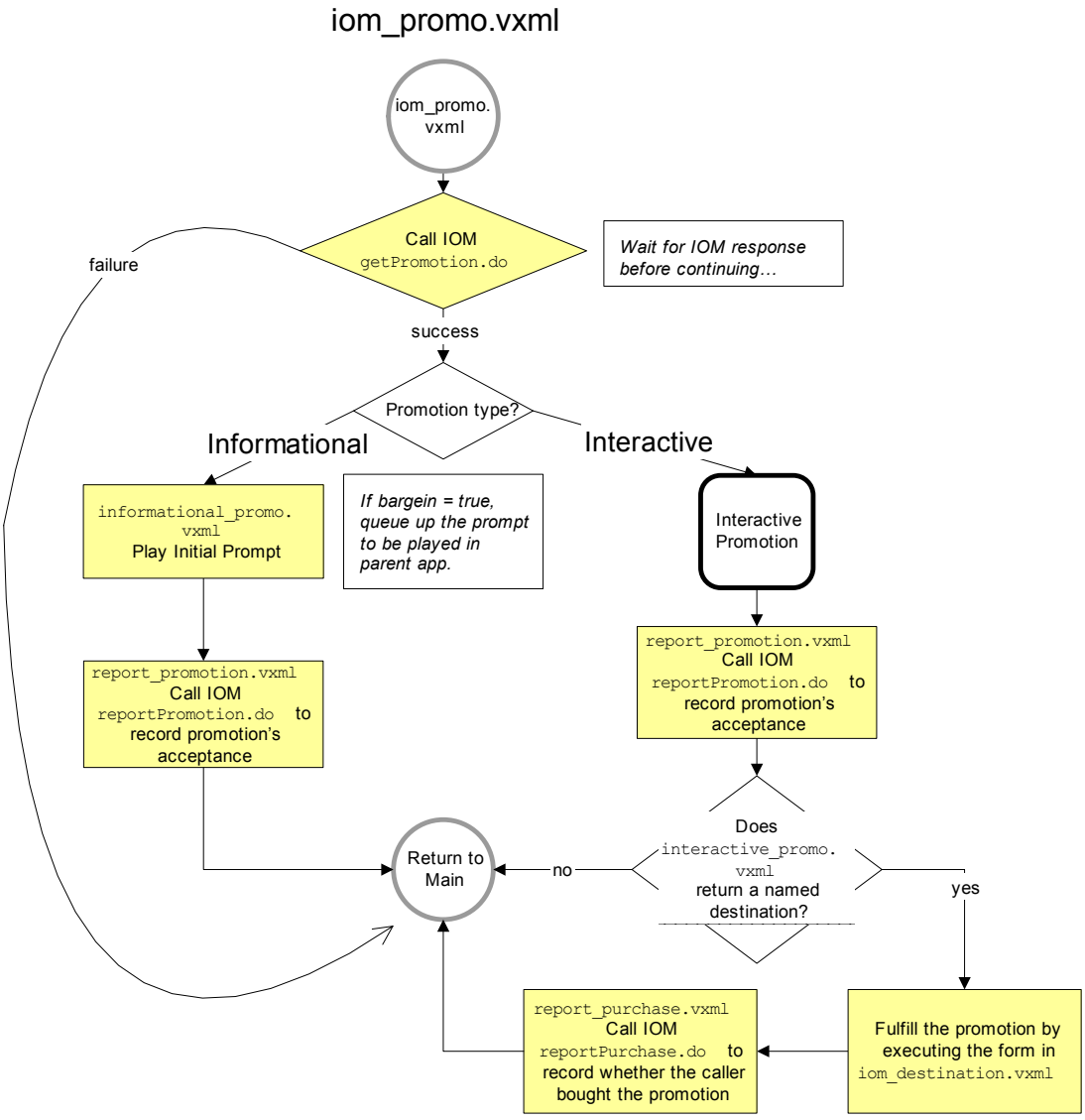
```
<!-- Play IOM Service Outage Slot (informational promotion) -->
      <block>
         <subdialog src="#get_promotion">
            <param name="isIomActive" expr="isIomActive"/>
            <param name="sessionId" expr="sessionId"/>
            <param name="slot" expr="'Service Outage Slot'"/>
         </subdialog>
      </block>

<!-- Play IOM New Promotion Slot (interactive promotion)-->
      <block>
         <subdialog src="#get_promotion">
            <param name="isIomActive" expr="isIomActive"/>
            <param name="sessionId" expr="sessionId"/>
            <param name="slot" expr="'New Promotion Slot'"/>
         </subdialog>
      </block>
```

We will look at how the `iom_promo.vxml` program works for each of these slots.

