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.

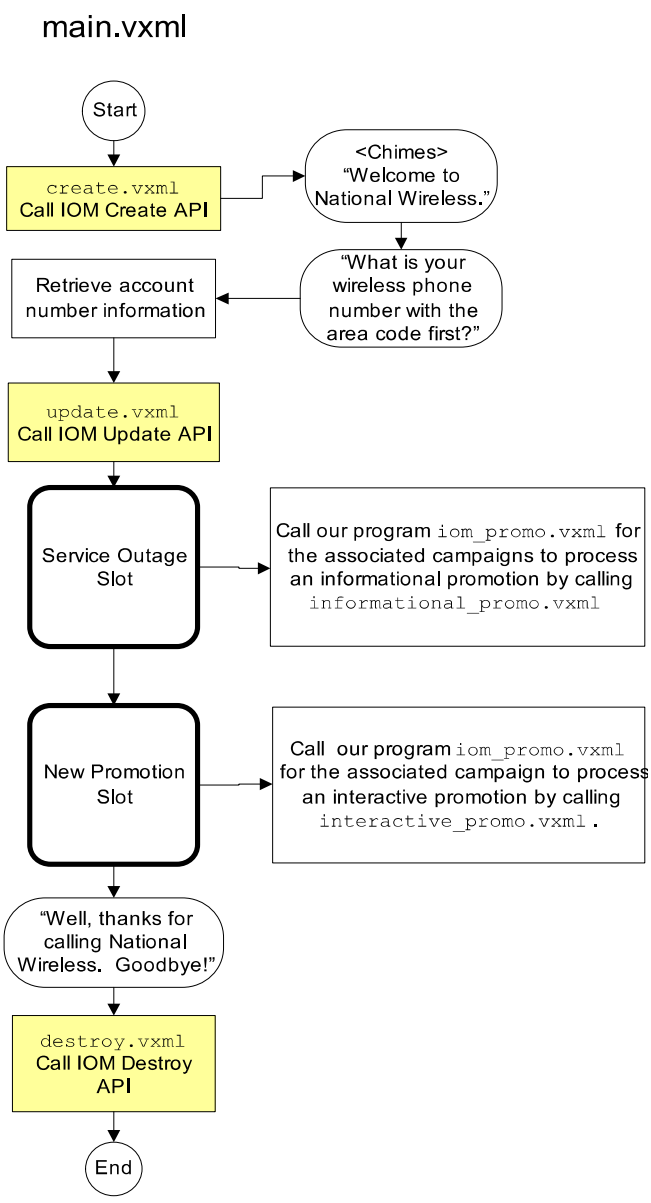
Table 5 Programs in the Sample Application

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).

