



Tutorial Application:

Receptionist

Complexity:

Moderate

Software Used:

Audium Builder for Studio

You Will Learn How To:

- Use standard Action elements
- Use Counter elements
- Use Transfer voice elements
- Use 2_Option_Menu voice elements
- Specify On Call Start and On Call End classes
- Use session variables
- Use Substitution
- Use standard Decision elements with a Java class

Advanced Topic:

- Say It Smart



What It Does

The application asks the user if they would like to be transferred to an office phone, or to a mobile phone. It then retrieves the chosen number from a text file, and executes the transfer.

Note! This application simulates back-end integration to a database by accessing data stored in text files; for an example of true back-end integration please see Audium's advanced sample applications.

How To Do It

Note! If this is your first time using Audium Studio, please read the tutorial for "HelloWorld" before continuing. In this tutorial, it is assumed that the user is familiar with how to create a new Audium Studio project.

1. Create a new Audium project named "Receptionist."
 - a. In the General Settings, choose "Class" from the drop-down menus next to On Call Start and On Call End (see diagram 1).
 - b. In the text field next to On Call Start, type "ReceptionistStart". This is the name of the Java class that will be run when a new call is received. In this case, it will retrieve the number of calls made to the application from num_calls.txt, and store the result in session data.

Note! There are two places where Audium application's can store variables. The first is in element data, which is where an element stores a variable within itself. Other parts of the application can access this element data, but only the element that owns the variable can change it. The second is called session data, which is where a variable is made globally available within a particular call (but not across calls). Other parts of the application can both access *and change* this variable.

- c. In the text field next to On Call End, type "ReceptionistEnd". This Java class will be run whenever the call is disconnected (by either party). In this case, it will increment the number of calls made to the application (which is stored in session data from step b, above), and then write the new value to num_calls.txt.

Note! If you are a programmer, you may have noticed that the behaviors of ReceptionistStart and ReceptionistEnd are not thread-safe (think of each call as a thread). For example, if two calls start within 1 second of each other, they will both read the same value during ReceptionistStart, and output the same value during ReceptionistEnd (rather than incrementing the value). For the purposes of this tutorial, we are not concerned with this issue because we are using these classes only to demonstrate how to setup On Call Start and On Call End. In a production environment, threading issues like this would need to be addressed.

d. Click on Finish.

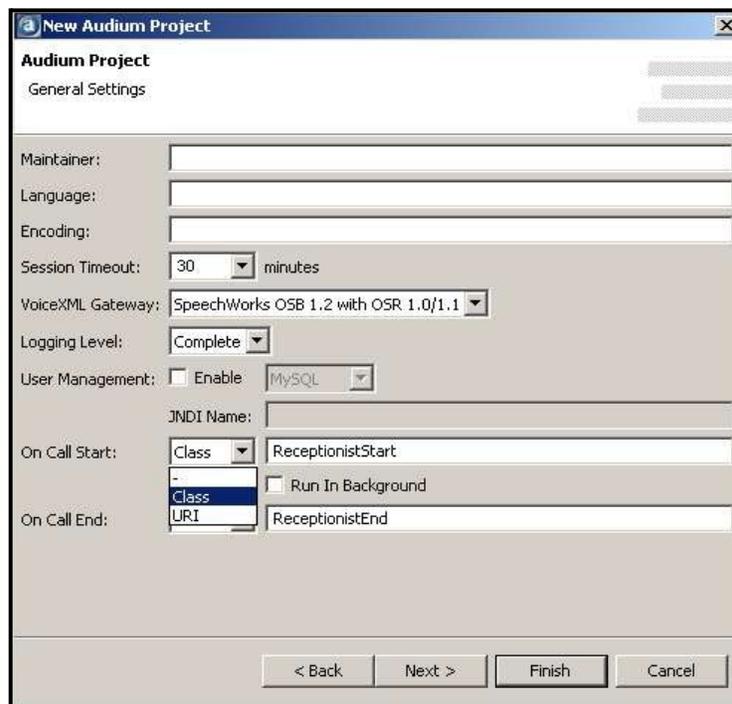


Diagram 1: Setting the On Call Start and On Call End classes

Note! The checkbox next to "Run In Background" on the General Settings dialog controls how the On Call Start class is run. If the box is unchecked (it should be for this application), the call flow will wait for the On Call Start class to complete before starting. If the box is checked, the call flow will start while the On Call Start class executes; the application designer needs to be careful not to use any information the On Call Start class is expected to produce, before it completes its execution.