We'll start with the Service Outage Slot to play an informational promotion, which is shown on the left side of the flow diagram below.

Figure 5 iom_promo.vxml



We'll look at the interactive promotion processing in later tutorials.

2: Holding the Results from getPromotion.do

Let's see how `iom_promo.vxml` retrieves the information about a promotion.

There are two primary IOM HTTP APIs that deal with promotions: `getPromotion.do` and `getPromotionData.do`.

- `getPromotion.do` retrieves information *about the promotion itself*, such as its type (interactive or informational), its timeout value, its barge-in setting, and URLs to call the IOM HTTP API `getPromotionData.do` to retrieve all of its prompts.
- `getPromotionData.do` actually retrieves the promotion's prompts.

Note: It is important to retrieve all the prompts of a promotion as early as possible in your voice application. This allows them to be queued by the VXML interpreter. Queuing the prompts avoids delays, because they are ready to be played when the caller is ready to hear them.

The Code

First, `iom_promo.vxml` creates an object called `promotion` to hold the results returned from `getPromotion.do`:

```
<!--  to hold the IOM service response on getPromotion -->
<var name="promotion"/>
```

It sets a variable that will call `getPromotion.do`:

```
<var name="IOM_GET_PROMOTION_URL" expr="IOM_SERVICE_URL +
   'getPromotion.do'
```

It makes the call and assigns the results to the `promotion` object.

`slot` was set in the subdialog in `main.vxml`.

```
<data name="getData" srcexpr="IOM_GET_PROMOTION_URL"
   namelist="sessionId appName slot locale format"
   fetchtimeout="8s"/>
<assign name="promotion" expr="xml2json(getData)"/>
```

(The `<data>` tag in VXML is to fetch an XML file. IOM wraps the JavaScript Object Notation results in an XML file. To extract the JavaScript object, we run the `xml2json` JavaScript function.)

The `promotion` object now has properties that we can test against or evaluate to play prompts.

3: Testing the Promotion Type

Let's look at the code in `iom_promo.vxml` that tests which kind of promotion has been returned. At its highest level, the program has an if-test against the `promotion.interactive` property:

```
<if cond="promotion.interactive == 'true'">
  <subdialog name="result" src="interactive_promo.vxml">
    .
   . <!-- Process the interactive promotions -->
   .
  </subdialog>

<else/>
  <subdialog name="result" src="informational_promo.vxml">
   .
   . <!-- Process the informational promotions -->
   .
  </subdialog>
</if>
```

Notice that the returned results will be stored in an object called `result`.

If the type of the promotion is "interactive," `iom_promo.vxml` calls `interactive_promo.vxml`. Otherwise, `iom_promo.vxml` calls `informational_promo.vxml`, which we will look at next.

4: Getting the Prompts with getPromotionData.do

Let's look at the code in `informational_promo.vxml` that retrieves and plays a prompt from an informational promotion.

The `getPromotion.do` and `getPromotionData.do` API calls work together so you can easily get the prompts and auxiliary files that were uploaded with the IOM Tool.

In the response from `getPromotion.do`, many of the returned values are formatted as calls to `getPromotionData.do` that will retrieve the desired prompt.

For example, the initial prompt of a promotion is represented like so (in JSON format):

An example of a complete response is in the *IOM Developer and Data-feed Reference*.

```
.
.
.
"initialName":"promo_sweeps_info.wav",
"initialUrl":"getPromotionData.do?promotionId=3181&resour
ceType=initialPrompt&locale=en-US",
.
.
.
```

The Code

So in `informational_promo.vxml`, after the response from `getPromotion.do`, the value of the `promotion.initialPrompt` property looks like this:

```
getPromotionData.do?promotionId=3181&resourceType=initial
   Prompt&locale=en-US
```

Thus, to play the prompt, we simply have to evaluate the expression in `promotion.initialPrompt`.

