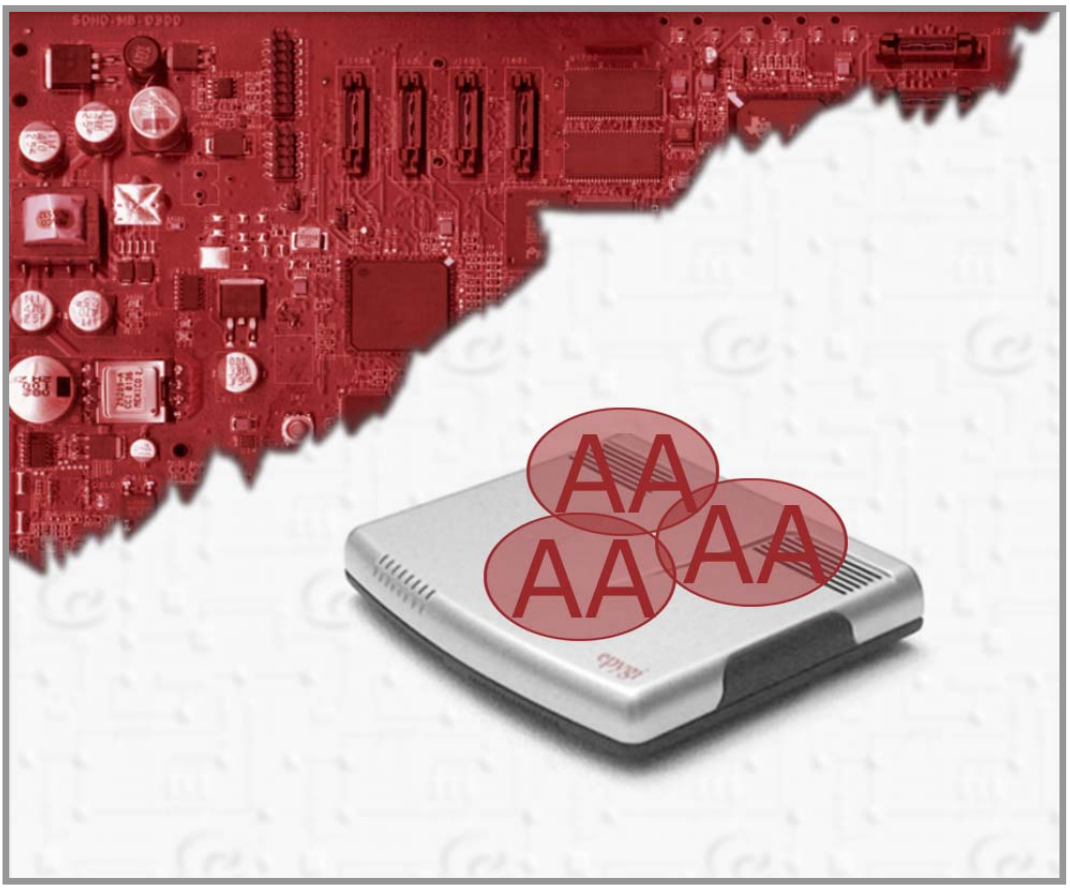
How to Set Up a Simple IVR on Quadro IP PBXs with VXML



Abstract: **This instructional document describes the steps necessary to set up a simple IVR on Quadro2x/4x/16x models. It also provides a list of supported VoiceXML elements and objects. It guides the reader through several examples with increasing complexity.**

Table of Contents:

1 Introduction

The Quadro IP PBX family allows you to create custom Interactive Voice Response (IVR) scenarios for use in its auto attendant(s). On the Quadro GUI they are called “Attendant scenarios”. They can be used to let a caller self-dial through a menu of voice messages with a DTMF phone to reach the correct person in a company. On Quadros this is done with the help of VoiceXML 2.0 (VXML).

1.1 Target audience

This document is targeted at Quadro administrators and consultant companies deploying Quadros for customers. The following information about the Quadro IP PBX is required prior to working with custom attendant scenarios.

General usage of the Quadro GUI and administrator menus	3
Managing PBX extensions and lines	4
Managing call routing	1

(1 – basic, 2 – limited, 3 – average, 4 – Good, 5 - Excellent)

1.2 Accompanying material

This instructional manual comes bundled with several example files. If you received this document without any example files, please visit the Download page of our technical support section at our website to receive the complete package. Each example in this instructional manual will list the example files needed to follow it.

1.3 Limitations

The Quadro custom attendant scenarios are limited in their functionality compared to the full power of VXML:

- The Quadro IVR is not capable of voice recognition. Only DTMF detection is available.
- The Quadro IVR is not capable of voice synthesis. Only prerecorded WAV files can be played back.
- Only a subset of VXML elements/attributes can be used in the Quadro IVR.
  (See **3 Appendix: Supported VoiceXML Elements**.)
- Only predefined VXML objects can be used in the Quadro IVR.
  (See **4 Appendix: Supported VoiceXML Objects**.)

Please note: Except for several elements/objects, the lists presented in this document are valid for software 3.0.x or higher releases. The elements/objects marked with “*” can be used starting with software 3.1.x (see appendices).

2 Getting it done

This instructional manual guides the reader through examples built on top of each other with increasing complexity. The first easy example described is a scenario requested by many of our customers. To begin with, some useful information for all examples is given.

2.1 Common information for all examples

This chapter shows you how to configure an Auto Attendant to IVR scenarios, where to upload the VMXL files and how to handle the prerecorded voice files.

Please note that only administrators can do all configuration changes regarding Auto Attendants and IVR. Normal extension users don't have the necessary rights to do so. So please log in as "admin" on your Quadro to follow the examples.

All examples in this instructional manual will be done with the Auto Attendant "10". You can also do the same with every other valid AA extension (00, 10-79).

2.1.1 Preparing an Auto Attendant

Every Auto Attendant (even the non-deleteable 00) on a Quadro can either use the "Default scenario" or a custom IVR scenario. When creating a new AA, the default behavior is the default scenario. To use an IVR scenario, the AA needs to be reconfigured. To do this, go to the menu "Users"-> "Extensions Management" and click on the Auto Attendant's number you want to change. This brings you to that AA's configuration page (see Figure 1). On this page, click the "Attendant Scenario" tab.