2. Double-click on Receptionist's app.callflow in the Navigator pane. A new workspace is now visible, including a Call Start element.
3. Drag a 2_Option_Menu voice element into the workspace.
 - a. You can find this element in the Elements pane, under Elements -> Menu.
4. Name the element "OfficeOrMobile".
5. Click on OfficeOrMobile to select it, then click on the Settings tab in the Element Configuration pane.
6. In the +Option 1 DTMF text field, type "1". This makes the first option in the menu activate when the caller presses 1. The plus sign (+) indicates that this option is repeatable, meaning that multiple touchtone values can map to the same result.

Note! To add an additional setting to a repeatable option (those with a red plus sign next to their name), right-click on the option name and choose "add <setting name>". In this example, if you were to right-click on +Option 1 DTMF, you would see "add Option 1 DTMF". This would add another setting where you could define another dtmf value for this choice. For now, we will just use the one setting we already have, and not add any additional ones. To erase any additional ones you created, right-click on them and choose "delete <setting name>" (e.g. "delete Option 1 DTMF").

7. In the +Option 1 Voice text field, type "office". This makes the first option in the menu activate when the caller utters "office".
8. In the *Option 1 Value text field, type "office". This means that whether the caller pressed 1 or uttered "office", the official result of the choice will be the string "office". This is a required setting (notice the red star next to the name) that is used to identify the choice the caller made later on in the application. This value is stored in element data representing the caller's choice.

Note! If you do not configure all required settings (those with a red star next to their name), you will receive errors when you attempt to validate and/or deploy your application. Errors are displayed in the Tasks pane at the bottom of your screen. Double-clicking on an error message in that pane will jump to the problem element, and the error message will describe the exact issue.

9. In the +Option 2 DTMF text field, type “2”. This sets the second touchtone option.
10. In the +Option 2 Voice text field, type “mobile”. This sets the second voice option.
11. In the *Option 2 Value text field, type “mobile”. This is the value for the second option (see diagram 2).

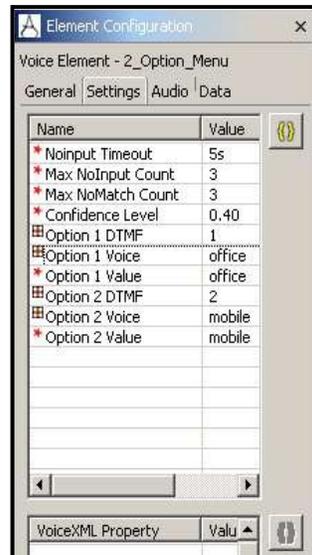


Diagram 2: Finished entering the OfficeOrMobile element's settings

12. Click on the Audio tab, and set the audio for this element.
 - a. Set the initial audio item to play the TTS string "Hello, I am the receptionist. Do you want to try the office phone or the mobile phone? For the office phone, say office or press 1. For the mobile phone, say mobile or press 2."
 - b. You can optionally set the nomatch and noinput audio items (3 each as per the Settings tab). For details on how to do this, please refer to the AgeIdentification tutorial.
13. Drag an Action element into the workspace.
 - a. You can find this element in the Elements pane, on the same level as the "Elements" folder (i.e. not in a subfolder).

Note! This type of Action element is often referred to as a "standard Action element". As we will see in a moment, it must be assigned a custom behavior each time it is added to an application, and has no default functionality.

14. Name the element "GetNumber".

15. Select the element, and choose "Class" from the drop-down menu in the the Element Configuration pane. This setting means that the Action element will perform its action by using a Java class.

16. In the text field next to the drop-down menu, type "ReceptionistGetNumber". This is the name of the class that this Action element will execute. This class will retrieve the requested number (as dictated by the caller's choice, which is held in OfficeOrMobile's element data) from a text file and store it in element data, in a variable named "NumberToCall".

17. Connect the following exit states (see diagram 3):

- a. Connect Call Start's only exit state (next) to OfficeOrMobile.
- b. Connect OfficeOrMobile's option1 exit state to GetNumber.
- c. Connect OfficeOrMobile's option2 exit state to GetNumber.
- d. Connect OfficeOrMobile's max_nomatch and max_noinput exit states to a Hang Up element (you will need to drag one into the workspace). Note that in a production application, you would not likely hang up on a caller that was having problems. This is done for the sake of brevity in this tutorial.