Playing the Prompt

To actually play the initial prompt, all we have to do is pass our `promotion` object to the VXML `<audio>` tag:

```
<audio expr="IOM_SERVICE_URL + promotion.initialUrl"
  maxage="0" fetchtimeout="10s"/>
```

Calculating the Call Time

We've played the informational promotion to the caller, so we're almost done. (Remember that an informational promotion does not require any action from the caller. It simply plays a prompt.) All we need to do in `informational_promo.vxml` is calculate the length of the call and set some other variables we will use to record the success of the promotion.

To calculate the length of the call, `informational_promo.vxml` sets a variable called `duration`:

```
<block name="Return">
<assign name="duration" expr="session.bevocal.timeincall
      - startTime"/>
```

```
<return namelist="status desc duration"/>
</block>
```

We now return back to `iom_promo.vxml`.

5: Reporting the Results of the Promotion

All we have to do now is record that the promotion was played (the status), the description, and the length of the call with the IOM HTTP API `reportPromotion.do`.

The Code

The end of `iom_promo.vxml` calls the `report_promotion.vxml` program with the properties that were set on the `result` object when `informational_promo.vxml` was invoked:

```
<assign name="status" expr="result.status"/>
<assign name="desc" expr="result.desc"/>
<assign name="duration" expr="result.duration"/>

<!-- report acceptance -->
```

Control is transferred to `report_promotion.vxml`.

```
<if cond="promotionId != undefined &amp;&amp; promotionId != ''
 &amp;&amp; !reportingLogged">
   <assign name="reportingLogged" expr="true"/>
   <subdialog src="report_promotion.vxml">
      <param name="IOM_SERVICE_URL" expr="IOM_SERVICE_URL"/>
      <param name="sessionId" expr="sessionId"/>
      <param name="appName" expr="appName"/>
      <param name="duration" expr="duration"/>
      <param name="promotionId" expr="promotionId"/>
      <param name="isAccepted" expr="isAccepted"/>
   </subdialog>
</if>
```

In `report_promotion.vxml`, the passed-in parameters are simply passed on to the IOM HTTP API `reportPromotion.do`:

```
<var name="IOM_REPORT_PROMOTION_URL" expr="IOM_SERVICE_URL +
  'reportPromotion.do' "/>
<block>
   <data name="getData" srcexpr="IOM_REPORT_PROMOTION_URL"
     namelist="sessionId appName duration promotionId isAccepted
     format" fetchtimeout="5s"/>
   <var name="results" expr="xml2json(getData)"/>
```

```
</block>
```

6: Working with Interactive Promotions

Now we will look at the `interactive_promo.vxml` program to learn how to work with multiple prompts, with grammars, and with destination files.

Here's a recap of what has happened in our voice application so far and how we have arrived at `interactive_promo.vxml`.

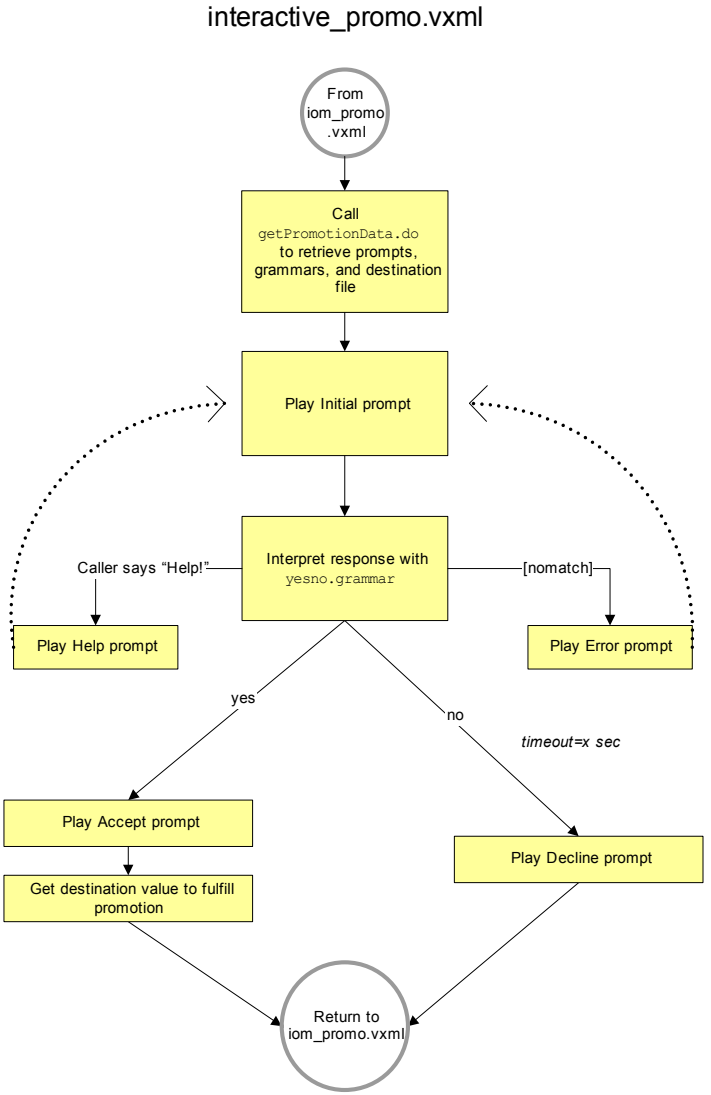
- In "1: A Look at Main" on page 91, `iom_promo.vxml` is invoked from `main.vxml` for the New Promotion Slot. The New Promotion Slot exercises the New Promotion Campaign, which has associated offers that contain interactive promotions, as detailed in "Description of the Campaigns" on page 27.
- In "2: Holding the Results from getPromotion.do" on page 94, we saw how to obtain a promotion's properties, such as its type (informational or interactive), its timeout value, its barge-in setting, and URLs to call the IOM HTTP API `getPromotionData.do` to retrieve all of its prompts.