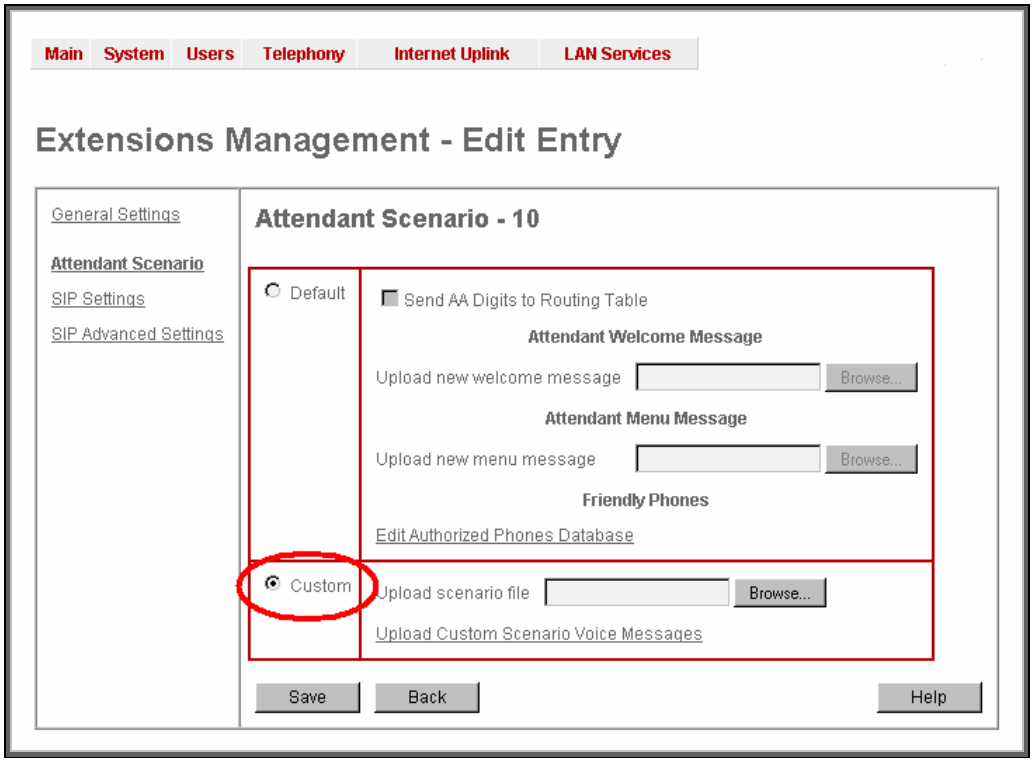


Figure 1 - Enabling IVR

Choose the radio button "Custom" and click "Save" at the bottom of the page. This AA is now using an IVR scenario. If you dial this AA now, you won't hear anything, as we haven't uploaded any Scenario File or Voice Messages.

2.1.2 Uploading a scenario file

To upload the scenario file, click on “Browse” and choose the VXML file you want for this Attendant Scenario, and then click “Save”. This step can be done at the same time when enabling the “Custom” radio button in the above paragraph.

After you uploaded a scenario file, you will see that two additional links have been added to the Attendant Scenario section (see Figure 2): “View/Download scenario” and “Remove scenario”. To view what the currently stored VXML file looks like, click on the “View Scenario” link. To delete, a no longer needed file, click the “Remove” link. If you want to update the existing VXML, simply upload a new one – the current one will be overwritten.

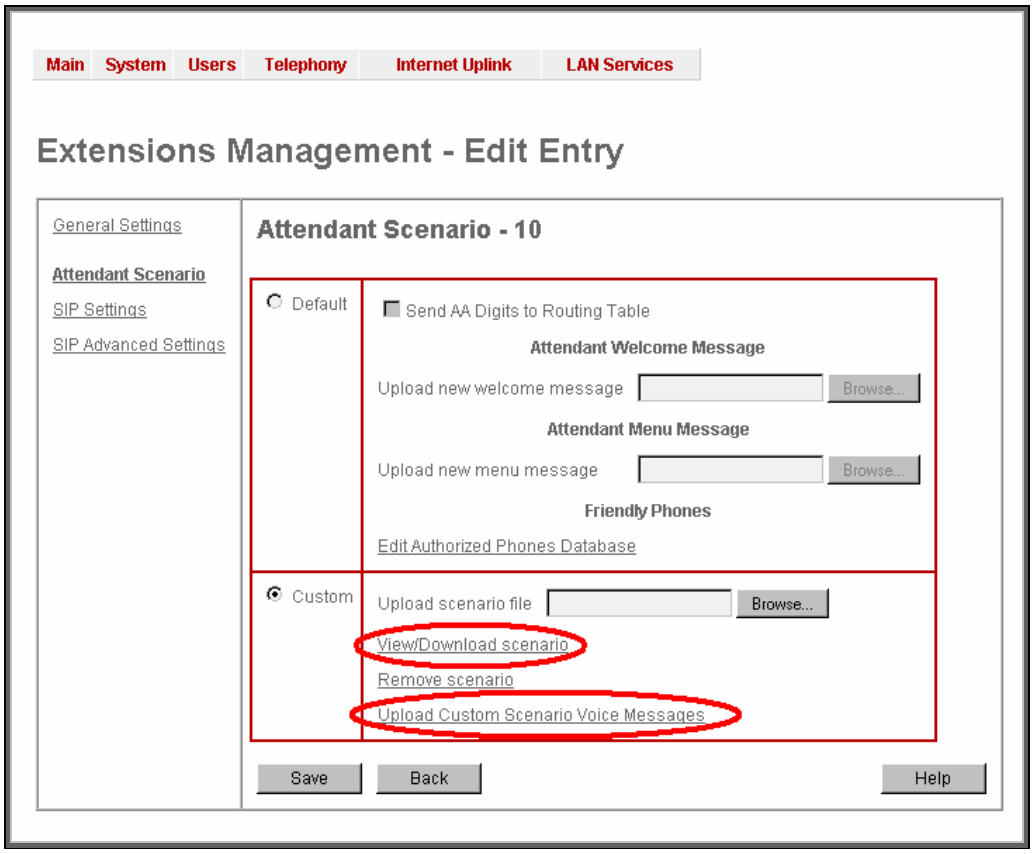


Figure 2 - Attendant with uploaded scenario file

2.1.3 Uploading voice files

Every IVR scenario will also make use of custom recorded voice messages. You can record them yourself with your PC by using a WAV recorder. Microsoft Windows operating systems like WindowsXP, Windows2000, or Wndows98 come with such a tool called the “Sound Recorder”. When recording a voice message, please make sure to save it in the following format so the Quadro can import it: 8KHz, 16bit, mono, and PCM u-law or PCM a-law.