Note! In most cases, the exit states of a 2_Option_Menu element will point to different places because the actions to take are different for each choice. In this case, we always want to retrieve the number after the user makes a choice (regardless of what the choice was). The action element's class will look at the choice the user made and retrieve the appropriate number.

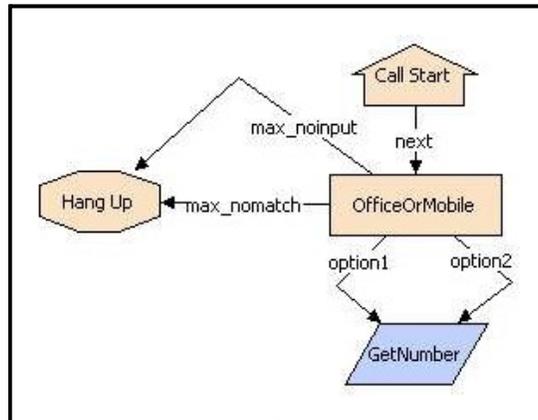


Diagram 3: The call flow so far

18. Drag a Decision element into the workspace. This element will be used to decide if the number that was retrieved by GetNumber is a valid 10-digit number.
 - a. You can find this element in the Elements pane, on the same level as the "Elements" folder (i.e. not in a subfolder).

Note! This type of Decision element is often referred to as a "standard Decision element". As we will see in a moment, it must be assigned custom logic rules each time it added to an application, and has no default decision functionality.

Note! When dealing with external data sources (e.g. databases, text files, etc.) there is always the possibility of unexpected data and data corruption. It is a good practice to validate data from external sources before using it.

19. Name the element "ValidateNumber".
20. Select ValidateNumber, and choose "Class" from the drop-down menu in the the Element Configuration pane, just as we did for GetNumber.
21. In the text field next to the drop-down menu, type "ReceptionistValidateNumber". This is the name of the class that this Decision element will execute to determine which exit state to follow.
22. Add two exit states to ValidateNumber (for details on how to setup exit states on Decision elements, see the AgeIdentification tutorial). Remember that exit state names are case sensitive.

- a. Name the first one "valid". This exit state will be followed if the data retrieved from a text file is a 10 digit integer.
 - b. Name the second one "invalid". This exit state will be followed if the data retrieved from a text file is not a 10 digit integer.
23. Connect GetNumber's done exit state to ValidateNumber.
24. We are already tracking the total number of calls our application receives using On Call Start and On Call End classes, but we would also like to keep track of the number of transfers a particular caller makes during a single call. Drag a Counter element into the workspace.
 - a. You can find this element in the Elements pane, under Elements -> Calculation.
25. Name the element "TransferCount".

Note! A Counter element is a pre-built Action element. Notice that we do not have to specify a class for it to work, it already has built-in functionality. It is an Action element because it performs a task and does not generate vxml (and thus does not have an Audio tab).
26. Leave all of the default settings in place for TransferCount. The default settings cause it to start counting at 0 and increment by 1 each time it is visited, and this is the behavior we want.
27. Connect ValidateNumber's exit state named valid to TransferCount, since we want to count valid transfers.
28. Drag a Transfer voice element into the workspace.
 - a. You can find this element in the Elements pane, under Elements -> Call Control.
29. Name the element "Transfer".
30. Select the Transfer element, and view its settings in the Element Configuration pane. Notice that one of the required settings, Transfer Destination is not set by default. This is where the 10 digit number to transfer to needs to be entered. Since this data is not known

until runtime (remember, the number will be retrieved from a text file by the GetNumber element), we will use substitution to fill in this setting.

Note! Substitution allows you to fill text fields with directions for how to find some desired data at runtime (this is available on text fields with a substitution button). In our example, we do not have the telephone number before runtime, so we will use substitution to tell the setting to find the number in GetNumber's element data, stored in a variable named NumberToCall (refer to step 16 to refresh your memory about this element). For more details on substitution, please refer to the User Guide.

31. Click in the Value column of the Transfer Destination setting, and click on the substitution button to the right. It looks like two yellow braces (see diagram 4).