You should read these sections before continuing.

- In "3: Testing the Promotion Type" on page 95, the `interactive_promo.vxml` program is invoked from `iom_promo.vxml` by an if-test on the type of the promotion returned by `getPromotion.do`.

The `interactive_promo.vxml` program has a straightforward structure.

Figure 6 interactive_promo.vxml



The program starts with function definitions to process destination files; this topic is discussed in "Looping Through Destination Keys" on page 104. However, we will first look at the simpler aspects of the program.

Getting All the Prompts

In "2: Holding the Results from getPromotion.do" on page 94, `iom_promo.vxml` runs the getPromotio.do API to retrieve the URLs associated with the promotion.

Thus, in `interactive_promo.vxml`, *we already have all the URLs we need* to retrieve the interactive promotion's prompts so we can play them. These URLs are stored as properties of the `promotion` object:

- `initialUrl`: the first prompt to play
- `helpUrl`: the prompt if the caller says "Help"
- `errorUrl`: the prompt if there is an error, such as a time out on the caller's response
- `acceptUrl`: the prompt if the caller accepts the offer
- `declineUrl`: the prompt if the caller declines

We also have the URLs to get the promotion's grammar and destination file:

- `grammarUrl`: the grammar to interpret the caller's response to the initial prompt
- `destinationUrl`: The URL to retrieve an uploaded destination file.

Playing the Help or Error Prompt

In "4: Getting the Prompts with getPromotionData.do" on page 95, we've already seen how easy it is to play prompts. We simply pass the value of the appropriate URL we obtained from `getPromotion.do` to the `<audio>` tag:

Help Prompt

```
<help>
   <if cond="promotion.helpUrl != undefined &amp;&amp;
      promotion.helpUrl != ''">
      <audio expr="IOM_SERVICE_URL + promotion.helpUrl"
             fetchtimeout="8s"/>
   </if>
</help>
```

Error Prompt

```
<nomatch>
    <if cond="promotion.errorUrl != undefined &amp;&amp;
        promotion.errorUrl != ''">
        <audio expr="IOM_SERVICE_URL + promotion.errorUrl"
               fetchtimeout="8s"/>
    <else/>
        <assign name="status" expr="'optout'"/>
        <assign name="desc" expr="'dialog_error'"/>
    </if>
</nomatch>
```

Exercising a Grammar File

We use the `<field>` tag to associate our grammar with the initial prompt.

```
<field name="destination_key">
.
.
.
    <grammar srcexpr="IOM_SERVICE_URL + promotion.grammarUrl"
        fetchtimeout="8s"/>
    <audio expr="IOM_SERVICE_URL + promotion.initialUrl"
        fetchtimeout="8s"/>
.
.
.
</field>
```

After the initial prompt is played, the caller's response is evaluated according to the grammar, and the program simply has to play either the accept or decline prompt.