To upload the recorded WAV files click the “Upload Custom Scenario Voice Messages” link (see Figure 2). On the following page (see Figure 3), click the “Browse” button and choose the first voice message on your PC. Then click “Save” on the bottom of the page. Repeat this for all voice messages you need.

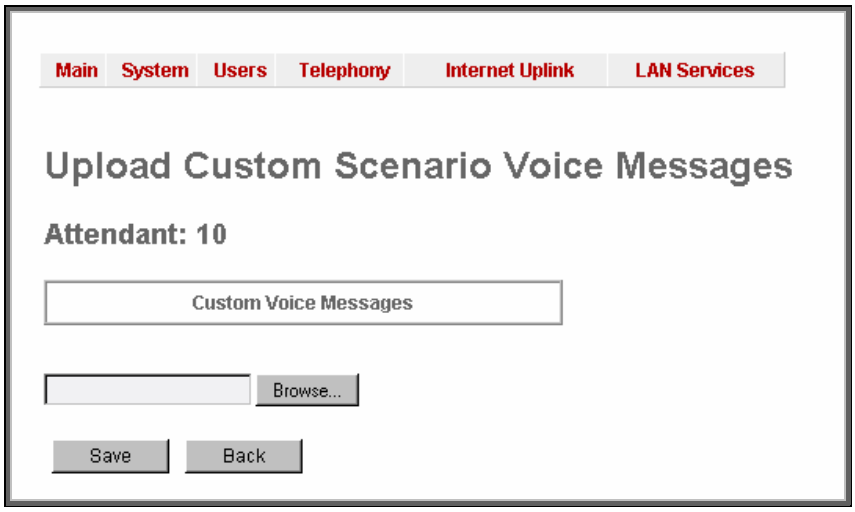


Figure 3 - Uploading voice messages

This page will always show all currently stored voice messages (see Figure 4). You can download and remove them from here. In case you upload a voice file with the same name as an existing one, it will replace the old file. This is only true within one Attendant. You may have different files with the same filename stored in different Attendant scenarios.

Note: In case you want to change an Attendant from IVR to a default scenario, make sure to remove the voice messages and the VXML scenario file first.

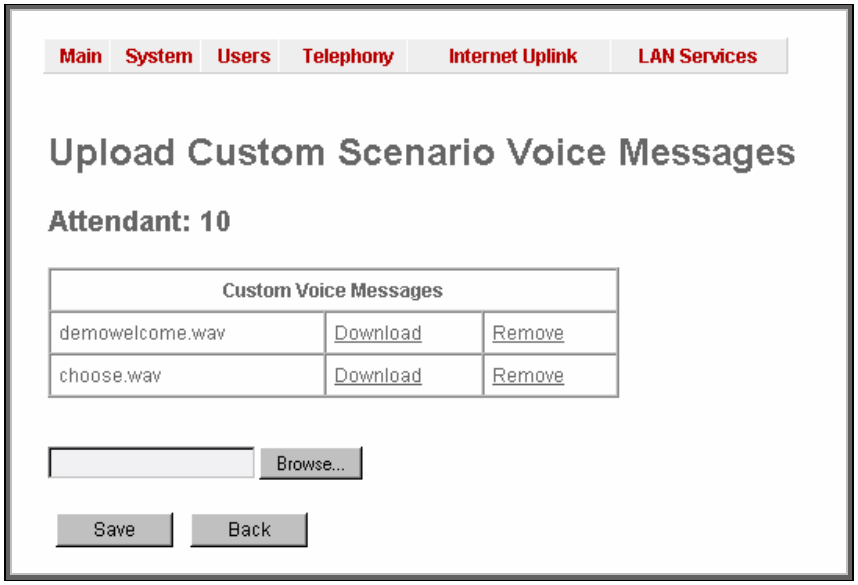


Figure 4 - Uploaded voice messages

2.2 Example 1: One level IVR with 1-digit dial-throughs

Now that we have addressed the general information on how to handle IVR on the Quadro IP PBX, the following is the first example.

2.2.1 Example files

The following files (see Accompanying material) are needed to reproduce this example:

example1.xml	The needed VXML file
demowelcome.wav	“Welcome to demo company”
johnjanejack.wav	“Please dial 1 to reach John, 2 for Jane, 3 for Jack or 4…”

Please note that VXML is case sensitive. All filenames must be lowercase prior to uploading to the Quadro.

2.2.2 Scenario

We want our Attendant 10 to receive all incoming calls on a certain SIP account. The caller should first hear the customized greeting “Welcome to Demo Company” (demowelcome.wav), followed by a customized voice file telling the caller he can reach John, Jane or Jack. The caller can also choose to speak to the receptionist.

We want to give our callers the easiest way of self-dialing through by way of a single digit being sufficient to make the connection. The customized greeting file could therefore read “Please dial 1 to reach John, 2 for Jane, 3 for Jack or 4 to reach a receptionist.” (johnjanejack.wav).

We assume that John has extension 11, Jane has extension 12, Jack has extension 13 and the receptionist can be reached on extension 14 on the Quadro.

In case the caller doesn’t make a choice within five seconds, the possible choices should be repeated. After repeating the choices three times, the call will be terminated by the Attendant to free up system resources.