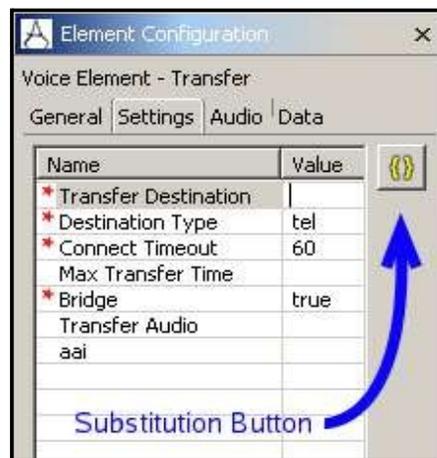


Diagram 4: The location of the substitution button

32. The Substitution Tag Builder will appear, this is where you construct substitutions. In this case, we want to substitute the NumberToCall variable from the GetNumber element, since this is where the telephone number will be stored at runtime.

- Click on the Element Data tab (this is the tab that is selected by default).
- In the Element drop-down menu, choose GetNumber.
- In the Element Data text field, enter "NumberToCall" (case sensitive).
- Click the Add Tag button to generate the substitution (see diagram 5).

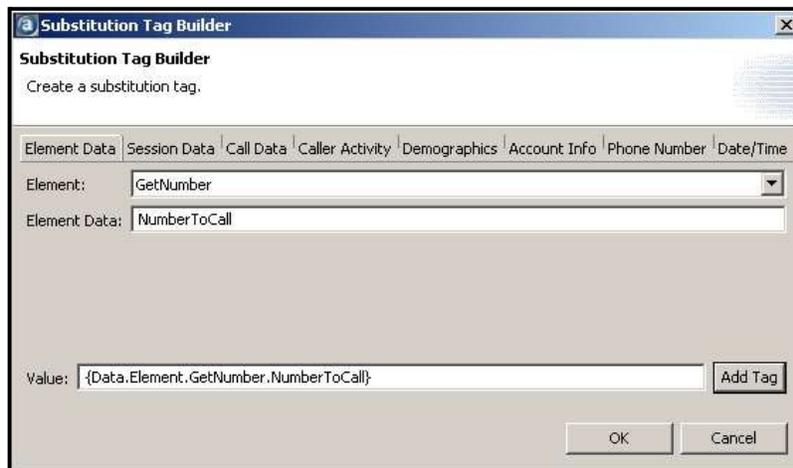


Diagram 5: Just clicked the Add Tag button

Note! It is a common error to inadvertently skip step d above, and click OK instead. If you skip this step, the substitution string will not be added to the setting, and your application may encounter problems at runtime (or it may not even validate and deploy).

e. Click the OK button.

Note! Substitution strings (the string you see in the Value text field in diagram 5) can be edited by hand, but it is a best practice to use the Substitution Tag Builder instead, to avoid errors.

33. Click on the Audio tab, we will now configure some audio groups.
 - a. Notice that there are no audio groups present by default (unlike Audio elements). We will define many of the optional Audio groups for this Transfer element; it is a best practice to give the user feedback on what is occurring during the call.
 - b. Right-click on Audio Groups, and choose each item listed under Add Audio Groups -> Transfer Audio. You should now see the following audio groups:
 - i. Initial - audio that is played before the transfer occurs
 - ii. Busy - audio that is played if the transfer fails due to a busy signal
 - iii. NoAnswer - audio that is played if the transfer fails due to no answer
 - iv. Phone Error - audio that is played if the transfer fails due to a telephony error
 - c. Notice that each audio group has one audio item by default. Configure the

audio items for Busy, NoAnswer, and Phone Error to play informative messages of your choice.

34. For the Initial audio group, we want to tell the user "Great, for transfer number x let me try dialing xxx-xxx-xxxx." The x's are not known until runtime, and so will be filled in with substitution.

a. Add three new audio items to the Initial audio group, so that there are 4 total. Do this by right-clicking on Initial and choosing Add Audio Item three times.

Note! An audio group contains one or more audio items. At runtime, the audio items within an audio group are concatenated and played in succession.

b. Set the TTS text of the *first* audio item to be "Great, for transfer number".

c. Set the TTS text of the *third* audio item to be "let me try dialing".

d. Set the TTS text of the *second* audio item. We will use substitution for the number of transfers (which is stored in the TransferCount element, refer to step 24 to refresh your memory). Click the substitution button to the right of the TTS text box to bring up the Substitution Tag Builder. On the Element Data tab, choose the element TransferCount from the drop-down menu, and enter "count" in the Element Data text field. Click the Add Tag button, and then click the OK button. Notice that the substitution string is now in the TTS text box. This will be replaced by the transfer count at runtime.