Playing the Accept or Decline Prompts

If the results of the grammar evaluation of the caller's response is "accept," the program passes the value of `acceptUrl` to the `<audio>` tag. Otherwise, it passes the value of the `declineUrl`.

```
<if cond="'accept'.equalsIgnoreCase(destination_key)">
    <assign name="isAccepted" expr="true"/>
    <if cond="promotion.acceptUrl != ''">
        <audio expr="IOM_SERVICE_URL + promotion.acceptUrl"
            fetchtimeout="8s"/>
    </if>
    <elseif cond="'decline'.equalsIgnoreCase \
```

```
            (destination_key) &amp;&amp; promotion.declineUrl !=
            ''"/>
        <audio expr="IOM_SERVICE_URL + promotion.declineUrl"
            fetchtimeout="8s"/>
</if>
```

Returning to a Named Destination

In the sample application, named destinations are used to fulfill the promotion if the caller accepts.

Read this section for an understanding of what a destination file is.

The `interactive_promo.vxml` program begins by defining support functions to extract the keys and values of named destinations from a destination file associated with a promotion; these functions are discussed in "Looping Through Destination Keys" on page 104. A destination value is usually the name of a `<form>` or `<subdialog>`.

First, interactive_promo.vxml tests the value of the `promotion.destinationUrl` property to determine if there is a destination file associated with this promotion:

```
<block>
   <if cond="promotion.destinationUrl != ''">
      <data name="destmappings" srcexpr="IOM_SERVICE_URL +
         promotion.destinationUrl" fetchtimeout="10s"/>
      <assign name="destinationXML" expr="destmappings"/>
   </if>
</block>
```

In the primary `<field>` portion of `interactive_promo.vxml`, after either the accept or the decline prompt is played, the program then determines the value of a named destination. It loops through all of the destination keys and stores the values to pass to `iom_promo.vxml` when it is finished.

Back in `iom_promo.vxml`, if a destination value is not null, `iom_promo.vxml` passes control to the `iom_destination.vxml` program, along with the destination value, which is the name of a `<form>` in `iom_destination.vxml`:

```
<if cond="destinationValue != undefined">
   <subdialog name="iom_destination"
      srcexpr="'iom_destination.vxml#' + destinationValue">
      <param name="productCode" expr="productCode"/>
      <param name="price" expr="undefined"/>
      <param name="isPurchased" expr="false"/>
   </subdialog>
```

```
.
.
.
```

For example, one of the destination values in the sample application is `Accessories_IOM_Slot1`. When this value is passed to `iom_destination.vxml`, the program goes to the `<form>` with that same name:

```
<form id="Acessories_IOM_Slot1">
   <var name="productCode" expr="undefined"/>
   <var name="price" expr="undefined"/>
   <var name="isPurchased" expr="false"/>

   <block>
      <!-- fulfillment menu -->

      <!-- if the fulfillment is successful return the price,
         productCode and isPurchased = true -->
      <assign name="price" expr="'5.99'"/>
      <assign name="isPurchased" expr="true"/>
      <return namelist="productCode price isPurchased"/>
   </block>
</form>
```

The form returns the details about the purchase back to `iom_promo.vxml`, which records the playing of the promotion; see "5: Reporting the Results of the Promotion" on page 97.

`iom_promo.vxml` then reports the purchase of the promotion.

Reporting a Purchase

In `iom_promo.vxml`, if the caller has accepted the offer and made a purchase, control is passed to the `report_purchase.vxml` program (along with the variables that hold the details of the purchase, such as its price):

```
<subdialog name="iom_report_purchase"
      src="report_purchase.vxml">
.
.
.
</subdialog>
```

`report_purchase.vxml` simply invokes the `reportPurchase.do` API to record the details:

```
<var name="IOM_REPORT_PROMOTION_URL" expr="IOM_SERVICE_URL +
       'reportPurchase.do' "/>
    <block>
       <data name="getData" srcexpr="IOM_REPORT_PROMOTION_URL"
          namelist="sessionId appName promotionId productCode
          price isPurchased format" fetchtimeout="5s"/>
       <var name="results" expr="xml2json(getData)"/>
.
.
.
```

7: Miscellaneous Useful Code

In this section we look at some useful coding techniques that can be helpful in your voice application.