2.2.3 Resulting VXML

Below you can find the VXML file (example1.xml) that corresponds to the requirements we made above. Please note the line numbers are only printed in this document for easy reference. Line numbers do *not* belong to VXML files.

```
 1  <?xml version="1.0" encoding="UTF-8"?>
 2  <vxml version="2.0">
 3        <noinput count="3">
 4              <prompt>
 5                    <audio src="timeover.wav" />
 6              </prompt>
 7              <exit/>
 8        </noinput>
 9        <form id="mainform">
10              <block>
11                    <prompt bargein="true"> <audio src="demowelcome.wav" /> </prompt>
12                    <goto nextitem="field1"/>
13              </block>
14              <field name="field1" type="digits?length=1" modal="false">
15                    <prompt timeout="5s">
16                          <audio src="johnjanejack.wav" />
17                    </prompt>
18                    <option dtmf="1" value="11" />
19                    <option dtmf="2" value="12" />
```

```
20                      <option dtmf="3" value="13" />
21                      <option dtmf="4" value="14" />
22                      <filled>
23                              <goto nextitem="field2"/>
24                      </filled>
25              </field>
26              <object name="field2" classid="connect">
27                      <param name="extension" expr="field1"/>
28                      <filled>
29                              <prompt> <audio src="timeover.wav" /> </prompt>
30                              <exit/>
31                      </filled>
32                      <nomatch>
33                              <goto nextitem="field1"/>
34                      </nomatch>
35              </object>
36      </form>
37  </vxml>
```

Now let us discuss this example VXML section by section:

- Lines 1 and 2 are the VXML header.
- Lines 3 to 8 define globally what should happen if the caller doesn't press a button after having heard the prompt message three times. `count=3` defines the number of cycles the Attendant should wait before starting the appropriate action.
  - Lines 4 to 7 specify the action in case of "noinput". In our scenario, the Quadro will play back an audio wav file "timeover.wav". (Lines 4-6) This specific file is a system message that doesn't need to be uploaded with the scenario, because it is available to all Custom scenarios. Line 7 uses the `<exit/>` command to tell the Attendant that it should drop the call.
- On Line 9 the actual "program" is started with the use of the `<form>` attribute.
- Lines 10 to 13 define a so-called "block". This block contains an audio file prompt that will be played initially after the Attendant picks up the call.
  - The wav file to be played is named "demowelcome.wav" (Line 11) which is a customized greeting file. So when using this scenario the according .wav file needs to be uploaded to the Attendant scenario for everything to work (see 2.1.3 Uploading voice files).
  - After the file has been played, in line 12 the Attendant is told to jump to the next item name "field1" with the `<goto/>` command.
- In line 14 the use of the `<field>` command defines a caller input with the name of field1 and a type of "digits?length=1". This tells the Attendant that this field expects the caller to dial exactly one digit to fill it.
- Lines 15 to 17 define that an audio wave file should be played prior the caller entering any input. In our case this is the file johnjanejack.wav. Again, this is a customized greeting file that needs to be uploaded to the scenario in order for it to work.
  - Line 15 <prompt timeout="5s"> tells the Quadro to be quiet for five seconds after playing the audio file. After five seconds the same audio file will be repeated.
- Lines 18 to 21 specify the extension to dial depending on the DTMFs the caller presses. `<option dtmf="1" value="11">`, for example, says that the caller should be routed to extension 11 in case he presses the DTMF button 1 on his phone.
- Lines 22 to 24 define from where the program should continue after the caller has "filled" the field, that is, after he dialed a DTMF. In this example the program should jump to "field2".
- Line 25 ends the input field with the `</field>` end delimiter.
- Lines 26 to 35 specify an object of type "connect". This object is an Epygi specific extension of VXML. This "connect" object is responsible for actually transferring the call to different locations. In our scenario the destination of the transfer is an extension.

- Therefore, on Line 27 we define a `<param name="extension">` and fill this parameter with the value of “field1”. In the section above, the user pressing DTMF tones fills in “field1”.
- The `<nomatch>` block below specifies what should happen in case the user presses a DTMF that is not defined above. In our case we simply redirect the program to the input field “field1” (that is to line 14) with the `<goto>` statement.

- Line 36 closes the program with the `</form>` tag.
- Line 37 closes the VXML file.

Please note that VXML strictly follows the XML specifications. Therefore, every VXML file must have the first two lines from the example above. Additionally, every tag must be closed by its according end tag (i.e. `<form…>` needs to be closed by a `</form>`). Single attribute tags can be closed by a trailing / at the end like `<option dtmf="2" value="12" />`.

If you try to upload a file that does not strictly comply, you will receive an error message telling you that the uploaded file is not a valid VXML file.

Also, the Quadro is not able to detect logical errors in your VXML file like endless loops, unused blocks of code or missing audio wave files.

2.2.4 Adjusting the example to your needs

This example can be easily adjusted to similar needs.

Perhaps you have fewer or more extensions you want to reach. Also, you may not want to reach extensions 11, 12, 13 and 14, but 34, 17, 55, 64 and 12. Simply adjust this in lines 18 to 21.

You want to give your callers more time to think. Increase the value of 5s in line 15 for more waiting time after the audio prompt. And/or increase the number of timeouts the Attendant will wait until it hangs up by changing the count value in line 3.

2.3 Example 2: Redirecting to an extension after caller timeout

With the following scenario, we will enhance the above example.

2.3.1 Example files

The following files (see Accompanying material) are needed to reproduce this example:

example2.xml	The needed VXML file
demowelcome.wav	“Welcome to demo company”
johnjanejack.wav	“Please dial 1 to reach John, 2 for Jane, 3 for Jack or 4…”

2.3.2 Scenario

As a customer sensitive organization, our Demo Company wants the undecided caller to be personally assisted. So when a caller doesn’t dial a DTMF, the Attendant should not hang up as in the first example. Instead, the Attendant should transfer the call to a certain extension.

2.3.3 Resulting VXML

Below are the first few lines of the resulting VXML (example2.xml).