Note! You can find out where each of the pre-built elements stores its data (i.e. under what variable name), in the Element Specifications manual. Note that the Counter element stores the count in its element data, with the variable name "count". On the other hand, since our GetNumber element uses a custom class, the programmer that made that class chose the variable names it will use. In this case, they chose to store the telephone number in a variable named "NumberToCall".

e. For the *fourth* audio item, we will again use substitution, this time for the telephone number. Repeat step d, but this time set the TTS text for the fourth audio item, and substitute the variable named "NumberToCall" from the element named "GetNumber".

Note! The output of this audio group could be accomplished with a single audio item, since all of our audio is expressed through TTS. For example, we could have the entire phrase "Great for transfer number x, let me try dialing xxx-xxx-xxxx." in one audio item, and simply replace the transfer number and telephone number with substitutions. However, it is a best practice to split long audio groups into multiple audio items. This allows reusable custom audio to be recorded for the static parts (e.g. "Great for transfer number"), and either Say It Smart or standard TTS to be used for the dynamic parts.

35. Drag a new Audio element to the workspace, and name it "InvalidNumber". Configure it to play the TTS text "Sorry, the phone number you wish to call is invalid. Good bye."

36. Drag another Hang Up element to the workspace.

37. Connect all of the unsatisfied exit states (see diagram 6):

- b. Connect the invalid exit state of ValidateNumber to InvalidNumber.
- c. Connect the done exit state of InvalidNumber to the new Hang Up element.
- d. Connect the done exit state of TransferCount to Transfer.
- e. Connect the busy, noanswer, and phone_error exit states of Transfer to the new Hang Up element. Note that this is done for simplicity, and is not a best practice.
- f. Connect the done exit state of Transfer to OfficeOrMobile (so the caller can be transferred multiple times within a single call, if each transfer is successful).

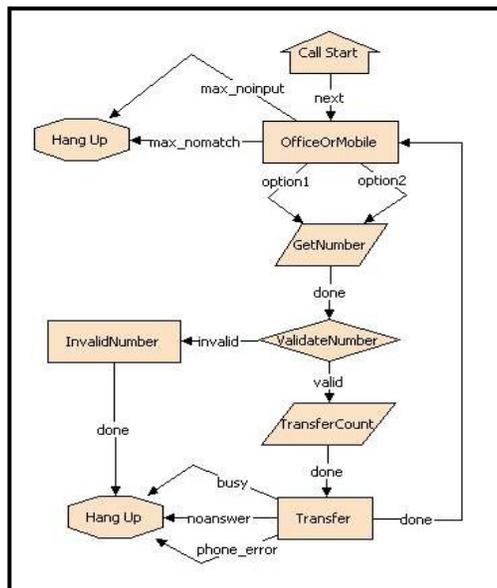


Diagram 6: The complete call flow

38. The Java classes used by this application (ReceptionistStart.class, ReceptionistEnd.class, ReceptionistGetNumber.class, and ReceptionistValidateNumber.class) can be found in the pre-built Receptionist tutorial application. They are in the deploy\java\application\classes directory of the application.

Note! If you have not already downloaded the pre-built Receptionist application, you cannot continue beyond this point.

39. Copy these class files into the Receptionist -> deploy -> java -> application -> classes folder in your Navigator pane by dragging the files directly into it (see diagram 7). You should now see the class files listed under the classes folder in your Navigator pane.