Looping Through Destination Keys

The `interactive_promo.vxml` program includes two useful functions for extracting the keys and values from a destination file associated with an interactive promotion.

Example of Destination File

For each promotion, the destination file is composed of key/value pairs of grammar response values and the corresponding forms or subdialogs to which IOM should transfer after a caller's responses have been interpreted. A destination map can have multiple items, for example:

```
<?xml version="1.0"?>
<config>
   <item key="accept" value="Accessories_IOM_Slot1" />
   <item key="2" value="Loyalty_IOM_Slot1" />
   <item key="3" value="textmsg200_IOM_Slot1" />
</config>
```

Note: In the sample application, each interactive promotion has its own associated destination file with only a single key/value pair, rather than a single destination file with multiple keys.

The Loop Functions

We need to loop through all the items in a destination file and store the values so our voice application can act on the values. The following JavaScript functions, `doesDestinationExist` and `getDestinationValue`, do this.

```
<script>
        <![CDATA[
            function doesDestinationExist(destinationXML, destinationKey) {
                if(destinationXML.nodeType == 9) { // 9 = DOCUMENT_NODE
                    var itemElement = destinationXML.getElementsByTagName("item");
                    for(var i=0; i < itemElement.length; i++) {
                        var key = itemElement.item(i).getAttribute("key");
                        if(key == destinationKey) {
                            return true;
                        }
                    }
                    return false;
                }
            }

            function getDestinationValue(destinationXML, destinationKey) {
                if(destinationXML.nodeType == 9) {
                    var itemElement = destinationXML.getElementsByTagName("item");
                    for(var i=0; i < itemElement.length; i++) {
                        var key = itemElement.item(i).getAttribute("key");
                        if(key == destinationKey) {
                            if(itemElement.item(i).hasAttribute("value")) {
                                return itemElement.item(i).getAttribute("value");
                            } else {
                                return "";
                            }
                        }
                    }
                }
                return "";
            }
        ]]>
    </script>
```

Error Catching

A successful response from IOM is indicated by the status code 200. The code is always in the `status` key, as in the following example response:

Any status code other than 200 indicates a failure.

```
<?xml version="1.0"?>
<result>
   <item key="statusCode" value="200"/>
    .
    .
    .
</result>
```

If you encounter a failure, you should set a flag that will prevent the running of any other IOM HTTP APIs and so you can gracefully exit your voice application.

The sample application uses the `isIomActive` flag. For example, in `create.vxml`, if the `create.do` API fails, `isIomActive` is set to `false`:

```
<block>
   <if cond="isIomActive == true">
      <!-- call create.do -->
      <data name="getData" srcexpr="IOM_CREATE_URL"
         namelist="sessionId appName format account did"
         fetchtimeout="5s"/>
      <var name="result" expr="xml2json(getData)"/>
      <if cond="result != undefined &amp;amp;
         result.statusCode != '200'">
         <assign name="isIomActive" expr="false"/>
      </if>
   </if>
   <return namelist="isIomActive"/>
</block>
```

Then, after the return to main, when we run `destroy.vxml` to end the IOM session, we catch the error and avoid executing the `destroy.do` API, because at the top of `destroy.vxml` we have a `catch` block:

```
<catch>
   <var name="isIomActive" expr="false"/>
   <return namelist="isIomActive"/>
</catch>
```

Appendix A IOM Change Management and Forms

The IOM Change Request (ICR) accommodates all types of requests for the system. A change request can apply to any part of the IOM delivery system, whether it is a simple update to an informational promotion or a more complex addition of a new datafeed or segment variable.

Specific IOM changes are designed to have an almost immediate turnaround. For example, in the case of an outage the customer can simply fax an ICR form to the Nuance Network Operations Center (NOC) to change to an outage message. The Nuance NOC then enables an outage campaign that would only affect those users in the outage area when and if an outage happens.

Other IOM changes require a short recording and QA cycle, such as when defining a new promotion with new prompts that utilize the segmentation variables and data already defined in the system. The steps to create the campaign are relatively quick—the IOM tool facilitates this process. However, as the new promotion may affect the user experience of the IVR, Nuance will always institute a review of the promotion within the IVR before enabling it for its clients

The ICR also covers changes such as gathering new data from existing feeds or implementing a new datafeed from a different source. However, these types of changes require a small amount of development, testing, and deployment time to ensure that the changes work as expected and do not affect the existing behavior of the client voice application.