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <vxml version="2.0">
3        <noinput count="3">
4              <assign name="field1" expr="'12'"/>
5              <goto nextitem="field2" />
6        </noinput>
```

Everything after line 6 is the same as in Example 1, so it is left not included here.

- Line 3 again defines that a “noinput” exception should take place in case the caller doesn’t select a DTMF button after the audio prompt has been played three times.
- Line 4 and 5 specify the action to take in case of the “noinput” exception. Instead of exiting as in Example 1, this time we fake a user input that normally takes place in the <field> section by using the <assign> method and fill the variable “field1” with the value “12”. Then the program is directed to “field2” which again is the “connect” object described in example 1.

This results in the caller automatically being transferred after 3*5 seconds = 15 seconds of idling after the initial greeting. In our case the caller is transferred to extension 12, which happens to be Jane.

You can use any other existing extension on your Quadro as a destination. You can also use an extension that is not part of the <option…> section.

2.4 Example 3: Adding a multi-level hierarchy and groups

With the following example, our Demo Company has more than four employees. In fact, we have several departments with employees in each department.

2.4.1 Example files

The following files (see Accompanying material) are needed to reproduce this example:

example3.xml	The needed VXML file
demowelcome.wav	“Welcome to demo company”
departments.wav	“Please dial 1 to reach Sales, 2 for Marketing, 3 for Technical…”
sales.wav	“Please dial 1 for Steve or 2 for Jennifer”

2.4.2 Scenario

An incoming caller reaching Attendant 10 will hear the customized greeting message “Welcome to Demo Company” (demowelcome.wav). After the greeting, the caller will be asked to decide which department they want to be connected to: “Please dial 1 to reach Sales, 2 for Marketing, 3 for Technical Support or 4 to reach a receptionist” (departments.wav).

The Demo Company’s sales department consists of two employees, Steve and Jennifer. Therefore, when the user dials 1 to reach Sales he will be asked to decide between the two: “Please dial 1 for Steve or 2 for Jennifer”. Steve has extension 31 and Jennifer has 35.

The marketing department currently consists of one person, so the Attendant should directly transfer to the marketing employee assigned to extension 17.

For the best possible customer satisfaction, the Technical Support group would like to have all their phones ring at once if the caller presses 3 to reach them. For this we assume a Virtual Extension (VE) has been created with number 79. This VE will then have “Many Extension Ringing” activated to all the actual support employees’ extensions. How to set this up is not within the scope of this instructional manual. Please refer to the administrator’s guide or other instructional manuals we provide.)

Finally, by pressing 4 the call will be directed to the receptionist, who is available on extension 14.

In case of a timeout, different things should happen depending on where the caller is currently located. That is, if the caller timed out during the initial choice (departments.wav), they should automatically be transferred to the receptionist. If the caller doesn’t press a DTMF at the Sales sub-selection, they should be transferred to Steve.

2.4.3 Resulting VXML

Below is the VXML (see example3.xml) matching the requirements above.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <vxml version="2.0">
3         <form id="mainform">
4                <noinput count="3">
5                      <assign name="field1" expr="'14'"/>
6                      <goto nextitem="field2" />
7                </noinput>
8                <block>
9                      <prompt bargein="true"> <audio src="demowelcome.wav" /> </prompt>
10                     <goto nextitem="field1"/>
11               </block>
```

```
12              <field name="field1" type="digits?length=1" modal="false">
13                    <link dtmf="1" next="#salesmenu"/>
14                    <prompt timeout="5s">
15                          <audio src="departments.wav" />
16                    </prompt>
17                    <option dtmf="2" value="17" />
18                    <option dtmf="3" value="79" />
19                    <option dtmf="4" value="14" />
20                    <filled>
21                          <goto nextitem="field2"/>
22                    </filled>
23              </field>
24              <object name="field2" classid="connect">
25                    <param name="extension" expr="field1"/>
26                    <filled>
27                          <prompt> <audio src="timeover.wav" /> </prompt>
28                          <exit/>
29                    </filled>
30                    <nomatch>
31                          <goto nextitem="field1"/>
32                    </nomatch>
33              </object>
34        </form>
35        <form id="salesmenu">
36              <noinput count="3">
37                    <assign name="sfield1" expr="'31'"/>
38                    <goto nextitem="sfield2" />
39              </noinput>
40              <field name="sfield1" type="digits?length=1" modal="false">
41                    <prompt timeout="5s">
42                          <audio src="sales.wav"/>
43                    </prompt>
44                    <option dtmf="1" value="31"/>
45                    <option dtmf="2" value="35"/>
46                    <filled>
47                          <goto nextitem="sfield2"/>
48                    </filled>
49              </field>
50              <object name="sfield2" classid="connect">
51                    <param name="extension" expr="sfield1"/>
52                    <nomatch>
53                          <goto nextitem="sfield1"/>
54                    </nomatch>
55              </object>
56        </form>
57  </vxml>
```

There are some similarities to the previous examples. The following are the important differences section by section:

- Lines 4 to 7: The `<noinput>` section has been moved from the global scope down into the first `<form>` section named “mainform”. This means that the specific `<noinput>` section is only valid during the caller being in the “mainform”. By assigning “field1” the value “14” and then jumping to item “field2” in that form we tell the Attendant to redirect the call to extension 14 (our receptionist) in case the caller doesn’t press any buttons.
- Lines 12 to 23 define the input field named “field1” again. This is the same as in Example 1.
  - This time, in line 13 we define a direct link being triggered by a dtmf by using the command `<link dtmf="1" next="#salesmenu"/>`. This tells the Attendant to jump to the object named “salesmenu” in the VXML file instead of further processing this field. (“salesmenu” is defined further down at line 36). This is done to fulfill the requirement of playing a submenu for the sales department.
  - The rest of the field “field1” is similar to Example 1.
- Lines 24 to 33 define the same object of type “connect” as in Example 1.

- Lines 36 to 57 specify a second form with an id of “salesmenu”. By pressings DTMF 1, this menu can be reached from the field section “field1” by pressing DTMF 1 (see line 13).
  - Please note that lines 37 to 40 define a different `<noinput>` section that is only valid for the scope of form “mainform”. In this “noinput” section, the call is being transferred to extension 31 (Steve) by filling the variable “sfield1” with value 31.
  - “sfield1” is defined as an input field on line 41. The concept of that form is exactly the same as the “field1” field on line 12 to 23, except that no direct <link> is defined, but only dtmf options are being used. For a detailed description of the usage, please refer to the VXML of Example 1.

2.4.4 Adjusting the example to your needs

Assume our Demo Company’s marketing department is adding a new employee. The marketing department decides they would also like to add a submenu the same as the sales department currently has. For this we would add an additional <form> section with the id “marketingmenu” by copy and pasting the “salesmenu” form. Then we would simply adjust several values to match the requirements:

- The “noinput” section needs to be changed from 31 to the extension that should receive the out-timed calls.
- The prompt section needs a different wav file other that sales.wav. For example, marketing.wav.
- The <option dtmf…> section needs adjusting to reflect the marketing employees’ extensions according to the new marketing.wav.