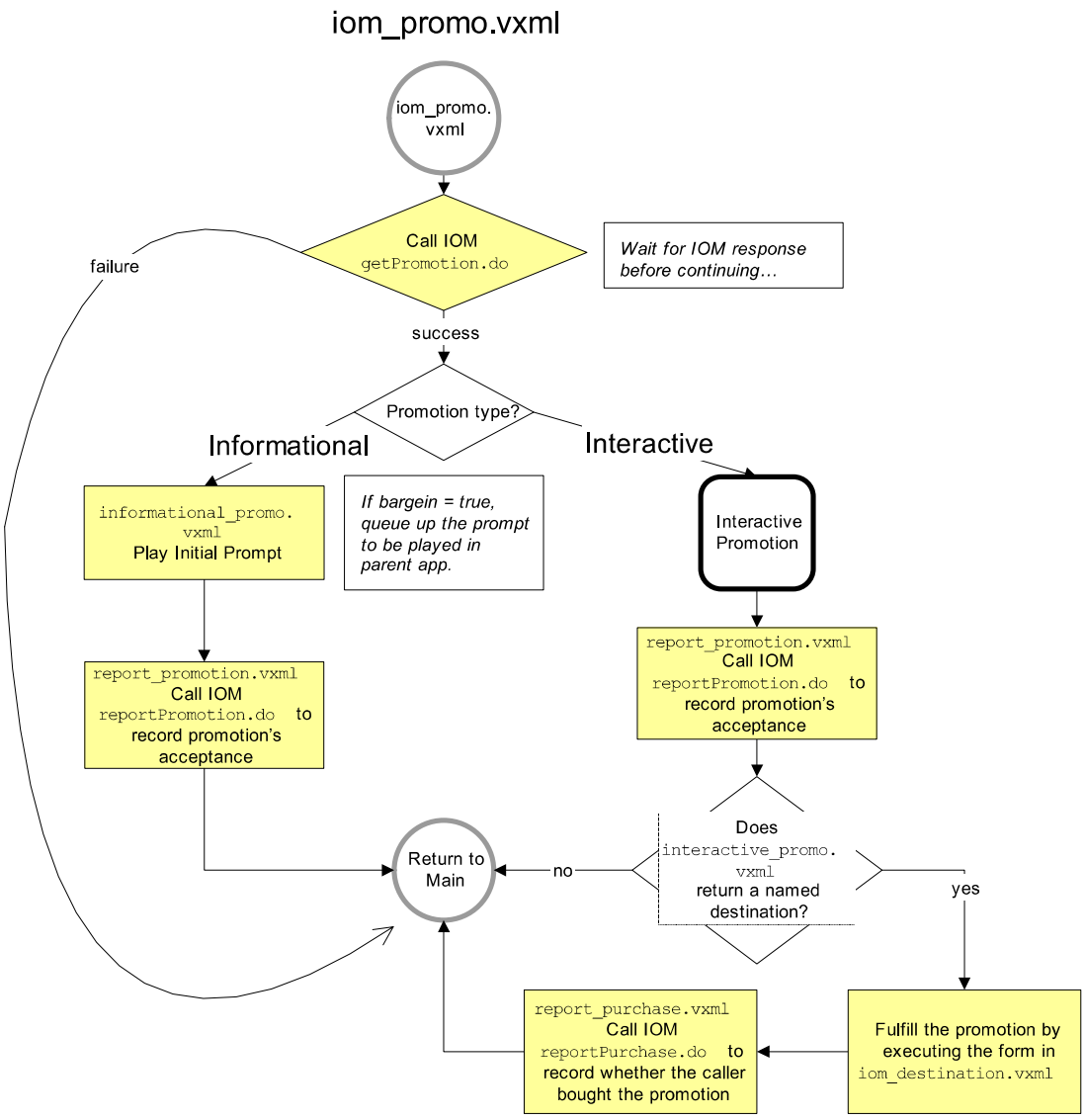
```
<!-- Play IOM Service Outage Slot (informational promotion) -->
      <block>
         <subdialog src="#get_promotion">
            <param name="isIomActive" expr="isIomActive"/>
            <param name="sessionId" expr="sessionId"/>
            <param name="slot" expr="'Service Outage Slot'"/>
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

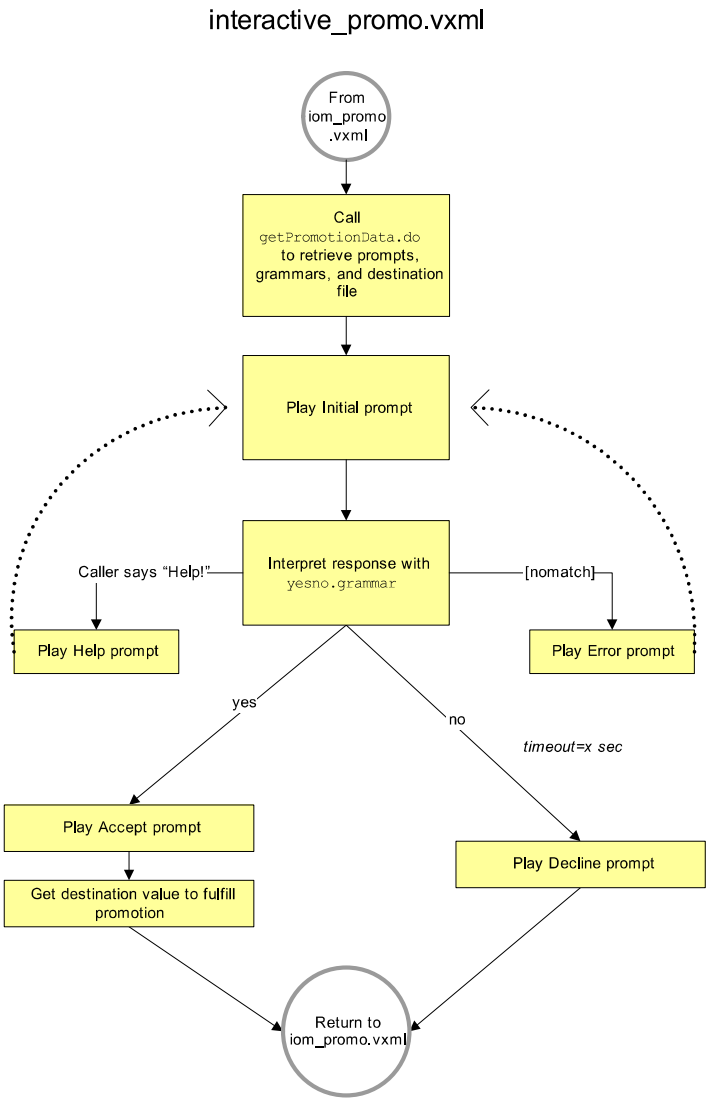
- In "1: A Look at Main" on page 91, `iom_promo.vxml` is invoked from `main.vxml` for the New Promotion Slot. The New Promotion Slot exercises the New Promotion Campaign, which has associated offers that contain interactive promotions, as detailed in "Description of the Campaigns" on page 27.
- In "2: 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;&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>
```