NUANCE	<Client> Engineering Change Request (ECR) IOM Campaign	<Client logo>

Document Version: 1.3 Created: Modified: Submitted:	This ECR is made pursuant to the Amended and Restated BeVocal Cingular Voice Application Services Agreement dated November 15, 2002 (as amended).
Engineering Change requested by : *Client Representative* *Tel.:* *Email:*	**Responsible Person** : *CSG Representative* *Tel.:* *Email:* *Fax:*
Short Description of Change Request:	
Application concerned (name) :	Backend system(s) concerned (name) :

Documents Affected			Flow diagram affected	
Name	Ver.	Path	Name	Path
Prompt List				

Changes Requested (Standard)	Changes Requested (Complex)
☐ Adding/Editing a campaign (section 1.0)	☐ Adding/Editing a variable
☐ Adding/Editing an offer (section 2.0)	☐ Adding/Editing a destination
☐ Adding/Editing a promotion (section 3.0)	☐ Adding/Editing a slot
☐ Adding/Editing a segment (section 4.0)	☐ Changing an existing data feed process
	☐ Adding a data feed
	☐ Creating a segment with multiple cases of a variable
	☐ Creating a segment with dependent variables
	☐ Adding a currently unsupported language

1.0 Campaign Change Description

(Add additional pages as needed for additional campaign changes)

1.

Campaign Information		
☐ Add ☐ Change ☐ Remove	Campaign Name:	
	Status:	☐ Enable ☐ Disable
	Brief Description:	
	Campaign Start Date:	
	Campaign End Date:	☐ Permanently
Campaign Offer Information		
☐ Add ☐ Change ☐ Remove	Offer Name:	
	Slot (Must be a defined slot):	
☐ Add ☐ Change ☐ Remove	Offer Name:	
	Slot (Must be a defined slot):	
☐ Add ☐ Change ☐ Remove	Offer Name:	
	Slot (Must be a defined slot):	

2.0 Offer Change Description

(Add additional pages as needed)

1.

Offer Information		
☐ Add ☐ Change ☐ Remove	Offer Name:	
	Brief Description of Offer:	
	Segment to Use: *(Must be defined segment)*	
	Promotion to Use: *(Must be defined promotion)*	
	Priority:	☐ High ☐ Medium ☐ Low
	Weight (from 1 to 100):	

3.0 Promotion Change Description

(add additional pages as needed)

1.

Promotion Information		
☐ Add ☐ Change ☐ Remove	☐ Informational ☐ Interactive	
	Promotion Name:	
	Brief Description of Product:	
	Promotional Prompt Wording: *(must end in question that requires Yes/No answer)* Initial prompt: For Interactive promos: Accept prompt: Reject prompt: Error prompt: Help prompt: Grammar file: Destination map:	
	How many times to play promotion per user?	☐ Every time subscriber calls in ☐ Only first time subscriber calls in ☐ More than once over campaign duration: ☐ Once a day ☐ Once a week ☐ Once a month

4.0 Segment Change Description

(Add additional pages as needed)

Example

Subscriber Information	
Subscriber Base to Target *(describe in terms of currently supported variables)*	*Subscribers in the Dallas market with an account balance between $0 and $20 and within 20 days of their 4th month of tenure.* Bold terms indicate supported segmentation variables.

1.

Subscriber Information	
Subscriber Base to Target *(describe in terms of currently supported variables)*	

5.0 Additional Changes

For any changes other than those that can be described in the above tables. Please give a detailed description as to the change including the type of change (i.e. variable, destination, etc..), where the change may occur in the IVR, the corresponding name and what the purpose of the change may be. For interactive promotions please provide a list of files to use for the initial prompt error prompt, grammar file, and destination map including language variations.

Will these changes affect BeVocal client billing? ☐ Yes ☐ No

If so usage of this feature must be recorded as part of the BeVocal Service Usage Report

6.0 Fees & Payment Terms

- Total Fees for the work contained in this ECR is fixed fee of $XXXX.
- Payment of of Fees will be due upon execution of this ECR.

Signatures:

BeVocal, Inc	**Client**
By:_____	By:_____
Printed Name:_____	Printed Name: _____
Title: _____	Title: _____
Date:_____	Date:_____

Index

M

O

P

R

S

T

U

V

W

X