Additionally, the mainform’s “field1” section needs to be adjusted. Remove the `<option dtmf=”2”…>` line (line 17) and add a line `<link dtmf=”2” next=”#marketingmenu”>` directly under the `<link dtmf=”1”…>` line.

3 Appendix: Supported VoiceXML Elements

Elements	Attributes	Parents	Children
assign	expr, name	block, catch, filled, noinput, nomatch	*none*
audio	expr, src	prompt	*none*
block	name	form	assign, clear, exit, goto, prompt, reprompt, throw, var
break	time	prompt	*none*
catch	count, event	field, form, menu, transfer, vxml	assign, clear, exit, goto, prompt, reprompt, throw, var
choice	dtmf, event, eventexpr, expr, next	menu	*none*
clear	namelist	block, catch, filled, noinput, nomatch	*none*
else	*none*	block, catch, filled, noinput, nomatch	assign, clear, exit, goto, prompt, reprompt, throw, var
elseif	cond	block, catch, filled, noinput, nomatch	assign, clear, exit, goto, prompt, reprompt, throw, var
exit	expr, namelist	block, catch, filled, noinput, nomatch	*none*
field	expr, modal, name, type	form	catch, filled, link, noinput, nomatch, option, prompt
filled	*none*	field, transfer	assign, clear, enumerate, exit, goto, prompt, reprompt, throw, var
form	id	vxml	block, catch, field, link, noinput, nomatch, transfer, var
goto	expr, expritem, next, nextitem	block, catch, filled, noinput, nomatch	*none*
if	cond	block, catch, filled, noinput, nomatch	assign, clear, exit, goto, prompt, reprompt, throw, var
link	dtmf, event, eventexpr, expr, next	field, form, vxml	*none*
menu	dtmf, id	vxml	catch, choice, noinput, nomatch, prompt, property, script, value
noinput	count	field, form, menu	assign, clear, exit, goto
nomatch	count	field, form, menu	assign, clear, exit, goto
object	classid, name	form	catch, filled, noinput, nomatch, prompt
option	dtmf, value	field	
param	expr, name, type, value	object	
prompt	bargein, count, timeout	block, catch, field, filled, menu	audio, say-as*
reprompt	*none*	block, catch, filled	*none*
say-as*	interpret-as	prompt	value
throw	event, eventexpr	block, catch, error, filled, help, if	*none*
value*	expr	say-as	*none*
var	expr, name	block, catch, filled, form, vxml	*none*
vxml	version	*none*	catch, form, link, menu, script, var

3.1 Appendix: Supported VoiceXML Elements with Descriptions

Elements	Descriptions	Attributes	Descriptions
<assign>	Assigns a value to an existing variable explicitly declared by the <var> element.	**<name>**	The name of the variable being assigned to.
		<expr>	The new value of the variable.

```
<var name="Var1" expr="'value of the first var'"/>
<var name="Var2" expr="'value of the second var'"/>
<form id="Form1">
<block>
      <assign name="Var1" expr="'another value'"/>
      <assign name="Var1" expr="Var2"/>
</block>
</form>
```

<audio>	Plays an audio wave file within a prompt. (8bit, 8Khz, u-law (a-law) files)	**<expr>**	The expr attribute is used when we want to use a generated value rather than an explicit constant expression for our audio file's destination.
		<src>	The name (path) of the audio prompt.

```
<var name="Var1" expr="'attwelcome.wav'"/>
<form id=" Form1">
<block>
      <prompt> <audio src="mysoundfile.wav"/> </prompt>
      <prompt> <audio expr="Var1"/> </prompt>
</block>
</form>
```

<block>	A form-item container of (non-interactive) executable code	**<name>**	The value or 'ID' of the block used for navigational purposes.

```
<form id="Form1">
      <block>
      <prompt> <audio src="mysoundfile.wav"/> </prompt>
      <goto nextitem= "Block2"/>
      </block>
      <block name="Block2">
            <prompt> <audio src="mysoundfile.wav"/> </prompt>
      </block>
</form>
```

<break>	Designate a pause in the audio output.	**<time>**	A value of '5s' would indicate a 5 second pause within the audio output.

```
<form id="Form1">
      <block>
            <prompt> <audio src="mysoundfile.wav"/> </prompt>
            <break time="2s"/>
      </block>
</form>
```

<catch>	Catches an event. The content nested within the catch element will be executed when its particular *event* is thrown.	**<count>**	Allows different <catch> elements to handle the same event based on the number of occurrences of the event. When an event is thrown for the first time, the catch element with a *count* of 1 will be executed.
		<event>	The event or events to catch (separated by space). If the attribute is unspecified, all events are to be caught.

```
<catch count="1" event="noinput">
      <prompt> <audio src="mysoundfile.wav"/> </prompt>
</catch>
```

<choice>	Defines a menu item.	**<dtmf>**	The dtmf key linked to a specific menu choice.
		<event>	The event to be thrown upon a choice match.
		<eventexpr>	An expression evaluating to the event being thrown to the application.
		<expr>	A variable name whose value will be used to determine the dialog ID or document path
		<next>	The dialog ID or document path that the application transitions to upon a user's selection.

```
<menu id="Menu1">
      <prompt> <audio src="mysoundfile.wav"/> </prompt>
            <choice dtmf="1" next="#Form1"/>
            <choice dtmf="2" next="#Form2"/>
            <choice dtmf="3" expr="Var1"/>
            <choice dtmf="4" event="nomatch"/>
            <choice dtmf="5" eventexpr="Var2 "/>
</menu>
```

<clear>	Sets a variable(s) to an undefined value.	**<namelist>**	The space-separated list of variables to be reset.

```
<var name="Var1" expr="'some value'"/>
<form id="Form1">
      <block>
            <clear namelist="Var1"/>
      </block>
</form>
```

<exit>	Terminates the current dialog and returns control to the interpreter.	*none*	

```
<form id="Form1">
      <block>
            <exit/>
      </block>
</form>
```