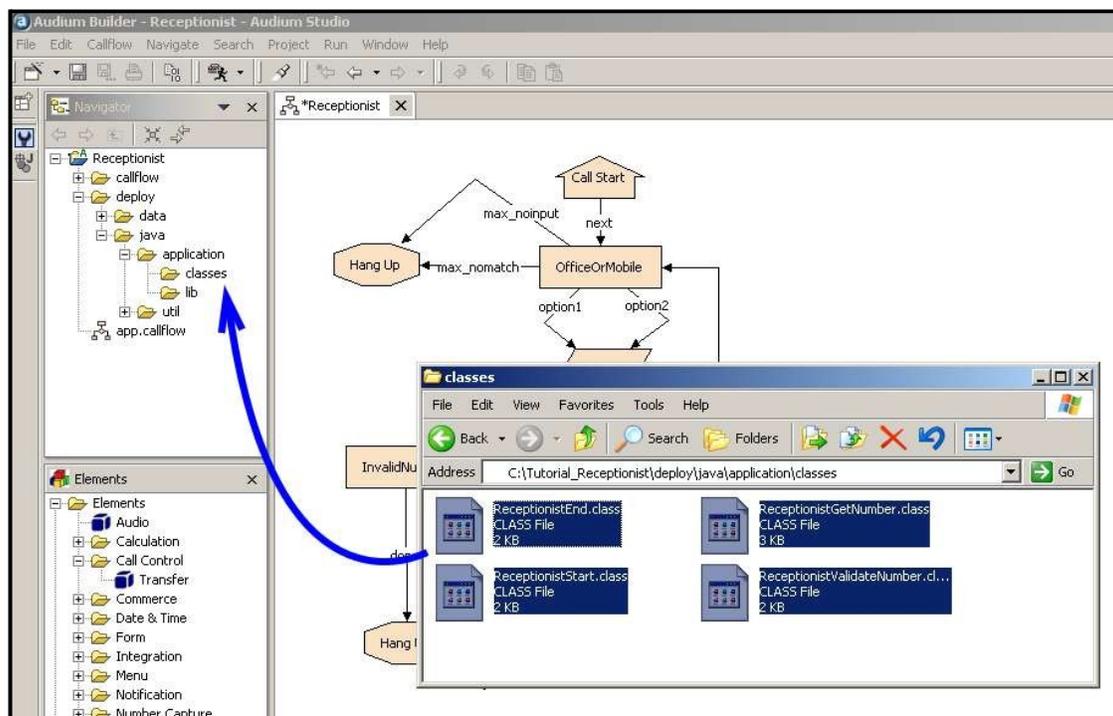


Diagram 7: Dragging class files into the project

40. At runtime, the application will attempt to access some external text files, so they need to be created.
- Create num_calls.txt in the text editor of your choice with the number "0" in it (and nothing else).
 - Create phone_office.txt, with a 10 digit telephone number in it, and nothing else (no spaces or punctuation are allowed, just the 10 digits).

- c. Create phone_mobile.txt, with a 10 digit telephone number in it (follow the guidelines in step b).
41. Copy these three new text files into the Receptionist -> deploy -> data -> misc folder in your Navigator pane (refer to step 39 for details).
42. Your application is complete. You can now test it on Call Services (refer to the HelloWorld tutorial for instructions).

Advanced Topic

In the application we just created, we used text-to-speech to read back all dynamic content (i.e. numbers). In a production application, it is preferable to use pre-recorded audio to improve the caller's experience. Additionally, when the TTS engine reads back the phone number before the transfer takes place, it will probably read it in an unusual fashion, such as "five hundred fifty-five billion..." (depending on your TTS engine). It would be better to have the number read back digit-by-digit, with pauses after the third and sixth digits, the way a natural speaker would read it.

Audium's Say It Smart plugins allows all of this to happen, and more. We will change the fourth audio item of the Transfer element to use Say It Smart. For the sake of simplicity, we will not record audio for the application, but even using Say It Smart with TTS vastly improves the audio output. No longer will the telephone number be read back as a whole number, but it will now be read as digits with pauses.

Follow these steps to make the change:

1. Select the fourth audio item of the Transfer element's initial audio group (the same audio item we were working with in step 34 part e of the tutorial).
2. Click on the "Say It Smart" radio button (see diagram 8). This changes the audio item from normal audio to Say It Smart audio.

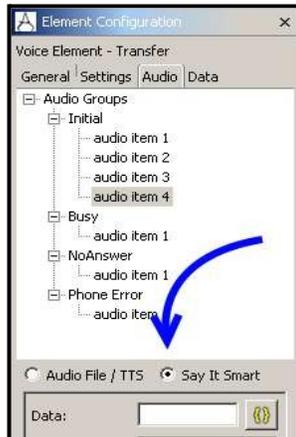


Diagram 8: Configuring an audio item to use Say It Smart

3. In the Data text field, insert a substitution string that refers to the GetNumber element's NumberToCall variable.
4. Change the Type drop-down menu to Phone Number. This will cause the number to be read with pauses after the 3rd and 6th digits.
5. Leave the other settings on their defaults.

Note! When you use another type of Say It Smart plugin (other than Phone Number), you need to make sure that the Input Format and Output Format are set correctly. For the Phone Number plugin, there is only one option for each, so we did not have to set them. The Input Format is how the data is stored in your application (e.g. 10 digits), and the output format is how it will be read back (e.g. digits with pauses).

6. You are done. This audio item will now read back the phone number as "d d d <pause> d d d <pause> d d d d" (where d is a digit).