<field>	Allows the interpreter to collect information from the user.	**<expr>**	The initial value of the element; if this value is 'undefined', (default), then the element will be executed. If not, then the element will not be visited until explicitly set to 'undefined'.
		<modal>	If set to false (the default) all active grammars are turned on while collecting this field; otherwise, only the field's grammars are enabled.
		<name>	The variable name that declares the field-item variable for the dialog.
		<type>	The type of field, i.e., the name of a built in grammar type.

```
<form id="Form1">
	<field name="Field1" type="digits">
		<prompt> <audio src="mysoundfile1.wav"/> </prompt>
	</field>
	<field name="Field2" type="digits?length=2">
		<prompt> <audio src="mysoundfile2.wav"/> </prompt>
	</field>
	<field name="Field3" type="digits" modal="true">
		<prompt> <audio src="mysoundfile3.wav"/> </prompt>
	</field>
</form>
```

<filled>	Specifies actions to perform when input items are filled.	*none*	

```
<form id="Form1">
	<field name="Field1">
		<prompt> <audio src="mysoundfile1.wav"/> </prompt>
		<filled>
			<prompt> <audio src="mysoundfile2.wav"/> </prompt>
		</filled>
	</field>
</form>
```

<form>	A container for all field-items.	**<id>**	The name of the form used to refer to the form within the document or from another document.

```
<form id="F1">
	<var name="Var1" expr="'variable with a form scope'"/>
		<field name="Field1">
			<prompt> <audio src="mysoundfile1.wav"/> </prompt>
		</field>
		<block>
			<prompt> <audio src="mysoundfile2.wav"/> </prompt>
		</block>
</form>
```

<goto>	Transitions application execution to a specific *form* within the current document, or to an entirely separate document.	**<expr>**	An expression that yields the ID of the next dialog to visit.
		<expritem>	An expression that yields the name of the next form item to visit.
		<next>	The ID of the next dialog to visit.
		<nextitem>	The name of the next form item to visit in the current form.

```
<var name="Var1" expr="'Block4'"/>
<form id="Form1">
	<block name="Block1">
		<prompt> <audio src="mysoundfile.wav"/> </prompt>
		<goto nextitem="Block2"/>
	</block>
	<block name="Block2">
		<prompt> <audio src="mysoundfile.wav"/> </prompt>
		<goto expritem="Var1"/>
	</block>
	<block name="Block3">
		<prompt> <audio src="mysoundfile.wav"/> </prompt>
		<goto next="#Form2"/>
	</block>
	<block name="Block4">
		<prompt> <audio src="mysoundfile.wav"/> </prompt>
		<goto nextitem="Block3"/>
	</block>
</form>
```

<if>	Provides a method to utilize conditional logic expressions which allow the developer to change the control flow within the application based on user utterances, variable values, or events.	**<cond>**	The cond attribute specifies any valid expression, which equates to the value to be evaluated.

```
<form id="Form1">
	<field name="Field1">
		<prompt> <audio src="mysoundfile.wav"/> </prompt>
		<filled>
			<if cond="Field1=='abc'">
		<prompt> <audio src="mysoundfile1.wav"/> </prompt>
			<elseif cond="Field1=='def'"/>
		<prompt> <audio src="mysoundfile2.wav"/> </prompt>
			<else/>
		<prompt> <audio src="mysoundfile3.wav"/> </prompt>
			</if>
		</filled>
	</field>

</form>
```

<link>	Allows to easily implement a document or application scoped *grammar* and transition/event handler for a caller's input.	**<dtmf >**	The dtmf key that can be used in conjunction with any other voice grammar specified within the *link* itself to validate a successful grammar match.
		<event>	An event to be thrown to the application.
		<eventexpr>	An expression that evaluates to the event being thrown to the application.
		<expr>	A value that defines the target URI. Either *expr* or *next* may be specified, but not both.
		<next>	The destination URI or form item to transition the caller to when the *link grammar* is matched.

```
<link dtmf="1" next="#Menu1"/>
<link dtmf="2" next="#Menu2"/>
<link dtmf="3" expr="Var1"/>
<link dtmf="4" event="nomatch"/>
<link dtmf="5" eventexpr="Var2"/>
```

<menu>	A dialog for choosing among alternative destinations.	**<dtmf>**	When set to *'true'*, assigns the first 9 menu choices implicit dtmf grammar values, unless the choices already have them explicitly assigned. Defaults to false.
		<id>	The navigational identifier of the form element that allows the menu to be the target of a <goto>.

```
	<menu id="Menu1">
		<prompt> <audio src="mysoundfile.wav"/> </prompt>
	<choice dtmf="1" next="#Form1"/>
	<choice dtmf="2" next="#Form2"/>
	</menu>
```

<noinput>	A shorthand for ***<catch event="noinput">***, handling the *"noinput"* event, thrown when the application expects DTMF input, but has received none.	**<count>**	Allows different <noinput> elements to handle the event based on the number of occurrences of the event. When an event is thrown for the first time, the catch element with a *count* of 1 will be executed.

```
<form id="Form1">
	<field name="Field1">
		<noinput count="1">
			<prompt> <audio src="mysoundfile1.wav"/> </prompt>
			<reprompt/ >
		</noinput>
		<noinput count="2">
			<prompt> <audio src="mysoundfile2.wav"/> </prompt>
		</noinput>
		<noinput count="5">
			<prompt> <audio src="mysoundfile3.wav"/> </prompt>
			<exit/ >
		</noinput>
	</field>
</form>
```

<nomatch>	A shorthand for ***<catch event="nomatch">***, handling the *"nomatch"* event, thrown when the caller inputs a value that is not recognized by any of the active grammars.	**<count>**	Allows different <nomatch> elements to handle the event based on the number of occurrences of the event. When an event is thrown for the first time, the catch element with a *count* of 1 will be executed.

```
<form id="Form1">
	<field name="Filed1">
		<nomatch count="1">
			<prompt> <audio src="mysoundfile1.wav"/> </prompt>
			<reprompt/ >
		</nomatch>
		<nomatch count="2">
			<prompt> <audio src="mysoundfile2.wav"/> </prompt>
		</nomatch>
		<nomatch count="5">
			<prompt> <audio src="mysoundfile3.wav"/> </prompt>
			<exit/ >
		</nomatch>
	</field>
</form>
```

<object>	Exposes implementation platform-specific functionality.	**<classid>**	The name specifying platform dependent object implementation.
		<name>	The variable name that declares the field-item variable for the dialog.

```
<var name="Var1" expr="'somevalue'"/>
<form id="Form1">
	<object name="Obj1" classid="Class1">
		<param name="Param1" expr="'22'"/>
		<param name="Param2" value="22"/>
		<param name="Param3" expr="Var1"/>
	</object>
</form>
```

<option>	Lists choices to the caller.	**<dtmf>**	The dtmf key that a caller may input to activate a particular listed choice.
		<value>	The string to use for the field item return when a user selects a particular option. If not specified, then the *dtmf* value is returned.

```
<form id="Form1">
      <field name="Field1">
            <prompt> <audio src="mysoundfile.wav"/> </prompt>
            <option dtmf="1" value="value1"/>
            <option dtmf="2" value="value2"/>
            <option dtmf="3" value="value3"/>
      </field>
</form>
```

<param>	Specifies the value passed to *objects*.	**<expr>**	An expression defining the value for the parameter.
		<name>	The name of the parameter
		<value>	The string to use for the field item return when a user selects a particular option. If not specified, then the *dtmf* value is returned.

```
<var name="Var1" expr="'somevalue'"/>
<form id="Form1">
      <object name="Obj1" classid="Class1">
            <param name="Param1" expr="'22'"/>
            <param name="Param2" value="22"/>
            <param name="Param3" expr="Var1"/>
      </object>
</form>
```

<prompt>	The prompt element allows the developer to output audio to the caller.	**<bargein>**	Specifies whether ('true') or not ('false') the caller will be able to interrupt the audio output with a dtmf keypress.
		<count>	A number that allows emitting different prompts if the user is doing something repeatedly, For instance, a *prompt count=2* could play more detailed instructions to the caller should his first input proves to be invalid. Defaults to 1,
		<timeout>	Specifies the amount of the time the interpreter should wait for a user response before throwing a *noinput* event. Defaults to 5s (5 seconds).

```
<form id="Form1">
      <field name="Filed1">
            <prompt bargein="false" timeout="5s" count ="1">
                  <audio src="mysoundfile1.wav"/>
            </prompt>
      </field>
</form>
```

<reprompt>	Increases the prompt counter by one and replays the most recent *prompt.*	*none*	

```
<form id="Form1">
      <field name="Field1">
            <prompt> <audio src="mysoundfile.wav"/> </prompt>
            <nomatch>
            <reprompt/>
            </nomatch>
      </field>
</form>
```

<say-as>*	This element allows the developer to specify how a particular value should be rendered.	**<interpret-as>**	The *interpret-as* attribute is used to denote the content type of the value construct. The allowable *interpret-as* values are: • number • digits • date • time • currency duration

```
<var name="Var1" expr="'123'"/>
<form id="Form1">
    <block>
            <prompt>
<say-asinterpret-as="number"><value expr="Var1" /></say-as>
</prompt>
    </block>
</form>
```

<throw>	Triggers an event that can be caught and handled by using the *catch* element.	**<event>**	The name of the event to *throw* to the application.
		<eventexpr>	An expression evaluating to the name of the event being thrown.

```
<var name="Var1" expr="'noinput'"/>
<form id="Form1">
      <block>
            <throw event="nomatch"/>
      </block>
      <block>
            <throw event="Var1"/>
      </block>
</form>
```

<value>*	The *value* element can be used to insert a variable value for say-as element.	**<expr>**	A variable name whose value will be used.

```
<var name="Var1" expr="'123'"/>
<form id="Form1">
    <block>
      <prompt>
<say-asinterpret-as="number"><value expr="Var1" /></say-as>
</prompt>
    </block>
</form>
```

<var>	Declares a VXML variable.	**<expr>**	The initial value of the variable (optional).
		<name>	The variable name.

```
<var name="Var1" expr="'somevalue'"/>
<var name="Var2" expr="'somevalue'"/>
<form id="Form1">
      block>
            <assign name="Var1" expr="'anothervalue'"/>
            <assign name="Var2" expr="Var1"/>
      </block>
</form>
```

<vxml>	The initial declaration that defines a document as a VoiceXML application.	**<version>**	The VoiceXML version number.

```
<vxml version = "2.0">
<var name="Var1" expr="'somevalue'"/>
      <form id="Form1">
            <block>
            <prompt> <audio src="mysoundfile.wav"/> </prompt>
            <goto next="#Menu1"/>
            </block>
      </form>
  <menu id="Menu1">
            <prompt> <audio src="mysoundfile.wav"/> </prompt>
  </menu>
</vxml>
```

4 Appendix: Supported VoiceXML Objects

Objects	Descriptions
checklogin	Verifies login information for specified as param “extension” and “password”.

```
<object name="objfieldname" classid="checklogin">
      <param name="extension" expr="’some extension’"/>
      <param name="password" expr="’some password’"/>
 </object>
```

dial	Dials to the destination (“pattern”) specified as param.

```
<object name="objfieldname" classid="dial">
      <param name="pattern" expr="’some pattern’"/>
</object>
```

connect	Dials to the extension specified as param.

```
<object name="objfieldname" classid="connect">
      <param name="extension" expr="’some extension’"/>

</object>
```

service	Activates service specified as param.

```
<object name=" objfieldname" classid="service">
      <param name="name" expr="'vm'"/>
</object>
```

directory*	Activates directory service.

```
<object name=" objfieldname" classid="directory">
      <param name="roll" expr="'true'"/>
</object>
```

datetime*	Retrieves current date/time. As a result following variables can be used • datetime_sec • datetime_min • datetime_hour • datetime_mday • datetime_month • datetime_year • datetime_wday • datetime_yday

```
<object name=" objfieldname" classid="datetime">
</object>
```

accessrequest*	Sends Radius message.

```
<object name=" objfieldname" classid="accessrequest">
    <param name="account" expr="'myaccount'"/>
    <param name="password" expr="'mypassword'"/>
</object>
```

accountingrequest*	Sends Radius message.

```
<object name=" objfieldname" classid="accountingrequest">
</object>
```

amounttransfer*	Sends Radius message.

```
<object name=" objfieldname" classid="amounttransfer">
    <param name="srcaccount" expr="'accoun1'"/>
                    <param name="destaccount" expr="'account2'"/>
                    <param name="amount" expr="'10'"/>
 </object>
```

* Not supported in Quadro software 3.0.x